-- --

Fault Management for Realtime Networks

by

Anindo Banerjea

B.Tech. (Indian Institute of Technology, New Delhi) 1989

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Domenico Ferrari, Chair
Professor Pravin P. Varaiya
Professor Richard E. Barlow

1994

-- --

The dissertation of Anindo Banerjea is approved:

Chair Date

Date

Date

University of California at Berkeley

1994

--

--

Fault Management for Realtime Networks

by

Anindo Banerjea

Abstract

Fault Management for Realtime Networks

by

Anindo Banerjea

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Domenico Ferrari, Chair

Realtime networks provide guaranteed performance communication to applications that require such guarantees. The Tenet Group, under Professor Domenico Ferrari, has developed a scheme to provide such Quality of Service (QoS) guarantees in a packet-switched internetwork. The scheme is based on the concept of the *realtime channel*, which is a network connection with associated traffic specifications and performance guarantees. Other schemes to provide realtime services exist, and share some fundamental similarities. However, none of these schemes address the problem of how to restore (or continue to provide) realtime service in the presence of network faults. This dissertation addresses the problem of dealing with faults in the context of realtime networks, using two classes of mechanisms: *proactive* and *reactive*.

Reactive mechanisms can be used to reroute realtime channels to surviving links in the network. This approach does not use any extra resources in the absence of faults, but involves a disruption of the service while the recovery action is being performed. In addition, some channels may not be successfully rerouted if the realtime load on the network is very high. Proactive schemes may be used to reserve redundant resources (such as on multiple paths in the network) so that there is no disruption (or only disruption that can be bounded *a priori*), as long as the fault scenario is one that is *covered*. The proactive scheme may be designed to cover fault scenarios such as single faults, double faults, and so on.

The first part of the dissertation describes the rerouting schemes for fault recovery. We start with a very general fault recovery model, and systematically refine our design till we are left with a well-structured and limited problem domain, which we explore through simulation. The simulation experiments were used to identify the best scheme within the chosen solution space, and to show that its performance is reasonable on a number of metrics of performance, such as speed, amount of traffic rerouted, and efficiency of resource usage, as well as for large variations in the external factors such as network load, fault scenario, traffic mix, and network topology. The dissertation also contains a high-level design of a protocol for fault recovery of Tenet realtime channels, the Real-Time Control Message Protocol, based on the results of the experiments.

The dissertation also describes the use of dispersity routing and forward error correction to provide fault tolerance for realtime channels. These techniques are used to design a variety of schemes that deliver various levels of service in the presence of restricted network faults. Some schemes provide transparent tolerance to multiple faults in the network, at the cost of increased network resource requirements; some others use no extra resources, as compared to a simple realtime channel, but the service degrades gracefully when a network failure occurs; still others suffer a total disruption in the event of a failure, but the duration of the disruption is limited to the time needed to notify the source of the fault. These mechanisms are end-to-end in nature, permitting implementation on top of a basic realtime service, as long as appropriate support for routing is provided by the network. The services are validated and the cost to the network is evaluated through simulation. The techniques are also compared to existing mechanisms that provide fault tolerance to realtime networks.

Committee chair: ______________________________

Professor Domenico Ferrari

--

--

To my parents, Maya and Bidyut.

Table of Contents

Table of contents iv
List of Tables vi
List of Figures vii
Acknowledgements ix
Chapter 1: Introduction 1
1.1 The realtime channel paradigm 1
1.2 General requirements from the control mechanisms 4
1.3 Two classes of control mechanisms 5
1.4 Previous work 10
1.5 Scope of the dissertation 20
Chapter 2: The Tenet Realtime Protocol Suite 1 22
2.1 Introduction 22
2.2 Network model and assumptions 22
2.3 Service model and assumptions 23
2.4 The Tenet Suite 1 26
2.5 Channel management 28
2.6 Other realtime schemes 35
2.7 Conclusions 36
Chapter 3: A reactive scheme for fault recovery 38
3.1 Introduction 38
3.2 Objectives of the fault recovery mechanisms 39
3.3 Network model and assumptions 40
3.4 A fault recovery framework 44
3.5 Design issues for a reactive scheme for fault recovery 45
3.6 Issues not to be investigated 53
3.7 Remaining issues 54
3.8 Conclusion 64
Chapter 4: Evaluation of Fault Recovery 65
4.1 Introduction 65
4.2 Simulation design 65
4.3 Results 88
4.4 Conclusion 104
Chapter 5: High level design of a fault recovery protocol 111
5.1 Introduction 111
5.2 Summary of design decisions 112
5.3 Description of the protocol 114

5.4 Changes to the RCAP+DCM software 122
5.5 Conclusions 123
Chapter 6: A proactive scheme for fault tolerance 125
6.1 Introduction 125
6.2 Background 127
6.3 Schemes for fault tolerance 135
6.4 Simulator design 151
6.5 Simulation results 155
6.6 Implementation issues 177
6.7 Conclusions 180
Chapter 7: Conclusions 186
7.1 Introduction 186
7.2 Summary of the dissertation 186
7.3 Requirements re-examined 188
7.4 Contributions 189
7.5 Future work 191
Bibliography 193

List of Tables

Table	Label	Page
2.1	Performance parameters of the Tenet Suite 1 interface	25
2.2	Traffic parameters of the Tenet Suite 1 interface	25
2.3	RCAP messages	30
4.1	Load index levels	85
4.2	Properties of the fault recovery scheme selected	109
6.1	Properties of various dispersity systems	181
6.2	Properties of some more dispersity systems	181

-- --

List of Figures

Figure	Label	Page
1.1	Link vs. Global rerouting	16
1.2	Dispersity routing - (3,2,2) system	18
2.1	Service model for realtime network	24
2.2	Protocols in the Tenet Suite 1	26
3.1	Avoiding collisions	51
3.2	The sector of the solution space to be considered	52
3.3	4-dimensional solution space to be explored	55
3.4	Local rerouting	56
3.5	Valid regions of the 4-dimensional solution space	63
4.1	Rate Control Static Priority scheduler	71
4.2	Properties of the load index.	76
4.3	Square mesh topology	81
4.4	Square mesh topology with trunks	82
4.5	Core topology	83
4.6	Effect of timing on (i) reroute success (ii) time to reroute	88
4.7	Effect of locus of reroute on (i) reroute success (ii) excess resources used	90
4.8	Effect of retry policy. (i) Immediate retries (ii)- (iv) Delayed retries	92
4.9	Effect of retry interval on the histogram of time to reroute (i) Exponential (ii) Fixed (100 ms)	94
4.10	Effect of state prediction on (i) time to reroute (ii) realtime load rerouted on first attempt	95
4.11	Effect of state prediction on histogram of time to reroute (i) Local prediction (ii) Global prediction	97
4.12	Effect of number of faults on (i) reroute success (ii) success of prediction (iii) time to reroute (iv) excess resources used	99
4.13	Effect of new load mix on (i) Success ratio (ii) Time to reroute (iii) Histogram of time to reroute	101
4.14	Effect of topology. (i)-(ii) Mesh with added trunks (iii)-(iv) Core topology	103
4.15	Effect of retry interval on core topology (i) 100 ms interval (ii) 60 ms interval	104
5.1	Interactions for RTCMP	112
5.2	Hysteresis loop to avoid oscillations	116
6.1	One set of isolated link failures which the IFI channel can survive	151
6.2	No IFI set of links exists in a square mesh topology	152
6.3	Structure of reassembly buffer for N=4	154
6.4	(3,2,1) dispersity system simulated	157
6.5	Packet loss vs. loss rate for dispersity systems with N-K=1, no network failure case	160
6.6	Packet loss vs. loss rate for dispersity systems with N-K=1, network failure case	164
6.7	Packet loss vs. loss rate for dispersity systems with N-K=2, network failure case	164
6.8	Number of connections established for dispersity systems without redundancy	167

Figure	Label	Page
6.9	Effect of allowing S = 2 on dispersity systems without redundancy	171
6.10	Number of connections established for dispersity systems with N-K=1	171
6.11	Effect of allowing S = 2 on dispersity systems with N-K=1	174
6.12	Number of connections established for dispersity systems with a N-K=2	174
6.13	Effect of allowing S = 2 on dispersity systems with N-K=2	175
6.14	Equivalent dispersity systems with no shared links and N-K=1	175

Acknowledgements

First of all I would like to acknowledge gratefully the invaluable guidance and support that I have received from my advisor, Professor Domenico Ferrari. He provided me with the freedom to explore my own interests, and at the same time with the guidance to ensure that I did not waste my energies on unproductive avenues of research. In addition, his detailed technical and stylistic criticisms of a draft of this dissertation caused me to rewrite major parts of it, and improved its coherence and readability immensely.

I would like to thank Professors Pravin Varaiya and Richard Barlow, for serving on my qualifying examination and dissertation committees, and for their helpful comments on the thesis. I also thank Professor Alan Smith for chairing my qualifying examination committee.

Part of the work presented in Chapter 4 of this thesis was performed in cooperation with Colin Parris. The simulator used in the experiments was developed initially by Hui Zhang, and modified by Colin Parris. Many invaluable discussions with them, as well as with other members of the Tenet Group, contributed to the ideas presented here. I would specifically like to thank Amit Gupta, Ed Knightly, Bruce Mah, Steve Mccanne and Mark Moran for allowing me to use them as sounding boards for many ideas, and Dr. Andres Albanese for many fruitful discussions about network reliability. I fondly remember many debates about *Life, the Universe, and Everything* with various members of the Tenet Group, and the Thursday night beer gatherings with the Dave, Krste and the Realization boys. I gratefully acknowledge Keshav Srinivasan and Dinesh Verma for influence and advice during my early days at Berkeley, especially for advising me to join the Tenet Group, and for continued interaction even after they left Berkeley.

Finally, I am indebted to my parents for the emotional support to persevere in my undertakings, to my brother for his ready ear and quick wit, and to Karen for her love.

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

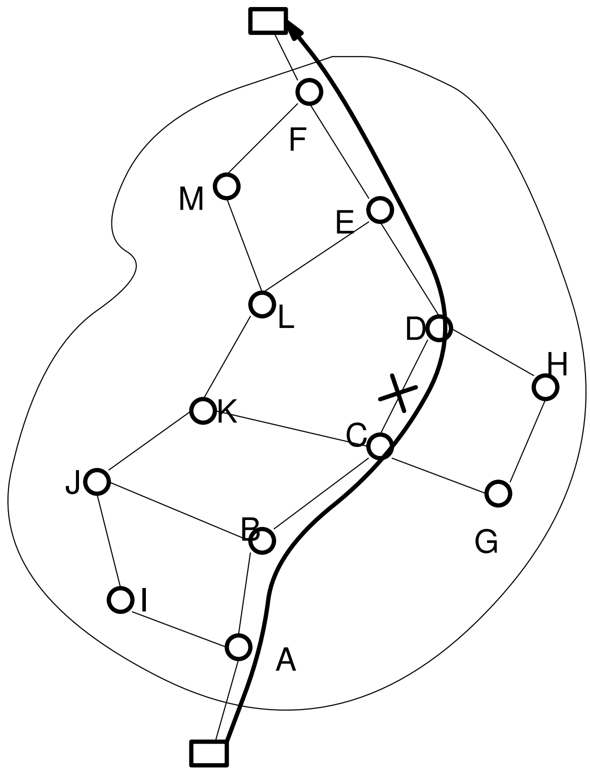

Figure 1.1: Link vs. Global rerouting

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, pre-computed, 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.

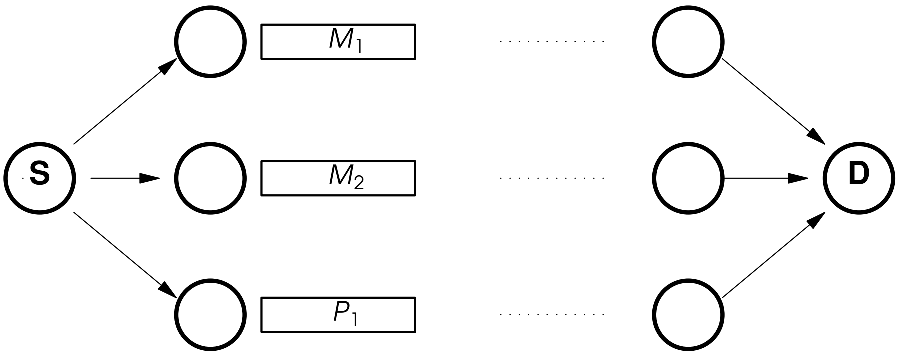

Figure 1.2: Dispersity routing - $(3,2,2)$ **system**

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.

Chapter 2: The Tenet Realtime Protocol Suite 1

2.1. Introduction

This chapter describes the Tenet Realtime Protocol Suite 1. In this process, it presents many of the ideas of the realtime paradigm by example. Our research into fault management is based upon this paradigm. Therefore, this background information is essential, in order to place the issues involving fault management for realtime networks in the correct context. The simulation evaluation presented in Chapter 4 and the high level protocol design present in Chapter 5 are also based on the Tenet suite.

We first present in Section 2.2 the assumptions about the underlying packet-switched network, on which the Tenet model is based. We also present the service model in Section 2.3, which is based on the idea of a contract between the network and the client. The basic approach to providing realtime guarantees, given the assumptions made, is described Section 2.4. Section 2.5 discusses channel management in further detail, and describes two specific control protocols, designed for the Tenet Suite 1, which provide different levels of network management support. We also describe the support for routing needed for realtime networks. Finally, Section 2.6 surveys some other realtime schemes, either paper designs or implementations, to see if the mechanisms to solve the problems of realtime communication differ vastly. These observations will form the basis of our arguments, presented in the next chapter, that the basic principles of realtime communications are fairly uniform across implementations, and fault recovery mechanisms based on these fundamental principles should be applicable to all such networks.

2.2. Network model and assumptions

Realtime communication can be provided on a general packet/cell switching store-and-forward network if some assumptions are satisfied. The primary assumption is some form of scheduling which supports differential grades of service on the switches or routers (henceforth called nodes) of the network. The first implementation of the Tenet

Suite 1 is based on EDD (Earliest Due Date) scheduling. Guarantees may also be provided based on other scheduling schemes such as RCSP (Rate Controlled Static Priority) (72) and multiple priority versions of FCFS (29); even networks with nodes running different scheduling disciplines can be supported (29).

The nodes of the network must be connected by links that have boundable delays. They might be physical links on which the delay is bounded by the laws of physics. They might be sub-networks on which the delay is bounded by some other mechanism, such as SONET ATM networks. They might also be sub-networks on which the delay is bounded using a hierarchic application of the realtime network protocols (7).

The nodes of the network must also have sufficient computing power and programmability to run an establishment protocol that includes admission control tests of moderate complexity. For switches that do not have such programmability, we assume the presence of a controller which can perform these tests and run the establishment protocol.

For the purposes of this thesis, we impose the additional assumption that the network is simple, i.e., each link in the network is a physical link and does not represent a sub-network. This simplifies the explanation of the protocols significantly. For details of how to extend the realtime paradigm to internetworks, the reader is referred to (29).

2.3. Service model and assumptions

We assume a service model based on a server-client relationship between the network and the application. We use the word *client* to refer loosely to the application or the human user running the application. We assume that the clients of a realtime service will require guarantees about the service provided by the network. The guarantees will cover all performance parameters of interest to the client. The specific parameters that will be included in a contract between the network and the clients depend on the nature of the application, and on what the network is prepared to support. For research

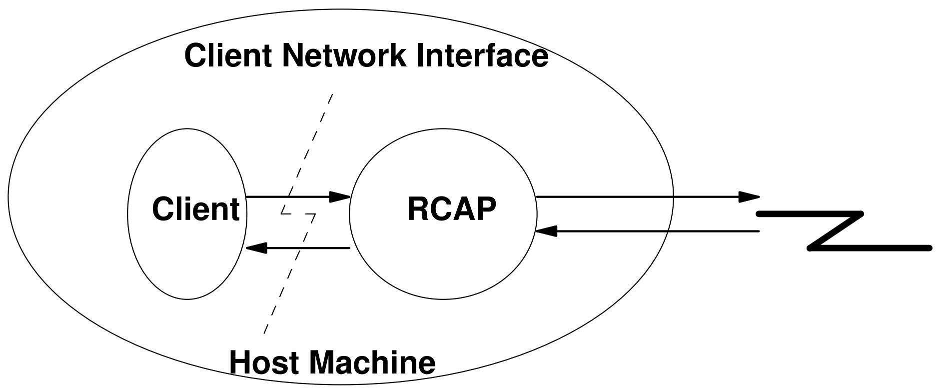

Figure 2.1: Service model for realtime network

purposes, we should try to support as many parameters as possible, and then evaluate the usefulness of each to the client and the cost to the network. The Tenet service model supports guarantees on delay, jitter and packet loss due to buffer overflow (see Table 2.1).

To support any realtime guarantees, the network has to limit the amount of load that the client may place on the network. This takes the form of traffic specifications, which the client declares when asking for performance requirements specified as described above. The exact set of parameters in the traffic description are also under investigation. More complex descriptions allow the network to allocate the resources more efficiently, leading to higher network utilization. However, simpler descriptions are less of a burden to the client (to choose the correct specification) and to the network (to process in order to allocate resources). The traffic specification used in Tenet Suite 1 uses a peak-average rate specification based on four parameters: X_{min}, X_{ave}, I, and S_{max} (26) (see Table 2.2).

The service model assumed involves a transaction between the network and the client. The client presents to the network a request for performance guarantees, and its traffic specification at the same time. The network processes this information based on its current load and resource state, and either accepts or rejects. If the request is accepted,

Parameter	Semantics
D_{max}	Upper bound on end-to-end delay
Z_{min}	Lower bound on probability of timely delivery
J_{max}	Upper bound on end-to-end jitter (optional)
W_{min}	Lower bound on probability of no loss due to buffer overflow

Table 2.1: Performance parameters of the Tenet Suite 1 interface

Parameter	Semantics
X_{min}	Minimum inter-message time
X_{ave}	Minimum average intermessage time
I	Averaging interval
S_{max}	Maximum messsage size

Table 2.2: Traffic parameters of the Tenet Suite 1 interface

the network is committed to delivering within the performance bounds each packet injected into the network, provided the client does not violate the traffic specifications. In the present service model, the performance is guaranteed except in the case of a network fault.

In the absence of sufficient experience with applications of such networks, we assume that any combination of performance requirements and traffic specifications is possible. The parameterized interface makes it possible to handle any combination, so that applications may find their exact requirements. This is especially important for research purposes, since we would like to find out what combinations are used by clients if no restriction is imposed on them.

Finally, we assume that the frequency of requests may be fairly high. We visualize such a realtime network being used to make video-phone calls, set up conference calls, access video servers, listen to digital recordings, and so on. As such, the lifetime of a connection will vary from minutes to hours. If a medium-sized network can support a hundred connections and the average lifetime is five minutes, this implies an average of twenty connections being established and torn down every minute. This frequency will increase with network capacity and size.

2.4. The Tenet Suite 1

The Tenet Suite 1 is a suite of protocols which provide realtime service on a network meeting the assumptions of Section 2.2. The protocols in the Suite are shown in Figure 2.2. The network-layer protocol, the Real-Time Internet Protocol (RTIP), provides a connection-oriented support for realtime communication in an internetworking environment. On top of this protocol, the Realtime Message Transport Protocol (RMTP) provides a message-based transport-layer interface to applications, while the Continuous Media Transport Protocol (CMTP) provides a time-driven interface for periodic traffic. These protocols coexist with the Internet protocol stack as

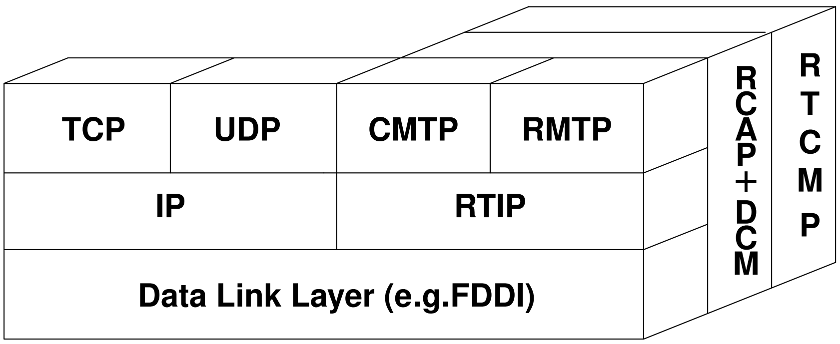

Figure 2.2: Protocols in the Tenet Suite 1

shown in the figure. The Realtime Channel Administration Protocol (RCAP) provides channel management services: channel establishment, teardown, and status enquiry. An extension to RCAP, the Dynamic Channel Management protocol (DCM), adds the ability to modify the parameters of the channel during its lifetime (51-53). To this suite, we plan to add a protocol for fault handling to be named the Real-Time Control Message Protocol (RTCMP).

The Tenet approach is based on the concept of *realtime channel*, which supports the service model described in the previous section by means of resource reservations on the network elements on the route of the channel. The resource reservation mechanism works in conjunction with the data forwarding protocols to ensure that sufficient transmission resources and buffers are available for the packets on the realtime channel as long as the client obeys the traffic specifications associated with the channel. This is made possible by admission control tests performed before each channel is accepted, which ensure that, if the channel is accepted, sufficient resources to meet the performance guarantees of all pre-existing realtime channels, as well as those of the new channel, are available at all nodes on the path of the channel.

The local tests (performed at a single node) ensure that the local performance requirements of the channel can be met without violating the performance guarantees of any pre-existing channel. The inputs to the tests are the local performance requirements and the traffic characteristics of the channel as seen at this node. The nature of the tests depends on the scheduling mechanism used in the data forwarding protocols. The tests for EDD scheduling are described in (27,28). This scheduling algorithm is implemented in the first version of RTIP.

These local performance bounds correspond to some resources reserved at the node on which the bound is given, since they cannot be used to establish any other realtime channel until the resources are released by the teardown of the channel. Since there is a limit on the number of realtime channels that can be supported, accepting a realtime channel implies that the number of subsequent channels that can be supported is reduced. For resources like buffers or bandwidth this reservation is easy to quantify, but the delay bound offered to the channel also constitutes a resource, which is harder to quantify. A lower delay bound corresponds to more resources being "blocked", since fewer low-delay channels can be supported, but the properties of this blocking effect depend on the scheduling algorithm implemented on the node. The control mechanisms must keep track of this resource state at each node, and quantify this "blocking" algorithmically, to ensure that, when resources are unavailable, new channels are not allowed to be set up.

To allow local guarantees to be extended to end-to-end performance guarantees, the traffic characteristics seen at each node must be computable. RTIP makes this easy by ensuring that the traffic specifications given by the client are obeyed at all intermediate nodes. We also need a mechanism to translate an end-to-end requirement to a set of local requirements, as well as a protocol to run all the local tests and make the resource reservations. The connection-oriented paradigm provides an efficient solution to this problem. The tests are run as part of the connection establishment procedure. This procedure is performed by the channel management protocol as described below.

2.5. Channel management

Channel management consists of the tasks of establishment, teardown, and status enquiry. In the Tenet Suite 1, the process of channel establishment is initiated at the source node, which computes a route for the channel based on its knowledge of the network topology and the resource state on each node in the network. In the first implementation, route selection relies on the underlying routing protocol available in the network.

Next, the source sends an establishment request message on this route. This message contains the performance requirements and traffic description specified by the client. Each node runs an algorithm to test the available resources against the requirements of the channel, and to choose the best performance bounds it can support. For example, the lowest delay bounds and the smallest jitter bounds are picked. The resources corresponding to these are tentatively reserved, information about the local reservations are added to the message, and the message is forwarded. When the message reaches the destination, the end-to-end performance achieved can be calculated (for example by adding the local delay bounds or multiplying the loss probability bounds). If the end-to-end performance is unacceptable, or if the destination chooses not to accept the connection, a denial message is returned. Otherwise, the excess performance values can be distributed among the nodes, so that they can relax the resource reservations accordingly. This is done on the reverse pass, when the destination sends an accept message on the path in the reverse direction. When this acceptance message reaches the source, the source can proceed to transmit data, and the network guarantees delivery within the performance bounds specified in the establishment request.

To see why we need to reserve the best amounts of resources on the forward pass and relax these amounts on the reverse pass, imagine a connection that needs to be established across a three-hop path through a network where the last hop is close to saturation. Let us say that the required end-to-end delay is 100 ms, and the links have transmission and propagation delays of 10 ms per link. This gives us a budget of 70 ms to distribute among the hops for queueing delay bounds. If the scheme naively allocates 20 ms per hop and reserves the appropriate resources at the first two hops to match that, the last node might be unable to meet the delay requirement if it is already highly loaded. On the other hand, if we allocate the lowest possible delays on the forward pass, we might end up with allocations of 10 ms, 10 ms, and 40 ms at the three nodes at the end of the forward pass, leaving us with a relaxation budget of 10 ms to distribute among the nodes.

2.5.1. Realtime Channel Administration Protocol

The Realtime Channel Administration Protocol (RCAP) implements channel establishment based on the approach described above. RCAP also performs the tasks of channel teardown and status reporting. During teardown the resources are released so that the blocking probability of future requests is reduced. Status reporting is used to find the resource reservation state of the channel at the intermediate nodes in the network and is useful for management and debugging.

RCAP is implemented as a daemon process that runs on each node in the realtime network. Each daemon is responsible for maintaining the resource state for the links outgoing from the node on which it runs, as well as for local CPU resources and buffers on the node. It maintains TCP/IP connections to neighboring daemons to exchange the

Message Name	Direction	Description
establish_request	Downstream	Message sent by source on forward pass to establish a channel; causes admission control tests to be run; best performance resources are reserved.
establish_accept	Upstream	Sent from destination to source on reverse path; indicates channel can be accepted with given requirements; causes relaxation of resources and confirmed reservations.
establish_denied	Upstream	Indicates that channel was rejected; causes resources to be removed;
status_request	Downstream	Requests information about resource reservation and channel state for given channel; proceeds along path of channel to destination.
status_report	Upstream	Carries collected information about status back to source.
close_request_forward	Downstream	Sent from source to destination to tear down channel; causes resources to be removed.
close_request_reverse	Upstream	Sent from destination to source to tear down channel; causes resources to be removed.

Table 2.3: RCAP messages

control messages described in Table 2.3.

The RCAP daemon also accepts messages from local processes on the same machine. The application programs on the source or destination machines communicate with RCAP by making procedure calls to an RCAP library, which constitutes the network control API. Procedures exported by the library enable the source application to request a connection or close it. The destination application is provided with registration, request reception, and request acceptance procedures. The procedure calls are converted by the library to control messages that the RCAP daemon recognizes.

The Tenet Suite 1, using the control protocol described above, has been implemented and deployed in several networking testbeds. Several studies of the performance of the suite as well as experiments of the use of the suite for video conferencing applications have been conducted (8,9,69).

2.5.2. Dynamic Channel Management

An extension to the basic services provided by RCAP is to allow the client or the network to change the parameters of the channel, such as the traffic specifications, the performance constraints, or the route, during the lifetime of the channel, if possible without affecting the flow of data. In (52) the Dynamic Channel Management (DCM) protocol is presented as such an extention. DCM is based on a round-trip approach similar to that used in RCAP; however, it needs to keep track of two sets of parameters, the old channel parameters and the new channel parameters. The forward pass consists of reserving the higher amounts of the resources required to support the two sets of requirements on nodes that are common to the old and new path. This ensures that, during the transition period, no packet misses its deadlines. On nodes that lie only on the new path, the new channel's requirements are used for the reservations. The reverse pass then performs the relaxation and informs the data transmission protocols of the new deadlines. On the common nodes, the higher amounts of the relaxed resources and the old resources are retained. A third pass is performed in the forward direction, along the path of the old channel. During this pass the resources for the old channel are removed from the nodes not on the new path. On the common nodes, the resources corresponding to the new requirements (after relaxation) are retained. The effect of such an alternate establishment is to change the parameters of the channel from the old set of parameters to the new (including possibly changing the route) without affecting the stream of data in transit.

2.5.3. Route update protocol

The route computation performed by the realtime scheme determines the path on which the channel establishment is attempted. For Suite 1, the implementors chose not to explore the routing function, by assuming that the routing protocol of the underlying network would be used. However, for DCM, a more complex routing function was worked out, where the route selection process is based on routing tables that summarize the resource state at all links in the network.

As a new channel is established or torn down in the network, the network state at the nodes in the path of the channel changes. The routing tables on all nodes must be updated to reflect the changed network state. Since the success of a following establishment attempt depends on the accuracy of the routing tables on which the route computation is based, the mechanism to maintain the routing information is important. For this study we assume a simple route update algorithm based on a reliable broadcast of any change to the resource state of each node to all nodes in the network, when a reservation is made permanent during the reverse pass, or when a channel traversing the node is torn down. Thus, each node has a picture of the resource state at all other nodes in the network, except that this picture may be slightly out of date. The further one node is from another, the more out of date its picture of the state of the other node is, since the route update message has further to travel.

During channel establishment, the admission control tests are performed on the actual resource state at each node on the path of the channel. During route computation, the same calculations are performed on the information about the resource state present in the local routing table. If the information were always completely up-to-date, the actual establishment would never fail. The only reasons why an establishment request might fail during the forward pass of the round trip are that either the information in the routing table at the node performing the route selection was out of date or another establishment proceeding concurrently caused the resource state at an intermediate node to change before the request reached the node.

2.5.4. Route selection

The selection of the route is based on the goal of reducing the resource usage in the network, subject to the primary constraint of meeting the delay bounds of the client if at all possible. In the Tenet Suite 1, this process is performed for each channel in isolation, without attempting to optimize the resource usage across the entire network, for reasons of scalability of the solution, and to prevent the requirement that other channels be moved when a new channel is set up. Since our network assumptions include a very dynamic channel state, where the establishment and teardown requests are frequent, the service provider cannot afford to optimize resource usage across the entire network for each channel request, especially since moving existing channels would involve additional channel management activity and may even affect the performance of the channel being moved.

In this context, the goal of minimizing the resource requirements of a channel is approximately equivalent to minimizing the number of links traversed by it, subject, of course, to the delay bound constraint.

A number of graph-theoretic procedures for selecting a route in a graph with various goals and constraints exist. The best known are the Dijkstra algorithm (24), which minimizes the cost of the route, and the Bellman-Ford algorithm (16), which minimizes the number of links in the path, subject to a cost constraint. If we set the cost of a link to the minimum delay bound which can be offered on that link to the channel under consideration, according to the resource state information available in the routing tables, the Bellman-Ford algorithm satisfies the requirement mentioned in the previous paragraph. This algorithm was also used to find the paths for realtime channels in (53), and is summarized below.

This smallest delay bound that can be offered to a channel on a given link can be computed knowing the traffic characteristics of the channel and the resource state of the link, using the same calculations that are performed during the forward pass of channel establishment. The cost of each edge on the graph representing the topology of the network is set to this value. Each node has a label, which is initialized to zero for the source and to infinity for all other nodes. This label represents the cumulative cost of reaching the node from the source given certain constraints. The algorithm works in phases. At the end of phase i, each node is labeled with the smallest cost of all i-hop or shorter paths from the source to the node. The algorithm stops when the destination is reached by a path with a cost less then the desired delay bound, or when i exceeds a given constant value. In the latter case, the algorithm fails to find a path.

2.5.5. Summary of Suite 1

This concludes our brief description of the Tenet Suite 1. We have, of course, focussed on the aspects of the suite that are important to an understanding of the rest of this thesis, and omitted many important and interesting details of the data forwarding protocols. Readers interested in the details of Suite 1 are referred to (9), which also includes a fairly comprehensive bibliography of the Tenet work. In this section, we have focussed on the channel management services, provided in the first implementation of Suite 1 by RCAP. We have also described RCAP extensions, such as DCM, the routing algorithm and the route update protocol, which were not part of the first implementation of Suite 1. Together, they form a networking environment that provides realtime communication with network management functionalities that allow the client or the network operator to make changes to the performance parameters, traffic specifications, and routes of existing channels. This environment is rich enough to support fault recovery and tolerance mechanisms. However, since we would like to design our mechanisms to be applicable to a wide variety of realtime networks, we will now provide a brief survey of other approaches to realtime communication in packet/cell switching networks.

2.6. Other realtime schemes

Variations on the realtime scheme described in this chapter have been suggested. For example, the establishment process may be speeded up by forwarding the establishment request before running the tests at each node; this method works well under the optimistic assumption that the tests will succeed. Data may also be transmitted before the acceptance message returns, with the understanding that, if the connection fails, the data will be lost. In this case, the first packet of data may also carry the establishment request (19).

Alternative approaches to the provision of realtime services on packet/cell switched networks have been proposed. In the predictive service proposed in (20), the network attempts to keep the performance given to the client from changing rapidly, but no guarantees are made. This approach does not meet the definition of realtime as used in this thesis. However, the guaranteed service proposed in the same paper does meet the definition, and the basic model, based on resource reservation during connection establishment, is similar to the model underlying the Tenet scheme. The Session Reservation Protocol (3) also fits the model described above closely. The differences lie in the underlying scheduling mechanisms and the details of the admission control tests.

The Heidelberg Resource Administration Technique (HeiRAT) (63) uses the ST-II network-layer protocol to provide unicast and multicast realtime connections. The ST-II implementation used in this investigation is enhanced with resource reservation algorithms very similar to the Tenet scheme (37). The Magnet-II Realtime Scheduling approach (38) supports a fixed menu of QoS classes. This simplifies the implementation of the admission control tests and the process of mapping end-to-end guarantees to local bounds. However, it limits the flexibility of resource allocation and the services provided.

Restrictive forms of realtime communication are proposed in (35,68), where the throughput is the main performance objective under consideration, and other parameters such as delay and loss rates can be computed secondarily. Resource reservation in the PlaNET network (19) uses the reservation model of (35). All of the above schemes are based on resource reservation during connection establishment.

Finally, there exist approaches to realtime communication which are fundamentally dissimilar from the Tenet model. The predictive service of (20) mentioned before falls into this category, since the performance provided is not guaranteed. The network merely *tries* to prevent the network performance from fluctuating rapidly, but performance is expected to deteriorate as load in the network increases. (70) uses the concept of soft reservations to avoid a completely-connection oriented model. It has certain advantages in the extremely heterogeneous environment of the Internet, but it does not offer true guarantees, since the reservations can be dropped without notice, or the path of data transmission may change during the lifetime of the conversation. (33) also proposes a connectionless approach in which the bandwidth partitioning model can be extended to approach realtime service, but the main goal of the system is not realtime service, but bandwidth partitioning.

2.7. Conclusions

While our experience with the Tenet Suite (8,9) demonstrated the benefits of realtime control in the network, it also highlighted some issues for further research. One key observation from our experiments with the realtime protocols is the need for fault handling mechanisms. This was especially keenly felt because the hardware and software that we were working with were experimental and crashed often in the earlier days of our experimentation. Providing fault recovery in those networks was not feasible, because the topologies did not offer alternate paths between end points. However, such redundancy would be needed in a commercial service network, in order to handle occasional failures without loss of service to clients. This aspect has been very carefully treated to date by network providers, and will have to be so treated as the new ATM based B-ISDN networks develop.

-- --

If realtime services are provided in an ATM or packet-switched network, they must follow some of the same principles of resource reservation and admission control that have been dealt with in this chapter. The next chapter isolates the common principles on which realtime service must be based, in order that we may study how to provide fault management services on networks that offer such services. Based on these fundamental principles, we designed fault management techniques that should be applicable to all networks that follow them. These techniques, and their evaluation through simulation, are described in the forthcoming chapters.

-- --

Chapter 3: A reactive scheme for fault recovery

3.1. Introduction

A realtime scheme as described in the previous chapter gives the application guarantees about the network performance that are only valid in the absence of network faults. Such a scheme is susceptible to link faults because of the connection-oriented nature of the network protocols, which dictates that the route followed by the packets during data transfer be pre-determined, so that resources on the links and nodes along the path can be reserved. This implies that, if any link on the route is affected by a fault, the flow of data is interrupted until and unless an alternate route can be found and the reservations transfered to it. Thus, we need to provide fault handling mechanisms to the realtime network. This chapter describes fault recovery mechanisms, which use reactive techniques to restore traffic in the event of a fault. Chapter 6 describes pro-active techniques that provide transparent tolerance to faults.

In the next section, we describe the goals of the fault recovery mechanisms, which drive the design and evaluation process. In Section 3.3, we discuss the assumptions about the network, derived from the basic underlying principles of realtime networks, on which we base the design of our fault recovery mechanisms.

We start the design process in Section 3.4 by providing a framework that consists of the basic tasks which any fault recovery scheme would have to perform. In Section 3.5, we consider some of the fundamental design issues, such as whether the scheme should be centralized or distributed, and whether we should attempt to find an optimal solution or use fast greedy algorithms. We make some fundamental design choices at this stage, and justify them on the basis of the principles of realtime networks. We also leave some of the fundamental issues open for investigation through simulation. In Section 3.6, we further narrow the solution space to be explored by choosing not to explore a number of issues. Finally, Section 3.7 identifies the design choices that remain to be explored. This leaves us with a well-structured problem domain, which we can systematically explore using simulations tools to be described in the next chapter.

3.2. Objectives of the fault recovery mechanisms

The fault recovery mechanisms are intended to provide fault handling that is efficient in the common fault-free situation. Accordingly, we will attempt to devise techniques that use no extra resources and cause no additional channel management activity in the absence of faults. When a fault does occur, some latency before service is restored to the affected realtime channels is acceptable. Of course, low latency is one of the objectives, and thus the basis of a metric, by which different schemes may be compared.

The recovery mechanisms are intended to restore realtime connectivity with the original performance guarantees to the affected channels. However, when a fault occurs, depending on the load in the network, all the affected channels may not be successfully rerouted. Rerouting as much traffic as possible is an important objective, and this defines another metric for comparison of recovery schemes. The effect of the rerouting process on the ability of the network to accept more channels immediately after the recovery process terminates is also a consideration. We would like the recovery process to use resources as efficiently as possible. Finally, we would like the recovery schemes to be as flexible as possible, to deal with different network topologies, load conditions, and traffic mixes. To summarize, the objectives of fault recovery are to:

- Use no extra resources during normal operation
- Restore original performance guarantees to channels
- Recover as much traffic as possible
- Minimize reaction time to the extent possible
- Use resources efficiently during rerouting
- Adapt flexibly to different topologies, load conditions, and traffic mixes

3.3. Network model and assumptions

The Tenet approach is one of the many approaches that can guarantee performance in a packet-switching network. Other solutions have been proposed by other investigators, some of which are mentioned in Chapter 2 of this thesis. All these schemes share some common features, which appear to be necessary to the provision of realtime guarantees on packet/cell switched networks. It has been argued (30, 32) that some fundamental assumptions must be satisfied in order to provide realtime services as defined here on packet/cell switched networks. If we can design our fault recovery schemes based on these fundamental assumptions about the network and the realtime scheme, then there is reason to believe that the fault recovery schemes will apply to any of the realtime schemes. These fundamental assumptions are listed below.

Resource reservation and admission control: Every network has some limit on such resources as bandwidth and buffers, and, as the offered load to the network increases, a point of congestion is eventually reached when the performance of the network deteriorates. In order to provide good performance to realtime applications even during periods of network congestion, it is necessary to set aside some fraction of the resources so that these resources are available to the realtime packets in preference to others. All the guaranteed performance schemes mentioned in Chapter 1 implement some form of resource reservation. If realtime traffic is added to the network, a point must eventually be reached when the addition of another stream could possibly lead to overload on the network, leading to violation of guarantees. While the algorithms or heuristics to detect this condition differ from scheme to scheme, all realtime schemes must employ some form of admission control.

Client traffic specification: Since the network cannot support an arbitrary amount of traffic, it is necessary for each realtime client to give the network a description of the traffic it will inject into the network. This traffic description is used to perform admission control, to make sure that the addition of a new client's traffic will not violate the network's ability to provide guaranteed service to all other clients, as well as to satisfy the new client's requirements. The client must adhere to these traffic descriptions, otherwise the traffic on the network will be higher than calculated, and realtime performance will suffer.

Connection-orientedness: The resources reserved for a realtime client, such as transmission bandwidth, buffers, and so on, are local in nature, that is, they exist in a given switch or router. Thus, the reservations must be made over a specific set of switches or routers on the network. This set of switches or routers would define one or more paths in the network, over which resources are reserved. During transmission, the packets of realtime data would follow these paths. These route(s) have to be computed before the transmission of data, in order that the resources be available when they are needed. Note that the route computation need not occur strictly before the establishment, since it may be done in a distributed manner as the establishment request makes its way through the network. However, before the data transfer starts, both processes have to be complete. Thus, the nature of the communication is connection-oriented.

Protective service discipline: The guarantees offered to a realtime client should be protected from disruption by misbehavior of other realtime or non-realtime traffic sources. This could be achieved by a variety of techniques, but a minimum seems to be providing priority of realtime traffic over non-realtime traffic, and some form of rate control or policing on each realtime source. The rate control ensures that the traffic introduced by a client into the network does not exceed its traffic specifications, thereby protecting other realtime clients. Various combinations of policing mechanisms and priority schemes have been proposed (26,50,71).

3.3.1. A general realtime networking model

For the purposes of designing our fault recovery schemes, we assume that the network meets the above set of requirements. We proceed to make this network model more concrete by describing the outlines of a scheme, which fills in some more details about the realtime network, but is still general enough to fit a number of the schemes described in Chapter 2 (3, 19, 20, 35, 38, 63, 68). This leads us to believe that this model will fit the schemes used to provide realtime switched virtual circuits (SVCs with QoS guarantees) on future ATM networks.

