INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

FAST LOW-COST FAILURE RECOVERY FOR REAL-TIME COMMUNICATION IN MULTI-HOP NETWORKS

by

Seungjae Han

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1998

Doctoral Committee:
Professor Kang G. Shin, Chair
Associate Professor Farnam Jahanian
Assistant Professor Sugih Jamin
Associate Professor Atul Prakash
Assistant Professor Kimberly Wasserman

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 9840551

Copyright 1998 by
Han, Seungjae

All rights reserved.

UMI Microform 9840551
Copyright 1998, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized
copying under Title 17, United States Code.

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

© Seungjae Han 1998
All Rights Reserved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To my parents and Sunju.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGEMENTS

I am indebted to many individuals for their continuous support in completing this thesis. First of all, I would like to express my deep gratitude to my advisor, Professor Kang G. Shin, for his guidance. He gave me opportunities to freely explore research ideas, and encouraged me when I had doubts. I would also like to thank the members of my dissertation committee, Farnam Jahanian, Atul Prakash, Sugih Jamin, and Kimberly Wasserman, for their constructive comments and suggestions.

Thanks also go to my officemates and friends who have enriched my life in Ann Arbor. I have enormously benefited from the interaction and collaboration with Harold Rosenberg, Atri Indiresan, Ashish Mehra, Hagbae Kim, Jaehyun Park, Tarek Abdelzaher, Emmanuel Propeta, Charles Meissner, John Reumann, Seok-ku Kweon, Sunghyun Choi, Sung-whan Moon, and many others. Special thanks to Harold Rosenberg for his contribution to the early stage of my research.

I fondly remember the help and kindness of Beverly J. Monaghan, the laboratory administrative assistant. I also gratefully acknowledge the financial support during the course of my ph.D. program by the Korean government, the Office of Naval Research, and the National Science Foundation.

Finally, a special appreciation should be given to my parents for their love and understanding. There is no way to replace their role to bring me this far in life. My wife, Sunju, is another person who deserves my deepest gratitude. She has been not only an inspiring supporter but also a discussion partner throughout the preparation of this thesis.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTERS

1 INTRODUCTION 1
- 1.1 Background 1
- 1.2 Dependable Real-time Communication 3
- 1.3 Existing Approaches 5
  - 1.3.1 Multi-Computer Networks 5
  - 1.3.2 Wide-Area Data Networks 7
  - 1.3.3 Telephone Networks 9
  - 1.3.4 Comparison with Our Approach 11
- 1.4 The Proposed Approach 12
  - 1.4.1 Design Goals 12
  - 1.4.2 An Overview of the Proposed Approach 13
  - 1.4.3 An Illustrative Example 17
- 1.5 Organization of the Thesis 19

2 DEPENDABLE CONNECTION ESTABLISHMENT 21
- 2.1 Spare-Resource Reservation 21
  - 2.1.1 Deterministic Multiplexing 22
  - 2.1.2 Probabilistic Multiplexing 24
  - 2.1.3 Scalability & Complexity Issue 28
- 2.2 Dependability QoS Negotiation 29
  - 2.2.1 A Dependability QoS Parameter, $\mathcal{P}_r$ 30
  - 2.2.2 Calculation of $\mathcal{P}_r$ 32
  - 2.2.3 QoS Negotiation Procedure 34
  - 2.2.4 The Number and Disjointness of Backup Channels 36
- 2.3 Backup-Route Selection 36
  - 2.3.1 Optimal Routing Problem 36
  - 2.3.2 Initial Route Selection 38
  - 2.3.3 Periodic Route Reconfiguration 40

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.4 Evaluation 42
2.4.1 Simulation Setup 42
2.4.2 Measurement of Spare Resource Overhead 43
2.4.3 Comparison of Routing Heuristics 47
2.5 Summary and Conclusion 50

3 CHANNEL FAILURE DETECTION 54
3.1 Channel Failure Detection Schemes 54
3.1.1 Channel Failure 54
3.1.2 Neighbor Detection Scheme 56
3.1.3 End-to-End Detection Scheme 56
3.2 Fault-Injection Tool Set Development 57
3.2.1 Fault-Injection in Distributed Real-Time Systems 57
3.2.2 Organization of DOCTOR 58
3.2.3 Software-implemented Fault Injection 60
3.2.4 Non-intrusive Data Monitoring 62
3.3 Fault-Injection Experiment Setup 64
3.3.1 Testbed 64
3.3.2 Experiment Goal 66
3.3.3 Experiment Specification 68
3.3.4 Fault-Injection Experiment Sequence 70
3.4 Analysis of Experimental Results 71
3.4.1 Fault Manifestations 71
3.4.2 Failure Detection Coverage 74
3.4.3 Failure Detection Latency 77
3.4.4 Workload Dependency 78
3.5 Summary and Conclusion 78

4 RUN-TIME FAILURE RECOVERY 80
4.1 Connection Restoration Procedure 80
4.1.1 Failure Reporting and Channel Switching 82
4.1.2 Priority-based Backup Activation 84
4.1.3 Recovery from Multiplexing Failures 85
4.2 Resource Reconfiguration Procedure 85
4.2.1 Channel Closure or Repair 86
4.2.2 QoS Maintenance 87
4.2.3 QoS Degrade & Upgrade 88
4.3 Bounded-Time Failure Recovery 90
4.3.1 The RCC Network 90
4.3.2 RCC Message Delay 92
4.3.3 Failure-Recovery Delay Bound 93
4.4 Dependability QoS Measurement 94
4.4.1 Fault-Tolerance Level of Various Backup Configurations . . 94
4.4.2 Per-Connection QoS Management 96
4.4.3 QoS Support for Heterogeneous Connections 98
4.4.4 Comparison with Brute-Force Multiplexing 100
4.4.5 Graceful QoS Degradation 100
4.5 Summary and Conclusion 105

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5 ADAPTIVE RESOURCE MANAGEMENT 106
5.1 Elastic QoS Control 106
5.1.1 Range-QoS Model 107
5.1.2 Excess-Resource Allocation 108
5.2 Network-Triggered Performance QoS Adaptation 110
5.2.1 Run-time QoS Adaptation 111
5.2.2 Evaluation 112
5.2.3 Discussion 115
5.3 Application-Triggered Performance QoS Adaptation 115
5.3.1 Run-time QoS Re-negotiation 117
5.3.2 Evaluation 118
5.3.3 Discussion 120
5.4 Conclusion 121

6 CONCLUSIONS AND FUTURE WORK 122
6.1 Research Contributions 122
6.2 Future Work 124

BIBLIOGRAPHY 127

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF TABLES

Table

1.1 Comparison of existing approaches with our approach 11
3.1 Fault manifestations (Experiment-1) 72
3.2 Fault manifestations (Experiment-2) 73
3.3 Detection coverage and latency of the neighbor scheme (Experiment-1) . . . 75
3.4 Detection coverage and latency of the neighbor scheme (Experiment-2) . . . 76
4.1 Cases requiring resource reconfiguration 87
4.2 R_{fast} with deterministic multiplexing 94
4.3 R_{fast} with probabilistic multiplexing 95
4.4 R_{fast} with mixed multiplexing degrees 99
4.5 R_{fast} comparison with brute-force multiplexing 101
4.6 R_{fast} comparison with brute-force multiplexing in case of hot spots 102
5.1 Statistics of two segmentations 119
5.2 QoS re-negotiation results 120

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF FIGURES

Figure

1.1 Example of a SFI channel 6
1.2 Three rerouting strategies 10
1.3 Overview of self-healing failure recovery. 14
1.4 Failure recovery by reactive rerouting 17
1.5 Failure recovery by the proposed scheme 18
2.1 Deterministic multiplexing algorithms 23
2.2 Probabilistic multiplexing algorithm 26
2.3 Data structures for reducing the algorithm complexity 28
2.4 Example Markov models to derive $R(t)$ 30
2.5 $R(t)$ of a $\mathcal{D}$-connection with a single backup 31
2.6 The effect of non-disjoint routing on $R(t)$ 35
2.7 Boundary routing 40
2.8 The iterative optimization method 41
2.9 Simulation networks 43
2.10 Average spare-bandwidth reservation under deterministic multiplexing . . . 44
2.11 Average spare-bandwidth reservation under probabilistic multiplexing . . . 46
2.12 Case 1 (network load = 30%, backup load = 36%) 48
2.13 Case 2 (network load = 30%, backup load = 36%) 49
2.14 Case 3 (network load = 30%, backup load = 36.2%) 50
2.15 Case 4 (network load = 25%, backup load = 30%) 51
2.16 A counter example 52
3.1 Two nodes connected by dual simplex links 56
3.2 The organization of DOCTOR 59
3.3 Architecture of HMON 63
3.4 Configuration of the experimental platform 64
3.5 The protocol stack in NP 66
3.6 Real-time message passing at run-time 67
3.7 Two implementations for heartbeat transmission 68
3.8 Failure-detection latency 69
3.9 An example of false alarm 74
3.10 An example of channel failures undetected by the neighbor scheme 74
3.11 An example of early failure detection 77
3.12 Comparison of detection latency distribution 78