- The network handles a request for a realtime channel in a distributed fashion. We do not consider centralized approaches for reasons of scalability of the scheme.
- Channel establishment is based on a round-trip message exchange, during which admission control tests are performed at all the nodes on the route of the channel, based on local resource state information available at each node. The single round-trip paradigm is assumed because of its efficiency and compatibility with the virtual circuit model of ATM networks.
- The best performance possible is reserved on the forward pass. The resources are relaxed to the appropriate performance level on the reverse pass of the round trip; the necessary computations may be performed at the destination, or in a distributed fashion during the reverse pass. The two-pass scheme allows us to maximize the chances of acceptance within one round-trip time, yet does not waste resources within the network because of over-reservation.

Note that the above outline does not place any restrictions on the route selection process. It may be performed on the originator of the establishment request, at a route server, or in a distributed fashion as the establishment request moves through the network on its forward pass. The specifics of the admission control tests performed and the resources reserved are also unspecified, and depend on the particular scheme being used.

To allow route modification of realtime channels, we may either assume support from the realtime scheme, or incorporate the appropriate channel management functionality in the fault recovery protocol. We chose the former approach, since route modification is already supported in a version of the Tenet Suite 1 by the DCM protocol. Thus, we need to make another assumption about the realtime scheme.

- The realtime scheme supports modification of the route of the channel during its lifetime. Using a three-pass mechanism, the resources of the channel can be modified without violating any guarantees during the transition process. The first pass makes tentative reservations for the new route or performance parameters, the reverse pass relaxes the reservations and also makes them permanent. At this point data starts to follow the new path. The third pass removes the resources along the old route.

The above model of the realtime scheme leads to the following observation, which must also be kept in mind while designing routing and recovery algorithms.

- The realtime network has a fairly complex resource state, distributed across the nodes of the network, which must be considered while seeking a new route for a channel. The exact parameters of the resource state depend on the particular realtime scheme under consideration. But the information required to perform the admission control tests is also required by the routing algorithm, in order for it to find routes on which the channel establishment request will be likely not to fail.

In addition to the above assumptions, we also need to keep in mind the assumptions about the underlying network and the service model described in the previous chapter. Two of these are repeated below, because they are significant to the choices made in this chapter.

- The nodes of the network are programmable computers with sufficient computation power to run the channel management protocols with admission control tests. These same computers can also be used to run the routing algorithm, as long as the algorithm is of reasonable time complexity.
- The channel establishment and teardown requests are frequent. Thus, the set of channels, and hence the set of routes, existing in the network at any given time is very dynamic.

--

--

3.4. A fault recovery framework

In this section, we impose a framework on the fault recovery scheme by enumerating five tasks which must be performed in order to restore traffic in the event of a fault. These tasks are *detection*, *instigation* of recovery, *route selection*, *connection management*, and *return to normal*.

Detection is performed by the nodes adjacent to the failed component. The latency requirement dictates that the time from the occurrence of the fault to the detection must be small. On the other hand, the requirement that channel management activity during normal fault-free operation be kept to a minimum implies that faults should not be declared without some level of validation. We will not explore this task in any further detail in this chapter. Chapter 5 contains some considerations on the details of the detection mechanisms.

Instigation of the restoral process is performed by the node that detects the fault. This consists of informing the rest of the network of the fault so that recovery can be performed. The information that needs to be communicated includes the identity of the failed component(s), the affected channels, and, depending on the recovery scheme, possibly the new routes. We will look at this process in more detail in this chapter and in Chapter 5.

Route selection is one of the most important stages of recovery. In this stage, the network chooses the new set of routes, to which the channel management process will transfer the channels affected by the fault. This process may be completely centralized, so that the entire set of routes are computed on one computer. It may be partially distributed, so that each route is computed on a single computer, but different routes are computed by different computers. Each single route may also be computed in a distributed fashion, on a hop-by-hop basis as the establishment message works its way through the network. Finally, the order and timing of the route computations can also be varied.

Channel management is the process of actually moving the network resources from the old path to the new. Channel management attempts to reserve sufficient resources on the new path to meet the end-to-end performance requirements. If the attempt fails, we may return to the route selection process again, perhaps with additional constraints defined by the specific retry policy. If the attempt succeeds, we remove the remaining resources from the old path.

Return to normal is the action taken to balance the network load more evenly when the link is declared good by the detection and validation mechanisms. Since the longer routes used after rerouting consume more resources, the capacity of the network can be increased by returning the channels to shorter paths using the newly restored links. The important consideration here is not to affect the traffic during this process. The scheme should also work well with the admission control protocol. One possible return to normal scheme is described in Chapter 5.

3.5. Design issues for a reactive scheme for fault recovery

In the framework described above, the tasks that are of particular interest to us in this chapter are the processes of instigation, route selection, and channel management. We assume, for the moment, that detection and return to normal can be performed. These are discussed in more detail in Chapter 5.

In this section, we look at some of the general design choices that exist in designing a reactive scheme for fault recovery, in the context of the assumptions that we have made about the realtime network. We looked at some of these issues in Section 1.4.2, in the context of survivability for the cross-connect layer of telecommunication networks. However, the assumptions regarding realtime channels, which were described in Section 3.3, are different from the ones on the basis of telecommunication networks are designed.

For example, in the cross-connect layer of the telecommunication network, the possible lack of intelligence or computing power of the DCS was an important issue in favor of centralized schemes. Here, we can assume that the switches (or the controllers attached to the switches) will have sufficient computing power to run distributed algorithms of reasonable complexity, since we already need such processing ability to run the admission control protocols. The state information associated with a realtime channel is also larger than for a cross-connect trunk, since, instead of just bandwidth, each channel has several traffic and performance parameters associated with it. The network state at intermediate nodes, which summarizes the local performance bounds allocated to existing channels, also needs to be considered during rerouting, in order that the algorithm be able to compute routes on which sufficient resources exist to support the new channel. This network state is also much more dynamic than that of the cross-connect network, since realtime channels have shorter life-times than cross-connect trunks. We need to look at these fundamental design issues again, with these different assumptions in mind.

The questions that we will consider in this section are the following. Should the algorithms attempt to find optimal solutions or should they aim at fast approximate solutions using a greedy strategy? Should they be distributed or centralized? Should we use precomputation of routes to speed up the recovery or should the reroutes be computed dynamically when the fault occurs? How much time should we spend on information exchange during the recovery process? Some of these questions will be answered in this section, while others will be kept open for investigation through simulation.

3.5.1. Optimal vs. greedy

The problem of finding the optimal flow in a network given bandwidth constraints on the links can be mapped onto a linear programming problem, for which reasonably efficient algorithms (23,61) exist. The routing problem at the cross-connect layer of telecommunication networks can be approximated by the above model, and quasi-optimal solutions can be found efficiently. However, when the admission control test becomes more complex than checking a simple bandwidth constraint, especially with non-linear constraints used in the traffic specifications, as is the case for the X_{min}, X_{ave}, I model, the problem can no longer be simplified to a linear program. and no known efficient algorithm for computing an optimal solution exists. The brute force algorithm, which enumerates all possible solutions, evaluates them in terms of some criteria of success, and picks the best, is exponential in time complexity. This makes such schemes infeasible for large graphs or if the state information per node, on the basis of which the optimal solution is chosen, is complex, as is the case for realtime networks.

Greedy algorithms are a class of algorithms that deal with problems involving many similar steps one at a time, finding the best incremental solution for each. In the context of establishing realtime channels, this implies dealing with channels one at a time and picking a route placement and resource allocation that minimizes the increase in resource usage. Greedy algorithms may produce non-optimal solutions. However, since each application of the algorithm attempts to minimize the increase in resource consumption caused by the single step, the overall solution tends to stay close to the optimal. Greedy algorithms provide reasonably good, fast solutions to otherwise computationally intractable problems. The greedy approach is also used in our network model for channel establishment (see Ch. 2), i.e., the routing and resource allocation of one channel does not involve making adjustments to the route or resource allocation of any other channel. This approach is much faster, since the time complexity of the resource allocation is now linear in the number of channels and nodes, and no channel management activity to move existing channels is required. This is especially important considering the dynamicity of the realtime load on the network, since we either have to optimize every time a new realtime channel is added or removed, or settle for a non-optimal solution. We will explore a greedy approach to fault recovery, because it fits well with our realtime network model, operates in linear time, and produces reasonably good solutions.

3.5.2. Centralized vs. distributed

A centralized scheme gathers all the important network state at one computer, where a new configuration for the network is computed. Since all the information is available, it is possible to compute a configuration that is optimal according to some target function. This optimal scheme might be able to reroute more channels successfully under heavy-load conditions. However, this gain comes at the expense of a possibly higher latency, since the time to gather the network state to one location and then distribute the network configuration to the switches must be added to the time to run the algorithm. The time to gather the network state has to be considered here, since, unlike the cross-connect networks, realtime networks have a very dynamic load, with connections coming and going as applications are started and ended. Since the establishment protocols are distributed, no single computer has the complete network configuration at the time of the fault. All the network state must be gathered to the central computer before the centralized algorithm can start operating. This latency would be negligible for a LAN, but might be significant for a WAN. Furthermore, this higher latency would translate to a much higher number of bits lost in a network with a high bandwidth-delay product.

Centralized algorithms are susceptible to single-point failures, since all recoveries would be stalled by a failure of the central computer, or of the control network which connects the central computer to the switches. In the telecommunication networks, this problem is mitigated by having more than one fault recovery computer and having redundancy in the control network, so that the control network itself is fault-tolerant. However, this increases the cost of the solution. We do not assume a separate control network; rather, the recovery action is carried out over the data network itself. This fits in better with our model of the realtime establishment protocols, which exchange control messages over the data network. However, this makes the centralized approach much less desirable, since the failures in the network that we need to recover from make the collection and distribution process itself unreliable. The distributed algorithms deal explicitly with the problem of failed links. For example, the exchange of control messages with neighbors may be used as part of the process to determine the working topology of the network, since the inability to exchange messages indicates a failure of the corresponding link.

A distributed algorithm also scales better to larger networks, since the processing power available increases with the network size. The centralized scheme, on the other hand, must run an optimization algorithm of exponential complexity on a single computer. Thus, the major advantage of using a centralized solution (i.e., finding an optimal solution) is infeasible for all but the smallest realtime networks. The establishment protocol assumed in the network model is distributed for much the same reasons. A distributed fault recovery scheme has the advantage of fitting better with our distributed network model.

A distributed algorithm would find a non-optimal solution based on a limited exchange of information faster than a centralized scheme and without relying on a single central computer. This comes at the expense of possibly finding a solution considerably worse than the optimal. Since the distributed elements may have inconsistent views of the network, the overall behavior of the system may be hard to understand for an implementor, and may be far from the expected and desirable behavior, even though each individual element has a simple and understandable behavior. In spite of these disadvantages, this approach appears more feasible for the case of realtime networks, and will be explored here.

The channel management activity described in Chapter 2 is completely distributed, since some admission control tests are performed on each node in the path of a candidate route. Selection of a candidate route may be either partially or completely distributed. For a partially distributed route selection, each channel is routed separately on a separate computer, but each individual route computation takes place on a single computer. For a completely distributed scheme, each route is computed hop-by-hop in a distributed manner as the establishment request progresses through its forward pass. While a fully distributed scheme is worth investigating, the partially distributed route selection strategy is known to work well (it is used in the basic channel management protocol), and will be used here.

3.5.3. Pre-computed vs. dynamic

The task of rerouting may be partially pre-computed in order to decrease the response time of the recovery after a failure. This idea has been explored in the context of the cross-connect layer of telecommunication networks, as mentioned in Section 1.4.2. In the context of realtime networks, multimedia traffic is expected to be much more dynamic than the configuration presented by the cross-connect connections. A trunk is a relatively long-lived connection. As such, the requirements of a set of trunks, characterized by source, destination, and bandwidth, are fairly static. After the set of required trunks has been decided, the current configuration, as well as the recovery configurations for all likely fault combinations, may be computed and stored. On the other hand, for a realtime network, the set of active connections changes much more dynamically. In addition, the time and space complexity of computing and storing the recovery configurations for any such set is also larger, since the channel has a more complex state than a cross-connect trunk. This makes it infeasible to pre-compute the set of recovery configurations for all possible fault conditions each time the network state changes. A dynamic approach, which computes the reconfiguration when the fault occurs, will be developed and explored here.

3.5.4. Global vs. local knowledge

The extent to which knowledge is universally shared among the elements of a distributed algorithm can have a tremendous impact on the performance and correctness of the algorithm. At one extreme, we might allow all nodes to gain complete information about any change in the network state before computing the next step in the algorithm. Since this would imply that all nodes have completely up-to-date knowledge of the network state throughout the reconfiguration process, this is equivalent to a centralized algorithm in terms of goodness or optimality of the solution. At the other extreme, we may choose to allow all nodes to run their reconfiguration algorithms with the view of the network state that they had at the moment of the fault, without allowing any additional communication. This would have the fastest response time of any algorithm, but would reroute fewer connections successfully, since the inconsistencies among the network views of the separate network elements would lead to poor cooperation between the nodes. Thus, by allowing more time for communication, we can improve the success of the rerouting algorithm.[1]

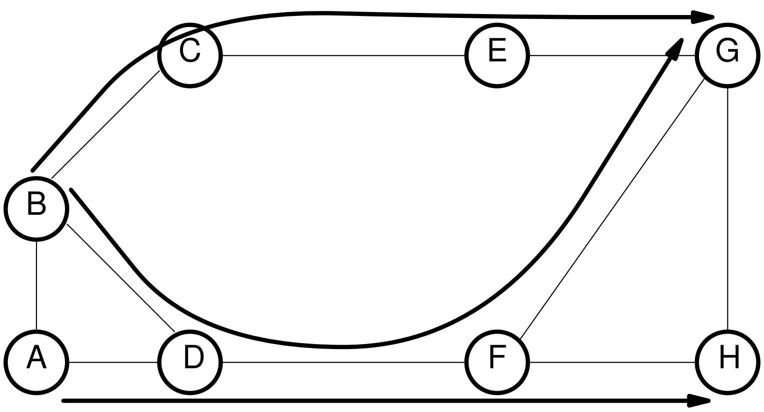

Figure 3.1: Avoiding collisions

A certain amount of communication is needed just to establish a consistent configuration. For example, in Figure 3.1, after the process of reconfiguration, if node A believes that a connection passes through node A and node B, and the local state on node B does not contain the same information, then the configuration is not consistent, and data transfer cannot take place correctly along the connection. This level of cooperation is taken care of at the connection management level. If we use an underlying

[1] Distributed algorithms may operate at any point on the trade-off curve, making them more convenient from the point of view of this research, where we are interested in exploring the trade-offs.

protocol such as RCAP or DCM, consistency of the configuration is ensured by the round-trip mechanism for channel establishment. However, even beyond this level of communication, opportunities exist for non-cooperative behavior between nodes. If node A and node B both decide to route a connection through link DF, and the resources on link DF are insufficient to support both, then one of them will fail at the admission control stage. This may even happen if sufficient resources for both channels exist on the link, due to the excess forward pass reservations of the first channel. If this information had been available to node B, then it could have chosen a different path (e.g., BCEG), on which this collision would not have occurred.

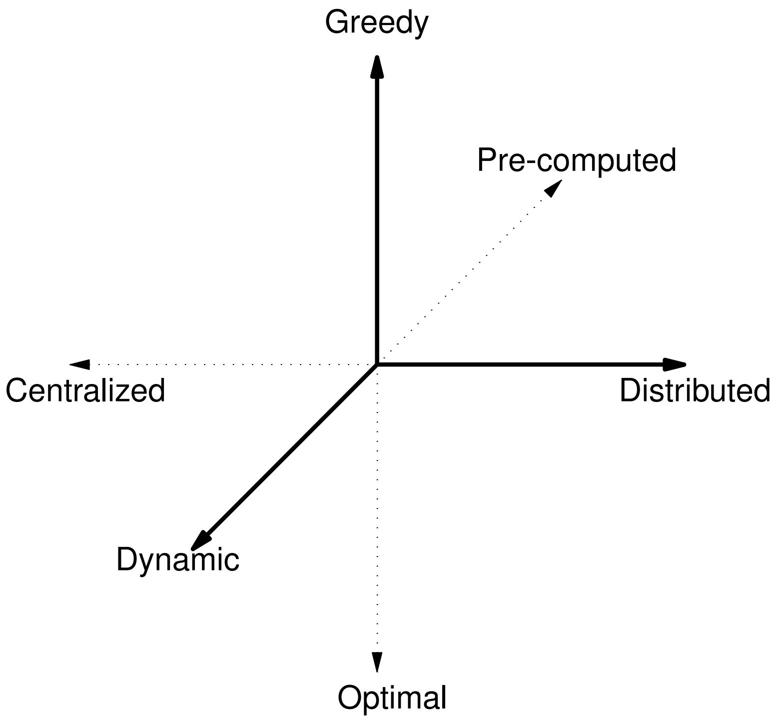

Figure 3.2: The sector of the solution space to be considered

Thus, the trade-off between speed and goodness of the solution computed by the distributed scheme needs to be explored. The best operating point for any network will depend on the network size, the communication latencies, the level of load on the network and many other variables. In addition, at present the details of the algorithm are completely unspecified. Thus, although we have restricted our solution space by considering only greedy, distributed, and dynamic solutions (see Figure 3.2), we are still left with a very large solution space to explore.

3.6. Issues not to be investigated

To reduce the size of the problem at hand, we will make certain simplifying assumptions about the solution. We have already decided not to explore fault detection and return to normal in any detail. Also, in the context of the Tenet scheme, the requirements of the channel management task can be satisfied by the services provided by the combination of RCAP and DCM. It fits all of the decisions taken so far, since it handles channels one at a time, using a distributed protocol to perform the admission control test, and does not require any pre-computation. Thus, we will also assume that channel management is performed by an existing realtime channel management protocol that supports route modification.

We will also not explore the mechanism by which the resource state information of the network, required by the routing algorithm, is distributed to all the nodes in the network. We assume that a reliable broadcast protocol, as described in Section 2.5.3, is used to maintain the routing table information. Of course, such a method implies that the tables are slightly out-of-date with respect to the true resource state of the network, and are updated as messages come in through the network. This also implies that more information can be obtained by simply waiting for the update messages to come in. Thus, there is a trade-off between the amount of information available on which routing decisions are based and the time that the scheme must wait to make the decisions.

When attempting to find a path to which a given channel may be rerouted, we have the same objectives as mentioned in Section 2.5.4, where we described the graph theoretic algorithm used to select a candidate route. The primary criteria are that sufficient resources exist to support the traffic parameters of the channel and that the end-to-end delay bound of the channel is satisfied. The secondary objective is to minimize the resource requirement of the candidate route. Since the route selection for the recovery process will also be carried out in the same greedy manner as before, we can use the Bellman-Ford algorithm, with certain modifications. These modifications are required to satisfy additional constraints, which are required by the recovery or tolerance schemes, and will be described as the constraints come up.

Thus, by fixing the channel management and route update protocols, and using the Bellman-Ford algorithm to find the min-hop path meeting the delay and resource requirements of a channel, we have simplified many aspects of the fault recovery activity. We will now describe the issues that remain to be investigated through simulation.

3.7. Remaining issues

The decisions made in the previous sections fix most of the variables in the scheme. We are left with a few more or less orthogonal factors: *the locus of reroute*, the *timing* of the rerouting attempt, the *retries* in the event of failure, and the use of *state prediction*. We will define these factors in this section.

For each of the factors named above, many solutions are feasible. Our objective is to explore the extremes of the range of possible approaches to each factor, as well as some intermediate solutions. Experimenting with the extremes will tell us if the factor being explored has any impact at all on the performance of the scheme as a whole. The experiments with the intermediate approaches will give us some idea of how the performance changes as the particular factor under consideration is varied. For example, if the performance according to some metric is roughly constant across all the approaches to a given factor, we can conclude that the particular metric is insensitive to variations within the factor. If the performance is best at one of the extremes, we can be reasonably sure that we do not need more careful experiments to search intermediate points for the best solution. If the performance of an intermediate approach to the factor is the best, more careful experiments are called for, perhaps in a more realistic environment involving an actual implementation, to find the best solution for that factor.

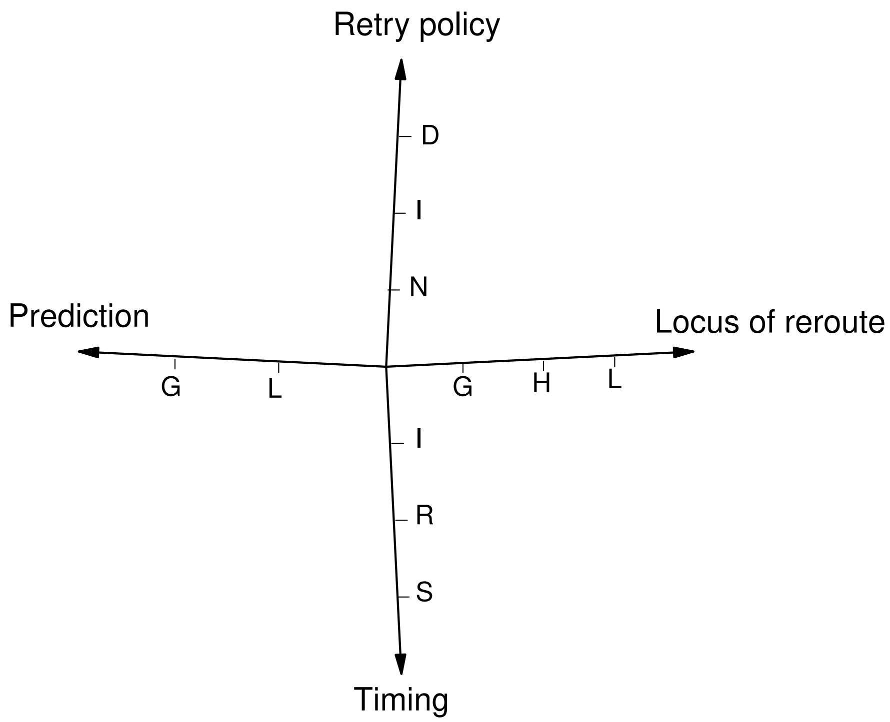

Figure 3.3: 4-dimensional solution space to be explored

We can think of the factors as defining a multi-dimensional space, and the various approaches as hyperplanes in which one coordinate is fixed. The schemes are points in the solution space, where all the coordinates are fixed by specifying an approach for each factor. In Figure 3.3, the approaches are marked on the dimensions by their abbreviated names (defined below). Not all the points in this 4-dimensional space correspond to reasonable schemes, as we shall see. Our simulation experiments will evaluate points in this space on multiple metrics of performance, attempting to find a scheme with good performance on all significant metrics.

3.7.1. Locus of reroute

The entire old route of the channel need not be subjected to a route selection process. In most cases, only one component of the channel, such as a link or a node, is affected by the failure, and it would suffice to find a new segment to replace a portion of the route. The locus of reroute determines the portion of the current route of the channel upon which a rerouting attempt will be made, and the node which performs the reroute selection. We consider three locus of reroute types: *Global reroutes* (G), *Local reroutes* (L), and *Hybrid reroutes* (H). In all types of locus of reroutes, the traffic and performance characteristics of any rerouted connection must be maintained after the rerouting is completed.

Global rerouting: With this approach, which corresponds to *end-to-end* rerouting (Section 1.4.2.1), the path of the rerouted connection is determined at the source node based on the current network load and the traffic and performance characteristics of the connection. This scheme considers all possible paths through the network. However, the source needs to have information about the resource state for all nodes in the network. The information on which this route selection is performed may be out of data, more so for nodes far from the source. This might make Global rerouting sensitive to the timing and prediction factors, which tune the level of information sharing and the time spent waiting for information.

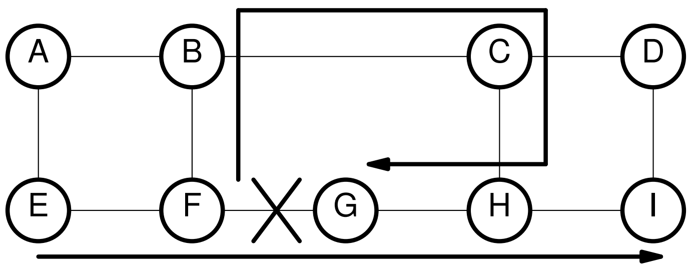

Figure 3.4: Local rerouting

Local rerouting: This approach to rerouting finds a new route from the node upstream of the fault to the node downstream of the fault. It combines this segment with the rest of the route from the original path. Since the path from the upstream node to the downstream node may include some links in common with the original path, such redundant sections are removed. For example, in Figure 3.4, the failed link FG is replaced by the segment FBCHG. However, on combining with the original path, the common portions GH and HG are removed, giving the new route EFBCHI. This process of route selection is performed at the fault detecting node. Since this is closer to the source of the fault, and the routing also takes place over the nodes in the immediate vicinity, Local rerouting should be able to route correctly with less waiting for information to come in. Thus, it should be less sensitive to the timing and prediction factors. The locality of the region affected might also make this scheme better able to handle multiple faults in different parts of the network, since one recovery attempt is relatively independent of the others. Finally, using as much of the old route as possible may also be beneficial, since the request is less likely to fail on these trunks, as some resources are already reserved.

Local rerouting is similar to *link* rerouting as discussed in Section 1.4.2.1. However, the establishment passes are carried out over the entire route, not just the new segment, for better resource balancing and to increase the chances of meeting the end-to-end delay requirements. This comes at the expense of a longer time to complete the rerouting, since the establishment passes must be made over the entire route.

Note that, although the need to perform end-to-end resource balancing does not allow us to get the shorter recovery time of link rerouting, the amount of traffic rerouted by Local rerouting should be as good as or better than that of link rerouting. Thus, to start with, we will explore Local rerouting as defined. Then, if the success of Local rerouting warrants it, we can explore the possibility of reducing the time to reroute by changing our resource balancing mechanism to exchange control messages over just the new segment.

Hybrid rerouting: Local rerouting considers a much smaller set of candidate routes than Global rerouting during route selection. This makes it much harder for Local rerouting to find a route that has sufficient resources. Hybrid rerouting attempts to retain the advantages of Local while mitigating its disadvantages, by first trying a Local reroute, then a Global reroute if the Local attempt fails.

3.7.2. Timing

The *reroute timing* policy selects the starting time of the reroute establishment attempt for each individual channel. This factor seeks to mitigate the effect of route collisions and database inconsistencies. *Route collisions* occur when an establishment request, during the forward pass along a link, consumes a large fraction of the remaining resources on the link, thus causing an immediately following forward pass establishment request to fail due to lack of resources, even though on the reverse pass the former establishment request would have relaxed its resource reservations so that the latter could have had its resource request honored. Reducing route collisions would increase the probability of establishing a connection. *Database inconsistencies* can also cause rerouting attempts within a small time interval of each other to interfere with each other. Had there been sufficient time between the two connection rerouting attempts for the route update protocol to exchange the necessary information, the routing database would have reflected a more accurate network load, and the latter of the two reroutes would have chosen a different path through the network.

Thus, by increasing the waiting time between rerouting attempts which use the same links in the network, we can improve the success of rerouting. Three possible approaches are considered to explore the effect of the timing factor: *Immediate* (I), *Random(n)* (R), and *Sequential* (S).

Immediate timing: With this timing approach, rerouting attempts are initiated by the controlling node as soon as a failure is reported to it. In the absence of any attempt to predict the behavior of other nodes, this approach is characterized by the least cooperation and potentially the most interference.

Random timing: With Random(n) timing, the rerouting attempt time is determined by generating a random value from a uniform distribution over an interval of duration n. This random value is added to the current time to obtain the reroute starting time. This randomization decreases the chances of two rerouting attempts being performed simultaneously over the same links in the network, and thus, should increase rerouting success. Different values of the randomization interval *n* yield different variations within Random timing.

Sequential timing: With this timing approach, all rerouting attempts are handled sequentially, with only one controlling node initiating a rerouting attempt at any given time. When all the reroute attempts of a single affected connection are completed, only then are rerouting attempts made on another connection. Sequential timing reduces route collisions and database inconsistencies. In fact, if we allow sufficient time between rerouting attempts, all nodes will perform route selection using up-to-date information. This is equivalent to a centralized scheme with respect to the rerouting success that can be achieved, but is obtained at the cost of a much higher average time to reroute. However, Sequential timing is useful since it represents the routing success that can be obtained by eliminating all collisions. We cannot expect to reroute a significantly larger amount of traffic without using global optimization techniques. As such, it serves as a good control experiment, against which we can measure the performance of the other approaches.

3.7.3. Retry policy

The *retry* policy determines the action that is taken if the route selection process fails, or if the establishment attempt along the selected path fails. We experimented with *No retries* (N), *Immediate retries* (I), and *Delayed retries* (I).

No retries: In the absence of retries, if the first attempt to find a route fails the channel is abandoned and the client is informed of the failure. The failed channel is torn down and the resources released immediately.

Immediate retries: With this approach to retries, if the route selection fails the channel is abandoned and torn down; if, however, the establishment attempt fails, new information provided by recent database update messages is incorporated, the saturated link (at which the previous reroute failed) is marked as a unusable for the route selection process, and a new route selection is performed immediately. An establishment is attempted on this route.

Delayed retries: With this approach, a retry is scheduled for a later time. The saturated link is not explicitly removed; rather, the route update information that comes in during the interval may cause the route selection to choose a different path. The method by which the interval is selected provides variations within this approach. We experimented with fixed intervals, exponentially increasing intervals, and randomization.

Retries are repeated on failure up to a pre-determined limit; different values of the limit give us different variations within each approach. We also experimented with using a cut-off time to stop further retries, instead of a number of attempts.

3.7.4. Prediction of network state

This factor determines the extent to which a node performing route computation modifies the information available in its routing tables at the time of route selection, on the basis of the predicted behavior of other nodes. The information in the routing tables models the actual resource state in the network but may be slightly out-of-date. Prediction aims to improve the accuracy of the modeling, to try and accurately determine the resource state likely to be found by the establishment request when it arrives at each intermediate node in the network. If prediction were perfect the effect would be to spread the different establishment attempts out onto different paths in the network, since the various nodes would cooperate to choose routes such that sufficient resources are available for each channel on each link.

Prediction has the same goal as the Random or Sequential timing approaches, where waiting for information to come in allows the correct routing decision to be made. Prediction attempts to obtain the same effect without the extra waiting time, by predicting the change in routing information which will occur.

We try two approaches within this factor: *Local prediction* (L) and *Global prediction* (G). These names (and abbreviations) are also used by the approaches for locus of reroute, but since we will always mention the factor being considered explicitly, no cause for confusion should arise. Since other intermediate prediction schemes that we can think of are fairly complicated, we decided to explore them only if the Global prediction approach offered sufficient improvement over the Local one. The two approaches are described below.

Local prediction: This is a minimal form of prediction, consisting of each node only trying to predict the result of its own establishment actions on its own local database. If a node performs two consecutive route computations, the latter is performed with the appropriate resources removed from the local database, so that it sees the state corresponding to what would exist after the forward pass of the former establishment request. This is the picture most likely to be seen by the second request as it propagates through the network after the first one. Also, in order to compute a reroute, the resources corresponding to the current channel are added to the network, since these resources are available to the alternate channel during the local admission tests. However, no attempt is made to make the behavior of the nodes globally predictable. This level of prediction requires no synchronization of reroutes, exchange of messages, or waiting time; thus, we make this the default mode of operation, and compare the performance of the Global approach against the performance of the Local one.

Global prediction: Although it is conceptually orthogonal to the other factors, Global prediction would work best with Immediate timing, since, with the other timing schemes, prediction is superfluous, as the information is already updated by waiting. In addition, the predictability of the timing of the Immediate approach makes the prediction scheme easier to implement, since we know that all the nodes are performing the route computations at roughly the same time. In contrast, the behavior of the Random timing approach is harder to predict, since the time at which the channel establishment is performed depends on the state of the pseudo-random number generation function. Thus, we only experiment with Global prediction in the context of the Immediate timing approach.

With Global prediction, all nodes wait for a sufficient interval of time without performing any channel administration activity, to ensure that the routing information has stabilized to the same value on each node. The fault message ensures that all nodes reach consensus about the set of channels to be rerouted. These channels are ordered using the same rule on all nodes. The route selection algorithm is performed for *all the channels* in the same order on all nodes. Since we only experiment with Global prediction in the context of Immediate timing, all of the nodes are running this algorithm at roughly the same time. After each route selection, the resources corresponding to the forward pass reservations are removed from the local routing tables. Since the same action is performed at all nodes, and the initial state of the routing tables are the same, all nodes make the same routing decisions. In fact, this is identical in effect to using a centralized routing algorithm. Thus, unless the assumptions of identical initial state and no extraneous channel management activity are broken, the routes found by this routing process will always have sufficient resources for the channel being rerouted, since the routing process models the actual establishment process. If the resource state during the forward pass might cause the establishment of any channel to fail on a given link, the algorithm should either find a different route for the channel or fail to find a route at the current time, since the resources used up by other channels are reflected in each local database.

Since the timing is Immediate, establishment attempts are generated at all the nodes at the same time. Thus, the resource states seen by the route selection process should be the same as the actual resources seen during establishment, and theoretically no attempt should fail during establishment. However, this might happen if our idealized assumptions about synchronization and complete coherence of the routing databases

-- --

are not satisfied. It might also happen if the order of channel establishment is different from the order in the routing sequence, and the second channel has a tighter delay constraint than the first one.

Since the forward pass reservations are higher than those after relaxation, it is important to use retries for the channels for which routes could not be found during the first synchronized routing attempt. Retries for these channels, as well as for any channels which failed during establishment, may be performed in synchrony, after a sufficient interval to complete all establishment attempts and update all databases again. Alternatively, the retries may revert to the Random delay model. Both schemes were tried in our simulations.

It is important for Global prediction that, apart from the recovery action, all channel management activity (i.e., establishment of new channels) be frozen during the time that the synchronized routing attempts are being performed. Otherwise, the states seen by the different nodes will be different during the routing process, due to the change caused by this activity and the action of the route update protocol. This freezing of new establishments would not be intolerable if the time interval for the synchronized phase were short. Note that data delivery for already established channels that were not affected by the fault is not interrupted.

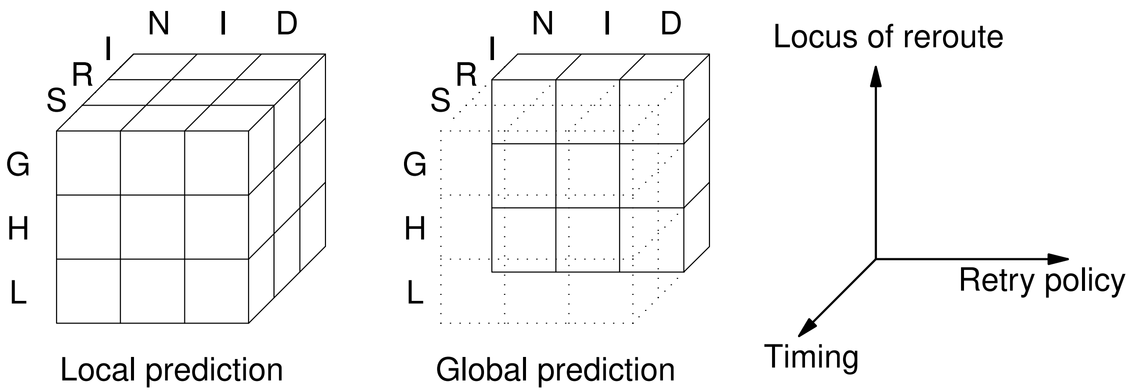

Figure 3.5: Valid regions of the 4-dimensional solution space

The Local prediction is valid with all combinations of the other factors. Global prediction, for the reasons mentioned above, is only useful with Immediate timing. Thus, the valid schemes in the 4-dimensional space lie in the regions shown in Figure 3.5, which shows the 4-dimensional space as two 3-dimensional spaces, one for Global prediction and the other for Local prediction. In the space for Local prediction, all the schemes are valid. These schemes are shown in solid lines. In the space for Global prediction, the regions containing the Random and Sequential timing approaches do not contain reasonable schemes. These regions are bounded by dotted lines.

3.8. Conclusion

In this chapter, we have formulated a framework for the systematic study of recovery of realtime channels. A promising portion of the solution space has been isolated by choosing greedy algorithms over ones which try to provide optimal solutions, distributed schemes over centralized control, and dynamic strategies rather than pre-computation methods. Figure 3.2 shows the sector of the solution space thus identified. We have chosen not to explore some issues by adopting the DCM protocol for changing the route of a channel, the Bellman-Ford algorithm to select the min-hop route meeting the performance and traffic constraints, and the reliable-broadcast-based route update protocol. We have also chosen not to address the problems of fault detection and return-to-normal, although we do return to them briefly in Chapter 5. The issues to be investigated are locus of reroute, reroute timing, network state prediction, and retry mechanisms. We have identified some possible approaches to each of these factors. The combination of these approaches define schemes, although not all of them are reasonable ones, as Figure 3.5 shows. In the next chapter, we proceed to explore the interesting schemes by simulation.

-- --

Chapter 4: Evaluation of fault recovery

4.1. Introduction

The problem of network fault recovery has been extensively studied. Chapter 1 provides a survey of existing work exploring various aspects of fault recovery in the context of telecommunication networks and computer data networks. The field is relatively mature and a good understanding of the issues involved in recovering from fault in these networks exists. However, as we saw in Chapter 3, the assumptions for realtime networks are considerably different. The situation is closest to that found in the cross-connect layer of telecommunication networks, but the solutions used there cannot be applied directly to the new environment without considerable thought and experimentation. In Chapter 3, we looked at the issues involved in redesigning the fault recovery mechanisms to work with realtime networks. We presented some initial design choices, which narrowed the solution space, and then proposed some orthogonal factors that would allow us to explore the remainder systematically. In this chapter, we present the results of the simulation experiments that we performed. We first describe, in Section 4.2, the simulation tools, and then present and justify the experimental methodology. This includes identifying metrics of comparison that reflect the objectives of fault recovery specified in Chapter 3. We also identify the internal and external factors that are expected to effect the performance of the schemes. In Section 4.3, we present a series of experiments and results, by which we explore the chosen solution space for algorithms to reroute realtime connections after a network failure. Our conclusions from these experiments are summarized in Section 4.4.

4.2. Simulation design

The recovery schemes were designed with as few assumptions about the underlying realtime schemes as possible. However, in order to evaluate the sensitivity of the schemes to the external factors (such as the topology, the load, the traffic mix, and the number of faults) and to pick the best combination of approaches for the internal factors of the scheme, we need to build a simulation model which specifies all the details of the underlying realtime network. Fortunately, the author had access to a simulation package for realtime communication on packet switching networks written by Hui Zhang. This was extended to include signaling and admission control using the Dynamic Channel Management protocol by Colin Parris. This simulator provides a detailed packet level simulation of the extended realtime channel scheme described in Section 2.5. The schemes for fault recovery described in the previous chapter were implemented on top of this platform by the author.

In this section we describe the framework of the set of experiments performed using this simulator. We first present an overview of the simulator. Then we consider the details of the realtime network being simulated, and briefly describe the scheduling mechanism used for packet forwarding and the admission control tests used during channel establishment, as implemented in this particular simulator. In the context of the scheduling discipline described, we present a metric of load that eases the task of comparing the performance of different rerouting schemes. We define metrics of performance that reflect the objectives of fault recovery presented in Section 3.2. Then, we describe the methodology of the experiments by identifying the internal and external factors which affect our chosen metrics. The external factors such as topology and load are elements which are not under the control of the scheme; thus, we study the sensitivity of the schemes to these factors. The internal factors are the variables within the scheme that we discussed in the previous chapter. The simulation experiments are intended to measure the effect of each factor on the overall success, as well as to evaluate the choices that can be made within each factor, in order to maximize the reroute success. This forms the body of our simulation experiments, where we systematically try the various combinations of approaches to each factor, and observe the effect on our metrics of performance.

4.2.1. The simulator

The simulator is based on the DCM simulator built by Colin Parris and Hui Zhang (51). It allows the user to specify a network topology, by defining the set of vertices of a graph (which correspond to the routers and hosts in the network) and the set of edges which join the vertices (which correspond to point-to-point links in the network). The edges have properties such as link speed, and scheduling discipline, which are specified by the user. The user also specifies a set of initial network events, which cause other events to be triggered in the simulator. The main type of event specified by the user of the DCM simulator is the channel establishment event. Each channel establishment event is characterized by the traffic and performance parameters of the channel to be established. The traffic parameters supported by this simulator are $Xmin, Xave, I$, and S_{max}. The simulator only supports deterministic delay bounds; thus, the only performance parameter is D_{max}.

The channel establishment event causes the simulator to trigger a round-trip channel establishment. When the establishment is complete, packet transmission events are triggered from the source node at intervals defined by the traffic parameters. This simulates data transfer along the channel.

The DCM simulator also allows its user to specify alternate establishment events, in which an existing channel is subjected to parameter or route modification. The alternate establishment event requires the user to specify the new traffic and performance parameters; a new route may be optionally specified.

Thus, the simulation is started with a set of events from an event file, each of which is associated with a time and other necessary parameters. As the simulation progresses, these events are processed in chronological order. The processing of an event such as a request for establishment in turn causes several events to be created, such as a message being sent from the source of the request to its neighbor, or the generation of data packets after the successful establishment of the channel. All events are maintained in a calendar queue (13), and, as simulation time elapses, the next event is removed from the queue and processed.