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.1 Channel state transition 81
4.2 Three channel-switching schemes 83
4.3 Unsuccessful channel repair 87
4.4 Three options of QoS degradation 89
4.5 The RCC message format 91
4.6 Message loss during failure recovery 92
4.7 Distribution of QoS differences 97
4.8 QoS maintenance in the under-loaded network 103
4.9 Graceful QoS degradation in the over-loaded network. 104
5.1 Example QoS/utility specification 108
5.2 Algorithms for excess-resource allocation at a link 109
5.3 Excess-resource allocation at a under-utilized link 110
5.4 QoS-update procedure 111
5.5 Comparison of excess-resource allocation policies 113
5.6 A case when the local-max policy flops 114
5.7 The impact of spare resource reservation on network utilization 114
5.8 Utility function change for QoS-upgrade re-negotiation 116
6.1 General protocol configuration 125
6.2 Fault-tolerant multicast using backup channels 126

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 1

INTRODUCTION

1.1 Background

Over the past decade, advances in the transmission-medium technology and the widespread deployment of powerful-yet-inexpensive computers have enabled a remarkable progress in computer networking. Initially, computer networks were designed with a different goal from telephone networks. While the motivation of telephone networks was human-to-human communication, that of computer networks was building a distributed computer system, in which resource sharing is possible irrespective of the physical location of resources like processors, disks, and peripheral devices. Distributed systems can also provide high reliability and cost-effectiveness as compared to their counter part, tightly-coupled systems. However, the growing trend of interconnecting computers by high-speed links (e.g., optical fibers) has introduced many new classes of applications over computer networks, including telephone services. On the other hand, telephone networks have also evolved towards accommodating computer-generated digital data. While ISDN (Integrated Services Digital Network) is an evidence of such efforts, the most drastic change at the telephone network side is the adoption of the packet-switching paradigm in place of the traditional circuit-switching paradigm, e.g., ATM (Asynchronous Transfer Mode). As a result, the distinction between computer and telephone networks in term of both target applications and underlying technologies is disappearing fast, and all communication services are being merged into a single network infrastructure, called *Integrated Services Network*. The cable television network is another candidate for network service integration.

A consensus in this convergence of network services is the necessity of supporting continuous media applications, i.e., real-time audio and video. The real-time transfer of continuous media has traditionally been achieved by circuit switching in telephony services or

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

by broadcasting over shared media in television services. In packet-switched computer networks, continuous media applications need a special care since the end-to-end packet delay and throughput of a media stream are inherently non-deterministic. Such end-to-end performance characteristics which are necessary to achieve the required functionality of these applications are often called *Quality-of-Service* (QoS). Today's representative computer network, Internet, also lacks QoS support for continuous media applications: the window-based flow control is unsuitable for traffic with end-to-end timing constraints. Nevertheless, many multimedia applications have already begun to run over Internet using such protocols as RTP [82], XTP [103], and IP multicast. However, these protocols do not meet the true multimedia requirements because they only support a best-effort service model. Though they are flexible to be integrated with non-IP-based protocols which can provide QoS guarantees, they themselves cannot guarantee timely packet delivery. The Next-Generation Internet (NGI) is expected to provide new services that meet the diverse QoS requirements of various emerging applications, and/or enhance the QoS support for existing applications.

In parallel with the growing demands for real-time communication services, recent years have seen considerable research efforts in developing various performance QoS-guaranteed (or real-time communication) service paradigms. Several survey papers [4, 107, 28, 97] discuss many of real-time communication schemes from various perspectives. Here, we only briefly discuss the basic idea behind the real-time communication schemes for multi-hop networks (i.e., point-to-point networks). Most (if not all) real-time communication schemes share three common properties: *QoS-contracted, connection-oriented*, and *reservation-based*. A contract between a client and the network is established before messages are actually transferred. To this end, the client must first specify its input traffic behavior and required QoS. Then, the network computes the resource needs (e.g., link & CPU bandwidths, and buffer space) from this information, selects a path, and reserves necessary resources along the path. If there are not enough resources to meet the QoS requirement, the client's request is rejected. The client's messages are transported only via the selected path with the resources reserved, and this virtual circuit is often called a *real-time channel*.

Unlike traditional datagram services in which average performance is of prime interest, guaranteeing QoS is the key requirement of real-time communication services. Numerous QoS-guaranteed service models have been developed ranging from the static CBR[1] service that resembles the telephony service to the 'controlled-load service' that mimics the best-effort service in unloaded networks [101]. In-between, there are VBR[2] deterministically-

[1]Constant Bit Rate
[2]Variable Bit Rate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

guaranteed services [85] (also called *hard* real-time communication) and VBR statistically-guaranteed services (also called *soft* real-time communication). Well-known hard real-time communication schemes include [29. 34. 19. 25. 74. 106]. and examples of soft real-time communication schemes are [37, 63. 19. 108. 51]. To cope with large time-scale burstiness. renegotiation-based schemes have also been proposed [76. 35. 109]. While all of the QoS-guaranteed schemes rely on some form of resource reservation and admission control. each differs in QoS parameters and/or the firmness of QoS guarantees. (Network utilization can be enhanced by relaxing the firmness of QoS guarantees.) There also exist feedback-based schemes [52. 84. 68. 24] which attempt to provide QoS support without resource reservation.

Despite the abundance of QoS communication schemes. very few implementations are available in the current Internet. mainly because resource reservation is not currently supported. IPv6 [23] and RSVP [111] are expected to alleviate this limitation. Unlike Internet. ATM networks are capable of various types of QoS communication services (i.e.. CBR. VBR. and ABR[3]) in addition to best-effort services (i.e.. UBR[4]).

1.2 Dependable Real-time Communication

Primitive real-time communication services will soon be available for such multimedia applications as Internet phone. WWW. digital libraries. etc. On the other hand. the increase of network connectivity and link capacity will expand the application domain of real-time communication to include business- or mission-critical applications. which require a different type of QoS support. That is, there will be a growing need for 'dependable' real-time communication services for such applications as remote medical service, collaborative scientific research. business net-meeting. real-time electronic commerce. or even remote battle-field command/control. Such critical applications require *both* dependable and timely communication services. Typical performance QoS includes message throughput. end-to-end message delay. and delay jitter. An example of dependability QoS is the connection availability defined as the probability of a connection being available at any given time. A guaranteed level of fault-tolerance is essential to these applications. Suppose. for example, there is a very important video conference and unanticipated network failures disconnect one or more participants from the conference for an unpredictably long period. This may lead to a failure or delay in reaching important strategic decisions. which can cause a significant economic loss.

[3]Available Bit Rate
[4]Unspecified Bit Rate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Network failures can cause even a large-scale disaster. Catastrophic social consequences of network failures have actually been witnessed in recent breakdowns of the US telecommunication network. For instance, the fire at an unmanned tall office in Illinois caused 3.5 million telephone calls to be blocked in 1988. Emergency 911 calls went unanswered, on-line business transactions were stopped, and flights were delayed because of air traffic controller failures. Hospital operations were affected and drug stores could not process prescriptions. Even banks had to be closed for security reasons due to disabled alarm systems [69]. In 1990's, several similar disasters have been reported for various reasons like a disrupture of a fiber cable during construction, earthquakes, outage of switching systems, or network overloading. The public telephone network usually provides a very high availability, better than 99.999% on average, thanks to various dependability techniques such as hardware duplication, standby power, dynamic rerouting, overload monitoring, reliable software, and resource distribution [59]. However, the consequence of improper failure handling could be devastating, thus making network reliability a major concern.

As for dependability, the current Internet with datagram services has successfully dealt with two types of network failures: *transient* and *persistent* failures. A typical example of transient network failures is temporary packet losses due to either network congestion or data corruption. Persistent failures include breakdown or crash of network components. Transport protocols like TCP can handle transient loss of packets by acknowledgment and retransmission, and the connection-less IP protocol deals with persistent failures by routing packets around the faulty network components. However, with some exceptions [26], retransmission is unlikely to be useful for real-time communication, because there is usually not enough time to detect and retransmit a lost real-time message before its deadline expires. Instead, for real-time communication, forward-error-correction (FEC) techniques should be used if no data loss is acceptable. The main drawback of FEC is its high overhead. We also face a serious difficulty in tolerating persistent failures for real-time communication, because a QoS guarantee is usually realized by reserving resources on a fixed path and transporting real-time messages only via the path. Hence, a real-time message, unlike datagram messages, cannot be detoured around faulty components on the fly.