The events simulated in the simulator include:

- Arrival, queueing, and scheduling of packets in nodes
- Transmission and propagation of packets on the links
- Arrival of requests from clients leading to channel administration messages being sent to neighbors
- Arrival of channel administration messages leading to admission control tests or tear down calculations being performed
- Arrival of route update messages leading to modification of the routing tables

The author modified the above simulator to permit the specification of fault events. When a fault event occurs, the simulator automatically generates the necessary alternate establishment events necessary to reroute the affected channels. The route, the parameter specifications, and the time of the event are generated according to the rerouting scheme being simulated.

A rerouting scheme is the combination of the approaches to each of the internal factors described in the previous chapter, i.e., one scheme may be Immediate timing, Global locus of reroute, No retries, and Local prediction. The input file allows each of these internal factors to be set to a particular approach. The simulator models the chosen scheme during the simulation, selecting routes, timing the establishment attempts and retries, and performing state prediction automatically. Thus, the simulator can model any of the schemes in the solution space depicted in Figure 2.5. All the approaches to the internal factors described in the previous chapter are implemented. The reasonable schemes in the region of the solution space described in the previous chapter were tried out in simulation.

The simulator also models the route update protocol, which modifies the local databases on which the nodes in the network make routing decisions, using a reliable broadcast method as described in Section 2.5.3. This allows us to study the effects of incomplete or out-of-date information, and the performance of algorithms to counteract these effects, in our experiments.

The simulator does *not* model the computational times for running the algorithms on the nodes. This time is assumed to be zero in the simulation model. This can be justified by measurements of the time to run the various pieces of code, as implemented in the simulator and run on a DECstation 5000/240 (53). The time to perform the simple RCSP admission control tests (described in Section 4.2.3) in the nodes on the forward and reverse pass of the establishment process (measured value = 3.3 μs) is negligible compared to the link propagation delays (10 ms). Since one execution of the test is performed for each link crossed, the inclusion of this computation time in the simulation would make a very small difference to the results. The time to run the routing algorithm on the "mesh" topology (Figure 4.3) is around 1 ms, which is small compared to the round-trip time to perform the establishment (average $\approx$ 100 ms). Again, since routing is performed once per round-trip establishment attempt, the inclusion of this computational latency in the simulation model would make no more than a few percent difference in the results.

4.2.2. The scheduling discipline

One of the fundamental processes being simulated is packet scheduling. As control or data packets arrive at a node in the simulator, they are queued for service, and served in the order specified by the scheduling discipline at the node. The link (together with the scheduling mechanism) can be viewed as a queueing server; there is one such server for each link on each node of the network. The scheduling discipline used for packet forwarding in our experiments is Rate Control Static Priority (RCSP) scheduling (71). We chose to use RCSP in our simulations for a number of reasons. RCSP separates the issues of rate control and delay control by conceptually dividing the RCSP server into two modules, as described below. This separation makes it possible to independently allocate delay and bandwidth, allowing simple and fast admission control tests. This simplicity allows us to analyze the effect of load on an RCSP network, making it easier to design metrics of load and performance, which we use to present our results. Finally, the simpler model facilitates our understanding of simulation results and eases the task of recognizing unreasonable results. This was especially useful in the earlier stages of our investigation, when we were debugging the simulator.

The rate control module enforces the traffic specification constraints on a per channel basis; if streams become more bursty (such as might happen as a result of passing through a number of servers), they are smoothed out by the rate controller, so that they will obey the traffic constraints again. This also serves to police the sources at the inputs to the network. The static priority module assigns each channel to one of several priority levels, each of which has an associated delay bound. Each priority level conceptually has a separate queue, and packets are served from the highest queue with a non-zero queue length. The admission control tests are responsible for keeping the admitted load for each level below the point where packets would violate the associated delay bounds. Figure 4.1 shows a conceptual picture of the operation of the RCSP scheduler, reproduced from (71).

4.2.3. The admission control tests

The admission control tests implemented in the simulator enforce a condition for call admission that is sufficient, but not necessary, for the satisfaction of performance guarantees during packet forwarding. In other words, the bounds are not the tightest ones that can be obtained. The calculations only take into account X_{min}; thus, the bandwidth reservation is peak rate. However, the tests require minimal computation during establishment. They are based on the following theorem, which is reproduced from (71).

Theorem 4.1: Let $d_1, d_2, \cdots, d_n$ $(d_1 < d_2 < \cdots < d_n)$ *be the delay bounds associated with each of the n priority levels, respectively, in an RCSP server. Assume that the* j^{th} *connection among the* i_k *connections traversing the server at priority level* k *has the traffic*

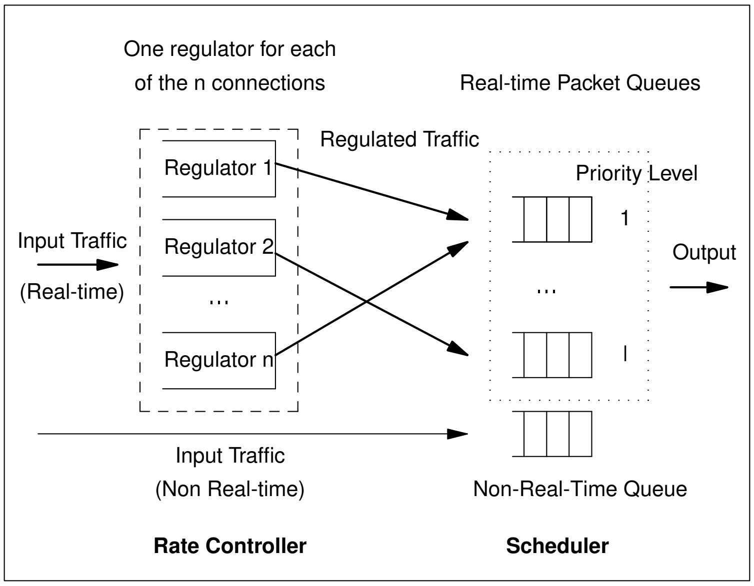

Figure 4.1: Rate Control Static Priority scheduler

specification $(X_{min}^{k,j}, X_{ave}^{k,j}, I^{k,j}, S^{k,j})$*. Also, assume that the link speed is* l*, and the size of the largest packet that can be transmitted onto the link is* S_{max}*. If*

$$\sum_{k=1}^{m} \sum_{j=1}^{i_k} \left\lceil \frac{d_m}{X_{min}^{k,j}} \right\rceil S^{k,j} + S_{max} \leq d_m l \tag{4.1}$$

the waiting time of an eligible packet at level m is bounded by d_m*.*

Theorem 4.1 gives the admission control condition that needs to be tested. For each priority level m, a state variable B_m is maintained and initialized to S_{max}. When an establishment request for a new connection with traffic specification (X_{min}, X_{ave}, I, S) comes in, the following tests are performed for $m = 1$ to n.

$$B_m + \left\lceil \frac{d_m}{X_{min}} \right\rceil S \leq d_m l \tag{4.2}$$

The new connection can be placed into priority level k without jeopardizing the performance guarantees of other existing connections if and only if Equation 4.2 holds for $m \geq k$. If the condition is satisfied and the decision to put the connection into level k is made, then B_m is updated for $m \geq k$ as follows:

$$B_m = B_m + \left\lceil \frac{d_m}{X_{min}} \right\rceil S \tag{4.3}$$

The intuitive interpretation of these tests is that Equation 4.1 computes a bound on the maximum busy period of the queue at priority level m. This is defined as the maximum contiguous period of time during which the queue at level m can contain one or more packets. A busy period starts when a packet arrives at an empty queue, and ends when the last packet in the queue is transmitted. Since the maximum delay any packet at this level can suffer is bounded by the maximum period, this serves as a sufficient (but not tight) upper bound on packet delays for packets on this level. The last term on the left hand side of Equation 4.1, S_{max}, accounts for the possibility that a best-effort packet was already in service when the busy period started, since the service is non-preemptive. Since the tests ensure that the new channel will always be rejected if the admission of the new channel causes the busy period bounds to exceed the delay bounds for any level, we can be sure that no packet in a channel belonging to level m will suffer a delay bound greater than d_m.

The above test takes care of the local delay bound for the channel. To ensure that sufficient buffers exist, a buffer test is also performed, and the required memory is reserved for the use of the channel. The details of the buffer test are not mentioned here.

4.2.4. The load index

One problem facing any investigation that attempts to compare schemes involving realtime channels is the one of choosing an index that characterizes the load imposed on a network by one or more channels. Finding a single number that captures the load is hard because it depends on at least the following components of the load: the number of realtime channels, the number of hops each channel traverses, the traffic specifications of each channel (which has peak and average rate components), and the delay requirements of the channels. A general solution to this problem is quite complex, and some simplification is usually chosen. For example, the sum over all existing channels of the product of the average rate (in bits per second) of the channel and the number of links over which the channel passes could be used as a load index:

$$\sum_{j=1}^{N} \frac{S^j I^j H^j}{X^j_{ave}} \qquad (4.4)$$

where N is the number of channels existing in the network, S^j, X^j_{ave}, and I^j are from the traffic description of channel j, and H^j is the number of links channel j traverses. This would capture the average rate and the number of links affected. However, this load metric would completely ignore the delay requirement of the channels.

Fewer low delay channels can be supported by a realtime network. Thus, if a low delay channel is accepted, more of the resources are reserved, and fewer channels can be subsequently accepted. The reservation due to delay bounds is harder to quantify than the reservations for buffer or bandwidth, but the admission control tests of Equation 4.1 express it algorithmically for the RCSP scheduling discipline. The establishment of a channel leads to a decreased capacity of the network to accept further realtime connections, and this decrease in capacity is greater if the delay bound required is lower. A good load metric should be able to quantitatively capture this effect. The exact mathematical properties of the reduction in realtime capacity depend on the admission control tests being used, which in turn depend on the packet service discipline in the network elements. The general problem is, thus, considerably harder, and we restrict ourselves to the specific case of Rate Control Static Priority (RCSP), with the simple admission control tests described above.

The static priority module of an RCSP server consists of n priority queues, each of which has an associated delay bound. For each level, the admission control mechanism maintains a variable, B_m (Eq. 4.3), which bounds the busy period for the corresponding level; the test consists of recomputing B_m for all levels $m \geq k$ when a new channel is introduced at level k, and ensuring that the new values are less than the corresponding delay bounds (d_m). Thus, adding a new channel at level k increases B_m for each level $m \geq k$ (Eq. 4.3). The busy period bound B_m is, therefore, a direct measure of the load on a particular server at level m. Since the placement of a channel at a higher priority (lower numbered) queue effects all lower priority (higher numbered) levels, the sum of B_m over all levels, if used as a index of load, has the desirable property that the placement of a channel at a higher priority level has a larger affect on it. Since $B_m = S_{max}$ in the absence of any channels, we need to subtract S_{max} from B_m to make the load index zero at a server without any established channels. We can normalize the index by dividing by the link speed, l, to make it it possible to compare the loads on links with different speeds. Finally, if we want to take into account the effect of load on the entire network, we can add the index at a server over all servers in the network. There is one RCSP server for each directed edge in the network graph (one in each direction for every link). The final index of load is defined as follows:

$$load(t) = \sum_{e \in \mathbf{E}} \sum_{k=1}^{n} \frac{B_{k,e}(t) - S_{max}}{l_e}, \tag{4.5}$$

where $load(t)$ indicates the network load at a particular time t, and changes as channels are set up or torn down, $\mathbf{E}$ is the set of directed edges in the network, n is the number of levels in the RCSP priority queue module, $B_{k,e}(t)$ is the value at time t of the maximum busy period as defined in Equation 4.3, for level k at the server for the directed edge e, and l_e is the link speed for the directed edge e. Load is measured in time units.

This index of load has many desirable properties. It is

- monotonically decreasing as a function of the delay requirements of the channels if all other parameters are kept constant
- linearly increasing in the peak rate requirement of the channels (since the reservations are peak rate)

- linearly increasing with the number of links affected
- additive to a first approximation over sets of channels

The additive property only holds if the channels in the set do not interfere with each other, as is the case when, for instance, a channel takes a different route or a different priority level because some other channel occupies a given set of resources. These properties are shown in Figure 4.2.

Apart from the intended use of this load index to design metrics to compare recovery schemes, we can think of several other applications of the index. For instance, since the index captures the effect of a channel or a set of channels on the network, it could be used for pricing. Such a pricing scheme, where the price of service is directly related to the effect on the network, has several advantages. For example, it would encourage users to use the network efficiently, since minimizing the cost to the user has been linked to minimizing the effect on the network. In addition, since the metric captures the resource state of an RCSP server in a single number, it may be used to guide the route selection process. This would reduce the amount of information to be carried for each link by the route update protocol, and also reduce the computation to be performed for route selection, which may be valuable for extremely large networks. We will not explore these possibilities any further in this dissertation.

4.2.5. Metrics of comparison

In order to compare the performances of different schemes, we have to be able to quantitatively define what we mean by good performance. There are several orthogonal criteria that must be considered. The amount of realtime load successfully recovered is a major concern, since it has a significant impact, to both the network provider and the clients. The time to perform the recovery is another criterion of importance. Finally, the effect on network resources is of concern to the network provider, since the recovery may come at the cost of significantly lower remaining network capacity after the

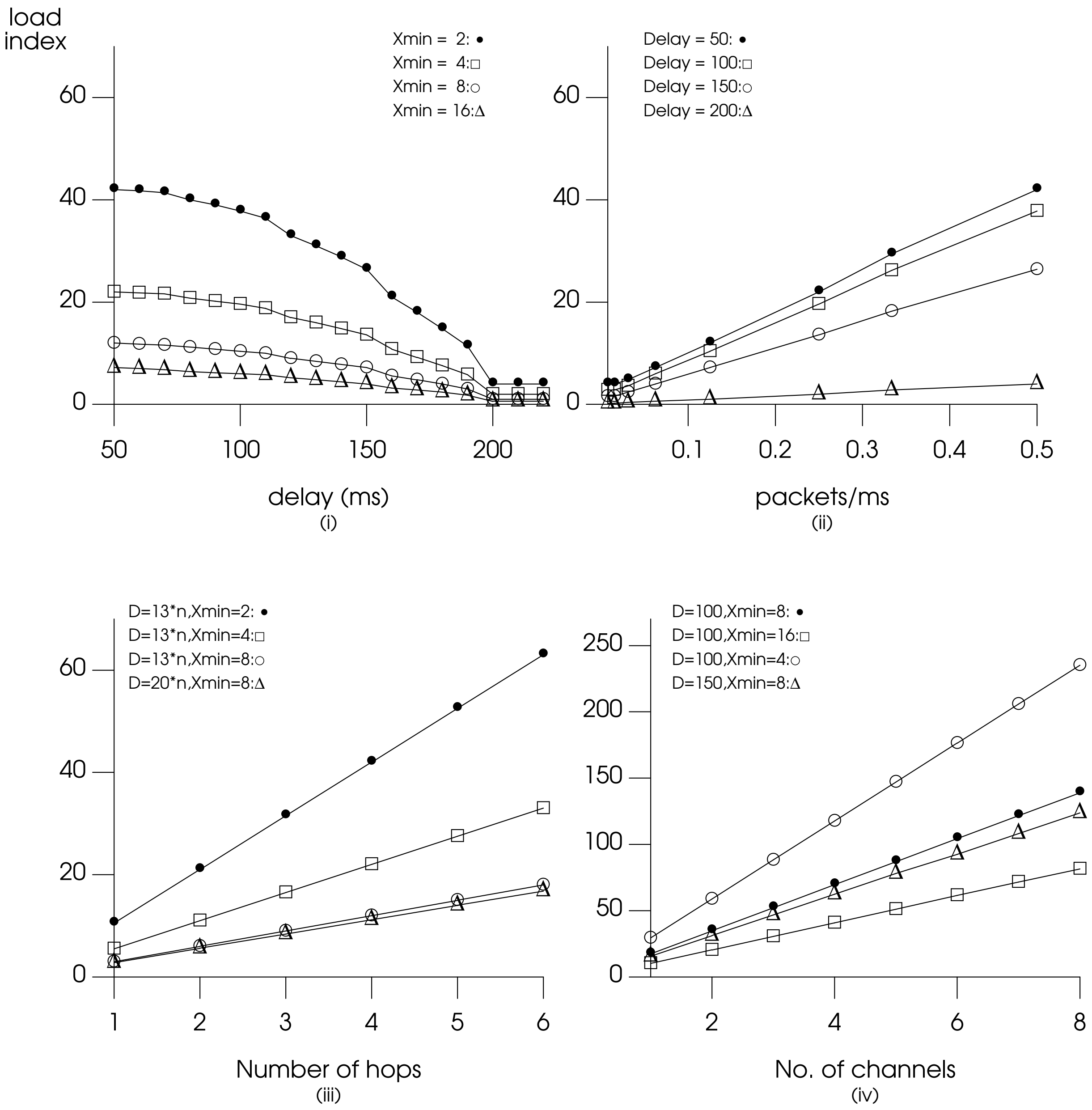

Figure 4.2: Properties of the load index

recovery process has completed, if the resources are inefficiently allocated.

Fraction of realtime load rerouted

We would like to compare the amount of realtime load successfully rerouted by different rerouting schemes, but this is hard to do unless the term "amount of realtime load" is well defined. The problem here is that the sets of channels are heterogeneous, with

-- --

different traffic specifications and performance requirements. Hence, it is difficult to say if the amount of realtime load represented by one set is larger or smaller than another, unless we reduce all these factors to one number. The load index defined in Section 4.2.4 allows us to do that.

We first define $load(c)$, the amount of realtime load contributed by a single channel c, for a given network topology and assuming a well defined deterministic routing algorithm, as the load index as defined in Equation 4.5, measured when the channel is set up in the network with no other channels established in it. Of course, this is only well defined if the routing algorithm is deterministic, i.e., always computes the same route given the same input. The Bellman-Ford algorithm described in Section 2.5.4 and used in our simulations meets this requirement.

$load(c)$ is the minimal increase in network load that the establishment of channel c can cause on this network, and is indicative of the intrinsic resource requirements of the channel. The difference in the load index measured in a non-empty network might be larger than this, due to the channel taking a longer path or being assigned to higher priority levels if the lower levels are occupied. However, under the assumption that the routing algorithm always picks the most efficient path and the relaxation strategy always moves the channel to the lowest priority levels available, it can never be smaller. Thus, $load(c)$ is a measure of the lowest possible amount of resources required by the channel c.

We define the amount of realtime load of a set of channels as the sum of the loads of the individual channels. Thus, the amount of realtime load corresponding to a set of channels $\mathbf{C}$ is:

$$load(\mathbf{C}) = \sum_{c \in \mathbf{C}} load(c) \tag{4.6}$$

Note that the load index in the network, even with a fixed set of channels $\mathbf{C}$, will depend on the order and timing of the route computations and the establishments. On the other hand, *load*(**C**) is uniquely defined as long as the routing algorithms and establishment protocols are fixed. Its value is independent of the ordering of the channels. Under the assumptions that the routing algorithm and relaxation mechanism always pick the shortest paths and lowest priority levels possible, *load*(**C**) is less than or equal to *load*(*t*) after establishing the set of channels **C** in the network.[1]

Finally, we can define the metric to measure the fraction of realtime load rerouted following the occurrence of failure. The Success Ratio (SR) is defined as the fraction of affected realtime load that is successfully rerouted, expressed as a percent:

$$SR = \frac{load(\mathbf{R})}{load(\mathbf{A})} * 100 \tag{4.7}$$

where **A** is the set of affected channels, and **R** is the set of successfully rerouted channels.

Speed of recovery

The speed of recovery can be measured in terms of the average time to restore service to a channel in the event of a failure. This is measured over the set of successfully rerouted channels. The maximum time to recover any channel is also of interest, since this is a measure of the time for the network to stabilize, as well as an indication of the worst case recovery time. Both these numbers can be easily obtained from our simulations.

Efficient use of network resources

We can expect the rerouted channel to use more resources than the original channel, since the shortest path is blocked by the failed link. In general, the more highly loaded the network, the longer the alternate routes found will be, leading to more and more inefficient use of resources. However, the rerouting scheme may itself lead to longer reroute paths, and hence, to inefficient resource use. For example, the Local rerouting approach will always tend to use longer paths than the Global approach, and the

[1] We have overloaded the term *load*(X), but there should be no cause for confusion, since the type of the argument indicates which definition is to be used.

penalties of this must be assessed. We would like to do this in a way more general than comparing path lengths. The load index defined previously allows us to do that by defining the excess resources (ER) consumed by a set of reroutes performed on a network in the event of a failure as the difference between the actual resources used in the network to support the successfully rerouted channels using the set of paths chosen by the rerouting scheme and the minimum resources required to support the set of successfully rerouted channels **R**, expressed as a percent of the minimum resources required:

$$ER = \frac{actual - minimum}{minimum} * 100 \tag{4.8}$$

The resources actually used can be experimentally obtained (in simulation) by measuring the network load after the recovery process terminates, then tearing down the set of channels **R** and measuring the load again. The difference in the two loads corresponds to the resources actually used. The minimum resources required is approximated by *load*(**R**). Thus, we have:

$$ER = \frac{load(t_R) - load(t_{TD}) - load(\mathbf{R})}{load(\mathbf{R})} * 100 \tag{4.9}$$

where $load(t_R)$ is the load at the time the last successful reroute completes, $load(t_{TD})$ is the load after tearing down the set of successfully rerouted channels R, and $load(R)$ is the load as defined in Equation 4.6 for the set of successfully rerouted channels **R**.

4.2.6. Factors affecting performance

Now that we have defined what we mean by the performance of a recovery scheme, we can move on to identifying the set of factors that will affect the performance. The experiments will then consist of changing the factors and observing the effects on the performance indices.

The factors affecting performance can be broken into two classes. Those in one class, the external factors, are independent of the schemes, and we need to make sure that the schemes work well in the face of variations of these factors. This class of factors contains the network topology, the operating load on the network, the nature of the applications, and so on.

The second class consists of the internal factors of the recovery schemes. In the previous chapter, we have identified a set of orthogonal factors, such as routing constraints, timing, retries, and state prediction, that are expected to affect the performance of recovery in different ways.

4.2.6.1. External factors

The external factors that are considered during our simulation experiments are the network load, the failure mode, the network topology, and the traffic mix on the network. We will measure the performance of our schemes and how it changes when we vary the external factors. While, in general, a network will be designed keeping these factors in mind, so that in the common case the network is operating with desirable characteristics such as low blocking probability for new channel requests, or high percentage of recovery in the event of a failure, we should try to design schemes that degrade gracefully when the operating environment becomes harsher. Thus, one interesting set of experiments for any given scheme is to see if the performance is sensitive to variations of these external factors.

Topology

The topology of the network can be specified in the input file of the simulator. The topologies that we have chosen to examine all have high levels of redundancy. This is necessary for the success of the rerouting mechanisms, since they work by finding alternate paths from the source to the destination when the fault occurs. However, we still need to look at the effect of changing the topology on the schemes, since we would like the scheme to be general.

The first topology we picked for our initial experiments is referred to as the "mesh" topology in the rest of the thesis. It is shown in Figure 4.3. Each link in the network has a

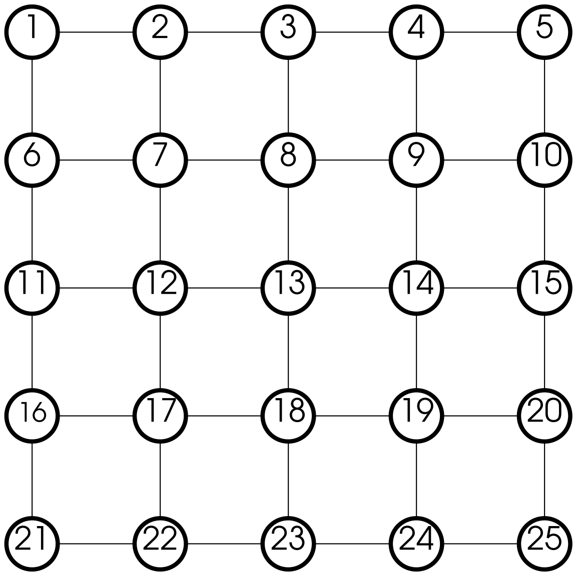

Figure 4.3: Square mesh topology

propagation latency of 10 ms, and a capacity of 10 Mbps. We chose this topology because it is simple and regular, which made it possible for us to interpret our results more easily. It also made it easier to recognize unreasonable results, facilitating the tracking down of bugs in the initial phases of our experiments. At the same time. the mesh topology has multiple disjoint paths between every pair of nodes. It is 2-edge-connected, since vertices near the corner have only two disjoint paths between them. However, vertices near the center have larger connectivity, as high as four. We considered folding the graph into a torus to make the situation symmetric, but, we felt that it would not be representative of realistic topologies. Real local and metropolitan area networks have almost planar topologies, which can be abstracted by a square mesh as we have used.

Of course, real computer networks may have links joining vertices which are not geographic neighbors, making them not perfectly planar. Thus, we chose to enhance the mesh graph with a few “trunks” to increase its connectivity and make it less regular (Figure 4.4). The trunks have a propagation latency of 25 ms and a capacity of 50 Mbps, all the other links have the same characteristics as in the mesh network. We used this as a second test topology, to check whether the change in the topology would change our

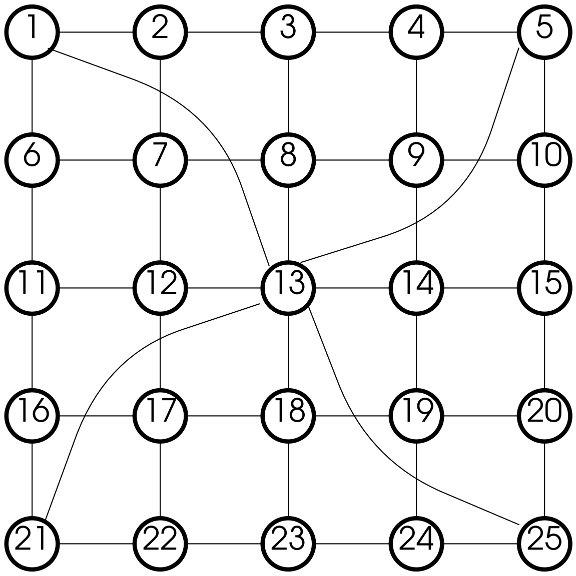

Figure 4.4: Square mesh topology with trunks

results completely. The new graph is 3-edge-connected, since the nodes in the corners have improved connectivity. We refer to this topology as the "mesh with trunks".

The two above graphs are similar, hence, in order to make sure that a completely different topology did not adversely affect our schemes, we tried one more topology based on the idea of a highly connected core, with peripheral nodes connected to the core. Many networks found in real life (such as telephone networks and airline route maps) look like this, since such topologies provide good connectivity, with short paths between any pair of nodes, at reasonable cost. Figure 4.5 shows the third network topology, "core", used in our simulations. The central core (nodes 1 to 6) is a complete clique of size six, with all links having a capacity of 10 Mbps and a latency of 10 ms. The peripheral nodes are locally gathered into two rings, and connected to three of the geographically closest central nodes to ensure redundancy. The ring is composed of links, each of 5 Mbps capacity and 10 ms latency. The links connecting the peripheral nodes to the central core are also 5 Mbps and 10 ms links. The graph is 3-node-connected, since we can find at most three node-disjoint paths between peripheral nodes in separate rings. However, we can find five *edge*-disjoint paths between any pair of nodes. Nodes within

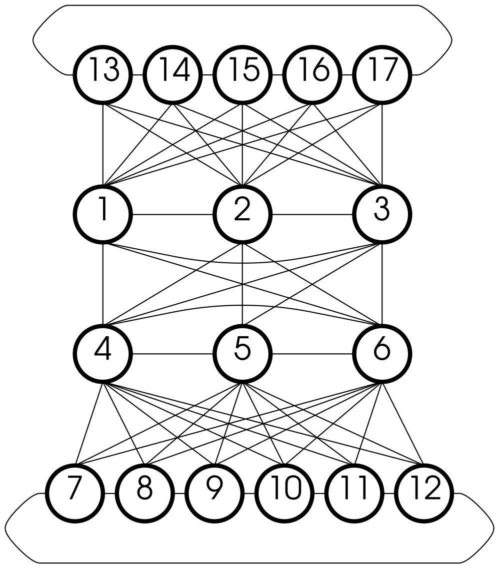

Figure 4.5: Core topology

the same ring have both a node-connectivity and a link-connectivity of five.

All our sample topologies are well connected. We must keep in mind that our schemes will not work well on poorly connected network such as trees. However, if our schemes work well on these three topologies, we can feel more secure that their success is not dependent on a particular topology, but that they will work well on any well connected network.

Traffic Mix

The nature of the applications on the network would determine the traffic parameters and the performance requirements demanded of the network. The choice of these parameters in a simulation is a difficult one, since no clear picture exists as of today of the traffic characteristics and performance requirements of applications which will run on realtime networks when they become widely available. We must make the best guesses we can about the classes of applications, their requirements, and the relative numbers of different types of applications using the network. For our experiments we chose three classes of channels: one-way JPEG-compressed medium-quality video conferencing channel: 2 Mbps (class A), CD quality audio channel: 200 Kbps (class B), and telephone quality audio channel: 64 Kbps (class C). The admission control tests used in the simulator only consider X_{min} and the delay bound d_m, so we keep $X_{ave} = X_{min}$. The value of I, therefore, is not significant. S_{max} was set to 10,000 bits for all packets, since this gave a reasonably small (1 ms) transmission time on the 10 Mbps links. X_{min} was calculated accordingly, from the rate requirements of each class.

We generated our first set of workloads from a distribution containing 30% class A channels, 30% class B channels, and the rest from class C. The source and destination of the channels were selected at random from the nodes on the periphery of the mesh. The delay requirement for the channels was generated from a uniform distribution within $(x + 50, x + 100)$ ms, where x is the one-way minimum propagation delay from the source of the channel to its destination. Thereafter, different workloads were created by using different seeds for the random number generator to get different sets of channels. The number of channels in the set was varied to get workloads with different values of the load index, as explained in the next section. The workloads generated with these distributions are referred to as the "original mix" in the rest of this chapter.

To see if the nature of the application mix had a serious impact on our simulation results, we generated another set of workloads with the following characteristics. The distribution of the channels was changed to 10% class A channels, 40% class B channels, and 50% class C channels. The distribution of source destination pairs was also changed so that channels would always go either left to right or up to down in the mesh. Finally, we changed the values of the load levels for this distribution (as explained in the next section). This distribution is referred to as the "new mix" in the results.

Load

We perform all our simulation experiments for four levels of network load, keeping all other factors constant. This allows us to see how the scheme under consideration behaves as load in the network increases. The load for the simulation is determined by the set of channels alive in the network when a fault occurs. This is determined by the set of establishment events in the event file before the fault event. We generate different loads by creating event files with different numbers of channels from the distributions defined above.

We use four load levels in each set of experiments, which we refer to as "low," "medium," "medium high," and "high". They correspond to different values of the load index defined in Section 4.2.4. The values used in one set of experiments (part of the "original mix" workload) are shown in Table 4.1. Also shown are sample utilization values from one of the workload files, to help the reader get a better feel for the meaning of the load index in terms of a more familiar metric of load. The average utilization is the total reserved rate expressed as a percentage of link capacity averaged over all links in the network after the establishment of all the channels in the workload file, while the maximum is the utilization on the most heavily loaded link. We must, however, emphasize that the load index captures more information than is captured by utilization, since a set of channels with identical rate parameters but different delay requirements would have the same utilization values but different load indices.

level	load index	average utilization	max utilization
low	270	15.8 %	63.3 %
medium	616	32.0 %	84.7 %
medium-high	777	41.2 %	87.3 %
high	858	44.5 %	86.7 %

Table 4.1: Load index levels

The different network topologies have different capacities; thus, different values of the load index were used in experiments involving different topologies. In addition, for the "new mix" distribution we also increased the load values to see if that would change the behavior of our schemes. Thus, while each set of experiments has four load levels, the values may differ and will be printed on the load axis of each graph.

Each set of experiments was conducted with twelve workload files, consisting of three workload files at each of the four load levels. The three workload files at a given level were generated with the number of channels picked to give a load value close to the desired load index when all the channels were set up. The results from each set of three were averaged to increase our confidence in the results. Each of these experiments takes an hour or more to run, depending on the class of workstation the simulator is running on. Moreover, each experiment had to be repeated across a large number of external factors and internal factors (described below). Thus, conducting a significantly larger number of experiments at each load level, while desirable, was not feasible.

We are using the results to compare the performances of schemes. The numerical values obtained as results are specific to the network topology and load mix. In addition, the numerical values depend very much on the specific set of channels to be rerouted, so that even at the same load level they vary widely. However, some clear trends are observable about the *relative* performance of the schemes. Thus, even though the numerical values do not converge, some schemes perform consistently better than others. This is the kind of result that we shall present in the simulation section. Since we did not perform each experiment a large number of times, statistical tests of confidence could not be conducted.

Failure model

The nature of the failure being simulated has an influence on the ability of the schemes to recover from the failure. We place certain restrictions on the failures being simulated. We only deal with link failures.[2] We also deal only with *fail-stop* failures, in which the failed component does not continue to transmit bad messages into the rest of the network. *Byzantine* failures of nodes are especially hard to deal with, since it may not even be possible to detect and isolate the failure. In the simulations performed, when a

[2] Node failures may be dealt within this model by treating them as simultaneous multiple failures of all links connecting to the failed node. However, for simplicity we restricted the scope of the simulations performed to link failures.

link fails, it simply stops transmitting data.

Finally, multiple failures are harder to deal with than single failures, because more of the network load is affected, and less of the network is available to service the recovery. Multiple faults also aggravate the problem of collisions, discussed in Section 3.5.4. In addition, the physical distance between the two failures, and thus, between the two sets of channels being rerouted, would affect the timing behavior of the route update process.

Some links of wide-area computer networks may be leased lines, which are themselves routed on top of the cross-connect layer of a telecommunication network. Thus, it is possible for a single physical failure to be manifested as the multiple failures of two or more links of the computer network, unless care is taken to place logically different links (at the higher layers) in physically separate links.

We simulate single faults and double faults in our experiments. However, since a single fault is more likely, we consider recovery performance for the single fault cases more important than that in the double fault cases.

4.2.6.2. Internal factors

As described in the previous chapter, the internal factors of the recovery scheme are: locus of reroute, timing, retries, and state prediction. The simulator is programmed to accept parameters from an input file, which allows the user to select combinations of values for the internal factors of the scheme. We will experiment with the effect of the internal factors on the performance, to identify how much of an effect each factor has on it, to select the correct approach for the important factors, and to identify interactions between factors. The objective of these experiments is to come up with a scheme that provides good performance on all the metrics of comparison and is tolerant to variations in the external factors.

4.3. Results

In this section, we discuss the most interesting results of the simulation experiments performed using the framework described above. We first present the results concerning the effects of the internal factors, choosing as levels of the external factors the "mesh" topology, the "original mix", and single failures. Results are shown for all four load levels. The next four subsections show the effects of the four internal factors (timing, routing constraints, retries, and state prediction).

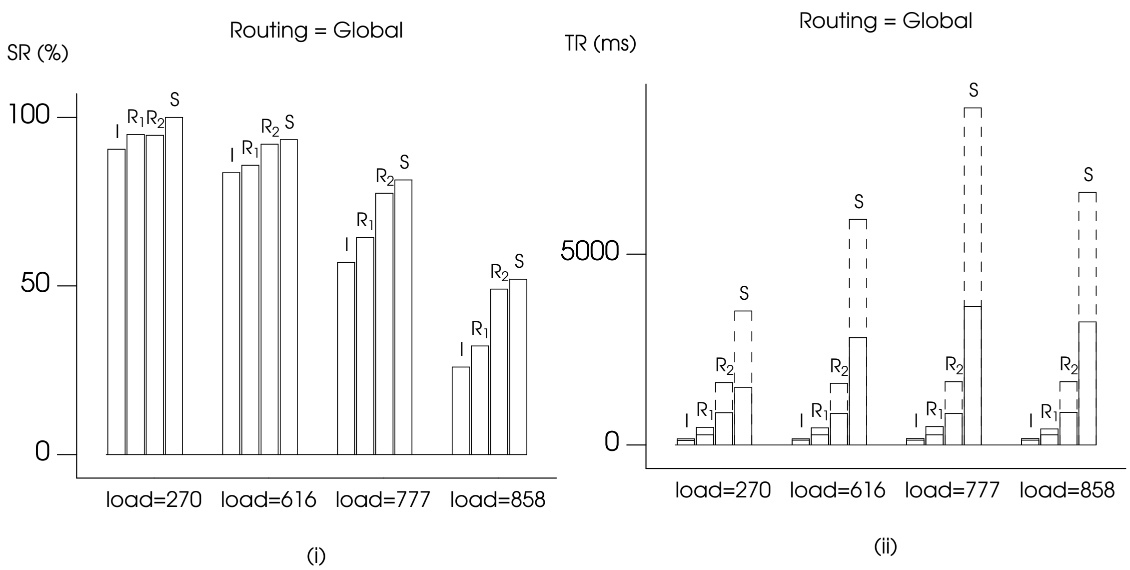

Figure 4.6: Effect of timing on (i) reroute success (ii) time to reroute

4.3.1. Effect of timing

Figure 4.6 shows the effect of varying the timing, with the other internal factors fixed (Global locus of reroute, no retries, and Local prediction). The timing is varied across Immediate (I), Random over 300 ms (R_1), Random over 1500 ms (R_2), and Sequential (S). Graph (i) shows the reroute success in terms of the Success Ratio (SR) defined in Equation 4.6. Graph (ii) shows the time to reroute (TR). The lower solid box shows the average time to reroute, averaged over the set of successfully rerouted channels, measured from the time of the fault to the time of successful re-establishment. The upper dashed box shows the time to reroute for the last channel to be successfully rerouted.

We observe that, while Sequential successfully reroutes far more realtime load than Immediate (roughly twice as much under high network loads), the time to reroute for Sequential rules it out as a practical scheme. However, as we observed before, it is a useful control scheme to compare the reroute success of other more practical schemes. Ideally, we would like a scheme with the success of Sequential, but the timing characteristics of Immediate. Randomization achieves this goal to a limited extent, since we reroute some fraction of the difference between Immediate and Sequential at a more reasonable time to reroute. However, there is scope for improvement, both in terms of the success ratio and the time to reroute.

The average time to reroute a realtime channel using Immediate timing is about 130 ms. The average path length (from the distribution of source-destination pairs that we used for this experiment) is five. Since each link has a propagation latency of 10 ms, the expected round-trip time would be close to 100 ms. The remaining time can be attributed to the time for the fault message to reach the source node from the location of the fault. The maximum time to reroute for the Random timing case is close to $n + 150$ ms, where n is the randomization interval, which is as expected. The timing characteristics of Immediate and Random are independent of the load, since they are determined by the time taken to perform a round trip, and the value of the randomization interval. However, the time to reroute for Sequential changes with the load, since it depends on the number of channels being rerouted. It increases initially, as more channels are being affected by the fault as the load increases. However, at high load it drops off, since less channels can be successfully rerouted, and the maximum and average are calculated over the set of successfully rerouted channels.

We only show the results for Random over 300 ms and Random over 1500 ms. Other values of the randomization interval follow the same trends.

4.3.2. Effect of locus of reroute

Figure 4.7 (i) shows the effect of the locus of reroute on the reroute success. The timing is fixed to Random over 300 ms. The results for other timing values follow a similar trend. We see that Local reroutes a far smaller set of channels than either Global or Hybrid, especially at higher loads. This can be explained as due to the limited choice in routes, since Global rerouting looks for a new route from the source to the destination, while Local rerouting only tries to replace the section of the route corresponding to the failed link. Thus, the set of possibilities considered by Global is much larger than that considered by Local. This reduces the chances of Local

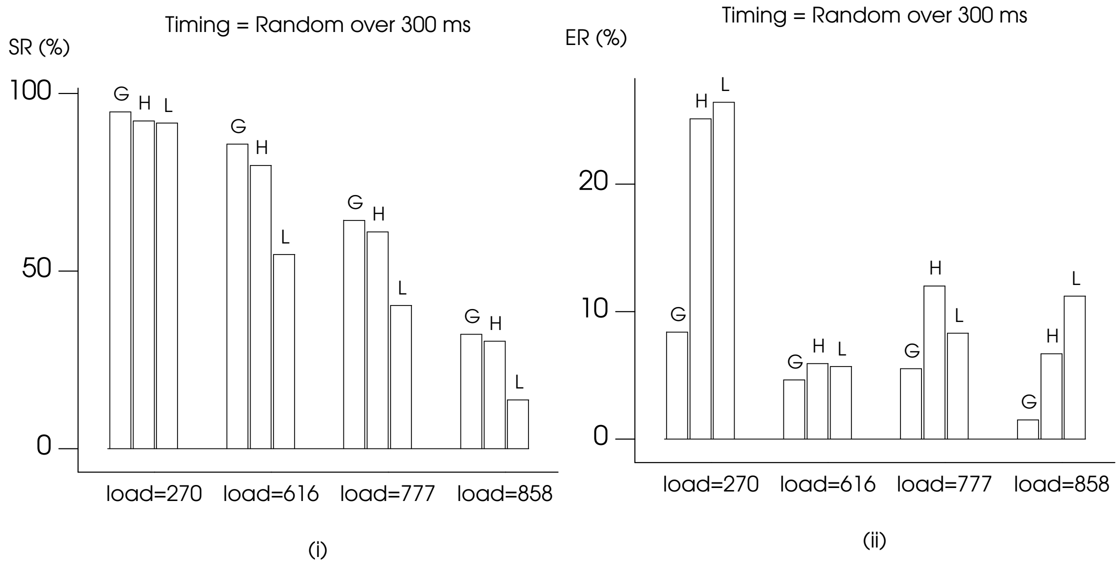

Figure 4.7: Effect of locus of reroute on (i) reroute success (ii) excess resources used

finding a route in heavily loaded networks, as well as makes it likely that any route found by Local will be longer than one found by Global. The performances of Global and Hybrid are comparable, since, during Hybrid rerouting, any time the Local route cannot be found, a Global route is sought. On the basis of these results, we have a strong reason to reject Local rerouting as the correct approach to locus of reroute, but have no clear choice between Global and Hybrid.

The effect of the locus of reroute on the efficiency of resource usage (ER) is shown in graph (ii). Global rerouting is more efficient than the other routing schemes, across all load values. This result holds across the other timing schemes too, and is mainly because Global tries to find the shortest existing path from the source to the destination. This goal also minimizes resource usage. Thus, from the point of view of resource usage, the performance of Global rerouting is best.

Changing the locus of reroute does not change the time to reroute; hence, we do not show a graph for it. The time to reroute for Local and Hybrid are very similar to that seen in Figure 4.6 (ii), for all the approaches to the timing policy. For example, for Immediate timing, the average time to reroute is close to 130 ms, regardless of the locus of reroute approach chosen. This is because the time to reroute is dominated by the round-trip time for establishment, and the random waiting time in the case of the Random approaches. Since the round-trip time is only marginally affected by the locus of reroute, and the waiting time not at all, the overall effect of the locus of reroute on the timing is negligible.

4.3.3. Effect of retry policy

The effect of the retry policy on the performance metrics is shown in Figure 4.8. Graphs (i) and (ii) show the success ratios for Immediate retries and Delayed retries. We fix the locus of reroute to Global and look at the performance across Immediate, Random over 1500 ms, and Sequential. At each load level we show six bars. The dotted bars depict the performance for the scheme with retries, the plain bars show the performance without retries. The performance for Immediate retries, with up to four retries in the event of failure to reroute, is shown in graph (i). As we can see, the success ratio improves when we add retries, but the improvement is smaller than the improvement achievable through randomization. Increasing the number of immediate retries does not appreciably improve the success ratio.

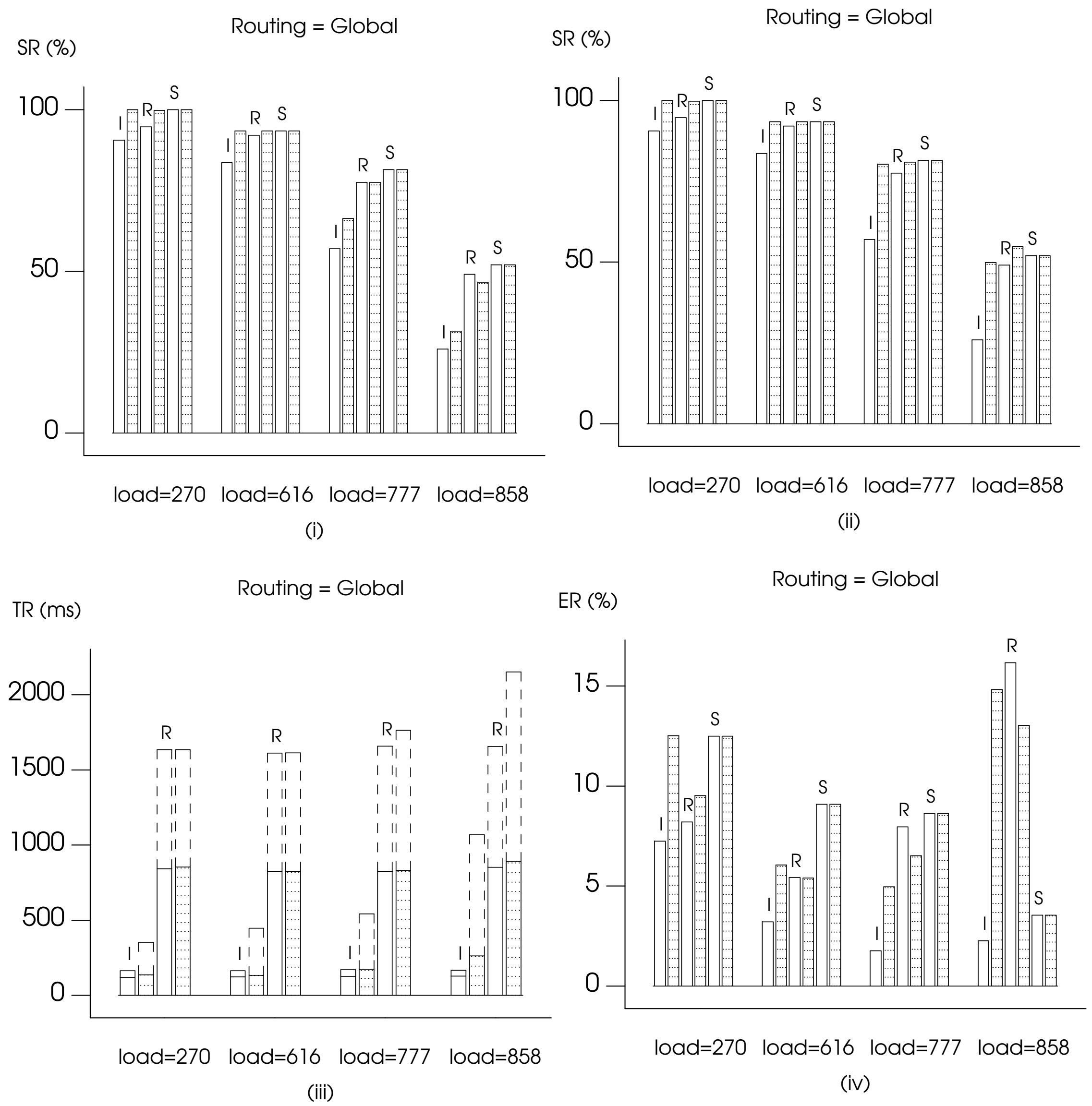

Figure 4.8: Effect of retry policy. (i) Immediate retries (ii)- (iv) Delayed retries

The success ratio for Delayed retries is shown in graph (ii), where we can see that this approach is much more successful. In fact, Delayed retries can reroute as much realtime load as Sequential timing, which is about the best we can do assuming a greedy strategy, with the same routing algorithm and locus of reroute constraints.

The performance of Delayed retries, as regards the time to reroute metric, is shown in graph (iii). We only show the time to reroute for Immediate and Random, with and without Delayed retries, since the time to reroute of Sequential is known to be very poor (Fig. 4.6, (ii)), and plotting it would make the graph harder to read, because the scale for Sequential is much larger. We note that the Immediate timing scheme, when we add retries, suffers from an increase in the average time to reroute, and a much larger increase in the maximum time to reroute. The retry policy used to generate the graphs shown is based on an exponentially increasing period[3] over which the random delay is chosen. Thus, the later retries are likely to occur after longer waiting periods.

From graph (iv) we can see that adding retries causes the resource usage to go up, but this is reasonable, since the amount of realtime load rerouted is also increasing. The resource usage for Global does not exceed 15% over the minimum required. The performance of Global in this regard continues to be better than that of the other schemes.

The time to reroute can be improved significantly without loss of reroute success or of efficiency of resource usage, if we fix the retry interval to the *correct* value, instead of starting from a small interval and exponentially increasing it. Picking the right value for the initial retry interval can make sure that most of the reroutes never go into a third retry. For the "mesh" network this value was experimentally found to be around 100 ms. This is also the expected round-trip time for the traffic mix chosen. This can be explained as follows: if the initial attempt failed due to a collision with another establishment in the network, the first channel should wait for the establishment to finish and for the route updates generated by the establishment to reach the source node (of the first channel). The channel establishment will take on average half a round trip time to complete, since on average it will be half completed when the collision happens. The information from the farthest node will take on average half a round-trip time more to get to the source node. Thus, we should generate our retry at this time.

Figure 4.9 shows the change in the histogram of successfully rerouted channels

[3] The range from which the random delay is chosen starts from a small value, and is doubled for each additional retry.

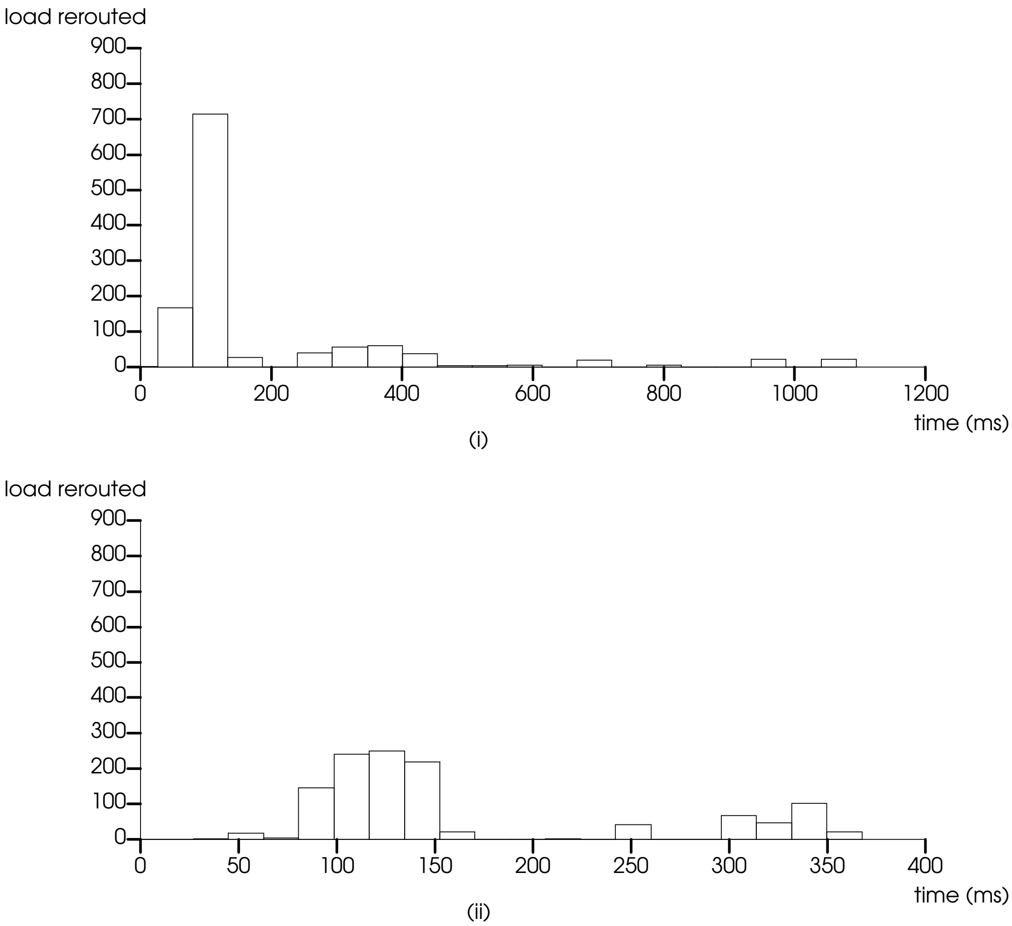

Figure 4.9: Effect of retry interval on the histogram of time to reroute (i) Exponential (ii) Fixed (100 ms)

(amount of realtime load vs. time to reroute), when we fix the retry interval to 100 ms. This was the best performance obtained with regard to the time to reroute metric out of a large number of experiments with different methods of selecting the delay for each retry, such as different fixed intervals, randomly chosen intervals, and increasing intervals. Picking the right retry interval can pull in the tail of the histogram dramatically. In general, the correct value of the retry interval will depend on the network topology, and the distribution of path lengths. Our hypothesis is that the average round-trip time is a good estimate of the value the retry interval should be set to. This hypothesis will be tested on a different topology in Section 4.3.7.

The performance is sensitive to the selection of the correct interval. Thus, this should be a tunable parameter of any implementation, to allow the value to be correctly selected for a particular network topology and channel path-length distribution. Also, evaluation of the variation of performance with respect to this parameter, in the context of a real implementation, would be valuable.

We also experimented with different numbers of retries, as well as using a threshold of time to halt the retry process. These allow the maximum time to reroute to be selected to adjust for different application requirements; any channel remaining after the retry process is halted is torn down and its user informed. While the number of retries and the threshold method have very similar performances, the latter appears to be a more natural way to specify the maximum recovery time needed from the network.

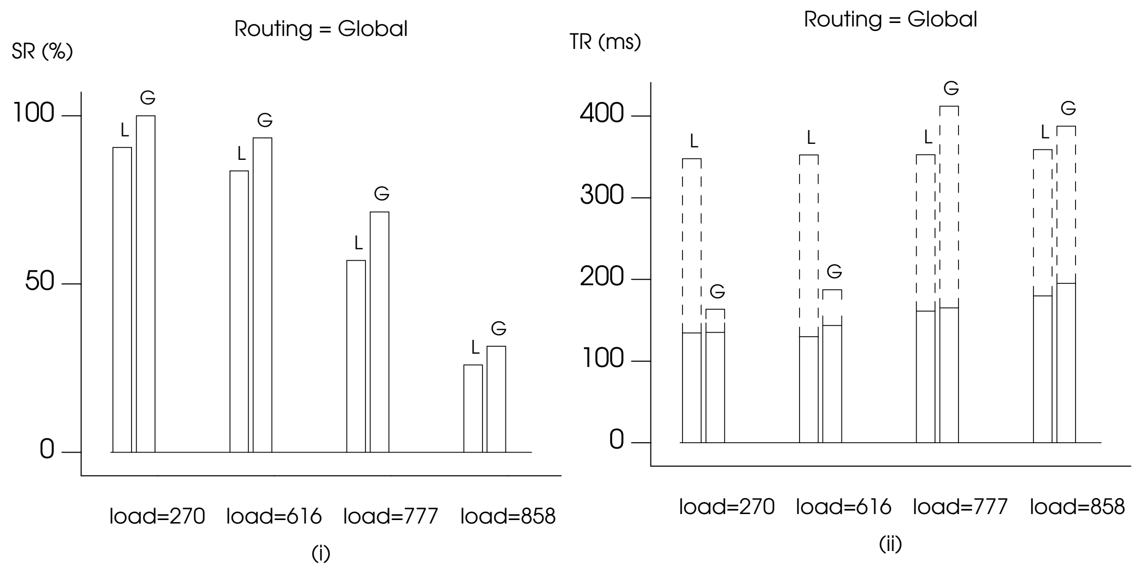

Figure 4.10: Effect of state prediction on
(i) time to reroute (ii) realtime load rerouted on first attempt

4.3.4. Effect of state prediction

Global prediction attempts to use knowledge of the behavior of other nodes in the network to place reroute attempts on paths where collisions will not occur. This improves the rerouting success on the first attempt, making the retry policy less important. This can be seen in Figure 4.10, graph (i), where the amount of realtime load successfully rerouted in the first attempt (without retries) is plotted. The two bars shown for each workload are "Local prediction" (L) and "Global prediction" (G). With Global prediction, the first attempt always reroutes more realtime load, leading to quicker recovery times for more clients.

However, this amount of realtime load is less than that recovered by the Immediate timing with Delayed retries scheme (the first dotted bar in Figure 4.8 (ii)). If we use Global prediction in conjunction with Delayed retries we can recover exactly the same amount of realtime load. However, the timing of the aggregate scheme is determined by the Delayed retry process, as shown in Figure 4.10, graph (ii). Both schemes use retries set to an interval of 100 ms, which we saw performed well. We see that Global prediction performs slightly better at low loads and comparably, or slightly worse, at high loads. The maximum time to reroute is affected more than the average. It seems that, as long as the load is low enough to allow the prediction algorithm to reroute all the realtime load in the first synchronized rerouting attempt, we get better timing with the Global prediction scheme. However, as soon as the load increases to the point where the scheme is forced to use retries, the timing is slightly worse than with the Local prediction scheme.

We can also see this in the histogram of the time to reroute (Figure 4.11). The histogram for the scheme with Global prediction, shown in graph (ii), has a larger area under 200 ms. However, because of the need to synchronize the reroute time across all the nodes, the scheme with Global prediction adds an extra 50 ms or so to the start of the recovery process. This shows up in the histograms as well as in the maximum time to reroute.

One element that is ignored in our simulations is the computation time to run the algorithms. We justified this in Section 4.2.1 by looking at the measured values of the computation time. However, that argument does not hold for Global prediction, since the

-- --

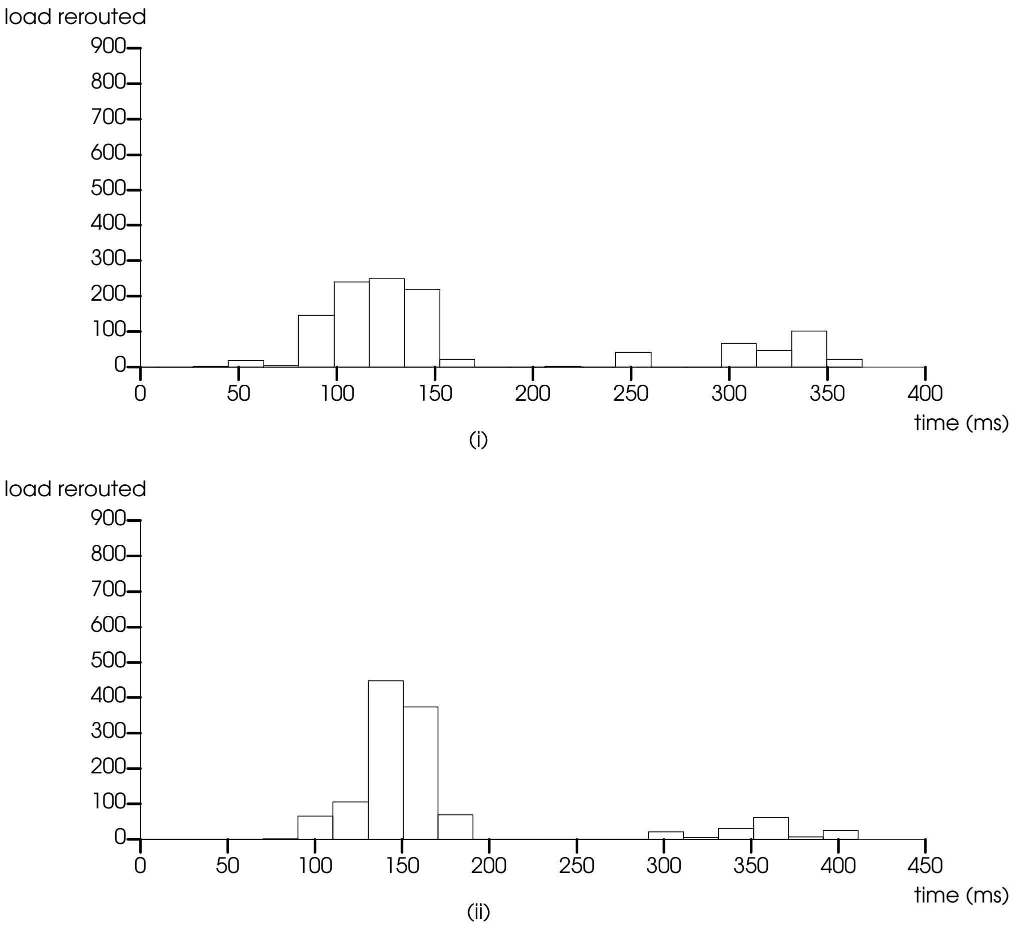

Figure 4.11: Effect of state prediction on histogram of time to reroute (i) Local prediction (ii) Global prediction

algorithm involves running the routing algorithm for all the affected channels at each node. The workloads being simulated have at most twenty channels affected by a single fault. Since the synchronized algorithm is run only once at the start of the recovery (if the retries are not using Global prediction, as was the case in the experiment that generated graph (ii)), we can approximate the effect of including this computation time by sliding the histogram to the right (increasing the recovery times) by 20 ms. This does not have a big effect on our results; however, it does make the Global prediction scheme, which already has a slightly higher maximum time to reroute than the Local prediction scheme, even less attractive.

Overall, it appears that, though Global prediction does improve the amount of real-time load rerouted in the first attempt, even more can be rerouted by using retries, and the timing of the Global prediction scheme with retries is dominated by the timing of the retry process. In addition, the extra overhead incurred by the Global prediction scheme at the start shows up in the form of a slightly higher maximum time to reroute.

Moreover, we note that the synchronization overhead to get all the nodes to start the establishments at the same time, as well as the extra computation time to run the Global prediction algorithm, would scale with the number of channels affected by the fault, and with the size of the network. For small networks, the Local prediction scheme seems to work already slightly better than the Global prediction scheme, and this difference should tend to increase with the size of the network. Thus, overall, the gains offered by Global prediction do not seem to justify the overheads and the increased complexity of the scheme.

4.3.5. Effect of number of faults

The (Global routing, Immediate timing, Delayed retry, Local prediction) scheme with a well-chosen retry interval works well in the case of a single fault. To see whether it continues to work for multiple failures, we simulated two faults in the network. Figure 4.12 (i) shows the success ratio for the Immediate (I), Immediate with Delayed retries(ID), and Sequential (S) schemes. The Delayed retry scheme works as well in the double failure case, leading to success ratios as high as in the case of Sequential timing. Graph (ii) compares the performance of the Global prediction scheme with that of the Local prediction scheme, by showing the amount of realtime load rerouted in the first attempt (i.e., with no retries). As before, the Global prediction scheme performs slightly better in the first attempt by rerouting slightly more realtime load. Graph (iii) compares the effect of two failures on average and maximum time to reroute. The two bars shown for each load are the Immediate without retries, and Immediate with Delayed retries set at 100 ms intervals, both with Local prediction. The timing of the Global prediction scheme with

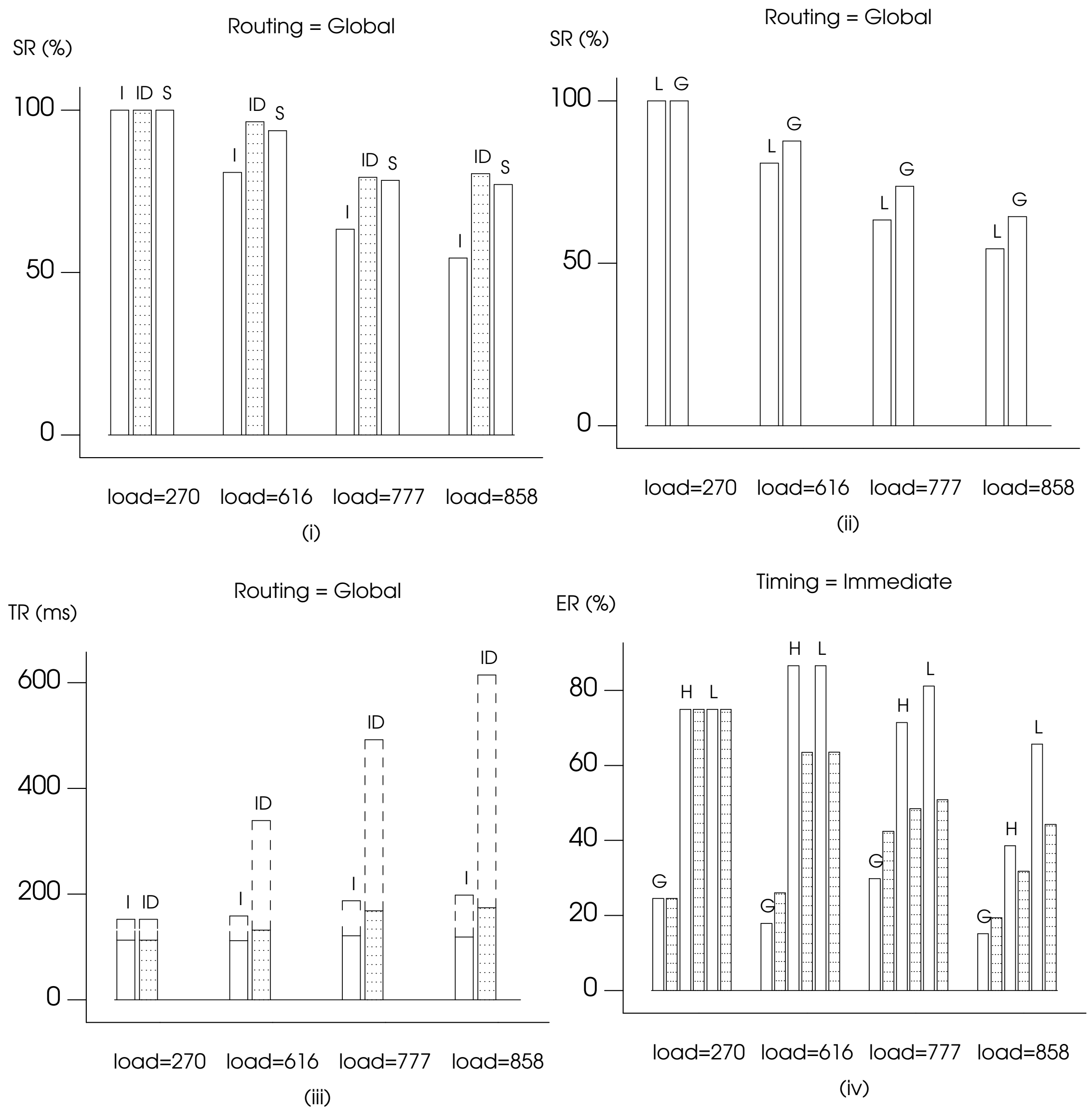

Figure 4.12: Effect of number of faults on
(i) reroute success (ii) success of prediction (iii) time to reroute (iv) excess resources used

retries is very similar that of the scheme with Local prediction (graph (iii)), and hence is not shown. The maximum time to reroute is longer for the two-failure case, since in Figure 4.9 we saw that the single failure maximum recovery time was less than 400 ms. However, the two failure experiment affected a larger number of channels; hence, a longer time to recover is acceptable. We experimented with different values of the retry interval, but the best performance, shown here, is still for retries at 100 ms.

Graph (iv) looks at the excess resource (ER) used. We can see that Global rerouting continues to perform well in this respect, but Hybrid and Local perform worse. The excess resource used decreases when adding retries for Hybrid and Local locus of rerouting. This is an artifact of the way in which retries are implemented; all retries are performed using Global locus of reroutes. Thus, in the presence of retries, the scheme tends to behave more like Global, and the efficiency of resource usage improves. This supports our earlier observation in favor of Global locus of reroute.

In general, the following conclusions are valid for both the single and double failure experiments:

- Immediate rerouting has the best time to recover.
- With Delayed retries the success ratio of Immediate retry can be brought up to match that of Sequential rerouting.
- By appropriately choosing the retry interval we can keep the time to reroute down to practical levels (average 200 ms, maximum 600 ms for two failures at high network load).
- Global locus of reroute is the most efficient in terms of resource utilization.

4.3.6. Sensitivity to load mix

Figure 4.13 shows the effect of a different load mix on the performance of some of the approaches to failure recovery. The load mix used to generate this graph is the "new mix" described in Section 4.2.6.1. The load levels used in this mix are shown on the x-axis of the graphs. They span a wider range of values than the ones used in the previous mix. The mix also uses different ratios for the different classes of traffic, and a different distribution of (source, destination) pairs.

The three bars shown at each load level in graph (i) are the success ratios for Immediate timing without retries (I), with Delayed retries(ID), and for Sequential timing(S). Adding Delayed retries to Immediate timing allows it to recover as much realtime load as

-- --

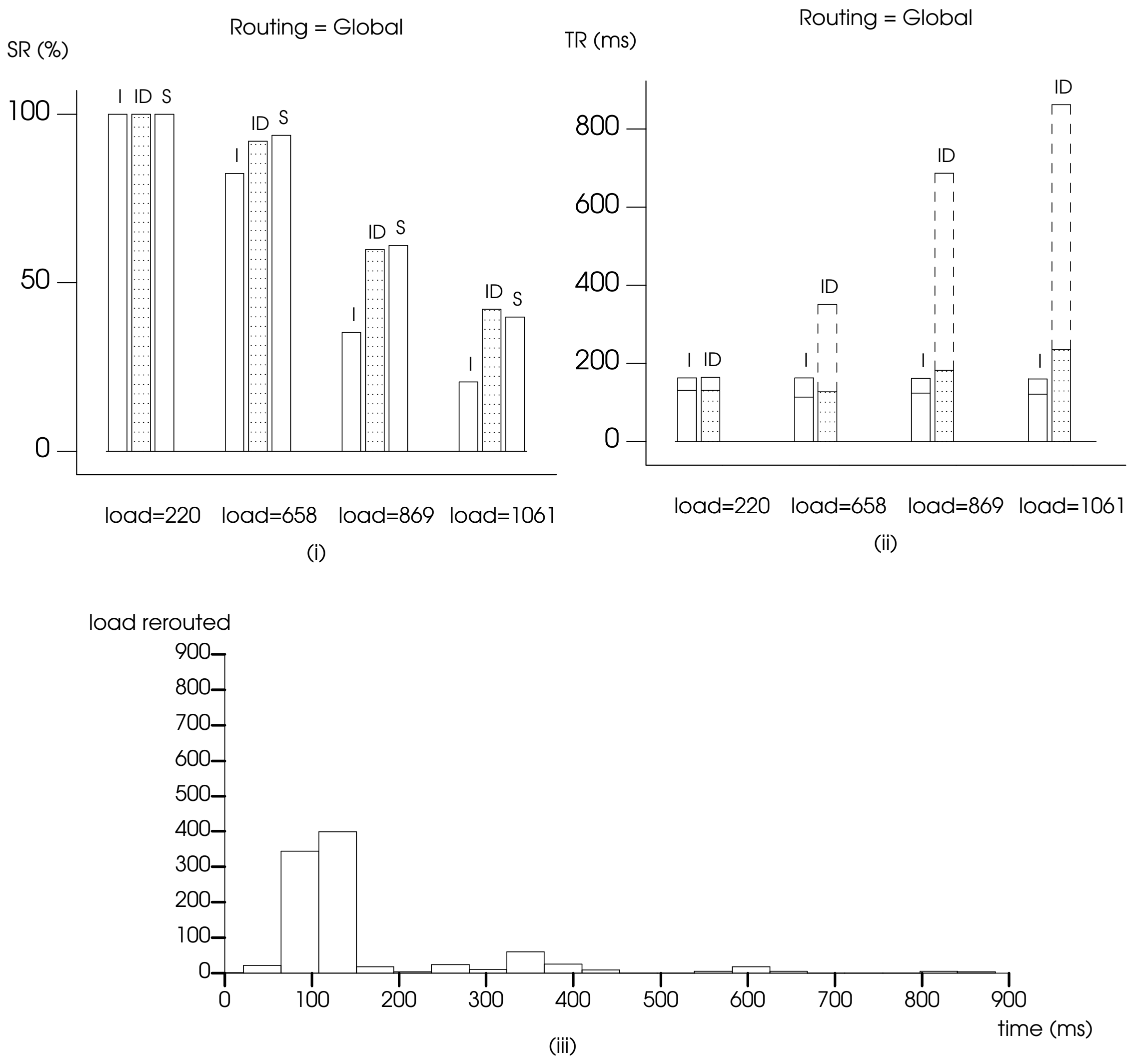

Figure 4.13: Effect of new load mix on
(i) Success ratio (ii) Time to reroute (iii) Histogram of time to reroute

the Sequential approach. At the same time, graph (ii) shows that the time to recover is not too long. The plain bar shows the time to recover for the Immediate timing scheme, while the dotted bar shows the effect of adding retries. The load at the highest level in this mix is more than in the last mix, so the recovery schemes are stressed more. This increases the maximum time to recover as compared to that shown in Figure 4.9 or Figure 4.12. The average is less affected. The histogram of time to recover for the Delayed retry scheme is shown in graph (iii). We can see that very little realtime load lies in the tail of the histogram; hence, most clients would see delays less than 400 ms. This is a desirable characteristic of the Delayed retry scheme. We can also easily trade the amount of real-time load rerouted for the maximum time to reroute, to accommodate tighter response requirements, by not generating retries beyond a certain time after the fault.

To summarize, the scheme with Delayed retries adapted well to the change in load mix. It continued to perform as well as the Sequential scheme in terms of the reroute success. The maximum time to reroute increased; this can be attributed to the increased load levels in the network. The change was not catastrophic, but graceful, since the average time to reroute did not shift much, and few channels were affected by the increased time to reroute. The efficiency of resource usage (not shown here) was also not changed significantly.

4.3.7. Sensitivity to topology

Figure 4.14 shows some of the results of the experiments we performed to verify the effect of changing the topology on the recovery schemes. In graph (i) and (ii) we look at the effect of adding some trunks to the mesh topology to yield the "mesh with trunks" topology shown in Figure 4.4, using the "original" load mix. Graph (i) shows the effect on the reroute success. Adding the trunks increases the capacity of the network, so that, with the same sets of channels, the network is less heavily loaded. This is seen in the lower values of the load index, shown on the x-axis of the graph. Thus, the Sequential timing scheme and the Immediate timing with Delayed retries can recover all of the affected realtime load in this case. Graph (ii) shows the time to recover for the Immediate scheme without retries (plain bars) and with Delayed retries (dotted bars).

We get similar results with the "core" topology (graphs (iii) and (iv)). With the given load mix and load levels, all of the affected realtime load is successfully rerouted by the Immediate timing scheme with retries within a reasonable time. However, the time to reroute can be further improved by changing the timing of the Delayed retry. The best timing comes for intervals close to 60 ms. We can reroute the same amount of realtime load in a shorter time, as shown in the histograms in Figure 4.15. Again, this is roughly the

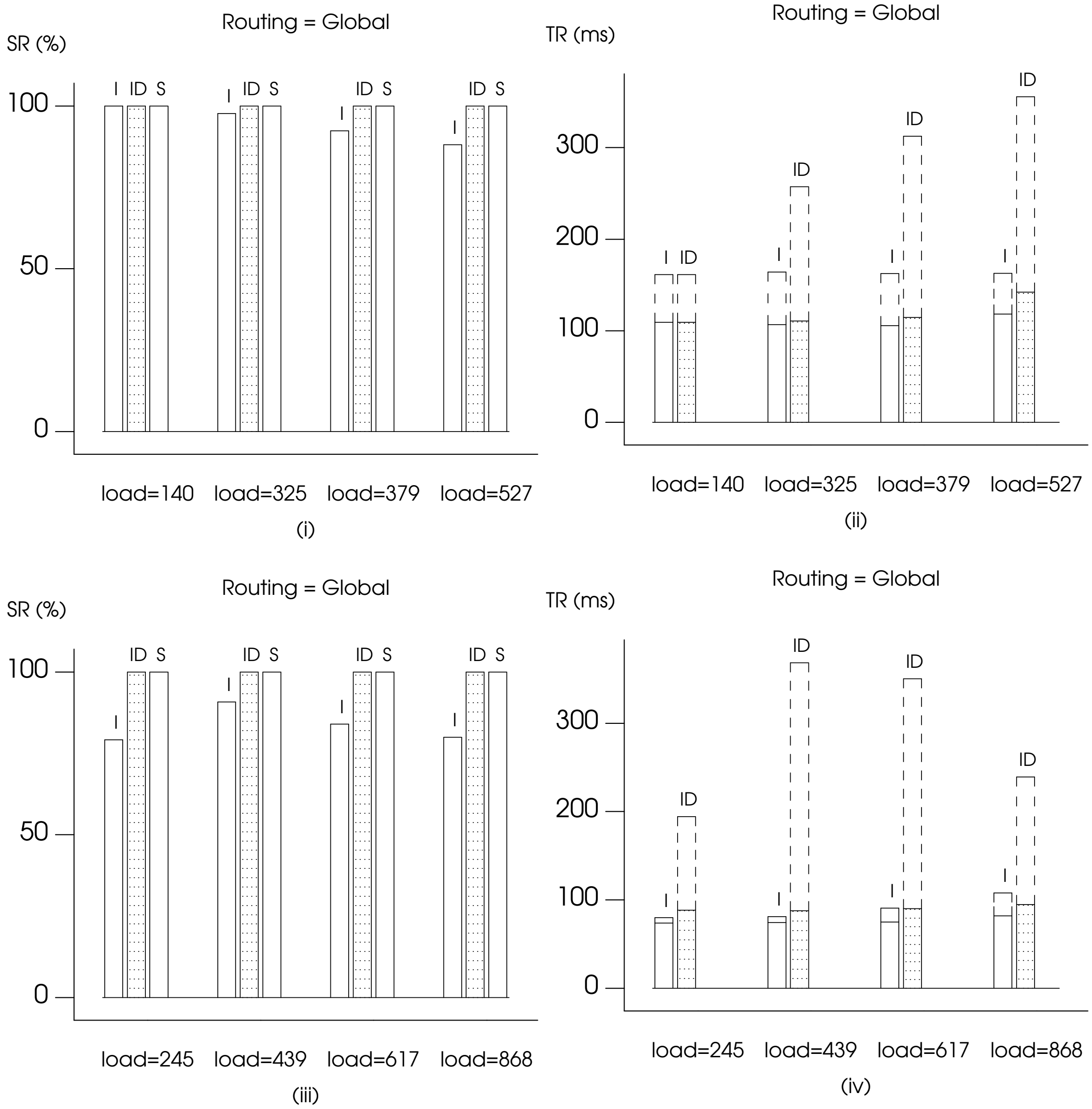

Figure 4.14: Effect of topology. (i)-(ii) Mesh with added trunks (iii)-(iv) Core topology

average round-trip time for this network, supporting our earlier hypothesis.

Thus, all the general observations made with the mesh topology continue to hold for the other topologies. The (Global routing, Immediate timing, Delayed retries, Local prediction) scheme continues to give good performance on all metrics of performance. It is quite insensitive to changes in topology, as long as the topology is well connected.

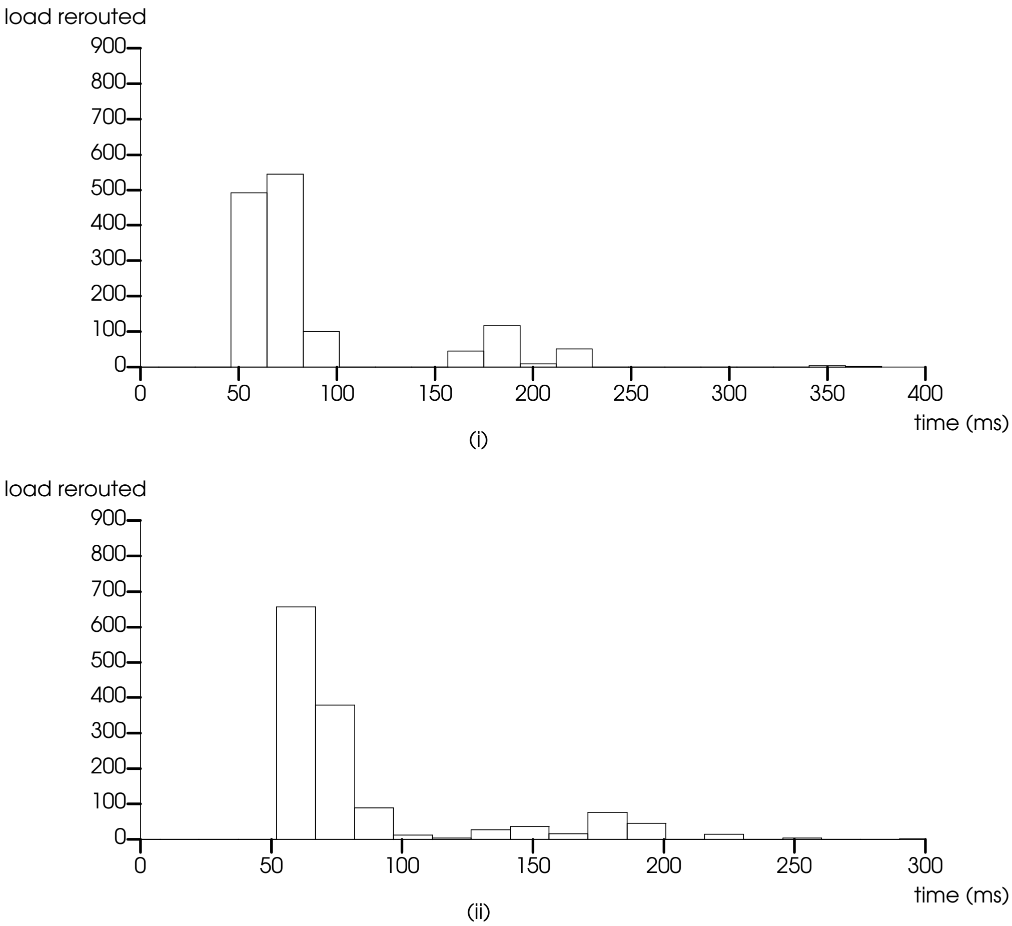

Figure 4.15: Effect of retry interval on core topology (i) 100 ms interval (ii) 60 ms interval

4.4. Conclusion

The main conclusion that we can draw from the results presented above is that schemes within the distributed, dynamic framework selected in the previous chapter can reroute a significant portion of the affected realtime load in the event of a link fault, within reasonable times and without wasting network resources excessively. The schemes are also reasonably tolerant of changes in load, traffic mix, number of faults, and network topology.

The experiments described in this chapter explored realtime channel recovery along the dimensions of reroute timing, locus of reroute constraints, retries, and state prediction. The effects of changes in these internal factors were evaluated on the three chosen metrics of recovery performance: amount of realtime load rerouted, time taken to reroute, and efficiency of resource usage.