The prevalence of optical fibers affects network dependability in two different ways. First, the probability of transmission errors in optical links is negligible; the error rate is dropped from 10^{-5} per bit in the 56 kbps links of initial ARPANET to below 10^{-10} per bit in optical links [92]. The chance of packet loss due to transmission errors is very low, and most packet losses are attributed to congestion control mechanisms. Therefore, for

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

real-time communication, tolerating transient-failures has become relatively less important because congestion-induced packet losses are avoided by resource reservation. Furthermore, occasional loss of messages is tolerable in many real-time applications, such as video conferencing. By contrast, the deployment of optical fibers exacerbates the difficulty in tolerating persistent-failure, because more connections will be running through each large-capacity link, and thus, even a single link failure can result in loss of a large number of connections. Unless the network is carefully designed to handle the huge amount of traffic lost due to failures, the increase of link capacity will seriously threaten the network dependability. Not only link failures but also node failures are getting more difficult to deal with. Usually, switching nodes of computer networks (i.e., routers) are not designed to meet such a stringent reliability goal as telephone network switches (e.g., AT&T 5ESS [94]). Higher-performance routers in future networks will become even harder to provide high reliability, due mainly to their complex software. Moreover, computer networks are more vulnerable to vandalism like virus or hacking than the telephone network which has a 'closed' architecture.

Considering the criticality of network dependability and the increasing threat of network failures, the development of effective mechanisms to cope with network failures is a *must* in future integrated service networks. This thesis focuses on how to effectively tolerate persistent failures for dependable real-time communication in multi-hop packet-switched networks.

1.3 Existing Approaches

In this section, we summarize the previous work on dependable real-time communication in multi-hop networks. Particularly, notable work in multi-computer networks, wide-area data networks, and telephone networks, is reviewed with comparative discussions against our approach.

1.3.1 Multi-Computer Networks

Dependable multi-computer systems have been used in mission- or life-critical real-time control applications such as transportation vehicles, military systems, life-support systems, or plant controllers. The traditional tightly-coupled multi-computer approach such as STAR [6], FTMP [47], CM* [88], SIFT [33], MAFT [60], FTP [64], and AIPS [65] has been dominant in such application areas. To meet the stringent reliability requirement of critical control applications, a fully-connected communication architecture with an excessive

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

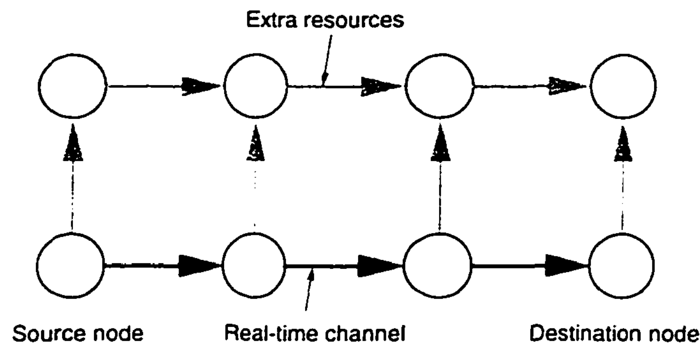


Figure 1.1: Example of a SFI channel

capacity has usually been employed. Though this approach provides a very high level of dependability. the property of the underlying communication architecture limits system scalability and flexibility. Also. developing proprietary hardware/software instead of using off-the-shelf products results in high costs that many applications cannot afford.

Meanwhile. distributed systems have emerged as a promising candidate for a new way of building real-time control computers. due mainly to their high potential for dependability and cost-effectiveness. Distributed systems are also suited for those applications which require physically dispersed environments. A successful paradigm in this approach is the broadcast-network-based architecture. some examples of which include MARS [61]. Delta-4 XPA [11]. Springnet [90].Cyclone [66]. and AAS [22]. This paradigm capitalizes on the simplification of the underlying communication structure and protocols. sacrificing scalability. flexibility and reliability to some extent. Some research results with regard to dependable real-time communication in broadcast networks can be found in [21. 2. 62. 15].

A more general approach toward the open-system architecture is to use point-to-point packet-switched networks with regular or arbitrary topologies. However. most work on this approach. such as Delta-4 [78]. ISIS [13]. and Consul [71]. has considered only fault-tolerance issues without taking real-time constraints into account. One of the hardest problems in the construction of a real-time system in a point-to-point network is timely and reliable message delivery. Correct and timely delivery of messages generated by real-time tasks is crucial. because failure of timely message delivery could cause tasks to miss their deadlines.

There have been roughly two types of approaches to achieving dependable real-time communication in point-to-point multi-computer networks. The first type is the multi-copy approach in which multiple copies of a message are sent simultaneously via disjoint paths [79. 56]. This method attempts to achieve both timely and reliable delivery at the same time. Thus. by transmitting multiple copies of a message over different paths. the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

chance that at least one copy is delivered within its deadline increases and the effects of possible failures are masked. This approach has an advantage that failures are handled without service disruption. but it is very expensive and timely message delivery is not guaranteed. since messages are delivered in a best-effort manner.

The second approach is to set up a single-path real-time channel using such real-time communication schemes as [55. 114]. When a real-time channel is disconnected due to component failures. the channel is recovered by establishing a new channel. If a temporary connection disruption during failure recovery is acceptable to the underlying applications. this is a cost-effective alternative to the multi-copy method. Actually. temporary message losses are tolerable in many real-time control applications because of the 'system inertia' characterized by the control system deadline [86]. The scheme proposed in [112] took this approach. It is called the SFI (Single Failure Immune) method because it provides guaranteed failure recovery under a single failure model. In the SFI method. additional resources are reserved in the vicinity of each real-time channel. and the failed components are detoured by altering the channel path using the reserved resources. Figure 1.1 illustrates the setup of a SFI channel. In [113]. the SFI method is extended to survive special patterns of multiple failures in a hexagonal mesh topology. The resource reserved for fault-tolerance are not utilized for real-time traffic in the absence of failures. (They can be used by best-effort traffic. though.)

Similar to the SFI method. our approach reserves some resources for failure recovery. but the amount of required extra resources is much smaller than the SFI method. In addition, our approach is not locked to a certain deterministic failure model. so that the connection dependability can be flexibly controlled according to the application criticality and environmental factors.

1.3.2 Wide-Area Data Networks

In packet-switched datagram networks. routers (or gateways) are mostly responsible for failure recovery functions such as isolating faulty components and selecting alternative routes. For instance, in Internet. current operational status is continuously exchanged between neighbor networks (i.e., by Exterior Gateway Protocols) or internal gateways (i.e., by Interior Gateway Protocols) for immediate isolation of failures [20, 91]. Since packets can traverse any routes, intermediate gateways between the source and destination of a connection can easily reroute its packets to a different path when they detect failures on the current path. Sometimes, the source host needs to be involved in failure recovery in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

response to the reports received from ICMP (Internet Control Message Protocol). While the packet redirection procedure of Internet is not applicable to real-time communication, its detection scheme of faulty components and rejoin scheme of repaired components may be useful.

The simplest way of recovering a real-time channel from a component failure is to establish a new real-time channel which does not include the failed component. This *reactive* method is studied in [10]. In the context of the Tenet approach [9], this scheme relies on the broadcast of all component failures to the entire network, so that all hosts can maintain a consistent view on the current network topology. When a source node detects the failure of its channel from this broadcast, it tries to establish a new channel to replace the disabled channel. Since no consideration is given *a priori* for the purpose of fault-tolerance, this method causes no fault-tolerance overhead in the absence of failures. However, it does not give any guarantee on failure recovery. The channel re-establishment attempt can fail due to resource shortage at that particular time. Even when there are sufficient resources, the contention among simultaneous recovery attempts for different faulty connections may require several trials to succeed, thus delaying service resumption and increasing network traffic. To regulate simultaneous recovery attempts, random delays can be introduced before starting each recovery operation.

By contrast, in our approach, a backup channel is established before failures actually do occur, so one can use it immediately upon occurrence of a failure to the original channel, without the time-consuming channel (re)establishment process. The time required to establish a real-time channel is relatively large and unbounded even without contention, since channel-establishment messages are usually sent as datagrams and non-trivial calculation is necessary at each node on the channel path for the admission test. In addition, since each backup channel is equipped with dedicated spare resources, simultaneous failure recovery attempts do not cause conflicts.[5] Thus, the failure-recovery delay of our approach is much smaller than that of the reactive method, and hence, '*fast recovery*' is possible.

In [8], a framework is presented to classify fault-tolerant real-time communication schemes using three factors: dispersity, redundancy (cold or hot), and disjointness. This framework is general enough to characterize most schemes including our approach, but cannot capture details to accurately address the pros and cons of each scheme. A more tangible contribution of [8] is the formulation of a forward error correction approach based on dispersity routing. The innovative aspect of this approach lies in combining error-coding techniques

[5]This is not exactly true when our resource-sharing technique is used. This issue will be detailed later.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with multiple-copy transmissions. This allows for a tradeoff between resource overhead and fault-tolerance capability. Its main shortcoming is the additional resource consumption for FEC, which is undesirable to many real-time applications that can tolerate temporary data losses. A similar idea has been employed to reduce the frequency and overhead of retransmission for loss-free message delivery, in which the FEC technique is combined with the retransmission technique [12].

1.3.3 Telephone Networks

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

Essentially, when a telephone connection is broken, the connection is rerouted by detouring the failure point. Failure recovery should be fast, so that people (or applications) may hardly perceive the service disruption. Even more important is assuring the success of failure recovery itself. If there are not enough resources available for rerouting all disrupted connections, some of the connections should be dropped. To avoid resource shortage during failure recovery, '*spare resources*' are reserved in advance. The allocation of spare resources is an important issue in the network design, and is closely related to problem of rerouting failed connections. For the selection of rerouting paths, there are three strategies: *local-rerouting*, *end-to-end rerouting*, and *local-to-end rerouting*. Each of these strategies is illustrated in Figure 1.2.