Of the internal factors considered, the retry policy has the most profound effect on the first two metrics: fraction of realtime load rerouted and time to reroute. While Immediate retries did not provide a significant improvement in amount of realtime load rerouted, by adding a delay before initiating the retry and computing the route based on route update information that arrived in the interval, we can improve the success of the scheme to the point where it rivals the amount of realtime load rerouted by the Sequential timing scheme. As pointed out in Section 3.7.2, the Sequential timing scheme eliminates all collisions at the expense of longer times to reroute the channels. Thus, we cannot expect to reroute a significantly higher amount of realtime load using a greedy strategy, where each channel is rerouted independently of the others.

By experimenting with the interval between retries, we found an empirical value of 100 ms, at which the scheme performed best for the "mesh" topology. This is also roughly the average round-trip time given the topology and the distribution of (source, destination) pairs. This fits the theory that a significant number of unsuccessful establishment attempts occur due to collisions with other attempts during the forward pass. Thus, if the retry is timed to occur shortly after the other establishment completes *and* the information reaches back to the node where the route computation is being performed, it has the best chance of success and the shortest waiting time. However, the value of this interval will, in general, change with the network topology. A rule of thumb to help select this interval for an implementation is to set it to the average round-trip time. Since the performance is quite sensitive to this parameter, this should, perhaps, be evaluated more accurately in the context of a real implementation. In any case, the retry interval should be a tunable parameter of an implementation, to make it easy to make it adapt it to different topologies.

The locus of reroute policy has a significant impact on the reroute success and efficiency of resource usage. Local rerouting, which is similar to *link rerouting* used in telecommunication network recovery, performs poorly because it looks at a smaller set of routes while selecting one for the channel, and because the longer paths found by this scheme use more resources. This leads less of the affected realtime load being rerouted at high network loads. Also, as implemented in our simulation, with a complete round trip to balance resources along the path, Local rerouting did not lead to a reduction in the time to reroute. If we had devised a scheme more faithful to *link rerouting*, which does not require a complete round-trip establishment, we would definitely have gained in terms of time to reroute, but would have been able to reroute even less realtime load, because many of the channels would fail to meet their delay requirement along the longer path if we did not adjust this delay along the entire path as we did in our implementation. This seems to rule out Local as a method of constraining the route selection.

Hybrid performed almost as well as Global in terms of success of rerouting. All of the timing schemes were almost identical in terms of time to reroute, because of the round trip for establishment. But in terms of resource usage efficiency, Global was the best, and, since the performances of Global and Hybrid are similar in all other respects, Global appears to be the correct approach to use for this aspect of the design.

Adding Global prediction to a scheme without retries allows us to recover a significantly larger amount of realtime load. However, even more realtime load can be restored by adding Delayed retries, and in this case the timing of the entire system is dominated by that of the retry process. Thus, the gains of using Global prediction are diminished. In fact, since all the establishment attempts need to start at the same time for Global prediction to work well, this adds a small, but not negligible, amount to the time to reroute. In addition, the Global prediction algorithm takes more computation time to run. To extrapolate this to other topologies, we note that the synchronization and computation time for Global prediction will grow with the number of sources affected by the fault and with the size of the network; in a large network, this will be an even larger problem. Thus, we reach the conclusion that Global prediction does not seem to be useful as a mechanism to improve the reroute success.

The effect of reroute timing on the amount of realtime load rerouted was significant, but this gain came at the expense of increased time to reroute. The improvement in performance attained by adding retries exceeded what could achieved by randomized timing, with lower increase in the time to reroute. In fact, the initial ideas that we developed for this internal factor were transferred to the retry policy when we added delays to the retry scheme. In the presence of Delayed retries, Immediate timing seems to be the best approach.

Of the external factors considered, load seems to have the biggest effect on the metrics of performance. Increasing the load reduces the amount of realtime load that a recovery scheme can reroute. For the schemes using retries, the effect on the amount of realtime load can be partially offset by trading off the time to reroute. By using retries, the scheme takes more time, but manages to reroute more realtime load. However, there is a limit on the extent to which this trade-off can be exploited, since beyond a certain load level, which depends on the network's capacity and topology, the amount of realtime load that can be rerouted drops off, regardless of how long the scheme spends retrying. Also, there is a practical limit on how long a rerouting scheme should take, because the affected channels are not transmitting data during this interval, and the applications using the channels will eventually time out.

We observe that the schemes degrade gracefully with increasing load. However, in order to give satisfactory performance, the network should be run at a load such that, in the event of a failure, a large fraction of the channels can be rerouted. This is also consistent with another requirement: that, in the absence of network faults, the probability that a new request is accepted should be high. If the network is well dimensioned, both these requirements should be satisfied. The problem of network dimensioning is not dealt with in this thesis, but the maximum load at which a given network should be run, given a target percentage of affected realtime load which we would like to recover within a given time in the event of a fault, can be evaluated from our simulations.

The number of simultaneous faults being dealt with has some affect on the metrics of performance. This works in three ways: less of the network capacity and redundancy is available for rerouting; there are a larger number of channels to reroute, so the recovery schemes are stressed more; and the multiple sets of channels being rerouted are physically separated by some distance in the network, so the route update process exhibits different timing characteristics. The overall observed effect on the performance, in the case of double failures and the Delayed retry scheme, is that the recovery takes a little longer, but can still reroute as much realtime load as the Sequential timing scheme.

The last two external factors, the load mix and the topology, do not seem to have a significant effect on the recovery success of the schemes. The time to reroute, of course, depends on the topology, and would increase with the average path length of the channels to be rerouted. However, this is also true of the delay bounds that can be offered on the channels. This implies that in any network on which reasonably low delay bounds can be offered, our schemes would be able to reroute a significant portion of the affected traffic in a reasonable time. The excess resource usage did not change significantly with the topology. Thus, the schemes are relatively insensitive to the nature of the network topology. We designed the schemes to be scalable and flexible, without any target load mixes or topologies in mind. The insensitivity of the schemes to these factors validates our design choices. We must bear in mind, however, that the topologies tried in our experiments are all well-connected topologies, and that our results would not hold for poorly-connected topologies. This cannot be helped, since the basic idea on which the schemes are based is that of finding alternate paths between the source and destination; if none exists, the scheme will not work. However, our experiments seem to indicate that the results are not restricted to any specific well-connected topology.

The scheme with the best performance according to all our metrics of performance is the combination of Global locus of reroute, Immediate timing, Delayed retries, and Local prediction. The salient features of this scheme is summarized in Table 4.2. As with all presented in

Property	Evaluation
Overhead in the absence of faults	None
Amount of unrecoverable realtime load	As low as possible with greedy algorithms
Duration of disruption	Depends on topology
Guarantee of recovery	None
Overhead in the presence of faults	Small
Adaptability to external factors	Good
Complexity	Moderate

Table 4.2: Properties of the fault recovery scheme selected

this chapter, it has no overhead, in terms of resources required or channel management activity, in the absence of faults. It reroutes as much of the affected traffic as can be expected from any greedy algorithm based on the Bellman-Ford algorithm. Since the Bellman-Ford algorithm minimizes the resource usage of each individual route computation, we should no expect any better performance without discarding the greedy paradigm. The time to reroute, which determines the duration of the disruption, depends on the topology. In the topologies tested it was found to be acceptable (average 200 ms on the mesh topology). This value is related to the average round-trip time along the channel; this is because all the major components of the time to reroute, such as the time for an unsuccessful attempt, the delay before the next retry, and the successful attempt, are all related to the round-trip time. For most applications of realtime networks, the round-trip time has to be small to allow interactive communication without intolerable delays. Thus, the recovery time offered by this scheme may be acceptable on most realtime networks. However, this recovery is not guaranteed, since, under high network load, some channels may not be successfully rerouted.

In the presence of faults, the overhead of this scheme is small, since it uses Global locus of reroute, which minimizes the resource usage to the extent possible. It is also fairly adaptive to the external factors considered, such as topology, load, load mix, and

-- --

number of failures. Finally, the scheme is among the simplest of the schemes that we considered, from the point of view of implementation. This point is considered in more detail in the next chapter, which deals with the high-level design of a protocol embodying the ideas presented here and in the last chapter.

Chapter 5: High level design of a fault recovery protocol

5.1. Introduction

The last two chapters presented a large number of decisions regarding the design of mechanisms to perform fault recovery in a realtime network. In this chapter, we will summarize this information and place it into the context of a high level description of a fault recovery protocol, intended to work with the Tenet Suite enhanced with the DCM protocol (see Figure 2.2). This protocol will be called the Real Time Control Message Protocol (RTCMP), after the Internet Control Message Protocol (ICMP), the equivalent protocol in the Internet protocol stack.

The support for routing required by the fault recovery process may be implemented by modifying the existing routing software of the realtime network. A routing function is included in the software implementing the combined functionality of RCAP and DCM. We will use the term "RCAP+DCM" loosely to refer to either the combined protocols or to the software implementing them. Thus, the RTCMP implementation involves modifying the routing function of RCAP+DCM, and extending the control interface to allow RTCMP to use it appropriately. In this chapter, we will describe the interface and the functionality required from the routing element. The changes in software described will be relative to the routing software as currently implemented in RCAP+DCM. However, this model should also be applicable to a separate routing module implementation such as in the Tenet Suite 2 (36).

The rest of the fault recovery process will be implemented as a separate protocol, namely, RTCMP, which will run on all the nodes where the Tenet control protocols run. The RTCMP entity on a given node would exchange messages with neighboring RTCMP entities, the routing module, and the channel management process on the same machine. The other events which would trigger action from the RTCMP protocol are timer events and fault events generated by the network hardware. The interactions of the RTCMP protocol with the rest of the world is shown in Figure 5.1. Since this high level description is not

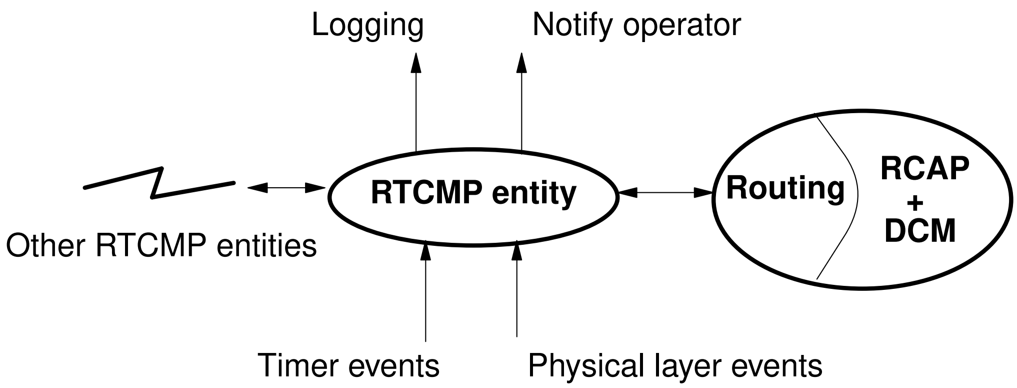

Figure 5.1: Interactions for RTCMP

targeted to a specific network or operating system, no implementation details will be provided.

5.2. Summary of design decisions

Starting from a completely general framework for realtime channel recovery in the beginning of Chapter 3, we narrowed down the possible choices, to the point where we can now describe the recovery process at a high level. In the previous two chapters, we made several design decisions, which we will summarize here.

The first set of decisions made were at a general level. We argued that a routing strategy that optimizes the network routes on a global level is too time consuming and unscalable, given the dynamics and scale of realtime networks. We chose a greedy strategy, in keeping with the strategy followed by the establishment protocol of the real-time network, where each channel would be routed in isolation, minimizing resource usage in this isolated context by using "min-hop" routing. In the context of this decision, the question of whether the fault recovery algorithm should be centralized or distributed became easier to answer, since distributed algorithms can implement greedy strategies equally well, and provide the benefits of scalability and robustness. We also chose to dynamically compute the routes from scratch for each failure, rather than pre-compute the configurations for all possible fault conditions, because of the dynamics of realtime channel establishment and teardown, and the space complexity associated with attempting to account for all possible configurations that a realtime network system could be in. We decided to further explore, in the course of our experiments, the question of the degree of information exchange during a distributed recovery.

Thereafter, we eliminated some of the variables in the experiments, such as the method used for failure detection, the route update process, the graph theoretical algorithm used to find the "min-hop" path, and the channel management process. We did this either by making simplifying assumptions in our simulator, or by assuming support from existing protocols. The remaining questions, open to investigation through simulation, were: the constraints to be used on the route selection process, the timing of the reroute and establishment attempts, the number and timing of retries, and the level of state prediction used to improve the recovery success. The issue of information exchange is implicit in the timing question and in the state prediction question, since the route update protocol exchanges information in the background continuously, so that, by waiting for the appropriate interval, the information needed is obtained without invoking an active request-response protocol.

The experiments conducted in Chapter 4 led us to the conclusion that Global locus of reroute, Immediate timing, Delayed retries, and Local prediction would provide a recovery success comparable to that of the schemes with maximum overhead for information exchange, but considerably faster. Methods that make decisions using the available information, rather than waiting for information to come in, seem to work best. The only case where waiting for information to come in seems to help is when an establishment request has failed, and the protocol is attempting a retry. The delay appropriate for the retry is close to the average round-trip latency of the network.

Fortunately, these choices are among the simplest that we looked at, which simplifies the task of designing a protocol based on them. For example, Local locus of rerouting is more complicated to implement that Global, since it involves a route computation model significantly different from the model described in Chapter 2. Global locus of reroute uses the same route computation model as RCAP+DCM, i.e., the route is computed at the source, and the shortest path which meets the delay constraint is used. Thus, the task of implementing this locus of reroute is the simplest, at least when the DCM protocol is present. Immediate timing is also the simplest to implement, since it does not require any timing cooperation between nodes. Similarly, implementing Global state prediction would have involved significant changes to the route computation software of RCAP+DCM. Instead, it turns out that Local prediction performs adequately. Local prediction does involve some minimal changes to the current RCAP+DCM routing software and its interface; these changes will be described in this chapter.

5.3. Description of the protocol

We will now provide a high level description of RTCMP based on the ideas developed in the first few chapters. The framework provided in Chapter 3 will be used throughout this chapter as the basis for our description. We will fill in the details, to the extent necessary for a high level design, but leaving out system-specific or implementation-specific information. This framework is based on the main tasks common to all fault recovery protocols. They are: detection, instigation, route computation, channel management, and return to normal. All of these tasks need not be supported on all the nodes, though an implementation might be easier to maintain if the same software runs on all the nodes of the network.

5.3.1. Detection

Detection must be supported on all the nodes, since we assume that any link in the network can fail. Thus, each node must monitor its neighboring links for failures. This would be more efficient with some support from the physical layer, for example, posting of events such as loss of transmission clock, physical layer protocol errors, and so on. The software may react to these events, either by handling interrupts posted by the hardware on the occurrence of such events, or by polling hardware status registers at intervals. Events such as retry timeouts of the reliable control message exchange protocol used by RCAP or RTCMP should also trigger detection. The specifics depend on the particular networking hardware and operating system structure, so we will not go into any further detail.

The support from the physical layer may be fast and efficient, but may not catch all faults. Sometimes, a computer may fail without affecting the physical layer's status of its outgoing links. To deal with all failure conditions, as well as to handle networks where the physical layer support may be inadequate, each node must ascertain the status of its neighbors by exchanging "ping" messages, whenever no data has been received on a link for some duration of time. Each node is responsible for ensuring the health of all its incoming links. Thus, if a link fails only in one direction, the node to which data stops arriving would send a ping message and after a few retries would detect the fault. This mechanism is slower than detection supported by the physical layer, and adds some control overhead to the network, but is general enough to be applicable to any computer network. It can detect any fail-stop failures that the physical layer does not detect. It is also general enough to support recovery from node failures, since the fail-stop mode failure of a node can be detected by the neighboring nodes by the ping method as the failures of all the links connecting to the node, even if the physical layer is not affected.

Finally, error rates are another mechanism to detect failures. If a link has very high error rates, so that the quality of the realtime data received from it is unacceptable, it would be better to declare it faulty for realtime purposes, and remove all realtime channels from it. This may be a temporary problem, so that, if the error rate improves after some time, we would return the link to realtime service. Such phenomena may be observed on links established over micro-wave transmission systems, where the error rate changes with the atmospheric conditions. The physical layer may maintain a count of physical layer protocol violations over small intervals, and the RTCMP software should poll this information, and maintain its own filtered counts as described below. If the data delivery protocols implement link level checksums, the errors detected by these checksums should also be counted. This error information may be used as described in (4), normalizing the error rate by the actual data transmission rate in the same interval, and using this metric to declare a failure when the normalized error rate crosses an upper threshold.

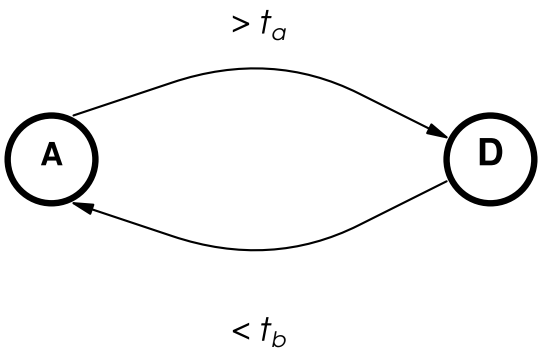

Figure 5.2: Hysteresis loop to avoid oscillations

To reduce the occurrence of oscillations of the link status between "alive" and "dead", with its associated instigation of fault recovery and return to normal actions, some low level validation of the faults detected is essential. A simple hysteresis method is presented in Figure 5.2. A link is classified as faulty if a permanent fault is detected by the other means described above, or if the number of transient errors detected at the physical layer in an interval I_a, expressed as a fraction of the actual data bits transferred, exceeds t_a. However, it is only returned to service if no permanent fault is present and the transient fault ratio measured over a larger interval I_b is less than a threshold t_b. I_a should be small, and chosen depending on the reaction time desired for detection. I_b should be relatively long to avoid oscillations and to ensure that the link is stable before returning it to service. It should be at least as long as the sum of the time to recover from a fault and the time to return to normal after a restoral, since in general we want to avoid declaring another fault on the same link before the return to normal procedure is complete. t_a should be chosen depending on the error rate tolerable for a typical application running on the realtime network. t_b should be at least an order of magnitude lower and may be chosen based on the expected error rates on the transmission medium.

When a fault is detected, the protocol should inform a human network operator to repair the fault, log the event for maintenance and tracing purposes, and also instigate automatic recovery.

5.3.2. Instigation

Instigation of the restoral process is performed by the node that detects the fault. This consists of reliably broadcasting a fault message, containing the link identifier of the failed link, to all nodes in the network so that the routing table in every node reflects the topology after the fault. We need the reliability to ensure that all the nodes receive the same information about the set of failed links. If the underlying network supports reliable broadcast, then this problem is solved. If not, the reliable broadcast is achieved by the RTCMP protocol using a constrained flooding method. The purpose of constrained flooding is to ensure that, as long as the network is not partitioned by a set of faults, the route update messages reach all the nodes in the network. We also want to ensure that messages do not traverse any link more than once. The arrival of the fault message at any node causes it to remove the failed link from the topological database.

For the following discussion of the constrained flooding method, we assume a lower layer transport protocol with checksums and retransmissions to take care of transmission errors. We also assume that clocks are synchronized using a time protocol such as NTP. Each fault message is tagged with a unique fault number, containing the unique identifier of the node and a time stamp. Fault messages have a maximum time to live (*MTTL*), which is a constant known to all the nodes, and which is larger than the maximum possible time to travel across the network's diameter. All RTCMP daemons keep a list of the fault numbers they have seen, and these numbers are not removed from this list until they are older than *MTTL*. Fault messages older than *MTTL* are never forwarded. A message with a fault number which is in the list is also not forwarded. Each RTCMP daemon forwards a fault message, if it is less than *MTTL* old[1] and the fault number is not in the list, to *all* its neighbors. This ensures that the message reaches all reachable nodes, and no link is traversed twice.

The description of the protocol in Chapter 3 also called for a separate reroute message to be sent to the source of each channel affected by the fault. This was needed to support the variety of options we were considering at that stage. However, if we look at the contents of the reroute message in the context of the decisions that have been taken, we find that it only contains the channel identifier of the channel to be rerouted. We can eliminate this message by including the entire set of channels affected by the failure in the fault message. This information must be obtained by querying the local channel management daemon, so the control interface of the channel management software must export a function which takes in a link identifier and returns the channel identifiers of the set of channels using the link. This set of channel identifiers is then included in the fault message broadcast to all the nodes.

5.3.3. Route selection

The route selection function needs to be implemented only on the host machines of the network. Each host node performs route computation for the affected channels originating at that node. To do this, it needs to identify the channels originating locally from the set of channel identifiers carried in the fault message. The RTCMP daemon should query the channel management process to find the set of channels currently originating from the node, and then find its intersection with the set of channels in the fault message. The recovery action described next is performed for this set of channels. Eliminating the reroute message saves any implementation complexity that could have been caused by the arrivals of the fault and reroute messages in different orders.

The process of route selection is performed at the source node, using the Global

[1] if (time stamp > current time - *MTTL*)

locus of reroute constraint, Immediate timing, and Local state prediction. The path selected is the minimum-hop path which has sufficient resources to support the channel and meets the delay constraints. This is computed using the Bellman-Ford algorithm. The information about resources and delay bounds on links is based on the information in the routing database at the time of the computation, subject to the modifications of Local prediction.

In the event of the first attempt failing, the reroute is retried after an interval, which is set to the expected round-trip latency of the network. The route computation during the retry would be based on the route update messages that came in during the delay interval, since the route update protocol would have continued to exchange messages and update the database in the interval. The retry interval should be a tunable parameter of the RTCMP implementation, since the performance of the scheme was seen to be very sensitive to the value of this parameter in our experiments. The retry process should be halted after a fixed interval from the time of the fault, based on the desired maximum response time for the recovery process. This may also be a tunable parameter.

Since the route computation is always performed at the source (the Local routing alternative was rejected due to its poor performance), we can either keep the route computation task in the software for RCAP+DCM or have a separate routing module, as long as the control interface supports the needed functionality.

The general functionality needed in the routing module and in its control interface is described below. Firstly, the control interface of the routing module should allow RTCMP to inform it of the failed links, so that they are removed from consideration during route computation. Secondly, RTCMP needs to specify a set of channels to be rerouted, rather than just one as in the interface provided to the clients. This makes it easier for the routing module to implement the third requirement, which is that the routing software must handle simultaneous route requests using the Local prediction algorithm.

In the specific context of RCAP+DCM, since all the parameters of the channels to be rerouted are already known, and they are not changed during the recovery process, the reroute request interface of the system is simplified. RTCMP just needs to pass to RCAP+DCM the set of channel identifiers. RCAP+DCM can look up the channel parameters from the state stored during the original establishment. This single request subsumes both the routing request and the establishment request, since the RCAP+DCM software performs both actions. We also make one request for all the channels that need to be rerouted at a time, so that the routing software may perform Local prediction on this set of channels. This call *must* be non-blocking, unlike the interface to the clients, since RTCMP might need to react to some other event while RCAP+DCM is performing the establishments. In order to have RCAP+DCM perform Local prediction, we would need to modify the routing software. RCAP+DCM already interacts with the route update protocol to modify the database on which routing in based.

Thus, we can simplify the software structure of RTCMP, as well as minimize the changes that need to be made to the existing realtime network control software, by appropriately extending the interface to the routing module, and implementing the necessary functionality there. The timing and retries are handled in the RTCMP software.

5.3.4. Channel management

The channel management task needs to be implemented on all the nodes of the network on which resources exist to be managed. This includes all the nodes in which RTIP is implemented, as well as any switches which can provide realtime data forwarding without RTIP, but need admission control and resource management support. However, the interface to the RTCMP protocol only needs to exist on the host machines of the network. In the context of RCAP+DCM, since the routing and channel management functions are implemented in the same software entity, the requests for route computation and channel management may be merged into a single call. Channel management, as implemented in DCM, involves a round trip for establishment along the new path, followed by a teardown pass along the old path. In case the attempt fails, RTCMP is asynchronously informed. This causes RTCMP to set a timer event, so that, after the appropriate delay, a retry is attempted. The number of retries is limited, as described in Chapter 4. No channel management action needs to be taken by RTCMP, apart from specifying the reroute requests and retries at appropriate times.

5.3.5. Return to normal

The return to normal task needs to be implemented on the host machines of the network. This action is performed on the detector node, adjacent to the failed link, which also detected the failure. A return to normal is detected when the hysteresis loop of Figure 5.2 restores the link to active status. Since the channels that were rerouted on the detection of the failure may be now on non-optimal paths, we need to reroute them to use shorter paths involving the restored link if necessary. When the link was detected to be in a failed state, the process was time-critical, because the service of the channels on the failed link were disrupted by the failure, and quick recovery was required. In contrast, the return to normal event is not time critical, since no channel is disrupted. If we use the DCM protocol to perform these reroutes, we get the added benefit that, even during the re-establishment process, none of the channels have their service affected. Thus, all that is necessary is to make sure that the rerouting load on RCAP+DCM is not excessive, since this might affect the service seen by other realtime users trying to set up new connections. We want all the channels to eventually move back to their optimal paths, to reduce the realtime load on the network.

On detecting a return to normal, the detector broadcasts the "return to normal" message, which contains the link identifier, to all nodes. The RTCMP daemon on every node contacts the routing module to restore the link into the topology and then triggers a reroute on every locally originating channel, which was rerouted during the fault recovery, or created during the period between the fault and the return to normal. This task is performed at low priority so that it does not affect other channel establishment requests handled by this node. The process looks at all eligible channels and tries to find one for which it can find a shorter route. The source routing algorithm is used without any additional constraint except the end-to-end delay bound. When such a channel is found, the process initiates a reroute onto the shorter path, and then waits for a random time interval with mean T, chosen appropriately to keep the expected channel management load on the network low. The randomization ensures that the retry processes from different nodes are not synchronized. The process is started at the time when the link is restored, and after a time interval of nT, where n is the number of affected channels, all channels would be back on their shortest paths. The process terminates when none of the remaining eligible channels can be rerouted to a shorter path. The DCM protocol ensures that the performance of any of the channels is not affected during the re-establishment process.

This idea can be extended to general load balancing by having this process run continuously at a very low frequency for all channels, rather than only for the channels defined as eligible above.

5.4. Changes to the RCAP+DCM software

The changes which must be made to the control interface of the routing module and the channel establishment software are summarized below.

- Inform routing function of failure (used on all nodes)
- Inform routing function of restoral (used on all nodes)
- Request reroute for set of channels (used on hosts)
- Query to find all channels traversing a link (used on the node detecting the failure)
- Query to find all channels originating from a node (used on hosts)
- Request establishment for channel on a specific route (used on hosts)

The first three features belong to the interface to the routing module, and the last three to the interface to the channel administration software. Since for RCAP+DCM both routing and channel administration are implemented in the same software entity, the third and sixth features may be combined into one call to the combined interface.

The main change needed to the routing software as implemented in RCAP+DCM is the addition of Local prediction. Since the update of the routing database by the route update protocol involves some latency, if the routing protocol is given a number of channels to route simultaneously (as is true for Immediate timing), then the database should be modified after each route is computed to remove the resources that would be reserved by the channel on its forward pass. This is the state that would be seen by the subsequent channels, if they happen to get routed on common links. However, this modified image of the network should be temporary, since the return-pass resource reservations will be lower, and the route update protocol will bring in the correct information, as each channel is established. Thus, in our simulation implementation, we used a temporary copy of the database, which we modified as we computed the set of routes, and finally discarded. The original copy of the database was only modified as the route update messages came in.

The interface to the routing algorithm must also be extended to take into account the fault tolerance schemes; these schemes and the extensions required for them are described in Chapter 6.

5.5. Conclusions

We presented a high level description of a protocol to support rerouting of realtime connections in the event of a fault. The protocol does not discriminate between channels, but attempts to recover as many as it can, within the desired maximum response time of the recovery process. For channels that cannot be rerouted within this time, the recovery protocol informs the user of the failure of the network recovery and tears the channel down. This should only happen when the network is operating close to the maximum realtime load that can be supported.

The recovery schemes designed fit well into the existing realtime network support protocols. The routing software and interface of RCAP+DCM must be modified to some extent, to allow links to be removed or restored by RTCMP, to implement Local prediction for multiple simultaneous route computations, and to support the fault tolerant channels that will be described in the next chapter. The channel management interface of RCAP+DCM must also be modified to support some query functions needed by RTCMP, and to combine the routing and establishment requests for multiple channels into a single call. However, the basic software model remains valid, and RTCMP can perform its tasks as a user of the RCAP+DCM interface.

Many questions have been left unanswered. For example, can multiple priorities for channel rerouting be supported in this paradigm? We can easily extend it to two levels: no recovery support for some channels, and support for other channels, by extending the channel performance specifications to include a flag to indicate this. The above actions would only be performed for channels with the recovery flag on. But to extend this to multiple levels of priority would involve non-trivial extensions.

Another interesting question is related to the interface of the Tenet Suite 2. How can this interface best be extended to provide support for fault recovery? This question too is beyond the scope of this dissertation.

There is also the problem of network security; that is, how do we prevent some unauthorized process from using the RCAP+DCM control interface to reroute channels that it should not be allowed to reroute? This, and other questions, can only answered in the context of a particular implementation, and a specific operating system; hence, we leave the questions open for future work.

Chapter 6: A proactive scheme for fault tolerance

6.1. Introduction

This chapter presents some possible schemes to provide fault tolerance on top of an underlying realtime service. The approach used is to make reservations on multiple paths, using the available capacity and connectivity of the network, to provide fault-tolerant realtime connections, which can offer a variety of grades of service in the presence of network faults. Forward Error Correction (FEC) techniques are used in conjunction with the multi-path reservation when completely transparent fault-tolerance is required. This approach is useful in a network where the transmission capacity is relatively abundant, and for applications where tolerance to failures and non-stop operation is essential. This approach would not make sense in a bandwidth-poor environment such as the current Internet, where depriving other users of bandwidth to increase the tolerance of one's connection would be regarded as anti-social behavior. However, with the gigabit per second networks envisioned in the future, bandwidth requirements may become less of an issue, and the improvement in the service provided may allow some applications to use a shared network, which would otherwise require a private network.

The ability to put many different classes of applications, including those that require some degree of fault tolerance, on the same network opens up the possibility of reducing the cost of the service by exploiting economies of scale. The target applications for this service include remote surgery, space missions, industrial control, and so on, where fault-tolerant realtime audio-visual communication is required. Some interactive multimedia applications may also benefit from fault tolerance, if the associated costs, in terms of increased resource requirement, are low enough.

The underlying realtime service allows for the reservation of buffers on a per-connection basis to eliminate loss due to buffer overflow. That still leaves two sources of packet loss in the network: transmission errors and network failures, which are not covered by the guarantees provided by the Tenet scheme. The primary objective of the mechanisms presented in this chapter are to handle network failures. However, tolerance to errors is provided are a side-benefit of using FEC to control loss due to faults.

An error can be looked at as a transient fault in the network. A long term fault, on the other hand, is one where the link does not transmit information for a long period of time, relative to packet transmission times. There is a continuum of error states in between these two extremes, and, for practical purposes, one may define a threshold of error rates, as described in the previous chapter, beyond which the link is removed from service. For simplicity, we will consider only the two extremes, and assume that either packets are lost with a constant (low) independent probability on all links, or that a link fails and does not transmit any information at all. We will discuss both the error-tolerant and fault-tolerant properties of the schemes.

The schemes for fault tolerance presented here are based on the idea of dispersity routing (44), described in Chapter 1. However, application to realtime networks requires a reconsideration of the capacity and performance requirements. Furthermore, the ability to continue to provide *realtime* service in the event of a fault must be built into the schemes, and validated through simulation. This chapter takes the idea of dispersity routing and modifies it in the context of realtime networks, to form several schemes which provide different levels of service, at different costs to the network. Some of the schemes provide completely transparent tolerance to restricted network fault scenarios (e.g., single or double faults) at the cost of increased bandwidth requirement. This is done by using multiple paths and FEC to introduce sufficient redundancy so that some of the paths can fail without disrupting the flow of data. The principle of realtime channels are used to provide guaranteed performance service on each of the paths, so that the overall service is also guaranteed.

Some other schemes offer gracefully degrading service as the number of faults in the network increase. They use the multiple paths through the network to provide a service whose capacity degrades gracefully, and use appropriate coding techniques to ensure that any of the surviving streams can be used independent of the others. Still other schemes provide a service, which is disrupted in the event of a failure, but the recovery is guaranteed and the recovery time is short. This is made possible by preserving the backup resources, but allowing non-realtime data to use this spare capacity until it is needed to recover from a fault.

A framework to classify the different schemes is developed in Section 6.3, which also describes the service offered by each scheme, and the expected impact on network capacity based on bandwidth requirements. In Section 6.4, we present the design of our experiments. Section 6.5 presents the results of the simulations, which validate the service offered, and provide a more accurate picture of the network capacity issues. We then discuss some practical issues in Section 6.6, such as the changes to the network interface required to support fault-tolerant realtime connections, and the interaction with the recovery schemes described in the previous part of the thesis. Section 6.7 concludes by summarizing the service provided and the network costs of the schemes, and comparing them to another existing approach to fault tolerance (74). We start with Section 6.2, which provides some background information to be used in the course of the chapter.

6.2. Background

This section summarizes some background information on compression algorithms and error-correcting codes. While the fault tolerance schemes to be developed in this chapter are applicable to any realtime data, video traffic requires some special consideration. The realtime paradigm is well suited to audio-visual information, because of the regular time-based nature of the information. Audio, having a lower bandwidth requirement, can be handled much more easily by the fault tolerance mechanisms. Video is harder, firstly, because the larger bandwidth forces us to seek solutions that use resources more efficiently, and secondly, because, due to the large bandwidth requirements and the inherent redundancy of video, it is usually transmitted compressed, and some of the compression techniques introduce dependencies in the video stream. This increases the fault and loss sensitivity of video . We devote Section 6.2.1 to a discussion of the compression techniques, with special consideration of how partial information may be decoded. We provide a brief description of intra-frame, inter-frame, block update and hierarchical compression ideas.

In the sub-section on error correction, we provide a description of diversity encoding and Priority Encoding Transmission (PET). These techniques can be used to increase the redundancy of a stream and to split the stream into multiple streams to enhance the error and fault tolerance properties. Readers familiar with the above can safely skip this section, since we will not go into extensive detail.

6.2.1. Compression techniques

Intra-frame compression

Intra-frame compression techniques take advantage of the spatial redundancy inherent in the digital representation of most pictures. The information required to describe one pixel is usually very similar to that required to describe the next one. The intra-frame compression technique proposed by the Joint Pictures Experts Group (JPEG) depends on dividing the screen into blocks of 8x8 pixels, subjecting each block to a Discrete Cosine Transform (DCT), quantization, and Huffman encoding. This process can typically compress the representation of the picture by a factor of 10:1; the level of compression can be traded off for picture quality, leading to a large range of possible compression ratios.

Pure intra-frame compression does not take advantage of the temporal redundancy inherent in video sequences, especially at the higher frame rates. However, each frame can be decoded independently of all others. Since each lost packet only affects one frame of the video sequence, compression codes like JPEG are relatively tolerant of packet loss. When a packet is lost, an unknown number of bytes of the current frame has been lost, and the decoder state is not synchronized with the bit stream contained in the next packet. Thus, most decoders discard the bits until the next marker, where the state can be synchronized and decoding restarted. Such markers are always present at the start of a new frame, but may occur more frequently, such as at the start of every new packet. However, many current JPEG compression/decompression hardware devices only arrange for resynchronization at the start of each frame. Since resynchronization can always be achieved by the start of the next frame boundary, no more than one frame is ever affected by a single packet loss.

Inter-frame compression

Exploiting the temporal redundancy of video sequences permits an additional improvement in compression ratio. Since the picture does not change much from one frame to the next, a frame can be expressed as the change from the last frame using a much smaller number of bits than required to independently code the entire frame. This idea is used in the coding standards proposed by the Motion Pictures Experts Group (MPEG), where frames can be of three types. I frames are coded using pure intra-frame compression, similar to JPEG compression. P frames are coded on a block basis; each block is coded either in predictive mode or intra-frame mode depending on the accuracy of the prediction based on the motion estimation. The intra-frame mode is the same as for the I frame blocks. In predictive mode, a block is represented as a set of motion vectors, interpolated relative to the closest preceding I frame, and prediction error information. I and P frames are also known as "anchor frames". B frames are interpolated based on the closest preceding and following anchor frames. The compression or decompression of B frames requires waiting for frames later in the sequence, leading to increased delays, buffer requirements, as well as compression complexity. In addition, the video service becomes even more susceptible to errors since there are more dependencies. Thus, interpolative coding (and, hence, use of B frames) may not be suitable for realtime applications (49).

Inter-frame compressed video suffers from dependencies between frames. Thus, the loss of an I frame affects the entire group of pictures (GOP) until the next I frame. Thus, if possible, the network should try harder not to lose an I frame than a P (or B) frame. Some ATM networks will provide differential cell dropping, so that, in case of buffer overflow, cells marked as lower priority will be dropped in preference to higher priority cells. However, the losses due to transmission errors and faults cannot be prioritized directly. In the next section, we will discuss a differential FEC scheme, which allows different levels of redundancy to be imparted to different portions of a video stream, to achieve a similar effect.

Asynchronous video coding

The basic idea behind asynchronous video schemes is to treat video as a series of updates to the screen, so that information can be displayed when the packet containing it arrives, independently of other packets in the same stream. For example, block update compression treats the screen as a matrix of blocks, and transmits a block only if it has changed (by more than a threshold) since the last time it was transmitted, thereby taking advantage of some of the temporal locality in video. Blocks may be compressed, using techniques such as DCT, to further reduce the bandwidth required. This technique is especially suited to video without much action, panning or scene changes, such as usually found in video conferencing, lectures, and so on. Each block is transmitted with the position information necessary to render it. If a packet is composed of randomly selected blocks, the loss of a packet has an effect distributed over the entire picture. This scheme is fairly tolerant of losses, especially for the classes of video applications just mentioned. The typical image is a human in front of a static background; the blocks being updated correspond to the human as she moves around. Thus, a block which is updated is likely to be updated again soon, as the human will move around in the same portion of the image. Even if an update is lost, it will be updated again soon afterwards.

The above technique has been used in a video conferencing tool called `nv`, with a fair degree of success. The compression can be performed in software in realtime, leading to applicability to a large class of hardware platforms. The major complaint with the technique is that people in the image tend to leave body parts scattered around the screen, when packets are lost. To partially compensate for packet loss, a background refresh process sends the entire picture, block by block, at a rate chosen depending on the overall bandwidth constraints. The technique can scale to the available bandwidth by changing the block size, the update rate, the threshold for transmitting a block, and the background refresh rate. The picture quality, as well as loss tolerance, increases with increasing the background refresh rate, increasing the update rate, decreasing the threshold for transmission, and decreasing the blocks size; this improvement comes at the cost of the increased bandwidth needed by the video stream.

Similar ideas are being investigated by at least two groups of researchers (45, 48). An additional twist to the basic idea is to generate multiple sub-streams by separating the regions of fast motion and slow motion, and sending more detail for the slow motion regions. The sub-stream for the fast motion regions contains more frequent updates, but less detail. This sub-stream is more important to the perceptual quality of the video. A third sub-stream may contain the refresh information. The idea of multiple sub-streams for compressed video is applicable to non-asynchronous video coding schemes as well, and is generally referred to as hierarchical compression.

Hierarchical Compression

Hierarchical compression consists of coding video sequences into a number of compressed sub-streams with the property that an image sequence can be reconstructed from many subsets of the set of sub-streams. If all the sub-streams are used, the highest-quality video sequence is obtained. However, if a lower bandwidth stream is required, some of the sub-streams may be dropped, and the decoding algorithm can still recover a lower-quality video, where the quality of the video is somehow related to the cumulative bit-rate of the received subset.

One possible scheme for hierarchical compression uses a stream of highly compressed video as the base low-quality signal. The other streams provide different levels of enhancement information, so that in combination with the base stream, various levels of higher quality video may be obtained. Such video compression schemes are also known as scalable or multi-resolution compression. They are useful for multicast, since, if the route to a particular receiver does not have enough resources to support the highest resolution, some fraction of the scalable stream may be dropped in the network, to allow communication at some reduced quality. They also allow variable quality video to be retrieved from a video server, depending on the bandwidth that can be supported by the server and the network at the time. In the context of ATM networks, the cells of the higher layers of a scalable video stream may be marked to be dropped first in the event of congestion. When congestion occurs, the video would degrade to the lower quality, since only the cells of the higher layers would be dropped.

MPEG-2 contains scalable modes which can offer such multi-resolution compression to a limited extent. Temporal scalability can be provided somewhat artificially by treating the I, P and B frames as separate streams of hierarchically encoded video. Another possible approach is to include more frames in the enhancement layer. Spatial scalability can be provided by improving the high resolution image by combining motion prediction, the base layer image, and prediction error information. SNR scalability is supported by using coarse quantization of the DCT coefficients for the lower layer, and sending additional residual information at the higher layer. The number of levels that can be provided, as well as the compression efficiency for the higher layers, is limited compared to true scalable compression techniques such as 3-D sub-band coding, where sub-band transforms are used in the temporal domain as well as in the spatial domain. Unfortunately, the delays incurred in the compression and decompression of true 3-D sub-band coded video are currently too expensive for interactive applications. Research into reducing these problems is in progress (62).