The local-rerouting strategy, also called a *span-restoration* method, has usually been

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

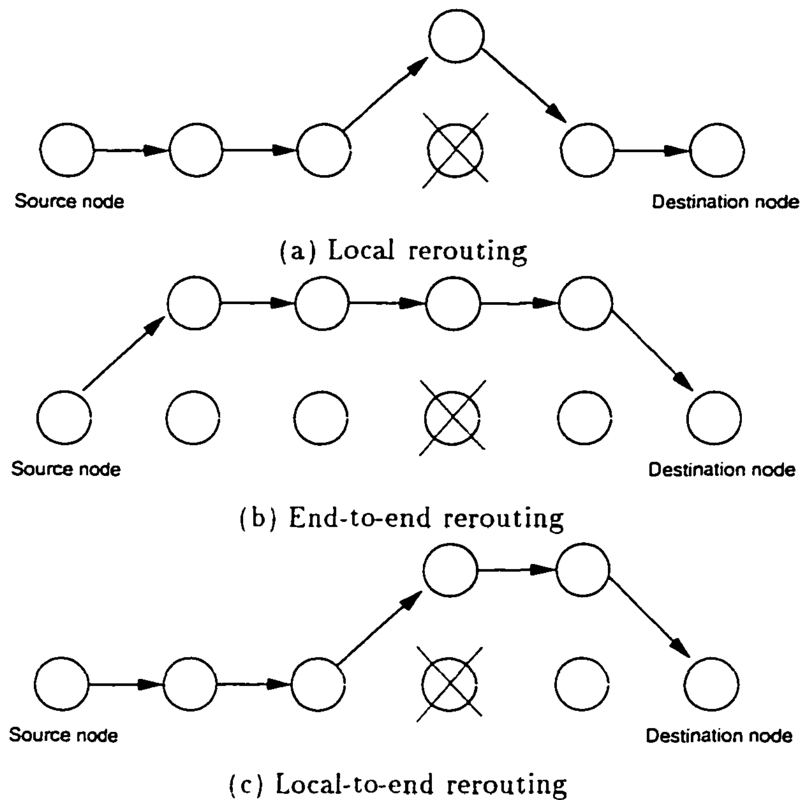


Figure 1.2: Three rerouting strategies

used in STM (Synchronous Transfer Mode) networks. In most research on this strategy [36, 104, 81, 7, 96, 46], the 'maximum flow' model is used to find the (semi-) optimal placement of spare-resources under a deterministic failure hypothesis — typically, a single-link failure model. A drawback of the local-rerouting approach is that the resource usage becomes inefficient after failure recovery, because channel path-lengths are usually extended by local detouring. Thus, its operation is the simplest among three rerouting strategies, but it suffers the lowest resource efficiency. According to [3], end-to-end rerouting is the best with regard to resource efficiency, and the local-to-end rerouting is the second, in mesh networks. The simulation with real telephone network topologies also reports similar trends [102]. The impact of topological characteristics on the performance of rerouting strategies is discussed in [73].

In the *end-to-end rerouting* strategy, also called a *path-restoration* method, a new connection is established between two end-points of each failed connection. There are two further variations in this strategy, depending on whether the failure recovery paths are pre-computed or not. In the former, the pre-computed recovery paths should be disjoint

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

Table 1.1: Comparison of existing approaches with our approach

with the paths of corresponding original connections, while in the latter the recovery paths can use the components of their original connections. From the viewpoint of spare-resource reservation, in the former, each recovery connection (or *backup* connection) reserves its own spare resources, so that there will be no conflict/contention between recovery attempts. In the latter (e.g., [50]), spare resources are shared and recovery paths are not determined until failures actually do occur. When failures occur, each faulty connection will establish a new connection by "claiming" the reserved spare resources. Some connections may need to attempt several recovery paths before they succeed.

The pre-computed backup-path approach has been studied mainly in the context of ATM networks. Some of recent efforts on this approach can be found in [32, 58, 3, 72, 102, 48]. Essentially, they derive optimal routing of channels (i.e., VPs) to minimize the spare-resource reservation while guaranteeing successful recovery under a deterministic failure model. They assume that all channel demands are known at the time of network design and change very rarely.[6]

1.3.4 Comparison with Our Approach

In Table 1.1, existing approaches are compared with our approach in terms of resource overhead, recovery delay, and recovery guarantee. The SFI method is similar to the span-restoration method in many aspects, except that it involves higher overhead, because, in the SFI method, the entire spare resources required by each connection is reserved under the worst-case assumption unlike the span-restoration method which optimizes the allocation of spare resources. Both methods will have a shorter recovery delay than end-to-end rerouting methods, because failures are handled locally without intervention of end-nodes.

[6]Each call setup is handled at the VC level without requiring a new VP to be set up.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The path-restoration method comes closet to our approach in that backup paths are established in advance for end-to-end rerouting, but there exist four main differences between the two. The first difference of this approach from ours is that they are unable to control the fault-tolerance level of each connection, and all connections are treated equally under the same failure model. We allow per-connection fault-tolerance control, so that more critical connections will get higher levels of fault-tolerance. Secondly, the path-restoration method assume that a fixed traffic demand (i.e., VP setup requests) is given beforehand and remains unchanged, while our scheme does not require global knowledge about all connections in the network. In the path-restoration method, all channel paths and spare resources are simultaneously determined, and hence, addition or removal of a channel requires recalculation of all channel paths and spare resources, which is computationally very expensive. Therefore, it cannot be applied to an environment where short-lived channels are set up and torn down frequently. Thirdly, though the control of recovery procedures might be distributed, centralized, or a hybrid of the two, the calculation/assignment of spare resources is centralized in the path-restoration method. In our scheme, we separate the spare resource allocation problem from the channel routing problem, so that (i) backup path may be selected by any algorithm and (ii) spare resource allocation may be done in a distributed manner. Finally, we provide an *integrated* solution to the problem of failure detection, channel switching, resource reconfiguration, and control-message transmission, which is not specific to a particular type of networks. For example, our behavior-based failure detection schemes are independent of the underlying physical media or protocols, in contrast to the special failure detection techniques capitalizing on physical layer characteristics as [3, 89]. Our control-message transmission mechanism is also applicable without relying on special mechanisms provided by a particular network as in [30].

1.4 The Proposed Approach

This section outlines our approach. We first describe the design goals and then introduce our solution to meet the goals.

1.4.1 Design Goals

To design a fault-tolerant service, one must first define the model of failures to be tolerated. We assume that (infrequent) transient packet losses are acceptable to the target applications, or are dealt with by other techniques like FEC. This thesis focuses on how to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

effectively handle "persistent" or "permanent" failures, e.g., *crash failures*. There are five goals that drive the design of our scheme:

- **Per-connection dependability guarantee:** Each connection can request a different level of fault-tolerance depending on its criticality. A successful recovery is guaranteed as long as the number and type of failures occurred do not exceed the fault-tolerance capability of a connection.
- **Fast (time-bounded) failure recovery:** The service-disruption time of a connection caused by failures should be acceptably short, and may be bounded if certain conditions are met.
- **Small fault-tolerance overhead:** The amount of the additional resource overhead required for fast/guaranteed recovery should be acceptably small.
- **Robust failure handling:** Failures should always be handled robustly even though failure occurrences may exceed the assumed failure hypothesis. By 'robustly,' we mean that the QoS of nonfaulty real-time channels are not affected, and as many faulty real-time channels as possible are recovered.
- **Interoperability/scalability:** The failure recovery scheme must be interoperable with existing and future real-time channel protocols, so that it can be used in a wide-area network equipped with various (heterogeneous) protocols. Also, it should scale well in a dynamic environment where short-lived connections are setup and torn down frequently.

1.4.2 An Overview of the Proposed Approach

Two of our main concerns are to reduce and bound the service disruption time caused by failures and to minimize the fault-tolerance overhead. To quickly restore real-time channels from failures, we set up *backup channels* in advance along with each *primary channel*; that is, each dependable real-time ($\mathcal{D}$-) connection consists of one primary channel and one or more backup channels. Upon failure of a primary channel, one of its backups is promoted to a new primary channel. Since a backup channel is established before failures, the network can use it immediately upon occurrence of a failure to the original channel, without the time-consuming channel (re)establishment process. To minimize the resource overhead for maintaining backup channels, resources for backups are cleverly shared. The dependability

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

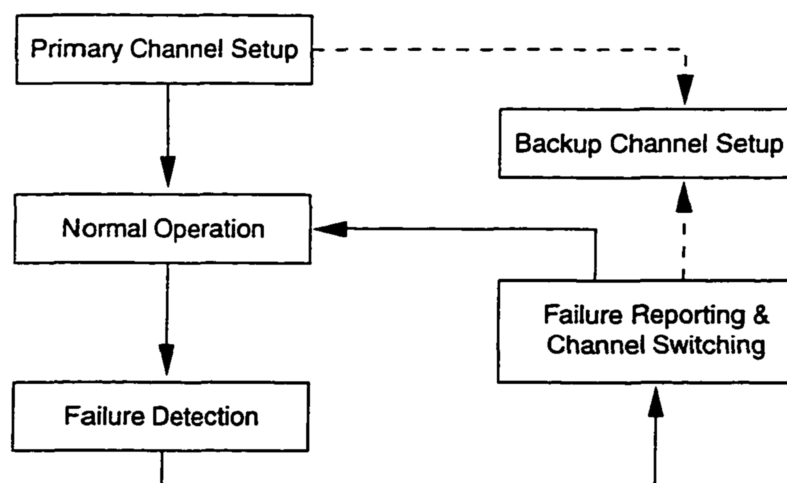


Figure 1.3: Overview of self-healing failure recovery.

of each connection can be flexibly chosen to reflect the application criticality by controlling the amount of spare resources reserved for each backup.

Figure 1.3 gives an overview of our failure-recovery scenario. The key steps in our approach are: (i) backup channel establishment, (ii) failure detection, (iii) failure reporting and channel switching, (iv) resource reconfiguration. All of these functions are performed by the (end and intermediate) nodes of each injured $\mathcal{D}$-connection in a distributed manner, thus the name 'self-healing' failure recovery.

The first step is to set up backup channels. A backup channel remains as a cold-standby and does not carry any data until it is activated, so that it does not consume resources in a failure-free situation.[7] However, a backup channel is not free, as it requires the same amount of resources as its primary channel to be reserved, in order to provide the same QoS as its primary upon its activation. We call the resources reserved for backup channels "*spare resources*". In a normal situation, spare resources can be used by non-real-time traffic, but they cannot be used to accommodate other real-time channels. It is because if a real-time channel is established by using spare resources which are reserved for other backups, its QoS guarantee may be violated when spare resources are claimed for failure recovery. As a result, equipping each $\mathcal{D}$-connection with a single backup routed disjointly with its primary reduces the network capacity by 50% or more (because the backup is likely to run over a longer path). Thus, raw backup channels are too expensive.

To cope with this problem, we have developed a resource-sharing method, called *backup multiplexing*, in which resources are shared among backup channels in such a way that the

[7]Only primary channels transfer actual messages.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

dependability of multiplexed backups is not compromised. Essentially, backup multiplexing reduces the amount of spare resources reservation by overbooking the same resources for multiple backups. To this end, instead of reserving the resources for each backup on its path individually, we determine the amount of total spare resources on a hop-by-hop basis by considering the relation among all backup channels on each hop. A heuristic is to allow resource sharing among those backups which are unlikely to be activated simultaneously. Different connections can be made to have different dependability by adjusting the parameters of backup multiplexing.

Under backup multiplexing, how to route backups has a significant impact on the amount of spare resources. We found that traditional routing algorithms such as minimum-hop routing or maximum load-balanced routing are less effective in backup route selection, compared to the routing methods which capitalize on the characteristics of spare-resource calculation.

During the backup establishment, we consider two dependability QoS parameters (P_r, Γ), where P_r is the probability of fast failure recovery and Γ is the estimated failure-recovery delay. In other words, with a certain rate of component failures, the probability that a $\mathcal{D}$-connection will suffer a disruption of real-time communication service longer than Γ is not greater than P_r. In making QoS contract between the network and client,[8] Γ is not negotiable while P_r is.

The service-disruption time of a $\mathcal{D}$-connection can be bounded, if at least one of its backup channels is available upon failure of its primary channel. When a healthy backup is available, Γ is the sum of failure-detection delay, failure-reporting delay, and backup-activation delay. As the fast failure recovery depends on the availability of backup channels, P_r increases with the number of backups set up for the $\mathcal{D}$-connection. For example, P_r of a $\mathcal{D}$-connection with a single backup and double backups are:

$$\begin{aligned}
P_r^1 &= P(\text{primary not fail}) + P(\text{primary fails} \cap \text{backup not fail}), \\
P_r^2 &= P(\text{primary not fail}) + P(\text{primary fails} \cap \text{first-backup not fail}) \\
&\quad + P(\text{primary fails} \cap \text{first-backup fails} \cap \text{second-backup not fail}).
\end{aligned}$$

Backup multiplexing also affects P_r. It reduces the amount of spare resources by reserving less resources than the summation of resources needed by individual backups, but creates a possibility of spare resource exhaustion. Thus, some backups cannot be activated because other activated backups have already taken all spare resources. In such a case, a

[8]This process is called QoS negotiation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

"*multiplexing failure*" is said to occur. When we account for the probability that a backup suffers a multiplexing failure, P_r of a $\mathcal{D}$-connection with a single backup becomes:

$$\begin{aligned} P_r^1 &= P(\text{primary not fail}) + \\ &\quad P(\text{primary fails} \cap \text{backup not fail } \cap \text{backup not suffer a multiplexing failure}). \end{aligned}$$

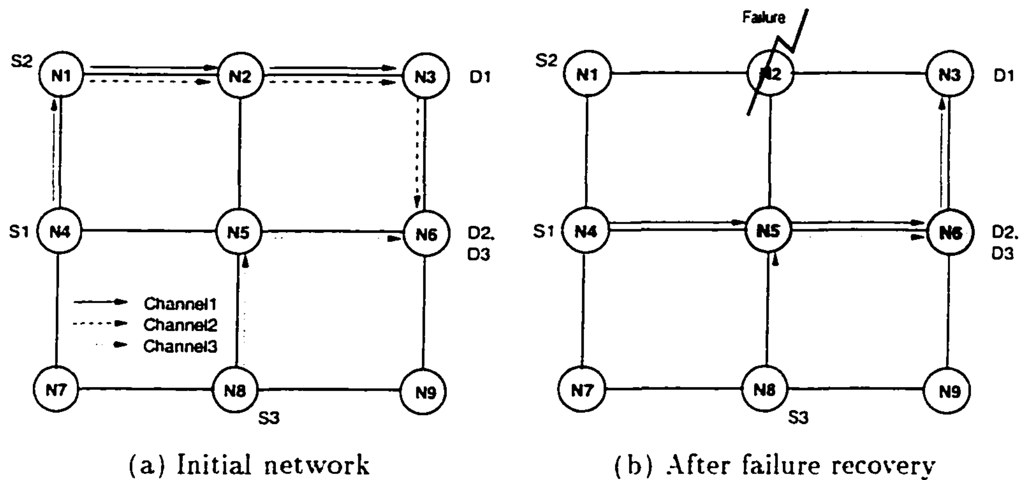
Since P(primary fails) and P(backup not fail) are functions of the component failure rate which is usually very low, these terms have relatively small impacts on P_r as compared to the multiplexing failure probability. Using this P_r calculation procedure, the number of backups and the multiplexing parameter are decided to meet the client's dependability QoS requirement.

So far, we have described the backup establishment procedure which is executed before a failure occurs. Now, we will explain the failure handling procedure after a failure occurs. The first step in handling a failure is its detection. Failure detection is essentially to discover anomalies in real-time channels, i.e., persistent losses of real-time messages. Applications can specify the failure semantic for each connection. The coverage and latency of failure detection methods are very important, because they directly affect the dependability QoS parameters, P_r and Γ, respectively. We have developed two behavior-level failure detection methods: a hop-by-hop (or neighbor) detection method and an end-to-end detection method. The effectiveness of these methods is empirically evaluated through fault-injection experiments on a laboratory testbed.

Once a failure is detected, the detected failure should be reported to the end nodes of the inflicted channels, which are responsible for the rest of failure handling. If the disabled channel is a primary channel, one of its healthy backups is activated to become the new primary. Such operations as failure reporting and backup activation should be done quickly, and, at the same time, these operations must be robust enough to insulate healthy connections from the recovery process for failed connections. To this end, we devised a special mechanism for the control message transmission, which enables timely and robust delivery of failure-report messages and backup-activation messages.

The next step of channel switching is resource reconfiguration. Thus, after failed primary channels are replaced by their healthy backups, the faulty primary channels will be torn down and new backup channels will be established to preserve the dependability QoS of the corresponding $\mathcal{D}$-connections. When the network suffers resource shortage due to coincident failures or slow failure recovery and can not establish new backups, the connection dependability is 'gracefully' degraded. When failed components are repaired and there

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Initial network (b) After failure recovery

Figure 1.4: Failure recovery by reactive rerouting

exist connections with degraded dependability, resource reconfiguration is performed e.g., migrating or establishing backups over new routes.