6.2.2. Error correcting codes

This section describes some forward error correction coding techniques. Diversity coding is used to recover from the loss of a known set of code words from a larger set of transmitted words. The Priority Encoding Transmission (PET) extends these coding ideas to multiple levels of redundancy for different layers of a hierarchical stream.

Diversity coding

Given a message, consisting of K equal sub-messages of equal length, can we create L additional code sub-messages, of the same length, such that the original message can be reconstructed from some fraction of the $K + L$ sub-messages? Forward error correction codes to solve this problem exist, and are easier to implement than standard error-correcting codes, because the decoding algorithm knows which contiguous portions of the encoded message have been lost. A simple example of a code which can recover the original message from any K sub-messages from a set of $K + 1$ sub-messages is the parity code. The original message is broken into K equal parts, and the $K + 1^{st}$ sub-message (of the same length as each of the K parts) is constructed as the bitwise parity of the K parts. If any one sub-message is lost, it can be reconstructed as the bitwise parity of the ones that were received. Codes which can recover the original message from any K of the received sub-messages are known as maximum distance separable codes. A large class of non-binary codes, the Reed-Solomon codes, achieve this limit (42). No code can recover from loss of more than L messages.

The parity code is simple enough to be implemented in software at realtime speeds. The non-binary codes can be implemented in inexpensive hardware at high speeds (5). They may also be implemented in software, and run at close to realtime speeds on current high end workstations. Experiments performed in the context of the PET project (described in the next section) at the International Computer Science Institute at

Berkeley show that a software implementation of the Reed-Solomon codes can run at rates of up to 1.2 Mbps on a SPARC 10 workstation with a 50 MHz processor (39). Recent work showed the feasibility of probabilistic decoding, with high probability for large *N*, using binary codes. Binary codes can be implemented in software at high speeds because bit operations such as XORs and shifts work very fast on general purpose CPUs. However, the restriction that *N* be large limits the applicability of the probabilistic techniques to our fault tolerance mechanisms. Work in progress in the PET project suggests that these ideas may be extended to deterministic binary maximum distance separable codes, which would have all the properties of Reed-Solomon codes, but run considerably faster. There is reason to believe that, with better coding techniques and faster general purpose CPUs becoming available every year, in the near future it may be possible to implement these codes at sufficiently high speeds in software.

Priority Encoding Transmission

The basic idea behind Priority Encoding Transmission (PET) is to provide different levels of redundancy to the different layers in a hierarchically encoded video stream. Various underlying FEC coding technique may be used to provide the redundancy. In (1), the coding is based on interpolation of polynomials. A faster implementation (39) is based on the Reed-Solomon codes described earlier, and work is in progress to speed up the implementation using the probabilistic and deterministic coding techniques described in the previous section. By appropriately encoding the different layers of the hierarchically encoded video with different levels of priority, we can create a combined stream with some overhead with respect to the original stream, depending on the levels of redundancy added. The stream has built-in redundancy, so it may be transmitted on a network which does not provide any further loss protection. On arrival at the destination, depending on the number of packets lost, we can extract some of the higher priority streams. The number of streams which can be extracted depends on the degree of redundancy provided to each of the streams and the number of lost packets. Thus, we can trade-off the bandwidth of the resulting stream against the level of tolerance.

To give a concrete example, consider an MPEG stream that consists of only I and P frames, and header information.[1] The size of the header stream can be neglected compared to the size of the other two streams. Let us assume an average I frame of 25 Kbytes and an average P frame of 5 Kbytes. Assume that the sequence of frames composing the group of pictures (GOP) are IPPPPPPPPP, which is repeated three times a second, giving us a 30 frames/second video stream. Thus, the I stream is 75 Kbytes/s, and the P stream is 135 Kbytes/second, for a total average stream rate of 205 Kbytes/second.

We may choose to send the header information with 3 times redundancy, the I stream with 1.5 times redundancy, and the P stream without any redundancy. The overall capacity requirement of the stream after PET encoding is 1.17 times the original. Any one third of the packets over which the information of an individual I frame is distributed may be lost without affecting the correct and complete reception of that I frame. However, the loss of any packet would affect the corresponding P frame. The header has the highest level of redundancy, since it is needed to decode any stream at all, but it occupies a very small fraction of the bit stream. The PET scheme also includes appropriate packetization techniques that spread out the original and redundant information among the packets, so that really *any* one third of the packets corresponding to the GOP can be lost without affecting the reconstruction of the I frame.

6.3. Schemes for fault tolerance

In this section, we will attempt to set up a framework for our experiments by classifying fault tolerance schemes along four dimensions. Then we will describe the individual schemes in some detail, and show how the application of various combinations of the levels of these four factors give us schemes with different properties and different capacity requirements.

[1] The header contains information such as the color map and is necessary to use any of the other information.

6.3.1. Classification

The schemes for fault tolerance dealt with in this chapter can be characterized by three variables: dispersity, redundancy, and disjointness. Finally, the schemes with redundancy may be hot or cold. These ideas are explained in this sub-section.

Dispersity

Dispersity is the idea of sending the information across a number of paths in the network. This idea was first proposed by Maxemchuk in (44), as mentioned in Chapter 1. Each message to be transmitted on a dispersity system with N paths is fragmented into N sub-messages. Each of the sub-messages are transmitted on one of these paths. At the destination, the original message is reassembled. The advantages of spreading the information out on N paths are:

- The transmission time is reduced to approximately $1/N^{th}$ of its single-path value.
- The load on any one specific path is smaller.
- The effect of bursts is spread out over the network.
- In the event of a network failure, the transmission capacity of the aggregate system is only partially affected.

Note that this system is not transparently fault-tolerant. It merely reduces the effect of the failure on the client.

In the context of a realtime network, the different paths of a dispersity system are subject to realtime constraints. These constraints can be met by using realtime channels as the component paths of the dispersity system. In other words, after choosing the set of paths which form the dispersity system, the realtime network performs channel establishment on each of the paths. We will use the term *sub-channel* to refer to one of the realtime channels of a dispersity system. One of the constraints the system should satisfy for efficient operation is that the paths should have roughly equal delays bounds. This should be a criterion in the path selection process, as well as in the choice of the end-to-end delay requirements specified in the establishment process for the sub-channels.

If the delays are not equal, buffers will be required at the destination to equalize delays, so as to deliver the packets in order. Buffers will also be required to smooth out network jitter. If the realtime network provides jitter control, the buffer requirements at the destination can be reduced. We will work out these requirements in the next section. An incidental benefit of this process of delay equalization is that the network jitter is removed from the information stream.

Since we use realtime channels as the component paths of the dispersity system, the delivery of messages at the destination is bounded by the performance guarantees in the absence of faults. Even in the presence of faults, the performance of the surviving sub-channels are still guaranteed. We will show how we can use this to provide guaranteed service to the application using the dispersity system.

The number of paths chosen, N, is one of the variables that characterize the system.

Redundancy

Redundancy is the idea of sending more information than the message, in order to be able to reconstruct the message in the event of loss in the network. It may be used independently of the idea of dispersity, by adding FEC to a single realtime channel. In conjunction with dispersity routing, it can be used by sending the redundant information along a separate path. For example, of the N paths, only K may carry the message stream. Thus, for a given message, the system would break it into K equal sub-messages and transmit them on the K paths. The rest of the paths may be used to transmit redundant information, which would be used to reconstruct the original message in the event of loss of some of the K original pieces of the message. A simple redundant system can be designed with $K = N - 1$. In this case the single redundancy sub-channel carries a bit-wise parity calculated over the $N - 1$ pieces of the message. If any $N - 1$ of the total sub-messages arrives at the destination, the message can be reconstructed. Error correction codes which work for arbitrary N and K exist. In the case of maximum distance separable codes, if any $N - K$ of the messages are received, the message can be recovered.

The variable K in relation to N defines the degree of redundancy. $K = 1$ is corresponds to duplicating the same information on N channel, uses N times the bandwidth of a non-fault-tolerant realtime channel, and has the largest fault tolerance. $K = N$ corresponds to a dispersity system without any redundancy.

A dispersity system with redundancy requires N/K times the bandwidth of a non-fault-tolerant realtime channel with the same traffic and performance requirements. When redundancy is used in combination with dispersity, we get the following additional advantages as compared to a dispersity system without redundancy:

- The system is error tolerant. A certain number of the pieces of the message can be corrupted or lost without affecting the decoding of the message. The number depends on N, K, and the error-correcting code. It can be no larger than $N - K$.
- The system is transparently fault-tolerant. A certain fraction of the paths can be affected by failure, without interrupting the flow of the information. Again, the specifics of the error-correcting code determine the number of failures that can be tolerated.

Moreover, since the service of the underlying realtime channels is realtime, we obtain performance bounds on the service provided by the dispersity system. The application sees fault tolerant realtime service, with guarantees on packet-delivery that continue to hold in the presence of restricted faults. The restriction on the types of faults covered depend on the level of redundancy of the system, and the nature of the FEC code used. Examples of types of faults that may be covered include single faults, double faults, and so on.

Disjointness

In general, the paths used in the dispersity systems should be disjoint. For these systems to work, the routing algorithm in the network must be able to recognize channels belonging to a dispersity system, and place them on disjoint paths. If the paths are not disjoint, then many of the advantages of dispersity systems are lost. In particular, the failures of the paths are no longer independent, since, if a shared link fails, two or more paths can simultaneously stop transmitting data. However, disjointness is a very harsh restriction, especially as N approaches the edge connectivity of the network topology. By allowing some links to be shared, we might be able to set up many more connections in the network. We would like to answer the question: can any of the advantages of dispersity routing still be provided after relaxing the disjointness criterion?

If the constraint is completely relaxed, then it is possible for a link to be shared by all the paths in a dispersity system, leading to the undesirable characteristic that if that link fails, the capacity of the system is reduced to zero. Therefore, we must put in some constraint, which, while looser than the strict disjointness constraint, should still allow the dispersity system to have provably good tolerance characteristics. If we allow a link to be used by at most two channel routes, then we know that a single failure will not affect more than two of the paths. Then, by imposing the restriction $N - K \geq 2$, the system can continue to be tolerant to single faults, assuming maximum distance separable codes. Of course, this system would require more network bandwidth than the $N - K = 1$ system with the same capacity. We look at the costs and benefits of such methods below. In terms of our characterization, we can define a variable S, which places a limit on how many paths can share a link.

The routing algorithm in the network must take the variable S into account while routing channels which belong to a dispersity system. Of course, even if the system allows links to be shared (i.e., $S > 1$), the routing algorithm should try to find paths which do not share links, and only use paths with shared links if no paths meeting the delay constraint exist, which do not share links.

Thus, a dispersity system can be characterized by the triple (N, K, S). This triple also tells us how fault-tolerant the system can possibly be. If we use a maximum distance separable code, the system can tolerate up to $\left\lfloor \frac{N-K}{S} \right\rfloor$ faults in the network transparently. For example, the (5,3,2) system can tolerate one fault with a maximum distance separable error-correcting code.

Hot vs. cold standby

In addition to the above three variables, a dispersity system with redundancy ($N > K$) can be run in a hot or cold standby mode. In case of hot standby mode, the extra sub-channels actually carry FEC information. In the event of packet loss or failure, the destination can recover without any exchange of messages with the source. In the case of cold standby, the extra sub-channels are not used, except in the event of a failure. This extra capacity can be used to transmit non-realtime traffic during normal usage, with the understanding that, in the event of failure, the capacity will be appropriated for use by the fault-tolerant realtime connection. When the failure occurs, we incur an extra delay before recovery due to the need to inform the source of the failure, so that it can shift the transmission to the back-up sub-channels. However, the capacity is guaranteed to be there, unlike what happens in the reactive fault recovery mechanisms we described in the previous part of the thesis.

6.3.2. Description of schemes

In this section, we discuss in further detail some of the interesting regions in the 4-dimensional space described by the above framework. The gains in service provided by the dispersity systems falling into each region, as well as the attendant costs in terms of increased bandwidth, delay, or implementation complexity, are discussed. We show how to compute the traffic and performance specifications of the sub-channels, as well as bounds on the buffer requirements at the destination. We also mention any additional support, from coding techniques, or from the network that may be required to make each scheme work.

Dispersity routing without redundancy - $(N, N, 1)$ systems.

Schemes within this category spread the data packets out into the network using realtime channels on disjoint paths. If N paths are used, the bandwidth reserved on each path is $1/N^{th}$ of the total required capacity. In the case of network failure, some of the paths survive and continue to carry data while the others are being rerouted. Thus, some lower bandwidth communication is maintained under fault conditions. The system has no overhead in terms of extra bandwidth requirement. However, it suffers from a degradation in service, which can be partially characterized by the loss of $1/N^{th}$ of the bandwidth in the event of a single failure. In the case of video transmission, depending on the compression and/or error-correcting technique being used, the destination may adapt more or less gracefully to this loss instantaneously. In all cases, when the sender is informed, the system can be re-arranged to get the full benefit of the remaining capacity. We will now look at some of the compression techniques, and analyze how they would adapt instantaneously to the failure, even before the sender re-arranges its operation.

The system would work well with video compression codes in which a packet transmitted on a given path is independent of packets on the other paths. Compression codes without inter-frame compression, such as JPEG, fit this model. As long as we make all packets corresponding to a single JPEG frame follow the same path, we have each stream independent of all other streams. In this case, failure of any path would result in a reduction of the frame-rate of the video stream seen at the destination. There would also be a small increase in the irregularity of the displayed stream, since we would see $N - 1$ frames displayed smoothly, and then a discontinuity. This could be fixed as soon as the source is made aware of the fault, by adjusting the frame-rate to use the available capacity.

This idea would also work reasonably well with asynchronous video coding schemes, such as the block update scheme described in Section 6.2.1. The following considerations must be kept in mind. The number of paths N should be large, since the fraction of traffic lost in the event of a fault is $1/N$ and the block update scheme is loss tolerant only up to some limit. The source should be notified by the network, and should adjust to using only the surviving links until the fault recovery process restores the failed link. This could be done by scaling back the block compression algorithm to a lower bandwidth as described in Section 6.2.1, and transmitting the information only on the surviving sub-channels. The picture quality after scaling back would be better than the picture quality while $1/N^{th}$ of the packets are being lost.

Compression codes that use hierarchical encoding could also work with $(N, N, 1)$ dispersity systems, in conjunction with Priority Encoding Transmission (PET). By appropriately selecting the levels of redundancy on the various layers in the hierarchical compression scheme, we can create a coded stream which would degrade by layers as the number of faults in the network increases. The example worked out in Section 6.2.2, if split appropriately into 3 streams, and transmitted on a $(3, 3, 1)$ system, would provide both I and P streams in the normal case, and only the I stream in the event of a failure. Note that the degradation is present only during the time it takes the network to recover the failed sub-channel. The overhead in this example is 17%, as worked out in Section 6.2.2.

Audio data, having much lower bandwidth requirements, may be handled using the systems with redundancy, to be described below. Other realtime that can be partitioned into multiple streams in such a way that any of the streams can be dropped without affecting the utility of the other streams can enjoy the graceful degradation characteristics of the dispersity systems without redundancy. The graceful degradation is manifested as a loss of bandwidth when a failure occurs. The guarantees on the other performance bounds provided by the realtime network, such as delay, jitter, and loss due to buffer overflow, continue to be valid.

Realtime data that cannot be partitioned into multiple streams as described above can be carried on such systems with a temporary disruption during failure, notification of the source and destination, and scaling back and adjustment at the source to use the surviving paths. However, since the round-trip delay of the channel has to be small in order to be suitable for interactive use, the latency of this notification process will also be small, and may be acceptable for many applications. In addition, the notification time can be bounded, at least statistically, by reserving resources for the control activity.

Since the coding step is absent for these systems, we do not need to fragment each message across the paths. Rather, each message may be sent on a single path, and the different messages may be sent round-robin on the paths in the system. This saves us the fragmentation overhead. Thus, messages are sent on the sub-channels at a rate $1/K^{th}$ the total rate, but the message size remains the same. No delay is added due to fragmentation or encoding. Thus:

$$\begin{gathered} X'_{min} = X_{min} \cdot K \\ X'_{ave} = X_{ave} \cdot K \\ I' = I \\ S'_{max} = S_{max} \\ {D_{max}}' = D_{max} \end{gathered} \tag{6.1}$$

The number of buffers needed at the destination to equalize the delays between paths, absorb jitter, and reorder the packets can be computed using the following formula:

$$B = \left\lfloor \frac{{D_{max}}' - \min\limits_{j<N}(Dp_j)}{X'_{min}} + 1 \right\rfloor, \tag{6.2}$$

where ${D_{max}}'$ is the delay bound on the sub-channels in the dispersity system, as calculated in Equation 6.1, Dp_j is the minimum delay (transmission + propagation) on the path for sub-channel j, and X'_{min} is the minimum inter-packet time on the sub-channels. B is an upper bound on the buffers required. We will compare the buffers actually used in the simulation experiments to this bound. The buffer space for reassembly should be large enough to hold B packets from *each* sub-channel in the dispersity system. The memory required is, thus, $N \cdot B \cdot S'_{max}$.

The buffer space requirement at the receiver could be reduced in a network that implements jitter control by taking advantage of the fact that no packet can arrive before $Dmax' - Jmax'$ at the destination. Thus, the number of buffers required is reduced to:

$$B = \left\lfloor \frac{J_{max}{}'}{X'_{min}} + 1 \right\rfloor \tag{6.3}$$

Our simulation did not implement jitter control, but a description of how jitter control may be provided in a packet switched network may be found in (31).

Dispersity routing with redundancy - $(N, K, 1)$ systems with $K < N$

Dispersity systems with redundancy operate by breaking each message into K packets of equal size. Then, they compute $N - K$ redundant packets using an appropriate coding technique, and transmit the N resultant packets onto N realtime channels. At the destination, the original message can be reconstructed from any K packets, if the coding technique used in maximum distance separable.

Schemes within this category use N/K times the network bandwidth, compared to the bandwidth required to support a non-fault-tolerant realtime channel with the same specifications. These schemes use the FEC information on the $N - K$ extra paths to recover instantaneously and transparently from up to $N - K$ faults. The bandwidth overhead of such a system is, thus, $\frac{N-K}{K}$. For example, a one megabit per second video stream carried on a $(3, 2, 1)$ system would require three sub-channels each with half megabit per second capacity.

Note that we can build systems with fault-tolerance identical to a PET stream being carried on an $(N, N, 1)$ system as a combination of different $(N, K, 1)$ systems. The example provided for PET in Section 6.2.2 is equivalent to a combination of a $(3, 1, 1)$ system for the header stream, a (3,2,1) system for the I stream, and a (3,3,1) system for the P stream, using the same set of three paths. While the details of encoding are different, the amount of fault and error tolerance provided is the same.

The systems for $K = 1$ and $K = N - 1$ are easier to implement, since $K = 1$ corresponds to sending the same data on all N sub-channels, and $K = N - 1$ corresponds to the parity system. The other systems would need some hardware support, such as described in (5), or fast CPUs to perform the coding and decoding in software. New coding techniques are under investigation (39), which might solve this problem. The systems with $K = N - 1$, which we will call the parity code systems, are currently the most practicable systems, given CPU speeds and the complexity of the algorithms for coding and decoding maximum distance separable codes.

Dispersity schemes with redundancy provide completely transparent fault-tolerant service to the application. For fault conditions that are within the recovery ability of the error-correcting code (less than $N - K$ faults, assuming maximum distance separable codes), no degradation of service is seen on. Since we use realtime channels as the component paths of the dispersity system, all messages are delivered according to the performance bounds guaranteed to the application, even after a failure has occurred. Thus, the service provided is fault-tolerant *realtime* service, within the strict Tenet definition of realtime; that is, the guarantees provided are mathematically provable. The ability to recover transparently from the loss or corruption due to transmission errors of $N - K$ packets from the same message is also provided as a side benefit.

The parameters to be used for establishing the sub-channels of the dispersity system with redundancy can be calculated from the parameters of the request as follows. Since each message on the entire system causes a packet to be transmitted on each sub-channel, X_{min}, X_{ave}, and I are the same as in the original request. The packet size is $1/K^{th}$ of the message size transmitted on the system. The delay for fragmenting and encoding the packets must be accounted for while setting the delay bound for the sub-channels.

For some systems, such as the $(N, 1, 1)$ systems, d_{code} and d_{frag} may be zero or negligible:

$$
\begin{aligned}
X'_{min} &= X_{min} \\
X'_{ave} &= X_{ave} \\
I' &= I \\
S'_{max} &= \frac{S_{max}}{K} \\
D_{max}' &= D_{max} - d_{code} - d_{frag}
\end{aligned}
\tag{6.4}
$$

At the destination, the time to decode the packets must be added to the end-to-end delay, before the decision about acceptance for the entire system can be made. The buffer requirements at the destination for these systems are the same as described in Equations 6.2 and 6.3.

Dispersity routing with some shared links - $(N, K, 2)$ systems

In case a completely disjoint set of paths cannot be found, the scheme can still continue to work using some partially disjoint paths. Relaxing the constraint thus allows us to set up a larger number of connections in the same network. The increase in capacity provided by relaxing this constraint will be investigated in the simulations. We will only look at systems with $S = 1$ (links may only lie on one path) and $S = 2$ (each link may only be shared by two paths). The routing algorithm is constructed to use shared links only when no other path can be found.

The service offered by such systems in the absence of faults is very close to that offered by the $(N, K, 1)$ systems. If we assume that the loss rates due to transmission errors on on all links are identical, and time-independent, the probability of a loss which is unrecoverable is the same for both systems. However, on real links, bit errors are auto-correlated, and thus the probability of loss due to a transmission error seen by two packets which cross the same link closely together in time are not completely independent. Therefore, the service seen by the $(N, K, 2)$ system may be slightly worse than the service seen by the $(N, K, 1)$ system. This effect would decrease with the packet size, since the auto-correlation would not be observed at the larger time scale of the packet transmission time, and hence, the probability of loss for two different packets would be almost independent.

In the event of the failure of an unshared link, the effect on the two systems is the same. If the link which fails is shared, then this results in a failure of two sub-channels in the $(N, K, 2)$ system. There are three possible approaches to offering a meaningful service in this context.

Firstly, if the system under consideration is a non-redundant dispersity system ($N = K$), the loss in capacity of the connection in the event of a single failure will be either $1/N$ or $2/N$, depending on whether a shared or an unshared link failed. This may be an acceptable service for large N, especially since we are operating in a heavily loaded network, where the better $S = 1$ connection could not be established due to existing reservations. The service after a fault is still realtime, since the performance guarantees are still valid; however, the loss in bandwidth for an $S = 2$ system is larger than that for the $S = 1$ system if a shared link fails.

Secondly, in case the user asks for an $(N, N-1, 1)$ connection, but the routing algorithm returns an $(N, N-1, 2)$ connection, we know this can only happen because an $(N, N-1, 1)$ connection could not be created. At this point, the user may be informed of the paths found. From a knowledge of the number of links that are shared between two paths, and the probability of failure of each link, the user (or the network) can compute the probabilities of an uncovered failure (failure of a link which is shared between two paths) and of a covered failure (failure that affects a link that is not shared). The user can then decide if he wants to accept this limited fault-tolerance, perhaps for a lower cost than for full tolerance. In this case, the fault-tolerance provided cannot be guaranteed, since only some of the failures are covered.

Finally, a third way is to use extra redundancy in the coding, i.e., use an FEC code that can survive the failure of two sub-channels. If the original request is for a $(4, 3, 1)$ system and four disjoint paths cannot be found, the user might also be satisfied with a

(5,3,2) system or a (4,2,2) system, as long as the total data rate is the same, since the fault and transmission error tolerance characteristics of both systems are at least as good as that of the (4,3,1) system. The system is transparently tolerant to all single failures (and can even survive double failures, unless the shared link fails). Realtime guarantees continue to be met without disruption after the fault. The questions that remain to be answered are:

- In a heavily loaded network where an $(N, N-1, 1)$ system cannot be routed, is it likely that we find paths for an $(N, N-2, 2)$ system, given such as system requires more network bandwidth than an $(N, N-1, 1)$ system?
- How much of a detrimental effect would such a trade-off have on the capacity of the network to support realtime connections?

Hot standby systems

In the most general interpretation of the term, all the schemes described so far for dispersity systems with redundancy are hot standby systems. This means that the capacity of the extra sub-channels is used to transmit redundant information, which can be relied on to recover the data in the event that some of the sub-channels fail. However, the common usage of the term "hot standby" is for systems where no encoding or decoding is required for the standby. The most common example of a hot standby system is that provided by a (2,1,1) system, where the source transmits the same data down two disjoint sub-channels. The destination uses the data from either channel, whichever comes in first. This allows completely transparent tolerance, to both losses and faults, without the need for notification of either source or destination. There is no degradation of the realtime performance of the system for single faults. In addition, the system needs only two disjoint paths in the network, which is a much easier constraint to satisfy than N disjoint paths for large N. Further, there is no overhead for special fragmentation and encoding at the source, or reassembly and decoding at the destination. No extra buffers are needed at the destination to equalize delays. On the negative side, the system uses twice the network capacity of the non-fault-tolerant realtime channel. This idea could be extended to $(N, 1, 1)$ systems, for tolerance to $N-1$ simultaneous faults, but with increased capacity overhead.

Cold standby systems

For cold standby operation, an $(N, K, 1)$ system is operated with K sub-channels in use during normal operation. The capacity of the other $N-K$ sub-channels is free for best effort data traffic. In the case of a fault the realtime dispersity system switches the data which was being carried by the affected sub-channel to one of the alternate channels. This involves a latency for informing the source and for the source to switch channels. However, no channel setup is required, since the resources are reserved at the time the $(N, K, 1)$ system is established. Only the packets transmitted by the source on the failed channel after the failure would be lost. All packets transmitted on the backup will be delivered within the performance bounds, since the reservations are ready and waiting. The duration of the disruption would depend on the time for the notification message to reach the source, and could be statistically bounded by reserving resources for the control messages.

No extra support from the network is required to use the resources of the spare sub-channel for best effort data traffic during normal operation. The realtime network is already designed to allow the unused capacity of realtime channels to be used by best effort data traffic. Of course, leaving the spare capacity unused is only useful if sufficient best effort datagram traffic exists to use it.

IFI and SFI channels

Finally, we discuss two other schemes which have been proposed for fault-tolerant realtime networks, in order to compare the network requirements and the services offered to those of the schemes proposed here. These schemes were also mentioned in Chapter 1. Single Failure Immune (SFI) channels, proposed in (73), provide immunity to single failures, but use roughly three times the capacity required by a non-fault-tolerant realtime channel. The basic idea is to start with a realtime channel, remove single links from it and run routing algorithms to find the smallest additional set of links, which would allow the system to keep running if that link failed. The routing algorithm used to add the links is similar to *link rerouting*, but this process is performed *a priori* during establishment, rather than at the time of the fault. At each stage, the smallest set of links is added, which would allow the system to survive the fault, subject to the delay constraints. This process is repeated for all the links in the original realtime channel. Finally, resources are reserved on each link in the system. The aggregate set of links has the property that in the event of failure of any single link of the original realtime channel, the data stream can be rerouted by the node adjacent to the failed link, to follow an alternate path which consists only of links within the aggregate set of links (excluding the failed link), and still obey the delay constraints. In the example provided in the paper, the fault-tolerant channel has close to three times the number of links as the original realtime channel. Thus, under the approximation that the delay constraints on all the links are close to each other, roughly three times the resources are required by the SFI channel as compared to the non-fault-tolerant realtime channel. However, since the scheme is a cold-standby one, the reserved capacity can continue to be used by best effort traffic until the fault occurs. The system requires network support to allow the intermediate nodes to update channel identifier mappings after detecting the fault. The disruption of service is limited to the amount of time to perform this activity.

This idea is extended in (74) to cover isolated failures, i.e., to provide Isolated Failure Immune (IFI) channels. Node faults are defined as isolated as long as the source and destination are not faulty, and no two adjacent nodes are faulty. A link is defined to be faulty not only if it fails, but also if the node to which the link leads fails. Link failures are said to be isolated if any two faulty links are not originating from the same functioning node, or directed to the destination node of the channel. The routing constraints of SFI channels

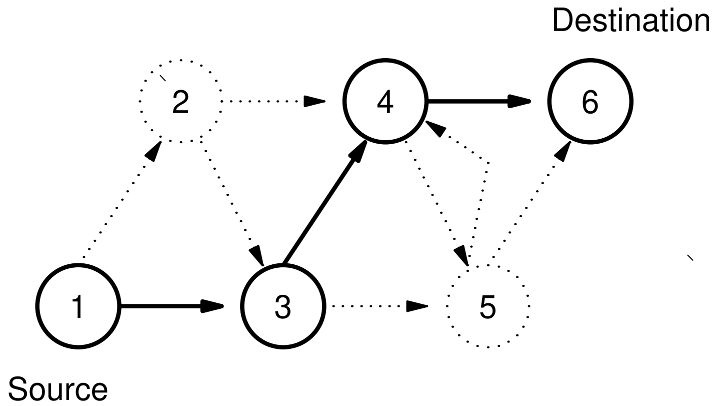

Figure 6.1: One set of isolated link failures which the IFI channel can survive

are modified to extend the fault tolerance to all sets of faults which do not violate the restrictions defined above. The overhead (number of extra links added to the realtime channel) continues to be roughly three. The other restrictions of the SFI scheme, such as network support to change the channel identifier mappings to make sure the data follows the new path in the event of a fault, hold. The IFI scheme offers a better service, since tolerance to multiple faults is provided. In fact as Figure 6.1 shows, a very large number of links can fail before an IFI channel is disabled. However, the IFI techniques are only valid in very restricted network topologies, since IFI sets of links do not exist in most common topologies. For example, in the mesh topology of Figure 6.2, the failures of links (2,3), (7,8), (12,13), (17,18), and (22,23), while isolated, break the graph into two disconnected components. Even if reservations are made on all of the links of the graph, the result is still not IFI. Thus, no IFI set of links exists in this (and most) graphs. The topology of the HARTS network, in the context of which the IFI channel is defined, is a wrapped hexagonal mesh. This is the only commonly-used network topology in which IFI channels exist; thus, the applicability of the algorithms and theorems of (74) is very restricted.

6.4. Simulator design

The simulator described in Chapter 4 was modified to conduct experiments concerning the fault tolerance provided and the network resources used by the dispersity

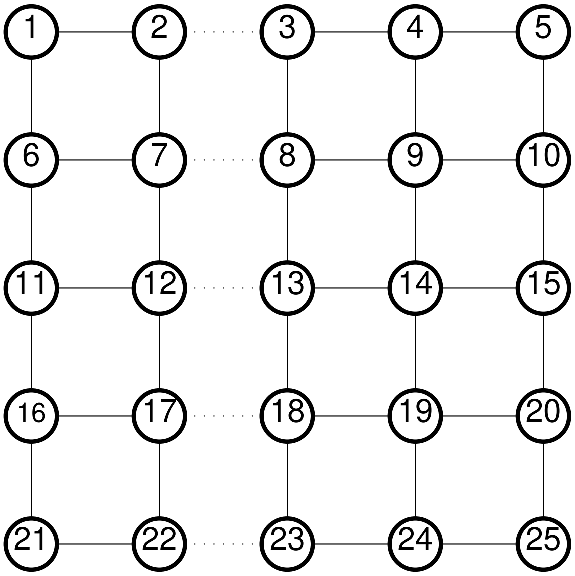

Figure 6.2: No IFI set of links exists in a square mesh topology

systems described in this chapter. The following changes to the simulator had to be implemented.

- The parameters for the sub-channels are calculated according the dispersity systems being simulated, as described in Equations 6.1 or 6.4.
- The routing algorithm had to be modified to route the sub-channels belonging to a dispersity system on disjoint paths. If disjoint paths cannot be found links, can be shared by at most S sub-channels. The routing algorithm attempts to keep the number of such shared links as small as possible.
- The routing algorithm tries to find paths with equal delay. Any differences in the delays on the different paths are equalized by buffers at the destination. Statistics are maintained about the number of buffers used.
- The generation of a packet at the source causes the transmission of N sub-messages, one on each path in the dispersity system.
- The statistics maintained by the simulation depend on the dispersity system being simulated. For example, for a $(5,3,1)$ system, the message is counted as being

successfully delivered to the application, when any three of the sub-messages corresponding to the message have arrived.

Routing for the sub-channels of a dispersity system is performed one by one. The first sub-channel is routed as a normal realtime channel. The additional constraints of disjointness are added for each subsequent route calculation. The changes to the routing algorithm were implemented as a set of preprocessing steps on the graph representing the network before the Bellman-Ford algorithm was run. To enforce disjoint paths, the links corresponding to the paths which should not be used are removed from the graph by setting the delay on the path to infinity. To allow links to be shared S times, we maintain a variable U_e associated with each link e, which is the number of times the link e has been used by sub-channels in the dispersity system. When $U_e \geq S$, the link e is removed from the graph by setting the delay to infinity as before. If $0 < U_e < S$, we want the routing algorithm to find paths including e only if other paths do not exist. We replace each such link by n links, each with a delay of $1/n$ times the delay on the original link. Since the algorithm returns the shortest path which meets the delay constraints, it finds paths which do not use the shared links before it find paths including them, up to a difference in path lengths which can be tuned by changing n. We do not change the end-to-end delay on any path by this replacement process; thus, we are guaranteed to find all paths eventually.

The method of reassembly used in the simulator consisted of a circular buffer, each slot capable of holding N sub-messages. The structure is shown in Figure 6.3. A newly arriving sub-message is placed into the appropriate slot in the circular buffer. The index into the circular buffer is the remainder after dividing the sequence number on the packet modulo B. As soon as K sub-messages have arrived for the same sequence number, the message may be reconstructed. Reconstructed messages are passed in sequence number order to the application, and the corresponding slot in the circular buffer is freed for reuse. Alternatively, we may hold each message up to the end of its delay bound, thereby removing all the jitter from the stream. This may be done using time-stamps on packets and the knowledge of the delay bound D_{max}, if the clocks on the source and destination machines are synchronized, as may be done using a time protocol such as NTP. Since we know that no sub-message can be delayed by more than D_{max}', the number of buffers required is bounded, and we are guaranteed not to wrap around in the circular buffer. Of course, the reassembly code checks this condition, to ensure that older packets do not get assembled with sub-messages from current messages. Such a case would be counted as a violation of the delay bound guarantee, and the old packet discarded.

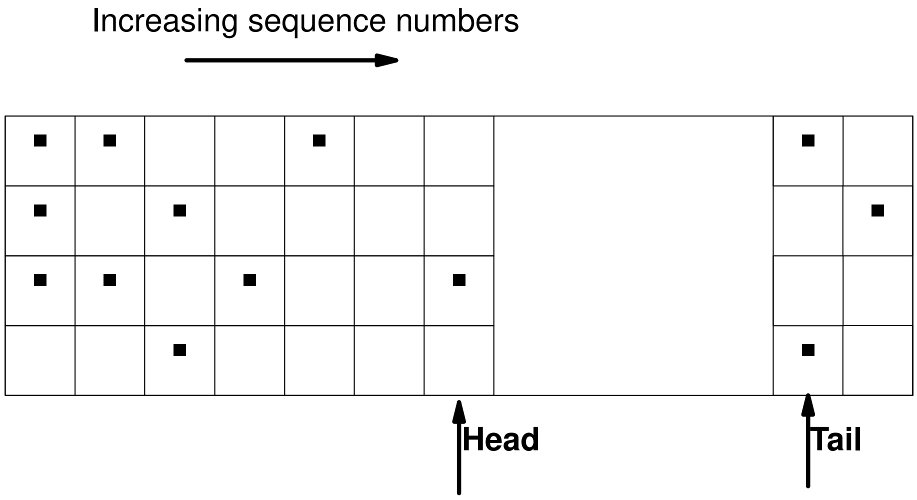

Figure 6.3: Structure of reassembly buffer for $N = 4$

The simulator models this behavior at the level of detail of individual packets. Thus, the encoding and decoding operations, which occur at the bit-level, are not simulated. The transmission of a message on the dispersity system is modeled as the transmission of packets on each sub-channel of the dispersity system, but the bit level contents of the packets are not simulated. The arrival of the appropriate number[2] of packets from the same message at the destination is interpreted as a successfully decoded message, and the statistics are updated accordingly. Encoding and decoding times are not modeled,

[2] Depending on the dispersity system being simulated.

since they are dependent on the coding algorithms, which are still under development, and on the hardware available on the workstation.

We modeled the process of notification of the source to initiate switching, since a fault message is sent from the detector to the source to initiate fault recovery, and the same message can be assumed to trigger the switching for the cold standby systems. The process of switching itself can be assumed to take a negligibly small amount of time, as compared to the notification time, which involves network communication.

The requirements of a realtime dispersity system are described, in the event file from which simulation is started, by the end-to-end traffic specifications and performance requirements and the variables N, K, and S. The parameters of the sub-channels are calculated as defined in Equation 6.1 or Equation 6.4. The additional restrictions on the routing algorithm are automatically generated, using the routing extensions described above.

6.5. Simulation results

The experiments performed using the simulator fall into two categories. The first set of experiments are intended to verify the level of fault tolerance provided by the dispersity systems. Since the reservations are pro-active, and the guarantees provided on the sub-channels are mathematical and deterministic in the absence of losses in the network, we can prove that the service guarantees hold, given that the assumptions about the fault(s) and other losses are satisfied. The simulations serve to confirm these expectations, as well as to demonstrate the workability of the systems to a fair level of detail. In the presence of network losses due to transmission errors and faults, which are not covered by the Tenet guarantees, the dispersity systems can provide lower packet loss than the basic realtime channel; this performance is experimentally evaluated and compared to values computed from a simple probabilistic model of network loss in the first part of this section. For the dispersity schemes with redundancy, the service provided is mathematically guaranteed even in the event of a failure; thus, we expect that, in the absence of transmission errors, no message will miss its deadline. This is confirmed in our simulation experiments. We also measure the buffer requirements of the dispersity systems, to cross check them against the computed values from Equation 6.2.

The second category of experiments performed deal with the impact of the dispersity systems on network capacity. The simple analysis provided in the previous section shows that the resources consumed by an (N, K, S) system are approximately N/K times the resources consumed by an equivalent non-fault-tolerant realtime channel. In simulation, we compare this result to the number of instances of each type of system that can be established in the network under varying load conditions. This provides us with a better empirical understanding of the effect of the more subtle issues, such as the effect on the network capacity of splitting the resource requirements among N sub-channels on different paths and the probability of finding resources on N disjoint paths.

6.5.1. Performance results

The first set of experiments conducted consisted of setting up one connection using the dispersity system under consideration, in a network carrying different levels of background load, and simulating the service provided in the presence and absence of faults and losses due to transmission errors in the network. The network being simulated is the "core" topology discussed in Chapter 4. The background load is obtained by choosing a set of channel requests from the "original" distribution of Chapter 4. Zero or one faults are simulated, depending on the experiment. Losses due to transmission errors are simulated by dropping each packet with a given probability on each link. Though the simulator supports specifying loss probabilities for each link separately, in the experiments described here all links were assigned the same loss probability.

All the experiments described in this section were performed for all of the dispersity systems. The performances observed for all the dispersity systems meet the expectations of Section 6.3.2. We will not present all the results for each separate dispersity system. Rather, we will use a running example, to present some specific results. The general

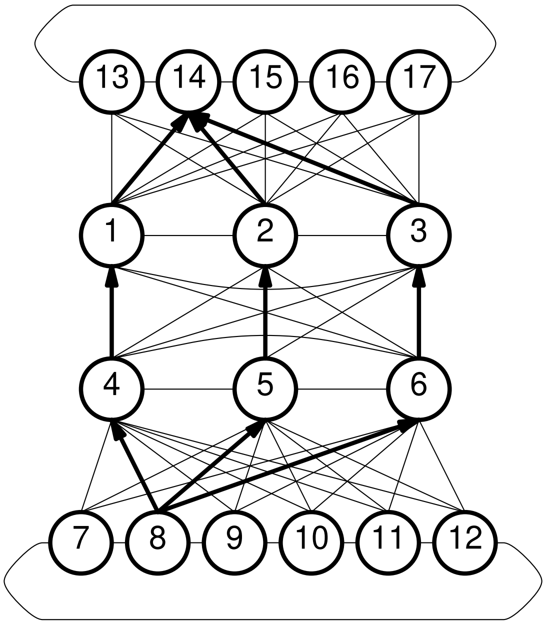

Figure 6.4: (3, 2, 1) **dispersity system simulated**

conclusions presented below from the specific experiment are applicable to the entire set of dispersity systems, unless otherwise noted. Where there are differences, we will show the performance of some of the other systems as well.

The example system is a (3, 2, 1) dispersity system, established from node 8 to node 14 in the network depicted in Figure 6.4. The traffic parameters for the fault-tolerant dispersity system were $X_{min} = 5$ ms, $X_{ave} = 5$ ms, $I = 500$ ms, and $S_{max} = 10000$ bits. This corresponds to a 2 Mbps JPEG compressed stream, called a class A channel in Chapter 4. The end-to-end delay bound for the system is seventy milliseconds, and the minimum propagation and transmission delay is thirty-five milliseconds. B as computed by Equation 6.2 is eight. The simulator does not support statistical delay bounds, jitter control, or variable buffer overflow probabilities. Thus, by default, $Z_{min} = 1$, $J_{max} = D_{max}$, and $W_{min} = 1$. In terms of Figure 6.3, we need at the destination (node 14) a circular buffer eight deep and three wide. We have $S'_{max} = 5000$ bits or 625 bytes. This is the size of each buffer unit in the circular buffer. The total memory needed is, thus, 15,000 bytes.