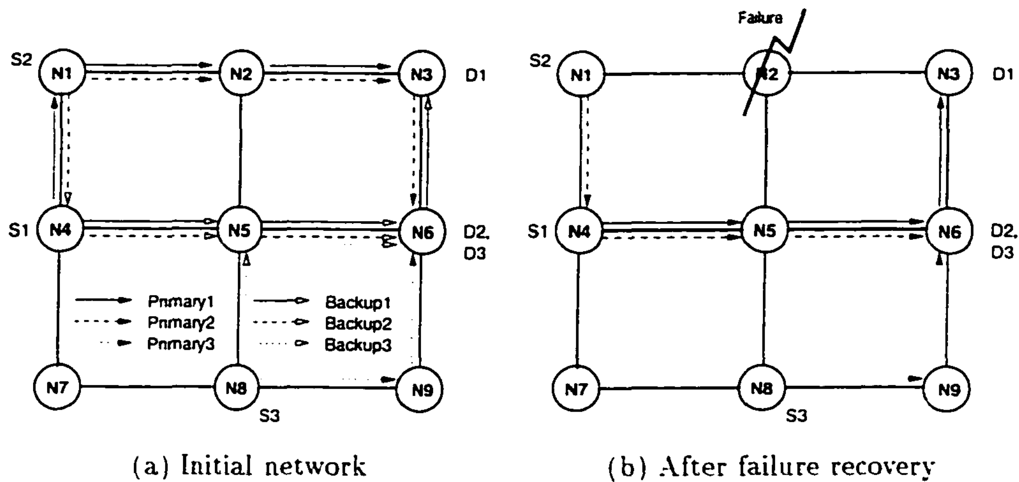
We have also developed an elastic QoS-control scheme to lower (or eliminate) the dependability cost associated with backups (i.e., the reduction of network's ability to accept future connection requests). In this new QoS control scheme, we combined our failure-recovery scheme with two adaptive QoS-control methods: network-triggered and application-triggered QoS adaption. Essentially, spare resources are utilized by active channels (i.e., primary channels) in a normal situation, so that active channels can utilize the entire network resources. This is carefully done so as not to unpredictably compromise existing connections' dependability QoS. The elastic QoS-control scheme enables "seamless" utilization of spare resources for both performance QoS and dependability QoS, so that the network can operate without incurring any dependability cost in a failure-free situation, while being able to *predictably* respond to failures.

1.4.3 An Illustrative Example

Here, we illustrate the benefit of our approach over the reactive method with an example.[9] In the reactive method, no backup path is pre-assigned and no spare resource is reserved in advance. When a component failure disables a real-time channel, a new channel will be established from scratch before resuming the service. Obviously, under this method, the application will experience an extended recovery delay during the establishment of a

[9]One can call our approach a *proactive* method.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Initial network

(b) After failure recovery

Figure 1.5: Failure recovery by the proposed scheme

new channel. Even without accounting the recovery delay, the reactive method cannot make any guarantee on successful failure recovery, because there may not exist proper detours for the failed channels. Figure 1.4 illustrates such a situation.

Figure 1.4 (a) shows a network which contains three real-time channels. Suppose two network nodes are connected by two simplex links, each of which can accommodate up to two channels. When node N2 fails, channels 1 and 2 need to be detoured around N2. Both channels may need to use shortest possible paths in order to maximize the chance of meeting their timeliness QoS requirements. As a result, the resource needs on the link from N5 to N6 exceed its capacity, and the link can accommodate only one of them, say channel 1, as shown in Figure 1.4 (b). Now, channel 2 has to be rerouted over a longer path. If channel 2's QoS requirement (i.e., end-to-end message delay) is too tight to fit the longer path, channel 2 cannot be recovered from N2's failure. An option is moving channel 3 to a different path in order to accommodate channel 2 at the link from N5 to N6. However, this is not a good idea, since moving an existing channel can cause domino effects without guaranteeing successful rerouting of the affected channels. A better solution is not to set up channel 3 over the link from N5 to N6 in the original network.

Figure 1.5 illustrates how the same failure in Figure 1.4 is handled in our scheme. Note the difference between the initial channel setups. In Figure 1.5 (a), primary-1 and -2 are routed over the same paths as in Figure 1.4 (a), but primary-3 is routed over N9 instead of N5. It is because, when primary-3 is established, backup-1 and -2 has already been established on the link from N5 to N6 with reserving all resources on that link. However, unlike primary-3, backup-3 can be routed over that link by multiplexed with backup-1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and backup-2. This backup multiplexing is possible since primary-3 does not share any component with primary-1 or -2. As a result of backup setups, the failure of N2 is tolerated without causing any connection teardown. In this example, we assumed that channels are established in the ascending order of their indices, using a shortest-path routing method.

1.5 Organization of the Thesis

Chapter 2 describes various issues in backup channel establishment. The mechanism of backup multiplexing is presented first. Then, dependability QoS parameters with which clients can express their fault-tolerance requirements are defined and their calculation procedure is presented. The problem of backup route selection is also dealt with in this chapter. A two-step routing method is presented, which can achieve both quick responses to connection establishment requests and optimal resource usage at the same time. Finally, the simulation results are presented to demonstrate the efficacy of the backup multiplexing and backup routing mechanisms.

Chapter 3 deals with the first step of run-time failure recovery, i.e., failure detection. Two behavior-based failure detection methods are presented. The performance of these detection methods are experimentally evaluated on a laboratory testbed which implements a real-time communication protocol developed in RTCL.[10] Experimental data gathered from the fault-injection experiments are analyzed and their implications are elaborated.

Chapter 4 explores the failure-handling procedure after a failure detection. Steps like failure reporting, backup channel activation, channel switching, resource reconfiguration after recovery, and graceful QoS degradation are described. A bound of failure-recovery delay is derived under a special mechanism developed for robust and timely delivery of the control messages associated with time-critical recovery operations. This chapter also investigates the fault-tolerance levels achievable by various backup configurations, and checks if the actual fault-tolerance level that each connection receives matches the negotiated dependability QoS.

Chapter 5 presents an adaptive QoS management scheme, the goal of which is to eliminate the fault-tolerance overhead in a failure-free situation. First, a network-triggered QoS adaptation scheme is described, which allocates spare resources to active channels for higher performance QoS. Resource allocation is adjusted to the network load condition. Second, an application-triggered QoS adaptation scheme is described, which allows applications to

[10] Real-Time Computing Laboratory in the University of Michigan

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

request QoS re-negotiation at run time. The use of spare resources to assure successful renegotiation attempts is presented.

Chapter 6 concludes this thesis by summarizing its contributions and suggesting possible avenues of future research. Earlier work on this thesis has been published in [41, 42, 40, 45, 43].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 2

DEPENDABLE CONNECTION ESTABLISHMENT

A dependable real-time ($\mathcal{D}$-) connection requires to set up a primary and one or more backup real-time channels. Assuming that the procedure for primary-channel establishment is exported from the underlying real-time channel protocol, we focus on the establishment of backup channels in this chapter. To establish backups for a $\mathcal{D}$-connection, dependability QoS should be negotiated between the network and clients, just as performance QoS is negotiated for primary-channel establishment. The selection of backup paths and the reservation of spare resources are key steps of backup establishment. Although the dependability QoS parameters may need to be described first, we begin this chapter by presenting our backup multiplexing schemes because the concept of backup multiplexing is essential to the entire connection establishment procedure.

This chapter consists of five sections. Section 2.1 describes two backup multiplexing schemes for efficient spare resource allocation. Section 2.2 introduces our dependability QoS parameters and presents their derivation process. Section 2.3 deals with the issue of backup-route selection. Section 2.4 presents simulation results, demonstrating the superior performance of the proposed scheme. The chapter concludes with Section 2.5.

2.1 Spare-Resource Reservation

The links/nodes used by a primary channel may preferably be avoided in routing its backups, in order to prevent a single failure from disabling all channels of the same $\mathcal{D}$-connection. As a result of disjoint routing, equipping each $\mathcal{D}$-connection with a single backup reduces the network capacity of accommodating $\mathcal{D}$-connections by 50% or more, as a backup channel requires at least the same amount of resources to be reserved as its primary channel. The large spare resources can seriously degrade the attractiveness of our

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

scheme.

To alleviate this problem, we have developed a resource sharing technique, called *backup multiplexing*. Its basic idea is that on each link, we reserve only a very small fraction of link-resources needed for all backup channels going through the link. In this thesis, we consider only link-bandwidth for simplicity, but other resources like buffer and CPU can be treated similarly. In what follows, we present two methods for backup multiplexing: (i) 'deterministic' method with 'resource aggregation' and (ii) 'probabilistic' method with 'admission overbooking.' Each method is explained assuming that the routes of primary and backup channels are given at the time of backup multiplexing.

2.1.1 Deterministic Multiplexing

This method adopts a deterministic failure model in which the maximum number of a particular failure type is assumed, and calculates the exact amount of spare resources which are just enough to handle all possible cases under the assumed failure model. As an example, the algorithm to calculate the spare resources s_ℓ at link ℓ under the single-link failure model is given in Figure 2.1 (a). Φ_ℓ^1 denotes the set of all primary channels whose backups traverse ℓ, and r_k is the resource required at each link by the primary channel M_k. Each connection is equipped with a single backup channel, because one backup for each connection is enough to tolerate any single link failure. The algorithm in Figure 2.1 (a) checks all possible single-link failures to extract the maximum spare-resource requirement at link ℓ. Whenever a backup channel is established, this algorithm has to be run at each link on its path. The algorithm for the single-node failure model can be easily devised by slightly modifying this algorithm.

Usually, spare-resource reservation based on the single failure model provides a sufficient level of fault-tolerance, since the time for channel failure recovery is much smaller than MTBF (Mean Time Between Failures) of the network components. Nevertheless, if a higher level of fault-tolerance is required, multiple backups can be set up — i.e., to tolerate simultaneous failures. As an example, the algorithm for double-link failure tolerance is presented in Figure 2.1 (b). To tolerate all possible double-link failures, each primary channel needs two backups. In Figure 2.1 (b), Φ_ℓ^1 denotes the set of all primary channels whose first backups traverse ℓ, and Φ_ℓ^2 denotes the set of all primary channels whose second backups traverse ℓ. $B1_k$ denotes the first backup of M_k.

Backup channels can be established to allow different $\mathcal{D}$-connections to have different fault-tolerance capability. For instance, when some connections require single-link failure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
01  loop  for each link i, i ≠ ℓ
02      loop  for each primary channel M_k ∈ Φ^1_ℓ
03          if  M_k contains link i then
04              s^1_{i,ℓ} ← s^1_{i,ℓ} + r_k
05          endif
06      endloop
07  endloop
08  s_ℓ ← max{s^1_{i,ℓ}}, ∀i ≠ ℓ
```

(a) An algorithm for single-link failure tolerance