The simulation was performed for zero and one fault in the network, at various probabilities of packet loss on the links, and various extraneous loads in the network. The network load was created by establishing varying numbers of simple realtime channels in the network, before starting the observation of the fault-tolerant system. The measurement was performed by allowing the experiment to run for 39 seconds of simulated time. This took an hour or more of actual time, depending on the speed of the machine running the simulation and the other load on the machine, because of the level of detail being simulated. The simulator kept track of the number of packets dropped, the number of messages decoded, the buffers used, and so on. When a fault was simulated, all packets on the failed link were lost. The fault recovery process described in Chapter 4 was activated, and the affected channels rerouted. Unlike the experiments in Chapter 4, data transmission across a link could also fail randomly with a probability determined by a parameter of the experiment, simulating losses due to transmission errors. At simulated time equal to 39 seconds, the measurements were written out to a file. Further simulation, involving tearing down the affected channels to compute the efficiency of resource usage, was performed, and at simulated time equal to 43 seconds, more statistics were collected and the simulation was stopped.

The general results which apply to all the experiments performed, but for which no graphs need to be shown, are described first. The service provided met the realtime guarantees in all cases. No packets were delivered after the delay bound. In the absence of losses due to transmission errors, no messages were lost for zero and one link failures. These observations were not affected by the level of the extra realtime load on the network, since the guarantees on the sub-channels hold irrespective of the other traffic.

In the experiments in which failure was simulated, one of the links in the central core was removed from service. This caused the fault recovery process described in the previous chapters to be triggered. The fault message reached the source in under 20 ms, because of the topology and proximity of the failed link to all sources. This is a measure of the time for notifying the source to perform switching for the cold standby sources, and is dependent on the topology. The fault recovery process also rerouted the failed sub-channel to an alternate path. In all our experiments, the reroute was successfully completed.

We also measured the maximum buffer occupancy in the circular buffers during the simulation. No more than eight buffers were needed for the running example of the (3, 2, 1) system, which matches the calculated bound exactly. The actual usage was less in the experiments with no faults and low loss rates due to transmission errors, since the maximum buffers are only needed when a message fails to be successfully decoded until its deadline. In case more than $N-K$ sub-messages from the same message are lost, the reassembly algorithm waits until the delay bound for the message expires. The message is then deemed lost, and the buffers for the sub-messages are freed. In the meantime, other messages may have come in with the minimum propagation delay in the network. Thus, in the case of experiments involving significant loss due to faults or transmission error the buffer size is predicted by Equation 6.2 exactly. The load due to the other realtime channels in the network did not affect the buffer requirement at all.

The numbers of buffers required by all the dispersity systems shown here are fairly moderate. The calculation for B is identical for all of them, since the delay bounds and the minimum propagation delay on the shortest path are the same. This analysis was confirmed by our simulations in all cases. The only difference in buffer requirements among the various dispersity systems is the width of the buffer, which is given by N. Thus, the largest amount of memory (25,000 bytes) is required by the systems with $N=5$. This amount of memory is fine for most general-purpose computers, and would also be acceptable for specialized devices like set-top boxes. The lesson here is that, by controlling the end-to-end delay for the component sub-channels, we have reduced the problem of buffering at the destination. In case the buffer requirements do become significant, as might happen for higher bandwidth connections than the 2 Mbps channels we used in our simulations, we can reduce them by using jitter control (Eq. 6.3).

When we introduce losses of packets due to transmission errors in the network, we demonstrate the loss-tolerant nature of the system. The transmission error process used in the simulation is very simplistic, since we drop packets with a fixed low probability on every link. This is not an accurate model of real transmission errors, since the error process displays temporal auto-correlation at the bit level. However, at the longer interval corresponding to the transmission time of a packet, this correlation is much less noticeable. In our experiments, the packet size is 10,000 bits, which would make the probability of loss even for consecutive packets almost independent. Thus, our assumption of stochastic independence for the transmission error process is reasonable.

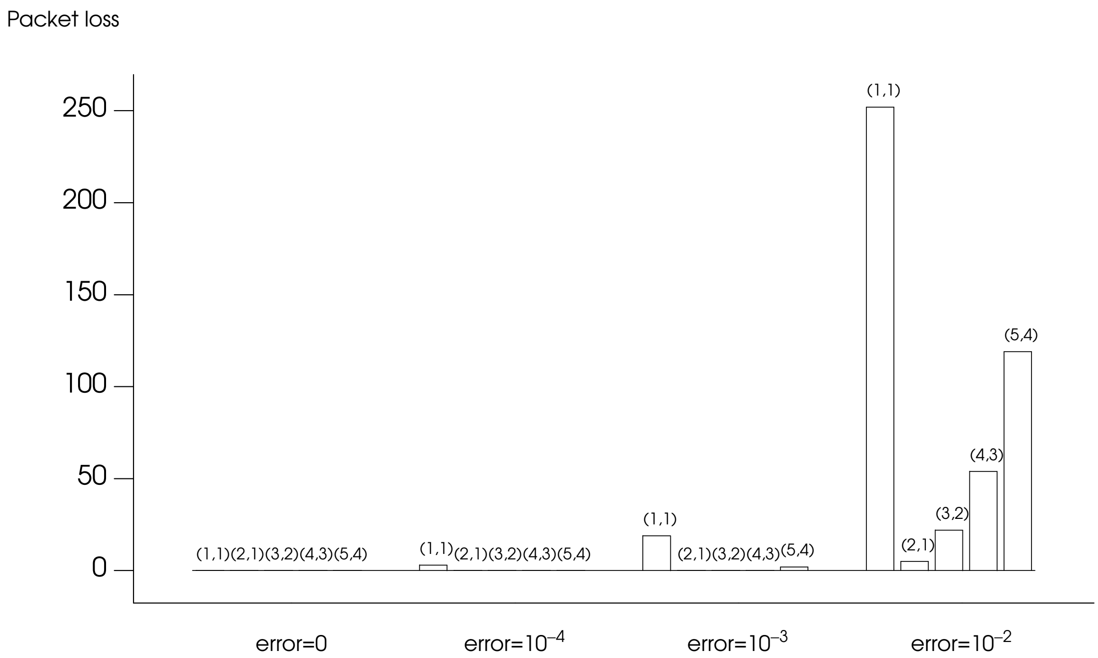

Figure 6.5: Packet loss vs. loss rate for dispersity systems with N - K = 1, no network failure case

Note that the loss process we are simulating does not include losses from buffer overflow at the nodes, since those are controlled by the realtime network. While the observed loss characteristics of packet switched networks do display auto-correlation at the packet level, the main source of such losses is buffer overflow due to congestion at network routers. This source of loss is eliminated in our experiments[3] by the realtime protocols, by reserving buffers on a per connection basis, so that buffers are guaranteed to be present when a realtime packet arrives at any node.

Figure 6.5 shows the effect of the loss rate due to transmission errors in the network on the loss of messages as seen by the application using the dispersity system. On the x-axis, we have the probability of losing a packet on any link. Note that the graph shows loss rates in the network up to a 10^{-2} probability of losing a packet during each transmission on every link in the network. These are not meant to be representative of transmission error rates in real networks. The actual error rates seen in fiber-optic transmission systems are much lower. We simulated high transmission error rates so that we could observe a significant number of lost packets and its effect on the overall dispersity system. On the y-axis we have the number of messages lost by the system. There are 5 bars for each loss rate, but up to a loss rate of 10^{-3}, most of them are of close to zero height. The height of each bar denotes the number of messages lost by the dispersity system due to lost packets. The number of packets lost on the sub-channels is, of course, larger, since a message is lost only if sufficient packets are lost to defeat the redundancy in the system. The tuple shown over each bar is (N, K) for the dispersity system represented by the bar. The variable S is 1 for all the systems simulated, so it is not shown. Only the systems with $N - K = 1$ and the $(1, 1)$ system are shown here. The $(1, 1)$ system is the simple realtime channel without any fault tolerance.

Since the simulator models packet loss by dropping packets with a fixed independent probability (one of the parameters of the experiment) during each packet transmission event, the loss rates are very close to that calculated from the knowledge of the loss rates on each link and assuming independent events. For example, the $(1, 1)$ system is a

[3] Buffer overflow probability is always zero in the performance model implemented in the simulator.

simple realtime channel with no added tolerance. With a loss rate due to transmission errors in the network equal to a probability of 10^{-3} of dropping a packet during a transmission event, and the path length equal to three, the probability of a message being successfully received is $0.999^3 \approx 0.997$. Since 7669 packets were transmitted during this experiment, the expected number of lost messages was 23. 19 messages were observed lost in the experiment. If the experiment were carried on for a longer time, the observed number would converge to the expected mean. The expected number of lost messages for the fault-tolerant systems can be similarly calculated, since the paths are disjoint, and therefore independent. The calculated numbers match the simulated behavior. The number of messages lost is also *independent* of the load on the network due to other realtime channels, because the effect of loss due to congestion is eliminated by buffer and bandwidth reservation at intermediate nodes by the realtime protocols.

The loss process being simulated is extremely simple and the graph for Figure 6.5 could have been easily computed analytically. We present it to give the reader an idea of the error tolerance benefits of the dispersity schemes. The results merely confirm that the simulator is behaving as expected, and demonstrate the workability of the schemes to a certain extent.

Figure 6.6 shows the performance of the dispersity systems when a fault also occurs during the experiment. Now the number of packets lost in the simulation includes the number lost on the failed channel. The system continues to provide realtime performance with increased tolerance to loss as compared to the non-fault-tolerant realtime channel, but the losses noted are slightly higher. The loss is controlled by two mechanisms. First, the duration of the outage of a sub-channel is limited by the *recovery* process that reroutes the faulty sub-channel. We will return to this point in the next section. Secondly, the redundant coding provided (for systems with $N > K$) controls the number of lost messages during the outage. Thus, though the non-fault-tolerant channel (the $(1, 1)$ system) lost 15 messages during the period of outage in the experiment with loss rate = 0, the other systems did not lose any messages. In a network with extremely high loss rates, the dispersity systems do lose messages, but far less than the system without redundancy. These high loss rates are not expected to be seen in most networks under normal conditions, since we are referring only to losses due to transmission errors here, and the transmission error properties of most current networks (and very likely that of future networks too) are far better than the rates at which we see lost messages in these experiments.

Figure 6.7 shows the performance of the systems with $N - K = 2$ and with a fault during the experiment. These systems are tolerant even at very high loss rates and one fault in the network. These systems also need the same amount of buffering at the destination to equalize delays and reassemble the packets. The cost of using the higher level of redundancy is the higher bandwidth requirement of the system. A $(4, 2)$ system needs twice the bandwidth of a non-fault-tolerant realtime channel, as opposed to a $(4, 3)$ system, which needs only 1.33 times the bandwidth. However, the effect on the network capacity is more complex than can be understood by just considering the bandwidth of the component sub-channels, for reasons which are explored in the next set of experiments.

Overall, in this set of experiments we have shown that, at the level of detail that our simulator models a realtime network, and subject to the limitations in the faults being simulated, the service provided by the dispersity systems meets our expectations, even in the presence of single link faults and reasonable transmission error rates. The realtime guarantees are never violated, and the error and fault tolerance is enhanced. Further, the performance is guaranteed independently of the load, since the component sub-channels of the dispersity systems are realtime channels. With the error rates seen in current transmission systems, any of these dispersity systems will provide adequate protection against single failures. Thus, the choice of which one to use is dictated in part by the question of the degree of tolerance required, that is, whether tolerance against a single fault, two faults, or *n*-faults is required. The degree of tolerance is determined by $N - K$,

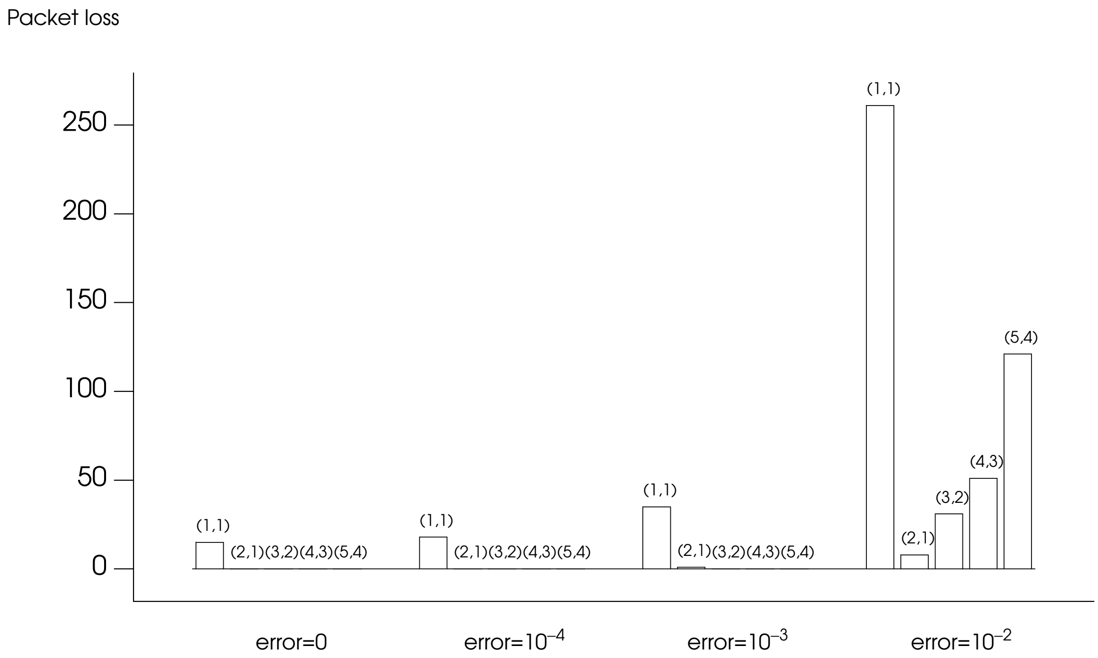

Figure 6.6: Packet loss vs. loss rate for dispersity systems with N - K = 1, one failure

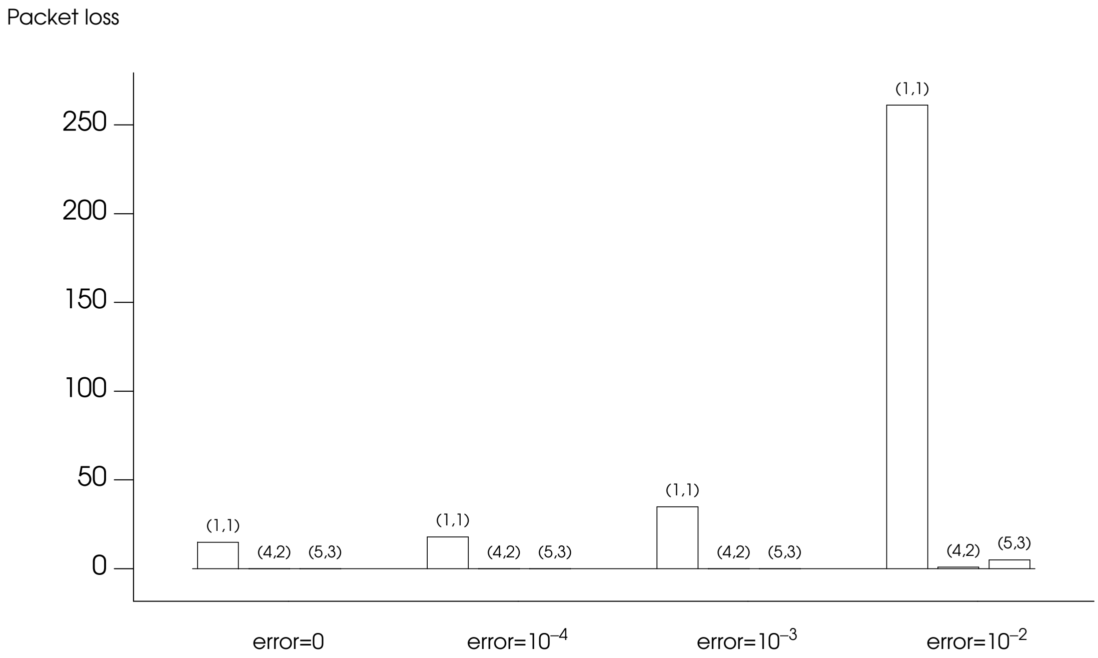

Figure 6.7: Packet loss vs. loss rate for dispersity systems with N - K = 2, one failure

assuming maximum distance separable codes. If the decoding needs to be performed in software using a low-performance CPU, the parity code restricts us to $N-K=1$ systems. Between systems with the same value of $N-K$, the simple analysis based on the amount of extra bandwidth used ($\frac{N-K}{K}$) seems to suggest that the higher the values of N (and K), the lower the impact on the rest of the network. This issue of network capacity will be explored in the second set of experiments with the simulator.

6.5.2. Network capacity

The effect of the different dispersity systems on network capacity was tested by seeing how many connections of each type could be established starting with a given initial load. The initial load is determined by the set of simple realtime channels already established in the network before the experiment starts, and is fixed by including a prefix with the same set of channel requests in the event file from which the simulation is started. The experiment is conducted for each type of dispersity system, for four initial load conditions. For each value of load, we repeat the experiment for three sets of channels generated from different random seeds, which have the same load index in the "core" network. The results are averaged from these three experiments. The number of connections that can be established is determined by attempting to set up a larger number than can be possibly successful, and then counting the number of successful establishments. The source and destination are chosen from opposite sides of the core topology randomly. The traffic and performance parameters of the dispersity systems are the same as in the last section. The parameters of the background load are picked from the "original" distribution again.

The effect of introducing dispersity without redundancy is explored first. These systems can be characterized by $N=K$. As explained is Section 6.3.2, the service provided by such systems degrades partially in the event of a fault, until the failed sub-channel is rerouted. During this time, a lower bandwidth connection is maintained. The bandwidth requirement of the dispersity system is the same as that of the non-fault-tolerant realtime channel; thus, one might expect that the same number of total connections can be established for each dispersity system as for the non-fault-tolerant realtime channel. However, the simple model based on bandwidth does not take into account the external fragmentation due to the size of the bandwidth requirement of the channel relative to the capacity of the link, and the probabilities of finding multiple paths meeting the delay constraints and having adequate resources. This depends very much on the particular network topology; for example, the fact that we can find five link-disjoint paths is based on the excellent level of connectivity of the "core" topology. It also depends on the background load. The experimental results show the effect of the level of dispersity on the ability to set up fault-tolerant connections for different levels of network load, and allow us to draw conclusions about the effect of the establishment of a fault-tolerant connection on the network's ability to accept subsequent realtime requests, as well as about the level of difficulty of finding the required number of disjoint paths in the network.

Figure 6.8 shows the number of connections of each type which could be set up, for the dispersity systems without redundancy, at different realtime load levels in the network. The $(1, 1)$ system is the non-fault-tolerant realtime channel, which serves as the control for the experiment. We note that, at all load levels, the number of connections that can be set up first increases and then decreases with N. This requires some explanation. The initial increase is due to the fact that each sub-channel in a dispersity system has a smaller-bandwidth requirement than the original request, and the requirement gets smaller with increasing N. Thus, the level of external fragmentation on the links decreases, since smaller bandwidth channels can be packed more efficiently on the links. This effect increases with the load on the network, since the remaining capacity of the network is smaller compared to the size of a single request, leading to more external fragmentation. This effect would be more pronounced in a network with small capacity, and less pronounced in networks where the capacity is much larger than the requirement of a single

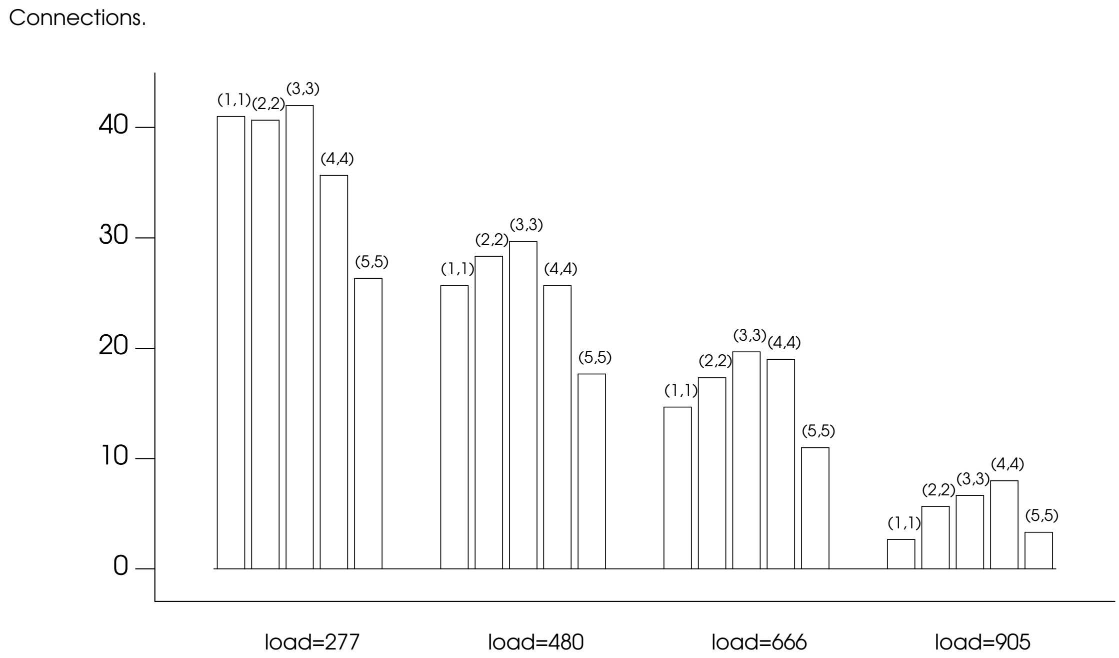

Figure 6.8: Number of connections established for dispersity systems without redundancy realtime channel.

The fact that the bandwidth requirement of each individual sub-channel is smaller than the total requirement of the dispersity system also has an important effect on the capacity of the network to accept subsequent realtime requests until the dispersity system is torn down. Since the amount of resources used on each path is lower, the effect of the request is spread out over the entire network, so more capacity remains on each individual path as compared to a realtime channel that places all its resource requirement on a single path. The amount of actual traffic from the dispersity system on each path is also lower than the traffic on the single path of an equivalent realtime channel. Thus, the dispersity system is friendly to other users, both realtime and non-realtime, of the network.

The subsequent decrease in number of connections established with increasing N is due to the limited number of disjoint routes in the network. As mentioned before, the maximum number of link-disjoint paths between nodes on different sides of the core topology is five. Thus, as N approaches five, the number of choices between alternative paths for the set of routes decreases. At $N = 5$, we need to find resources on all the available paths; if any path is too highly loaded, the request fails. Thus, the ability to establish a large number of fault-tolerant connections, and the probability of being able to establish one in a highly loaded network, decreases as N approaches the number of alternate paths available. For $N > 5$, no connections can be established in this network. This observation is important, since we do not anticipate that even very large integrated-services networks will have more than four or five disjoint paths available between any source-destination pairs. This shows that the practicality of any dispersity scheme with large N is limited by the network topology.

The number of connections that can be established starting from a lightly loaded network is an indication of the capacity of the network regarding the specific dispersity system. Thus, in this topology, the number of $(5, 5)$ connections that can be established is a little more than half the number of non-fault-tolerant realtime channels that can be established. As explained before, this is because N is at the limit imposed by the number of disjoint paths available in the network. On the other hand, we can actually establish more $(3, 3)$ connections than non-fault-tolerant realtime channels, because at this stage the effect of reduced external fragmentation is more important. Thus, bandwidth is just one of the three factors that determine the number of connections that can be established in a network.

The number of connections that can be established in a heavily loaded network is indicative of the relative probability of being successful in establishing a connection using a specific dispersity system in a similarly loaded network. In other words, we can interpret the sizes of the bars at load=905 to mean that we can expect to establish a $(4, 4)$ connection with greater probability of success than even a non-fault-tolerant realtime channel if the network is heavily loaded. This result is surprising, since it counters our initial intuition that finding resources on four paths successfully, especially when there are only five available, should be harder than finding resources on just one. However, we should interpret the result to mean that at the high load level, because of the small amount of resources left on each link, the influence of external fragmentation is more important than the effect of the disjoint path restriction.

An apparent contradiction can be pointed out in this argument. We note that the number of $(4,4)$ connections that can be set up at low loads is less than that number for the non-fault-tolerant realtime channels. But we are claiming that at high network loads, it is easier to set up $(4,4)$ connections. Thus, in the experiment starting at low loads, when the network is becoming saturated after establishing 35 connections, we should be still be able to establish more $(4,4)$ connections than $(1,1)$ channels, which is contradicted by the observed results.

The explanation for this apparent contradiction is that the nature of the high network load seen by the $(4,4)$ connections towards the end of the experiment starting with load=277 is not the same as that seen by the $(4,4)$ connections in the experiment starting with load=905. The network state in the second case is much more random, and therefore likely to be uniform, because of the distribution of the requests which created the load. They are composed of requests from three classes of applications, most of which are much smaller in bandwidth requirement. Thus, the resources reserved on all the links are much more likely to be smoothly distributed. On the other hand, the set of requests that created an equally high load index in the first experiment is not so random; they are all $(4,4)$ requests from Class A. The dispersity system breaks the request into four smaller sub-channels, but the capacity requirements of each is still larger than that of most of the requests in the distribution comprising the load in the second experiment, since the "original" distribution is composed mostly of Class B and C requests. The routing algorithm used is deterministic; so, it tends to send the sub-channels to the same set of links. Towards the end of the experiment starting at low load, some of the links had sufficient available capacity, but four disjoint paths with sufficient capacity could not be found, because of the unbalanced distribution of the load. On the other hand, in the experiment starting at high load, the load is evenly distributed on all the links, so that several $(4,4)$ connections can still be established. Thus, we should restate our earlier observation to say that, with the given topology, when the network is highly loaded and subject to the constraint that the load is evenly distributed, it easier to set up a $(4,4)$ connection than a non-fault-tolerant realtime channel.

This leads to the conclusion that the routing algorithm as implemented does not deal effectively with a large number of requests of the same class of dispersity system. The behavior of the network can be improved by suitably randomizing the routing algorithm for all channels, and specifically for the dispersity sub-channels. We needed the deterministic property of the routing algorithm to formally define the load indices, since they are only well defined if the path followed, and hence the load generated, is deterministic for the same network state. We also needed a deterministic behavior to make the experiments repeatable. But in an implementation of the realtime network, some care should be taken to design the routing algorithm to spread out the requests on the available paths, and randomization is one possible way of achieving this.

Figure 6.9 shows the change in behavior when we lift the strict disjoint-paths constraint and set $S = 2$. This allows the routing algorithm to return paths that share links between at most two sub-channels. Thus, in the event of a single fault, depending on which link failed, up to two of the sub-channels may fail. However, while the degradation of service will be more severe, some communication will still continue if $N > 2$. Thus, for $N = 5$, in the event of a failure in a shared link, the surviving capacity of the system would be 3/5 of the original, while in the case of an unshared link it would be 4/5. As mentioned before, intra-frame compression codes such as JPEG can continue to function under such conditions, since the data on one path is not dependent for its decoding on the data from another path. The graph shows that the negative effect of increasing N is

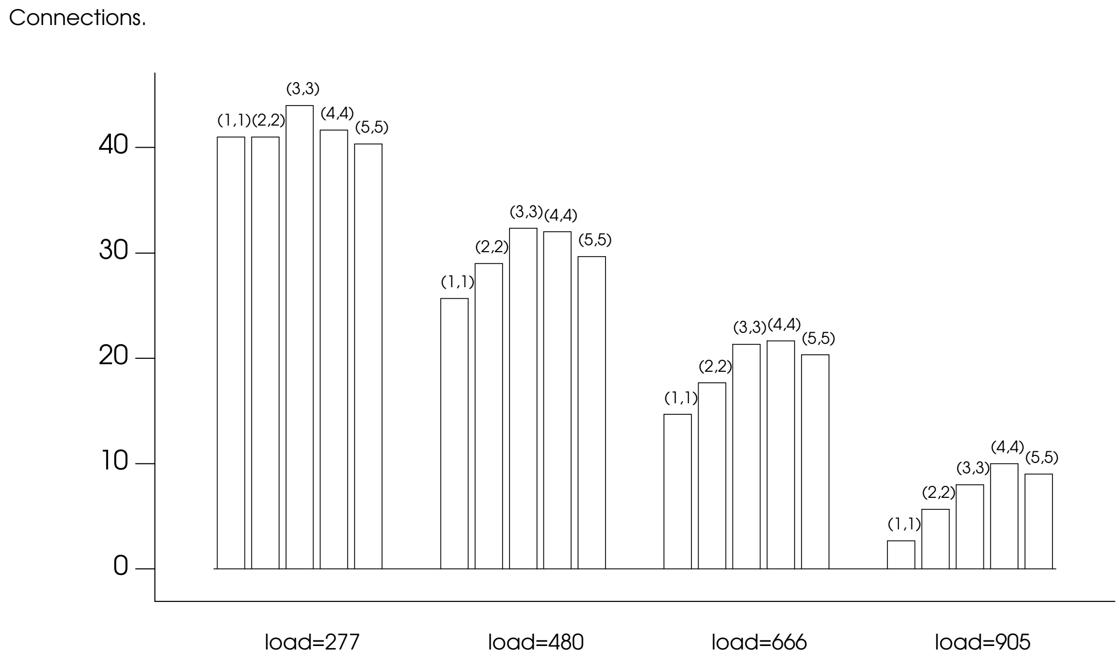

Figure 6.9: Effect of allowing S = 2 on dispersity systems without redundancy

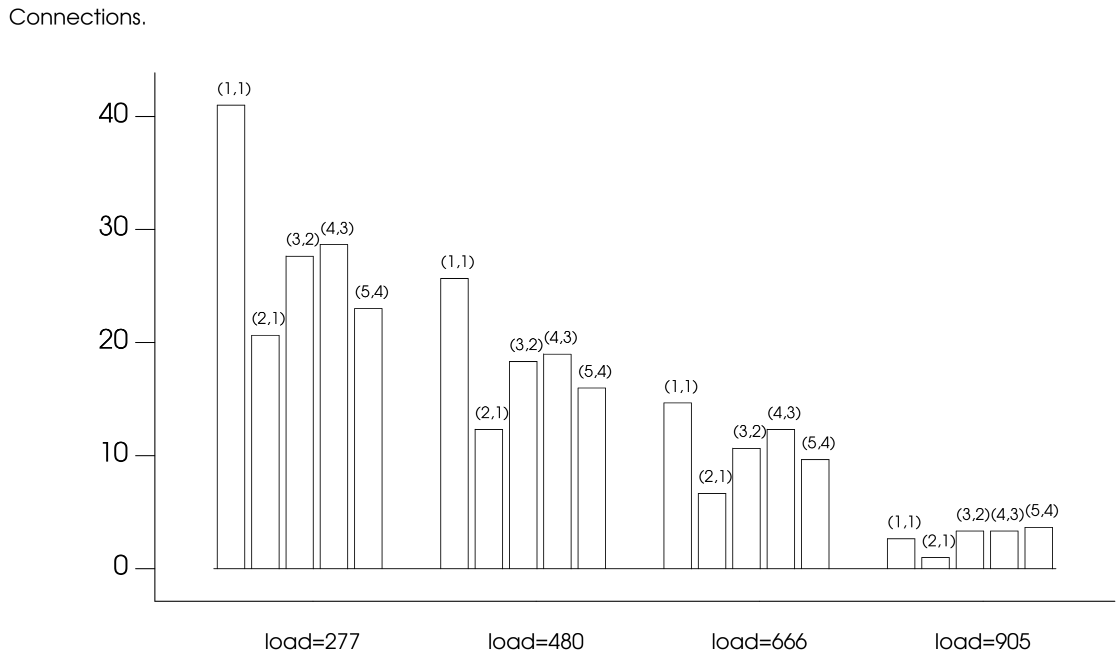

Figure 6.10: Number of connections established for dispersity systems with N - K = 1

greatly reduced, so that even the $N = 5$ system can establish as many connections as the non-fault-tolerant realtime channel in a lightly loaded network. At higher loads, the dispersity systems establish far more connections than the non-fault-tolerant realtime channel. They would also have a lower effect on future connection requests, since the effect is spread out over the network, and not concentrated on one route. The implementation of such systems is simpler than for the systems with redundancy, since no FEC computation is required at the source or destination, and fragmentation and reassembly are also avoided. Thus, these systems have no major disadvantage compared to non-fault-tolerant realtime channels, apart from the slight increase in implementation complexity to handle multiple paths, and the restriction about the kinds of data that can be carried. Only data that can be broken into N streams, each of which can be decoded and used independent of the arrival of the other streams, can enjoy the full benefit of the redundancy systems without redundancy. Other forms of data can also be carried, but in this case service will be interrupted in the event of failure, for the interval of time needed to inform the source, so that it can arrange to use the surviving links.

Figure 6.10 shows the effect of introducing redundancy into various dispersity systems with $N - K = 1$; the $(1, 1)$ system is the non-fault-tolerant realtime channel shown for reference. We see that adding redundancy reduces the capacity of the network to support these systems, since they add some overhead to the network. This effect is combined with the effect of the improvement due to reduced external fragmentation, and the effect of the disjoint-paths requirement. The dominant effect at low load and with small N is that of the bandwidth overhead. Thus, the $(2, 1)$ system can establish half the number of connections as the basic realtime scheme, and the $(3, 2)$ system roughly two thirds of the number. As N approaches 5, the effect of the connectivity of the topology starts being felt, so that the number of connections which can be established for the $(4, 3)$ system is less than three fourths. For $N = 5$, the effect of the network connectivity becomes large enough to drive the number of connections for the $(5, 4)$ systems below that of the $(4,3)$ system. However, as we increase the load, the effect of reduced external fragmentation becomes more important, since the remaining capacity on some links becomes comparable to the bandwidth requirements of the channels. At extremely high loads, we can actually establish as many or slightly more $(5,4)$ connections as a $(1,1)$ system. At this stage, the effect of reduced fragmentation, caused by splitting the requests into smaller sub-channels, dominates. Note that we do not get the benefit of reduced fragmentation for the $(2,1)$ system at all, since the sub-channels have the same bandwidth requirements as the original. Thus, for this system the total capacity requirements determine the number of channels that can be established at all load levels.

The conclusions we can draw from the above observations are:

- the effect of the increased bandwidth is partially offset by splitting a request into smaller sub-channels;
- this process also decreases the effect the scheme has on other traffic, and on future realtime requests;
- the advantage of splitting the request increases with the realtime load on the network, so that, in a highly loaded network, the dispersity systems can be set up as easily as a non-fault-tolerant realtime channel; and
- the disjoint-path constraint becomes significant as N approaches the number of available disjoint paths, reducing the number of dispersity systems with large N that can be established in a network, as well as making it more difficult to set up a dispersity system with large N in a highly loaded network.

Figure 6.11 shows the impact of allowing $S = 2$ on the ability of the network to accept dispersity systems with $N - K = 1$. We notice that the effect of the network connectivity is reduced, so that, for low loads, the number of connections that can be successfully established is determined primarily by bandwidth considerations. At higher loads, the reduced fragmentation caused by the smaller bandwidth requirements of the sub-channels allows many more connections to be set up for the dispersity systems. In

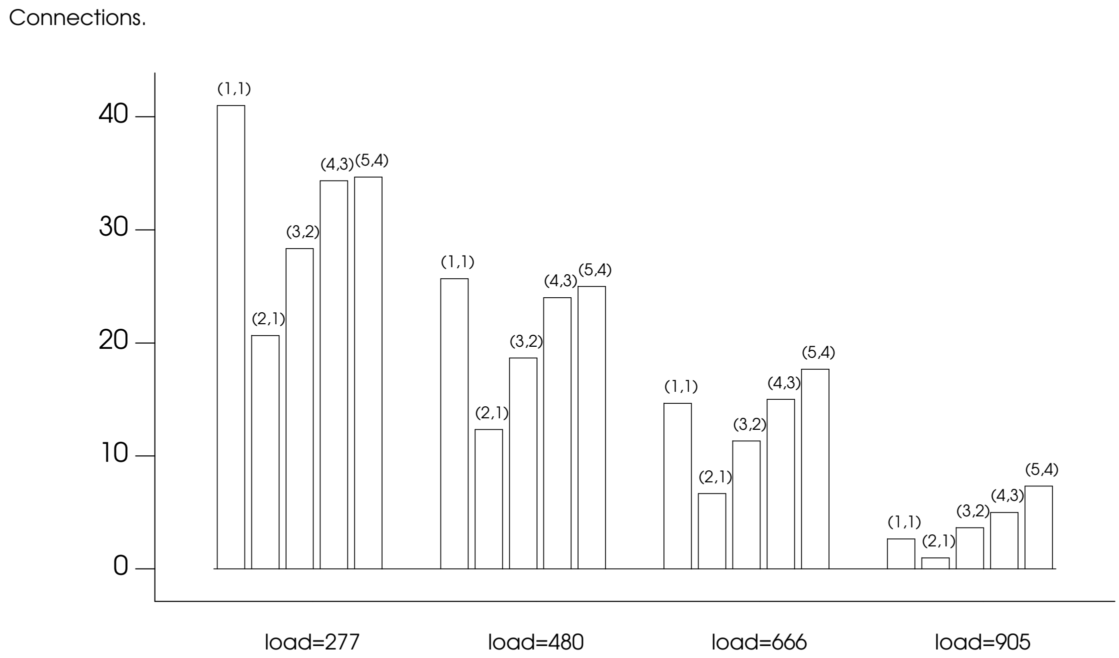

Figure 6.11: Effect of allowing S = 2 on dispersity systems with N - K = 1

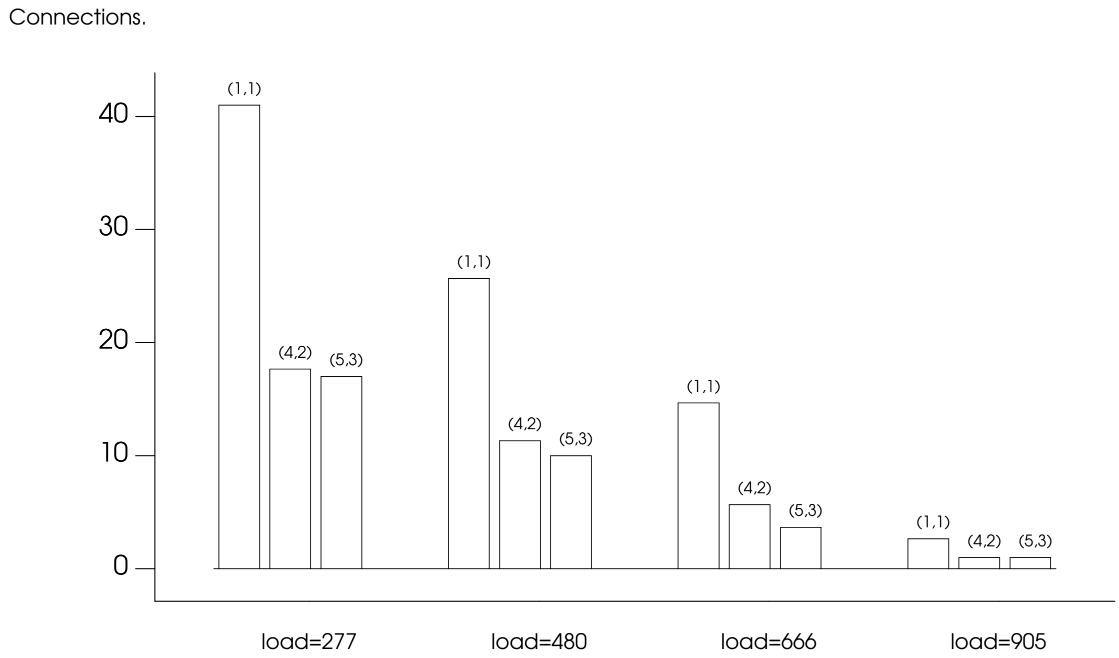

Figure 6.12: Number of connections established for dispersity systems with a N - K = 2

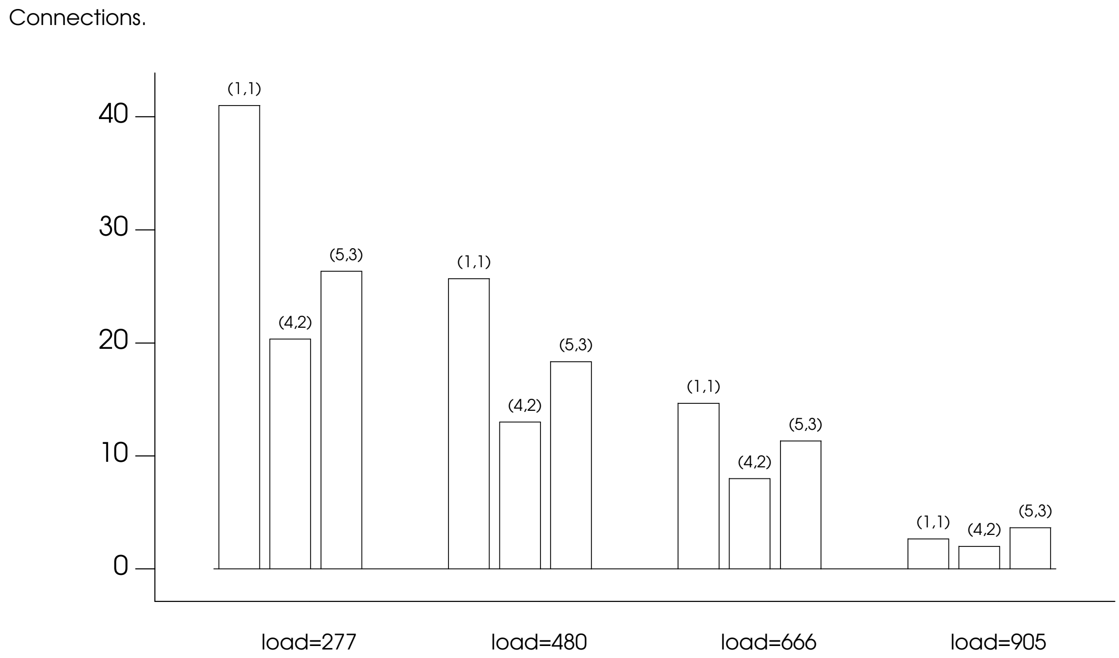

Figure 6.13: Effect of allowing S = 2 on dispersity systems with N - K = 2

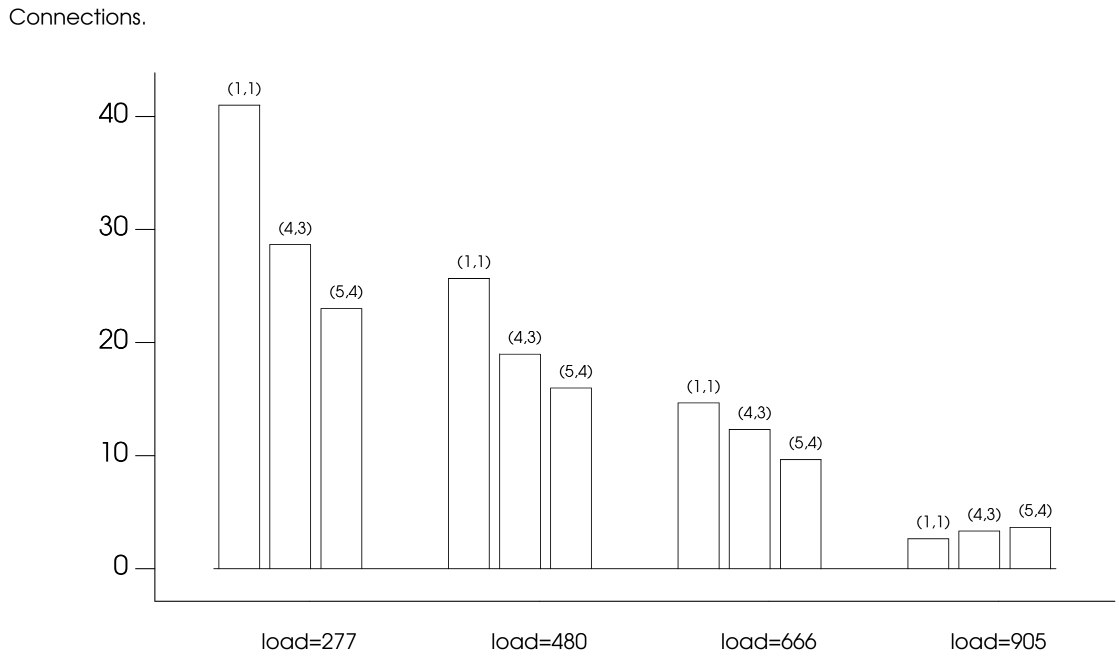

Figure 6.14: Equivalent dispersity systems with no shared links and N - K = 1

fact, the (4,3) and the (5,4) systems are comparable to the non-fault-tolerant realtime channel at low loads, and better at high loads, in terms of the number of connections established. Unfortunately, if a shared link fails, the service provided by such systems is interrupted, so the fault tolerance in these systems only applies to failures of the unshared links in the system.

Figure 6.12 shows the effect of increasing the level of redundancy. The dispersity systems shown have $N - K = 2$; the (1,1) system is again shown for reference. We note that now the effect of the increased capacity dominates, so that, even at very high loads, fewer dispersity connections can be set up than non-fault-tolerant realtime channels. The main reason is that the sub-channels are now as large as the original channel, so we get no decrease in fragmentation. The effect of the disjoint-paths constraint continues to be visible. When we modify the constraint, by setting $S = 2$ (Fig. 6.13), we get no improvement for the (4,2) system. The explanation for this behavior is that the reduced fragmentation effect is not applicable, since each sub-channel has the same bandwidth requirements as the original request. Thus, the only factor is the bandwidth requirement, which restricts the number of (4,2) connections to roughly half the number of basic realtime connections. For the (5,3) system, the fragmentation effect does play a small role, so that, at higher loads, the number of (5,3) connections is comparable to the number of non-fault-tolerant realtime channels. *If* the network had a higher degree of edge connectivity, higher values of N would yield better capacity characteristics for $(N, N-2, 2)$.

The fault tolerance characteristics of the $(N, N-2, 2)$ systems are comparable to or better than those of the $(N, N-1, 1)$ systems. Thus, if one were attempting to establish a (4,3,1) connection, and the network failed to find a fourth disjoint path, one of the options available to the user would be to set $S = 2$, and attempt to establish a (4,2) or a (5,3) system. Comparing the number of connections that can be established in Figure 6.13 and Figure 6.14, we see that the (5,3,2) system may indeed be a viable alternative to the (5,4,1) or the (4,3,1) systems, since the probabilities of establishing connections for these systems in heavily loaded networks seem to be comparable.

6.6. Implementation issues

The mechanisms to support the dispersity schemes can be easily separated from most of the network and transport layer issues. The dispersity mechanisms are mostly end-to-end ones and may be implemented on the source and destination of the connection, on top of a realtime service provided by the transport layer of a realtime protocol suite. In the OSI model, we may place this functionality in the presentation layer, since it deals with issues such as encoding and decoding the data. For the Internet model, we may imagine implementing the dispersity mechanism as a library of functions which the user may load into his applications. The services to be provided by this library would include the control interface to set up fault tolerant channels, the coding and fragmentation needed at the source, and the reassembly and decoding required at the destination.

An interesting question is whether a user-level implementation can operate fast enough to satisfy the typical application's bandwidth requirements, since data may need to be copied several times in the process of fragmentation, coding and finally transfer to the kernel. These issues can only be completely solved by full implementation in the context of a particular operating system. However, we believe that implementation at the user level is feasible, since techniques such as shared memory can be used to minimize the number of times data needs to be copied, and the problems of efficient user-kernel interactions have been successfully dealt with in networking applications before. We will leave these very important problems to future work, and focus here on two subjects: the changes to the control interface to the network, which must be provided to make the task of implementing dispersity systems above the transport layer possible; and the interactions with the fault recovery process described in the earlier part of this dissertation.

6.6.1. Network interface

The network must provide a richer control of the routing process to the higher layers than is currently provided by the control interface offered by the realtime protocols of the Tenet Suite 1. The control interface of the transport layer we assumed in Chapter 4, based on the DCM approach, allowed routing and rerouting requests of realtime channels to be specified, but did not permit the specification of relationships between channels. For the dispersity systems, we need to be able to specify relationships such as disjointness and partial disjointness.

The simplest extension to the control interface, which we implemented in the simulator, includes the optional specification of a set of channels **C** and the variable S, in addition to the parameters already provided by the client, while making a channel establishment request. The dispersity system, establishing connections on behalf of the client, uses this interface to inform the establishment protocol of the set of channels for which the level of disjointness specified by S is desired. This is property of the channel is maintained by the channel management software (RCAP+DCM), as part of the parameters of the channel, so that if the channel needs to be rerouted in the future, the disjointness constraints need not be specified again. The network uses these parameters, and its knowledge of the routes traversed by the channels, to enforce the correct level of disjointness as described in Section 6.4. A more general interface, which allows the specification of an extendible set of relationships, is provided in (36) in the context of the Tenet Scheme 2.

6.6.2. Interactions with fault recovery

One of the factors mentioned in the experimental verification of the loss tolerance provided by the dispersity systems is that the network automatically recovers the failed sub-channels, restoring the connection to full tolerance. The sub-channel affected by the fault is subjected to the same recovery process as the other realtime channels. In all the experiments performed to simulate the effect of a network fault, this recovery process successfully rerouted the affected sub-channel. As we saw in Chapter 4, this process is very fast, so that, for the "core" network topology, the average recovery time is around 100 ms, and the worst case recovery time is around 400 ms. Thus, in most cases, at least in a properly dimensioned network running at reasonable load levels, the recovery will be completed rapidly, restoring the connection to full fault tolerance, so that a second fault occurring at a later time is handled correctly.

However, the recovery is not guaranteed, since, as we observed in Chapter 4, at high loads some of the affected channels may not be successfully rerouted. In addition, the sub-channel has constraints on rerouting, determined by the paths of the other channels and the value of S, These constraints are part of the parameters which the channel management software (RCAP+DCM) maintains, so that when the reroute is requested by RTCMP the correct routing action is taken by the routing module. However, the constraints make it harder to find a valid path. Thus, it is possible that the network will fail to find a path within the limits imposed by the desired recovery response time.

For a non-fault-tolerant realtime channel, it makes sense to stop the retry process after the desired recovery response time has elapsed. However, for the fault-tolerant connections, the service to the application is either not affected at all, or partially degraded, by the failure. Thus, if the recovery protocol described in Chapter 5 cannot find a reroute path, the fault tolerance scheme should, at its own level, continue to try to reroute the remaining sub-channel. This should be done at a low frequency to avoid imposing a high load on the establishment mechanisms of the network. Even if the recovery is achieved several seconds after the failure, it is useful, since the connection is restored to full fault tolerance, so that the service is not interrupted in the event of a subsequent failure.

The mechanisms to continue to generate routing requests for a failed sub-channel, in the event that the automatic network-initiated recovery process fails, should be implemented in the software of the dispersity system. The resources corresponding to the failed sub-channel should be released, to prevent deadlock. This should also reduce the expected waiting time till a channel can be rerouted, since these resources will allow some other waiting channel to succeed. Then, at a low frequency, so as not to impose a high load on the establishment process, the dispersity software should generate routing requests with the appropriate disjointness constraints. Eventually, when the realtime load on the network decreases, the sub-channel will be rerouted and the system restored to full fault tolerance. We take the position that, in a properly dimensioned network, it should rarely happen that the network fails to recover a connection, which then must be reestablished by the dispersity system. If such events were frequent, this would imply that the network was being operated close to its realtime capacity. In fact, in a network where the realtime resources available are limited, so that only few realtime connections can be established, fault-tolerant realtime channels should probably not be offered.

6.7. Conclusions

In this chapter, we have proposed dispersity routing as a mechanism to provide fault tolerance to realtime communication networks. We worked out the detail of the schemes by looking at compression, error correction, routing, and reassembly issues. We presented a framework that we used to classify the various schemes. The schemes discussed include dispersity systems with various levels of dispersity (N), various levels of redundancy (K), and various levels of strictness of the disjoint-path constraint (S), and hot/cold standby systems. We looked at some other existing approaches to providing fault-tolerant realtime communication, to compare benefits and costs. Finally, we presented a simulation model, which we used to study the performance of the dispersity systems and the capacity of the realtime network needed to support them.

The properties of some of the schemes are summarized in Tables 6.1 and 6.2. For the dispersity systems without redundancy (i.e., the $(N, N, 1)$ systems), the service provided in the absence of faults in the network is the same as that provided by a non-fault-tolerant realtime channel. The service provided in the event of a fault is communication with the same delay constraints, but a degraded bandwidth availability. A fraction of the

Property	$(N, N, 1)$	$(N, N-1, 1)$	$(N, K, 1) \mid K < N-1$
B/W Overhead	None	$1/(N-1)$	$\frac{N-K}{K}$
Impact on network capacity	Increased capacity	Slight decrease	More decrease
Level of fault tolerance	None	1 fault	$N-K$ faults
Level of error tolerance	None	1	$N-K$
Duration of disruption	Recovery time	No disruption	No disruption
Service during disruption	Lower B/W realtime	No disruption	No disruption
Routing constraints	Easy for small N, hard for large N	Easy for small N, hard for large N	Easy for small N, hard for large N
Encoding/decoding complexity	None	Low	High

Table 6.1: Properties of various dispersity systems

Property	$(N, N-1, 2)$	$(N, K, 2) \mid K < N-1$	**hot standby**	**cold standby**
B/W Overhead	$1/(N-1)$	$\frac{N-K}{K}$	1	0
Impact on network capacity	Small decrease	Small decrease	Large decrease	No impact
Level of fault tolerance	Partial	$N-K-1$ faults	1 fault	1 fault
Level of error tolerance	1	$N-K$	1	0
Duration of disruption	Recovery time	No disruption	No disruption	Notification time
Service during disruption	No service	No disruption	No disruption	No service
Routing constraints	Easy	Easy	Easy	Easy
Encoding/decoding complexity	Low	High	None	None

Table 6.2: Properties of some more dispersity systems

realtime packets are lost due to the fault, but the rest reach the destination before their deadlines. For the systems with disjoint paths ($S = 1$), the extent of the degradation is a $1/N$ decrease in capacity; for the systems with $S = 2$, the extent of degradation is either $1/N$ or $2/N$, depending on whether a shared or unshared link failed. These systems require limited buffering at the destination, bounded by Equation 6.2, to equalize delays on the paths. This buffer can be also used to remove network jitter from the message stream delivered to the application, assuming NTP synchronized clocks on the source and destination hosts. The effect of non-redundant dispersity on the capacity of the network is good, since the resource usage is spread out over the network. The number of connections that can be established is larger than for the non-fault-tolerant realtime channels, except when N approaches the degree of edge connectivity of the network. In this case, relaxing the disjointness constraint allows more such connections to be established, with a controlled decrease in the reliability provided. The service in this case can be modeled based on independence of probabilities of failure and error events on each link, allowing simple calculations of the expected fault tolerance to be used in deciding which dispersity system to use. On the negative side, these systems require a small increase in complexity to set up multiple sub-channels, support from the network to do the constrained routing of sub-channels, and reassembly support at the destination. In addition, these systems can only be used with realtime data that can be split into independent streams, so that an entire stream can be lost temporarily without rendering the other streams useless. Video compressed with intra-frame compression schemes such as JPEG satisfies this requirement. Block update techniques can also be used on top of these dispersity systems, since each block can be rendered independently of the others. Scalable media, such as hierarchical video, can also be used in conjunction with PET encoding to provide a system that degrades gracefully with failed sub-channels.

The dispersity systems with redundancy (i.e., $(N, K, 1)$ systems with $K < N$) can be used to provide completely undegraded service in the presence of restricted network faults. They also improve the tolerance of the connection to losses due to transmission errors. The redundancy introduces some overhead in terms of the extra bandwidth required from the network. However, the dispersity spreads this effect over a number of paths, so that the effect of a single sub-channel on the links it traverses can be made smaller than that of the original request. Thus, the effect of the increased overhead is partially compensated. The smaller requirements of the individual sub-channels also reduce external fragmentation on the links, allowing more connections to be established. However, for large values of N the connections are hard to establish because of the disjointness constraint. This can be relaxed by setting $S = 2$, thereby allowing more connections to be established. Since as many unshared links are used as possible, systems with

$S = 2$ may provide some of the fault tolerance properties of disjoint-path systems, in cases where disjoint paths cannot be found, but this tolerance is only applicable to the unshared links in the system. Further, extra redundancy ($N - K = 2$) can be used to improve the fault tolerance, allowing an $S = 2$ system to be fully tolerant to single faults. This gives the user a number of options in case the disjoint paths cannot be found. Dispersity systems with $K = N - 1$ can be implemented using parity codes, which allow deployment on low-end general purpose computers. Systems with $K < N - 1$ require more sophisticated encoding/decoding support.

All the systems described above can be considered hot standby systems. A canonical example of hot standby system is the $(2, 1, 1)$ system, where the same information is sent down to disjoint paths. Whichever stream reaches the destination is used for display. This system is very useful in networks with limited connectivity, since only two disjoint paths are required. However, the system uses twice the network resources that a non-fault-tolerant realtime channel would use. A cold standby scheme, which uses only one of the sub-channels, but reserves resources on the second sub-channel, so that in the event of a failure it can be sure of having the capacity, is less wasteful of network resources. It allows best-effort traffic to use the capacity of the second path in normal usage, but suffers disruption of service in the event of a fault, for a period of time determined by the time needed to inform the source and switch transmission to the standby channel. In WANs, this disruption may be substantial, since the delay-bandwidth product may be large. However, the duration of the disruption less than the time for a round trip on the channel, which must be small if the communication is interactive in nature. Most of the applications of realtime networks fall into this category.

Finally, in comparing the schemes described here with IFI realtime channels, we note that the IFI technique is usable in the highly regular wrapped hexagonal mesh topology of the HARTS network, but, in a general graph, the algorithm would not always find a valid set of links. In fact, in most networks IFI sets of links would not exist. The dispersity technique is much more general, since in most networks we can at least find two disjoint paths between two hosts. Of course, dispersity works better, in terms of resource usage, for larger N, but, if such a set of disjoint paths cannot be found, a number of alternatives exist. Secondly, the IFI scheme uses three times the network resources needed by the non-fault-tolerant realtime channel, while the dispersity systems offer a variety of services, with a corresponding variety of capacity overheads. To be fair, the IFI scheme is a cold standby scheme, so that the resources can be used by best-effort traffic until the fault occurs, but the resultant reduction in realtime capacity is significant. In addition, the scheme requires network support, to allow the nodes to change routing information automatically in response to the failure. It is not clear from (74) what the latency associated with this action might be. Most of the network support needed by the dispersity schemes is already provided by existing realtime protocols, such as DCM; the incremental support required is limited to adding to the network control interface the ability to specify the disjointness constraints.

The major advantage of the IFI channel is that it continues to provide realtime communication in the face of several failures. We would argue that such failures are extremely unlikely, except in the event of major natural disasters, and that in that event, most applications will have to stop transmitting to allow emergency communications to occur. For the vast bulk of applications, the tolerance provided by the dispersity systems is sufficient. For mission critical applications, something like IFI channels would be appropriate, though it would have to run in more general network environments than the HARTS network.

SFI channels are more general than IFI channels, and may be used with all topologies. However, the service provided is tolerance to single faults, and their the realtime resource usage of three times the requirements of a non-fault-tolerant realtime channel is excessive. Again, the technique is cold, so that the resources may be used in the absence of faults by non-realtime data, but it reduces a little too much the realtime capacity of the system. Dispersity systems provide tolerance to single faults at a fraction of the overhead, and a far larger range of useful levels of fault tolerance.

In conclusion, we have shown that dispersity routing is a very general method for improving the failure and loss tolerance of realtime channels. It can provide a very large variety of services: from transparent tolerance to graceful degradation of service; from instantaneous recovery to recovery within one round trip for message exchange; and from tolerance to single restricted failures[4] to complete tolerance to $N - K$ faults. The cost of the system, in terms of the network bandwidth needed, depends on the level of redundancy provided, but is ameliorated by the effect of spreading the resource usage among multiple paths, and by the reduced external fragmentation of the link capacity. Most of the network support required is already provided by realtime protocols such as those of the Tenet Suite. The additional support required, limited to supplementary routing constraints, has been described in Section 6.6. The remaining mechanisms of the fault-tolerant systems can be implemented at or above the transport layer.

[4] Restricted to failure of the unshared links in an $(N, N-1, 2)$ system.

Chapter 7: Conclusions

7.1. Introduction

This thesis started out to answer the question of how to improve the fault-handling characteristics of realtime networks. In this chapter we will summarize the arguments and results presented. Then, we will return to the original requirements spelled out in Chapter 1, to see if the mechanisms proposed in the thesis satisfy them. We will briefly present the main contributions of our work, and then close with a discussion of the issues open for future research.

7.2. Summary of the dissertation

In Chapter 1, we looked at the realtime channel paradigm, noted its promise in the light of the requirements of future broadband integrated-services networks, and felt the need to investigate mechanisms to deal with fault recovery and fault tolerance for realtime networks. Such mechanisms are beneficial, both to the client, who gets improved service, and to the network, which does not lose revenue due to lost traffic. We identified *proactive* mechanisms to be used if a disruption of service is not tolerable to the application and the cost of the extra resources required is a less important issue. *Reactive* schemes are useful for dealing with failures when the cost of providing redundancy is too high in a particular network or when the application can tolerate rare disruptions due to failures. We surveyed some work in designing and evaluating survivable network topologies. We looked at existing fault-recovery techniques, and noted that the assumptions, on which they are based, are not applicable to realtime networks. We also looked at fault-tolerance techniques, and surveyed some forward error correction and dispersity routing strategies that might be useful for realtime channels.

In Chapter 2, we described the Tenet Realtime Protocol Suite 1. We outlined the basic approach, as well as the specific control protocols RCAP and DCM. We also surveyed some other realtime schemes. Chapter 3 extracted the common principles behind a number of these schemes. These basic principles, which are fundamental to realtime networks, may be taken as a starting point for designing the fault-handling mechanisms, so that these mechanisms may be applicable to as wide a set of realtime networks as possible. Thereafter, we described a very general framework for the fault-recovery process. We considered a number of basic design alternatives, and made some fundamental decisions. A further set of choices were made by default, when we chose not to further explore the issues of the fault-detection process, the route update protocol, the channel management protocol, and the algorithm to select the min-hop path from a graph. Finally, we identified a number of issues to explore by simulation.

Chapter 4 described our simulation experiments, by which we made decisions about the issues we had chosen to explore. We also showed that, with the correct choices, schemes within the design space we had chosen to explore could recover a significant portion of the affected traffic within reasonable amounts of time. The schemes were also shown to be tolerant to changes in network load, network topology, traffic mix, and simultaneous failures of two links. In Chapter 5, we put the ideas together into a description of a protocol to reroute realtime channels in the event of a fault, which would work in the realtime network environment provided by the Tenet Protocol Suite 1, including the DCM control protocol.

Finally, in Chapter 6, we examined the problem of fault tolerance, and explored a specific proactive mechanism, namely, dispersity routing. We worked out the details of a scheme to use dispersity routing to provide fault-tolerant realtime communication, simulated the service provided, and explored the effect on network capacity. We concluded that dispersity routing could provide a wide variety of fault-tolerant realtime services when implemented on top of a basic realtime service. Even when redundancy was used to improve the fault tolerance and loss tolerance of the dispersity systems, the network costs were found to be lower than anticipated for many of the schemes, because the effect was spread out over a number of paths in the network. Implementation at the higher levels of the protocol stack is possible if adequate network support is provided. The necessary network support was also described in Chapter 6.

7.3. Requirements re-examined

We started out in Chapter 1 with a set of requirements for the control mechanisms of a realtime network. The control mechanisms were required to:

- Enhance the ability to deal with failure conditions
- Use resources efficiently in bandwidth-poor networks
- Provide fast control in high-bandwidth wide-area (high delay-bandwidth product) networks
- Provide a range of services: transparent fault tolerance for applications which need it, low disruption times for others
- Work efficiently in the common cases, but deal gracefully with the unlikely occurrences
- Be applicable to a variety of networks, under a variety of conditions

Some of these requirements may conflict. For example, the goal of providing efficient use of resources in a bandwidth-poor environment might lead to increased latency of response, which might be unacceptable in high delay-bandwidth networks. Thus, a range of mechanisms are needed. We looked at a bipartite classification of network control mechanisms, proactive and reactive. The proactive schemes have resources ready to handle worst-case scenarios. Thus, they react faster, but are more wasteful of resources. Reactive schemes involve a latency for reaction, but only use resources when needed, so they are more efficient. Thus, depending on the application requirements, and the resource availability in the network, the correct set of mechanisms may be chosen to satisfy the needed requirements. In the next section, we will describe the mechanisms that were proposed in this thesis, and evaluate the extent to which they satisfy these requirements.

7.4. Contributions

We described two sets of mechanisms in this thesis. The fault-recovery mechanisms developed in the first half of the thesis are designed to satisfy the requirements of efficient operation in the common fault-free case, but restore as many realtime channels as possible in the event of a failure. The fault-tolerance schemes provide a higher level of reliability at increased cost. We will see to what extent they meet the requirements described in the previous section.

The network recovery mechanisms described in Chapters 3, 4, and 5 meet the requirement of efficient use of resources. In the common fault-free case, the network uses no extra resources, as compared to the control mechanisms without fault-handling capability. In the event of failure, the simulation studies performed show that the reroute process uses minimally more resources than would be required in the best possible case (i.e., if each channel were to be routed in an empty network). Furthermore, in the networks of the scale considered, the reaction is fairly rapid, since the average reroute is completed within 100-200 ms depending on the network topology. The mechanisms also deal well with two simultaneous failures, different network topologies, load conditions, and traffic mixes. Only one level of fault-recovery service is provided, since all channels are subjected to the same recovery process.

Thus, the first, second and fifth requirements are satisfied. The requirement of speed is also partially satisfied, since the recovery is fast enough for many applications in the topologies tested, but may not be adequate for wide-area networks, where the latency for reaction would be higher. Of course, the recovery process itself is not realtime, since there is no guarantee that it will succeed at all.

The average recovery time is dominated by the network transmission and propagation times. Thus, the average recovery time would grow approximately proportional to the round-trip time on the realtime channels. Since most of the applications requiring realtime services require reasonably *low* delay bounds, the fault-recovery mechanisms will also perform adequately on any network where a such a service can be offered.

The fault-tolerance mechanisms described in Chapter 6 span a wide range of efficiencies for network resource usage. The systems without redundancy actually increase the network capacity, compared to the basic realtime channels. They also provide graceful degradation of service in the face of faults in the network. They work with a number of loss-tolerant compression schemes, such as JPEG and block compression, as well as with hierarchical compression schemes in conjunction with PET. The systems with redundancy add bandwidth overhead in return for increased fault tolerance. The cold standby schemes allow the spare capacity to be used by best-effort traffic, but suffer some latency to switch to the backup channel when the fault occurs. Thus, the set of mechanisms described allow us to provide a range of services at varying levels of efficiency for resource usage. The service offered is insensitive to the load on the network, since the proactive reservation provides service guarantees. The schemes are valid in any network topology with a high degree of edge connectivity.

Thus, the dispersity schemes satisfy all the requirements. Even the efficient resource use requirement is met, to various extents, by the different systems; the systems without redundancy meet it fully. Together with the recovery protocol, they provide a powerful way of improving the fault tolerance of realtime networks, offering a variety of levels of improvement in service, at different costs in terms of the reduced capacity of the network.

In addition to the actual schemes presented, the thesis also provides a very useful framework to evaluate ideas for the fault recovery of realtime channels. The ideas presented are applicable to any network that satisfies the assumptions of Section 3.3. These are the minimum possible set of assumptions about the realtime network. Further, we attempted to provide convincing arguments to justify the portion of the solution space for recovery mechanisms that we explored in detail. We hope to have convinced the reader that all interesting solutions to the problem should lie within the framework we provided. Thus, our framework can be used to compare and evaluate any further ideas of fault recovery for realtime networks that might be developed in the future. In particular, the load index we developed is useful, since it allows comparisons of a multi-dimensional quantity, the amount of traffic, in one dimension. While this load index is restricted to RCSP scheduling networks, the basic ideas may be applied to other bandwidth reservation and priority schemes, and the load indices developed may also be used for diverse applications such as routing and charging.

Finally, the chapter on fault tolerance presented a comparison of the existing approaches to fault tolerance for realtime channels with the realtime dispersity systems. In this process, we spelled out a set of criteria, which may be used to evaluate and compare other fault tolerance mechanisms for realtime channels, when they will be developed.

7.5. Future work

One very important task left to be accomplished is the implementation of the ideas proposed in this thesis in a realtime networking environment. Arguing about the validity of our ideas from first principles is important, in that we can be more certain that the ideas are worth the time and effort of implementation. However, no amount of simulation or analysis can prove a concept as effectively as implementation.

Unfortunately, the time is not yet mature for these ideas to be tested in an actual implementation. The assumption of our schemes is that a realtime network with high degree of edge-connectivity exists. So far, the Tenet protocols have only been implemented in testbeds, where the topology is too simple to try these ideas out. The only existing DCM implementation does not yet support route modification. No other realtime network implementations with the richness and functionality of the Tenet Suite exist as yet. Thus, the all important last step, implementation of the RTCMP protocol, must wait until the realtime environment is ready to support it. Given the rate at which realtime networking research and deployment is being conducted, this time should not be very far in

-- --

the future.

Many questions have been raised, elsewhere in this thesis, and left unanswered. Some of them are implementation-related, such as that about whether an implementation of the fault tolerance schemes in user space can be made sufficiently efficient. One may also ask how the recovery and tolerance model fits into the Tenet Suite 2 interface. Others are more basic, such as that about whether multiple priorities for failure recovery can be provided within the recovery model. Can network security be included into the model without major changes to the interfaces proposed? These questions, and many others, are left open for future research.

-- --

Bibliography

(1) A. Albanese, J. Bloemer, J. Edmonds and M. Luby, 'Priority Encoding Transmission', *35th Annual Symposium on Foundations of Computer Science*, 1994.

(2) 'FDDI Station Management Standard', ANSI X3T9.5 Revision 5.1, American National Standards Institute, September 1989.

(3) D. P. Anderson, R. G. Herrtwich and C. Schaefer, 'SRP: A Resource Reservation Protocol for Guaranteed Performance Communication in Internet', Tech. Rpt.-90-006, International Computer Science Institute, Berkeley, California, February 1990.

(4) M. Antonellini and L. Sebastiani, 'Error Rates: A Convenient Technique for Trigerring Fault Management Procedures', *Proceedings of the IFIP TC 6/WG 6.6 Symposium on Integrated Network Management*, Boston, MA, May 1989.

(5) E. Ayanoglu, C. I, R. D. Gitlin and J. E. Mazo, 'Diversity coding: Using error control for self healing in communication networks', *Proceedings of INFOCOM'90*, San Francisco, California, June 1990, 95-104.

(6) E. Ayanoglu, C. I, R.D.Gitlin and I. Bar-David, 'Analog diversity coding to provide transparent self-healing communication networks', *Proceedings of GLOBECOM'90*, San Diego, California, Dec 1990, 683-688.

(7) A. Banerjea and B. Mah, 'The Real-Time Channel Administration Protocol', *Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'91)*, Heidelberg, Germany, November 1991, 160-170.

(8) A. Banerjea, E. W. Knightly, F. L. Templin and H. Zhang, 'Experiments with the Tenet Real-Time Protocol Suite on the Sequoia 2000 Wide Area Network', *Proceedings of ACM Multimedia '94*, San Francisco, October 1994, 183-192.

also available as Tech. Rpt.-94-020, International Computer Science Institute, Berkeley, CA, April 1994.

(9) A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma and H. Zhang, 'The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences', Tech. Rpt.-94-059, International Computer Science Institute, Berkeley, California, November 1994.

(10) M. Barezzani, E. Pedrinelli and M. Gerla, 'Protection Planning in Transmission Networks', *Proceedings of SUPERCOMM / International Conference on Communications '92*, Chicago, Illinois, June 1992.

(11) 'SONET Add-Drop Multiplex Equipment Generic Criteria for a Unidirectional Path Protection Switched Self-Healing Ring Implementation', Bellcore Technical Advisory TA-000496, Bell Communications Research, August 1990.

(12) A. Bellary and K. Mizushima, 'Intelligent Transport Network Survivability: Study of Distributed and Centralized Control Techniques using Dept. of Comp. Sci. and ADMs', *Proceedings of Globecom'90*, San Diego, California, Dec 1990, 1264-1268.

(13) R. Brown, 'Calendar Queues: A fast O(1) Priority Queue Implementation for the Simulation Event Set Problem', *Communications of the ACM 31*, 10 (October 1988), 1220-1227.

(14) R. H. Cardwell, C. L. Monma and T. Wu, 'Computer-aided Design Procedures for Survivable Fiber Optic Networks', *IEEE Journal Selected Areas in Communication 7* (1989).

(15) V. P. Chaudhary, K. R. Krishnan and C. D. Pack, 'Implementing Dynamic Routing in the Local Telephone Networks of USA', *Proceedings of ITC-13*, Copenhagen, Denmark, June 1991.

(16) C. Cheng, 'A Loop-Free Extended Bellman Ford Routing Protocol without

Bouncing Effect', *ACM Computer Communication Review 19*, 4 (1989), 224-236.

(17) S. Chiou and V. O. K. Li, 'Diversity Transmissions in a Communication network with Unreliable Components', *Proceedings of ICC'87*, Seattle, Washington, June 1987, 968-973.

(18) S. Chiou and V. O. K. Li, 'An Optimal Two-copy Routing Scheme in a Communication Network', *Proceedings of INFOCOM'88*, New Orleans, Louisiana, April 1988, 288-197.

(19) I. Cidon, I. Gopal and R. Guerin, 'Bandwidth Management and Congestion Control in PlaNET', *IEEE Communications Magazine*, October 1991, 54-64.

(20) D. Clark, S. Shenker and L. Zhang, 'Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism', *Proceedings of ACM SIGCOMM'92*, Baltimore, Maryland, August 1992, 14-26.

(21) B. A. Coan, W. E. Leland, M. P. Vecchi, A. Weinrib and L. T. Wu, 'Using Distributed Topology Update and Preplanned Configuration to Achieve Trunk Network Survivability', *IEEE Transactions on Reliability 49*, 4 (October 1991).

(22) C. J. Colbourn and L. D. Nel, 'Using and Abusing Bounds for Network Reliability', *Proceedings of Globecom'90*, San Diego, California, Dec 1990, 663-668.

(23) G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.

(24) E. W. Dijkstra, 'A Note on Two Problems in Connection with Graphs', *Numerical Mathematics 1* (1959), 269-271.

(25) R. Doverspike, 'A Multi-Layered Model for Survivability in Intra-LATA Transport Networks', *Proceedings of IEEE GLOBECOM '91*, Phoenix, Arizona, December 1991.

(26) D. Ferrari, `Client Requirements for Real-Time Communication Services', *IEEE Communications Magazine 28*, 11 (1990).

(27) D. Ferrari, `Real-Time Communication in Packet-Switching Wide-Area Networks', Tech. Rpt.-89-022, International Computer Science Institute, Berkeley, California, May 1989.

(28) D. Ferrari and D. Verma, `A Scheme for Real-Time Channel Establishment in Wide-Area Networks', Tech. Rpt.-89-036, International Computer Science Institute, Berkeley, California, May 1989.

(29) D. Ferrari, `Real-time Communication in an Internetwork', *Journal of High Speed Networks 1*, 1 (January 1992), 79-103.

(30) D. Ferrari, A. Banerjea and H. Zhang, `Network Support for Multimedia - A Discussion of the Tenet Approach', Tech. Rpt.-92-072, International Computer Science Institute, Berkeley, CA, October 1992.

(31) D. Ferrari, `Distributed Delay Jitter Control in Packet-Switching Internetworks', *Journal of Internetworking: Research and Experience 4*, 1 (1993), 1-20.

(32) D. Ferrari, A. Banerjea and H. Zhang, `Network Support for Multimedia - A Discussion of the Tenet Approach', *Computer Networks and ISDN Systems 26*, 10 (July 1994), 1267-80.

(33) S. Floyd, `Issues in Flexible Resource Management for Datagram Networks', *Proceedings of the 3rd Workshop on Very High Speed Networks*, Maryland, March 1992.

(34) W. D. Grover, `The Self-Healing Network: A Fast Distributed Restoration Technique for Networks using Digital Cross-connect Machines', *Proceedings of IEEE Global Telecommunications Conference*, December 1987, 28.2.1-28.2.6.

(35) R. Guerin, H. Ahmadi and M. Naghshineh, `Equivalent Capacity and Its Application to Bandwidth Allocation in High-Speed Networks', *IEEE Journal on*

Selected Areas in Communications 9, 7 (September 1991), 968-981.

(36) A. Gupta and M. Moran, `Channel Groups: A Unifying abstraction for specifying inter-stream relationships', Tech. Rpt.-93-015, International Computer Science Institute, Berkeley, California, March 1993.

(37) R. G. Herrtwich, Personal communication, November 1992.

(38) J. Hyman and A. Lazar, `MARS: The Magnet II Real-Time Scheduling algorithm', *Proceedings of ACM SIGCOMM'91 Conference*, Zurich, Switzerland, September 1991, 285-293.

(39) M. Kalfane, Personal communication, October 1994.

(40) H. Komine, T. Chuja, T. Ogura, K. Miyazaki and T. Soejima, `A Distributed Restoration Algorithm for Multiple Link and Node Failures of Transport Networks', *Proceedings of IEEE Global telecommunications Conference*, December 1990, 403.4.1-403.4.5.

(41) K. R. Krishnan and T. J. Ott, `Forward Looking Routing: A New State Dependent Routing Scheme', *Proceedings of ITC-12*, Torino, Italy, June 1988.

(42) F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*, North-Holland, New York, 1977.

(43) B. Mah, *Design of a Realtime Channel Administration Protocol*, University of California at Berkeley, 1993. Masters Thesis.

(44) N. F. Maxemchuk, `Dispersity Routing', *Proceedings of ICC'75*, San Francisco, California, June 1975, 41.10-41.13.

(45) S. McCanne, Personal communication, October 1994.

(46) J. McQuillan, `The New Routing Algorithm for the ARPANET', *IEEE Transactions on Communications COM-28* (May 1980).

(47) K. Menger, `Zur allgemeinen Kurventheorie', in *Fundamenta Mathematicae*, 1927, 96-115.

(48) D. Messerschmitt, Asynchronous video coding, Infonet meeting at University of California, Berkeley, 15 November 1994.

(49) P. Pancha and M. E. Zarki, 'MPEG Coding for Variable Bit Rate Video Transmission', *IEEE Communications 32*, 5 (May 1994), 54-66.

(50) A. K. J. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*, Massachusetts Institute of Technology, February 1992. PhD Thesis.

(51) C. Parris, H. Zhang and D. Ferrari, Dynamic Management of Guaranteed Performance Multimedia Connections, to appear in ACM Journal of Multimedia Systems, April 1993.

(52) C. Parris and D. Ferrari, 'A Dynamic Connection Management Scheme for Guaranteed Performance Services in Packet-Switching Integrated Services Networks', Tech. Rpt.-93-005, International Computer Science Institute, Berkeley, California, January 1993.

(53) C. Parris, *Dynamic Channel Management*, University of California at Berkeley. PhD Thesis.

(54) S. Rai and S. Soh, 'A Computer Approach for Reliability Evaluation of Telecommunication Networks with Heterogeneous Links', *IEEE Transactions on Reliability 49*, 4 (October 1991).

(55) T. Rodeheffer and M. D. Schroeder, Automatic Reconfiguration in Autonet, Lecture at University of California, Berkeley, September 1992.

(56) H. Sakauchi, Y. Nishimura and S. Hasegawa, 'A Self-healing Network with an Economic Spare-Channel Assignment', *Proceedings of IEEE Global Telecommunications Conference*, December 1990, 403.1.1-403.1.6.

(57) M. Schroeder, A. Birell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite and C. Thacker, 'Autonet: A High-Speed Self-Configuring Local

Area Network Using Point-To-Point Links', *IEEE Journal Selected Areas in Communication 9*, 8 (October 1991), 1318-1335.

(58) M. Schroeder, Circuit Management in AN2, Lecture at University of California, Berkeley, October 1992.

(59) K. G. Shin, 'HARTS - A Distributed Real-time Architecture', *Computer 24*, 5 (May 1991), 25-35.

(60) M. Stoer, *Design of Survivable Networks*, Springer-Verlag, 1991.

(61) R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983.

(62) D. Taubman and A. Zakhor, 'Highly Scalable, Low Delay Video Compression', *IEEE International Conference on Image Processing*, November 1994.

(63) C. Vogt, R. Herrtwich and R. Nagaragan, 'HeiRAT - The Heidelberg Resource and Administration Technique: Design Philosophy and Goals', IBM Technical Report No. 43.9243 , IBM ENC, Heidelberg, Germany, 1992.

(64) C. H. Yang and S. Hasegawa, 'FITNESS: Failure Immunization Technology for Network Service Survivability', *Proceedings of IEEE Global Telecommunications Conference*, November/December 1988, 47.3.1-47.3.6.

(65) M. Yoshida and H. Okazaki, 'New Planning Architecture for Reliable and Cost-effective Network Design', *Proceedings of ITC-12*, Torino, Italy, June 1988.

(66) M. Yoshida and H. Okazaki, 'Cooperative Control over Logical and Physical Networks for Multiservice Environments', *IEICE Transactions E.74*, 12 (December 1991).

(67) W. T. Zaumen and J. J. Garcia-Luna-Aceves, 'Dynamics of Distributed Shortest-Path Routing Algorithms', *Computer Communications Review 21*, 4 (September 1991), ACM Press.

(68) L. Zhang, *A New Architecture for Packet Switched Network Protocols,* Massachusetts Institute of Technology, July 1989. PhD Thesis.

(69) H. Zhang and T. Fisher, 'Preliminary Measurement of RMTP/RTIP', *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'92)*, San Diego, CA, November 1992.

(70) L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, 'RSVP: A New Resource Reservation Protocol', *IEEE Communications Magazine 31*, 9 (September 1993), 8-18.

(71) H. Zhang and D. Ferrari, 'Rate-Controlled Static Priority Queueing', *Proceedings of IEEE INFOCOM'93*, San Francisco, California, April 1993, 227-236.

(72) H. Zhang, *Service Disciplines for Integrated Services Packet-Switching Networks*, University of California at Berkeley, November 1993. PhD Thesis, Tech. Rpt.-UCB/CSD-94-788.

(73) Q. Zheng and K. G. Shin, 'Fault-tolerant real-time communication in distributed computing systems', *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, Boston, MA, July 1992.

(74) Q. Zheng and K. G. Shin, Establishment of Isolated Failure Immune Real-Time Channels in HARTS, to appear in *IEEE Transactions on Parallel and Distributed Systems.*