```
01  loop  for each link i, i ≠ ℓ
02      loop  for each primary channel M_k ∈ Φ^1_ℓ
03          if  M_k contains link i then
04              s^1_{i,ℓ} ← s^1_{i,ℓ} + r_k
05          endif
06      endloop
07  endloop
08  loop  for each link pair (i,j), i ≠ j, i ≠ ℓ, j ≠ ℓ
09      loop  for each primary channel M_k ∈ Φ^2_ℓ
10          if  M_k contains i and B1_k contains j then
11              s^2_{i,j,ℓ} ← s^2_{i,j,ℓ} + r_k
12          endif
13      endloop
14  endloop
15  s_ℓ ← max{s^1_{i,ℓ} + s^2_{i,j,ℓ}}, ∀i ≠ ℓ, ∀j ≠ ℓ
```

(b) An algorithm for double-link failure tolerance

Figure 2.1: Deterministic multiplexing algorithms

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tolerance and others require double-link failure tolerance, the 10th line of Figure 2.1 (b) should be changed so that only those connections requiring double-link failure tolerance are accounted for in calculating $s^2_{i,j,\ell}$.

Under deterministic multiplexing, spare resources at each link is determined as an aggregated entity, s_ℓ. This type of backup multiplexing is possible only when resource reservation is completely interchangeable among channels. However, such a condition does not hold for all real-time channel schemes, but in general it is valid only for 'rate-based' schemes, not for 'scheduler-based' schemes (we borrowed this classification from [4]).

In the rate-based schemes [74, 110], QoS has a static relation with the traffic characteristics. For example, a higher message rate (hence, higher bandwidth) results in a smaller message delay. In these schemes, the admission test at a link simply examines whether the demanded resources exceed the available resources, since the amount of resources determines the QoS level. By contrast, in the scheduler-based schemes [55, 29], the QoS requirement (e.g., delay) of a channel can be specified independently of its traffic characteristics. In such schemes, the admission test checks for the schedulability of a channel by deriving a feasible priority assignment to meet its QoS requirement while considering the worst-case contention with existing channels. Because the priority of a channel is determined by considering not only its bandwidth requirement but also its delay requirement, resources needed to guarantee the QoS of a channel may not be sufficient for other channels with a different QoS requirement, even if they have the same traffic characteristics. The inapplicability of deterministic multiplexing to scheduler-based schemes is detailed further in Appendix 2.A.

2.1.2 Probabilistic Multiplexing

The probabilistic multiplexing method is designed to use a non-deterministic failure model. Thus, each network component is assumed to fail with a certain rate. Backup channels are multiplexed indirectly via a modified admission test (or *meta-admission test*). In meta-admission test, some existing backup channels are not accounted for in the admission test of a new backup channel, which is, in essence, equivalent to resource sharing between the new backup and those backups unaccounted for.

This multiplexing method is more generally applicable than deterministic multiplexing, in that 'admission overbooking' through meta-admission test is possible regardless of the underlying real-time channel schemes. Note that the inapplicability of deterministic multiplexing to scheduler-based schemes is not because of its deterministic failure assumption but because of the use of resource aggregation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Deciding which backup channels will not be accounted for in the admission test of a backup channel is a crucial problem. In other words, the key is to decide which backups will share the same resources. Our strategy is to multiplex those backups which are less likely to be activated simultaneously. The probability of simultaneous activation of two backups of two different $\mathcal{D}$-connections is bounded by the probability of simultaneous failure of their respective primary channels. This probability depends on the routing of the primary channels, and increases with the number of components shared among the primaries.

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

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

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

Based on this probability, the set of backups to be multiplexed together is determined for each backup on a link, i.e., multiplexing is done hop-by-hop. B_i and B_j are multiplexed if $\mathcal{S}(B_i, B_j)$ is smaller than a certain threshold ν, called *multiplexing degree*, which is specific to each backup. So, the rule to decide resource sharing in the single-backup configuration is:

$$M_i \bowtie M_j \;\Rightarrow\; B_i \parallel B_j,$$

where $M_i \bowtie M_j$ indicates that $sc(M_i, M_j)\cdot\lambda \geq \nu$, and $B_i \parallel B_j$ indicates that B_i and B_j are not multiplexible. The relation $\bowtie$ is not necessarily symmetric, so that each backup can use a different ν to determine the set of backups to be multiplexed with itself. The smaller ν of a backup, the higher fault-tolerance will result. This way, per-connection

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

01 **loop** for each backup channel B_k on link ℓ

02 **if** $sc(M_k, M_i) \cdot \lambda \geq \nu_i$ and $\nu_k \leq \nu_i$ **then**

03 $\Pi_{B_i,\ell} \longleftarrow \{\Pi_{B_i,\ell} + B_k\}$

04 **endif**

05 **endloop**

06 $s_\ell \longleftarrow max\{\sum_k^{\Pi_{B_j,\ell}} r_k + r_j\}, \forall B_j \in \ell$

Figure 2.2: Probabilistic multiplexing algorithm

control of fault-tolerance is possible, thus allowing more important connections to have higher fault-tolerance (e.g., tolerating harsher failures). We require each backup to have the same multiplexing degree on all of its links for easy management.

When this multiplexing method is applied to rate-based schemes, the spare-resource requirement at each link can be quantified as a single entity s_ℓ as follows. Let $\Pi_{B_i,\ell} = \{B_\alpha, B_\beta, \ldots\}$ denote the set of backups which are not multiplexed with B_i on link ℓ. One way to determine the spare resources at link ℓ is to find the highest resource requirement among all sets of $\{\Pi_{B_i,\ell} + B_i\}$, where all backups are considered equally regardless of their multiplexing degrees. However, this method may overestimate the amount of required spare resources at a link, when backups with different multiplexing degrees exist at the link. Suppose there are one backup with a very small ν and many backups with large ν on a link. Then, Π_ℓ of the backup with a very small ν will determine the amount of spare resources at the link, which may be much larger than actually needed. To get around this problem, we consider only backups with no greater multiplexing degrees than that of B_i when $\Pi_{B_i,\ell}$ is constructed. Figure 2.2 depicts an algorithm to calculate s_ℓ when a new backup channel B_i is established on link ℓ under probabilistic multiplexing. (The same notation as in Figure 2.1 is used.)

A straightforward way of enhancing the fault-tolerance capability of a $\mathcal{D}$-connection is to establish multiple backups with the same multiplexing threshold. Alternatively, different multiplexing rules from the rule for first backups can be applied to additional backups. For example, the following set of rules are for the double-backup configuration to tolerate double-component failures:

Rule-1: $M_i \bowtie M_j \Rightarrow B1_i \parallel B1_j$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Rule-2:	$B1_i \bowtie M_j \Rightarrow B2_i \parallel B1_j$.
Rule-3:	$B1_i \bowtie B1_j \Rightarrow B2_i \parallel B1_j$.
Rule-4:	$(B1_i \bowtie M_j)$ & $(M_i \bowtie B1_j) \Rightarrow B2_i \parallel B2_j$.
Rule-5:	$(M_i \bowtie M_j)$ & $(B1_i \bowtie B1_j) \Rightarrow B2_i \parallel B2_j$.

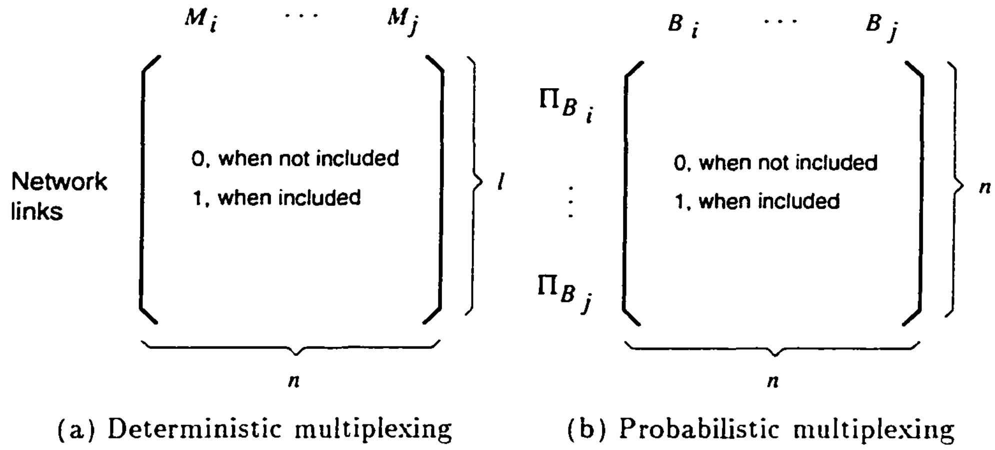
where $B1_i$ and $B2_i$ are the first and second backups of M_i, respectively.

Rule-1 is for the relation between first backups, which is the same as in the single-backup configuration. Rule-2 and 3 are for the relation between first and second backups (belonging to different connections), and Rule-4 and 5 are for the relation between second backups. For example, Rule-2 says that $B2_i$ and $B1_j$ should not be multiplexed to prepare for the case when M_i fails and a shared component between $B1_i$ and M_j fails, which causes the simultaneous activation of $B2_i$ and $B1_j$. Other rules can be reasoned similarly.

Probabilistic multiplexing supports per-connection fault-tolerance control in a finer grain than deterministic multiplexing. Under deterministic multiplexing, the dependability of a $\mathcal{D}$-connection is decided only by the number of its backups, allowing a coarse-grain fault-tolerance control like single failure tolerance or double failure tolerance. By contrast, under probabilistic multiplexing, the dependability of a $\mathcal{D}$-connection can be controlled by both number of its backups and associated multiplexing degrees. Even though equipped with the same number of backups, connections can have different fault-tolerance depending on the used multiplexing degrees. For instance, with a single backup, some connections may be 100% tolerant to all single failures, while some may be only 50% tolerant to the same failure type.

Under probabilistic multiplexing, the dependability of a connection is defined as a probability that the connection will be safely recovered from failures. However, 100% tolerance to deterministic failures is also achievable. For instance, if ν for a backup B_i is set to λ, failure recovery of the corresponding $\mathcal{D}$-connection from any single node/link failure is guaranteed, because B_i will not be multiplexed with any other backup whose primary overlaps with M_i. Similarly, if ν is set to 3λ, any single link failure can be tolerated, since no backup whose primary overlaps with M_i by more than three components (including the case of sharing a link and two nodes attached to that link) will be multiplexed with B_i. However, the amount of spare resources resulting from probabilistic multiplexing may not be as small as that from deterministic multiplexing to tolerate the same type of deterministic failures. It is because all of the primary channels of the $\Pi_{B_i,\ell}$ members may not overlap with M_i at the same component. In building $\Pi_{B_i,\ell}$, we only check if other primary channels overlap with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Deterministic multiplexing (b) Probabilistic multiplexing

Figure 2.3: Data structures for reducing the algorithm complexity

M_i at more than ν_i/λ components. As a result, spare-resource reservation based on $\Pi_{B_i,\ell}$ tends to overestimate the spare resource needs for tolerating a particular failure type. For instance, by setting ν to λ, probabilistic multiplexing actually can tolerate most of single channel failures instead of single component failures.

2.1.3 Scalability & Complexity Issue

Both backup multiplexing schemes use fully-distributed algorithms: they do not require each node to maintain global knowledge of the network traffic conditions or to generate any type of messages to be broadcast. Backup multiplexing is performed hop-by-hop, and therefore, at each link, only the knowledge of primary channels whose backups traverse the link is required. Such information can be easily collected, by making a backup channel-establishment message carry the path information of its primary channel. The efficiency of backup multiplexing does not degrade as the network gets large. Backup multiplexing would rather be more effective in large-scale and highly-connected networks, because such networks contain more versatile paths between the two end nodes of a connection, thus lowering the probability that primary channels overlap with one another.

In a large-scale network, the computational complexity of backup multiplexing mechanisms is a matter of concern. For deterministic multiplexing, the calculation of spare-resource requirement for each case of failure resides in the inner-most loop and decides the algorithm complexity. For example, the algorithm for single-link failure tolerance has a complexity of $O(n \cdot l)$, when n is the number of backup channels on the link under consideration and l is the total number of links in the network. The computational overhead can

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

be reduced by storing and reusing the once-calculated information. The data structure for such a purpose is depicted in Figure 2.3 (a). Whenever a new backup is established, new information is added to this data structure.

For probabilistic multiplexing, the essential part is constructing a set of non-multiplexible backups, $\Pi_{B_i,\ell}$, on each link ℓ. The complexity of this step is $O(n)$, where n is the number of backup channels on link ℓ. (This is because each calculation of $\mathcal{S}(B_i, B_j)$ requires constant time.) To find the largest set, we need to construct $\Pi_{B_i,\ell}$ for all backups on ℓ, which requires $O(n^2)$ time. However, if we store each $\Pi_{B_i,\ell}$ calculated before the new establishment request for B_j is made, we only need to update each $\Pi_{B_i,\ell}$ by calculating $\mathcal{S}(B_i, B_j)$. Hence, the complexity can be reduced to $O(n)$ at the expense of additional memory. The information needed to maintained at each link is shown in Figure 2.3 (b). Probabilistic multiplexing scales particularly well, because its complexity does not contain l, the total number of links in the network.

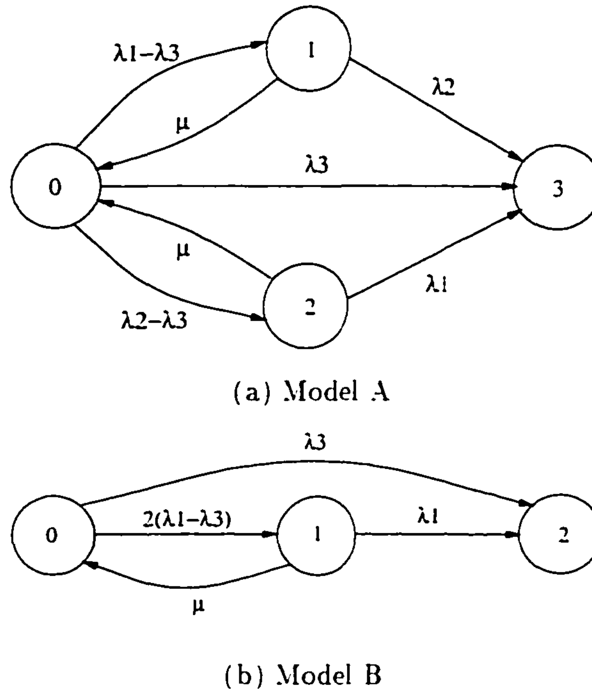
2.2 Dependability QoS Negotiation

Instead of providing an identical level of fault-tolerance to *all* connections, we allow each client to specify its fault-tolerance requirement. The network then establishes necessary backups to meet this requirement. Our scheme provides two dependability QoS parameters for this negotiation. One is about the guarantee on successful failure recovery, and the other is about the failure recovery delay. In QoS negotiation, the later is not negotiable while the former is. Only the former is considered in this chapter.

Deterministic multiplexing exports a simple QoS model. A client specifies its QoS requirement among a fixed QoS menu such as 100% single-failure tolerance, 100% double-failure tolerance, and so on. Then the network sets up a proper number of backups for each client.

Probabilistic multiplexing requires a more complicated QoS negotiation procedure. Two QoS parameters (P_r, Γ) are exported: P_r is the probability of successful failure recovery and Γ is the failure-recovery delay bound. In other words, with a certain rate of component failures, the probability that a $\mathcal{D}$-connection will suffer a disruption of real-time communication service longer than Γ is not greater than P_r. Described below are the QoS interface and the spare-resource reservation procedure for probabilistic multiplexing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Model A

(b) Model B

Figure 2.4: Example Markov models to derive $R(t)$

2.2.1 A Dependability QoS Parameter, $\mathcal{P}_r$

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

Let's consider how to derive $R(t)$ of a $\mathcal{D}$-connection. Assuming a Poisson failure process with rate λ, we derive $R(t)$ of each network component to be $e^{-\lambda t}$. For the convenience of presentation, we further assume that the failure rates of all network components are same and all failures are statistically independent. Then, $R(t)$ of a channel can be expressed as $e^{-n\lambda t}$, where the channel path consists of n components. In other words, the failure rate of the channel is $n\lambda$. Finally, the reliability of a $\mathcal{D}$-connection can be modeled with a Markov process using the failure rates of its channels. For example, Figure 2.4 (a) shows a continuous-time Markov model to derive $R(t)$ of a $\mathcal{D}$-connection with a single backup channel, where μ is the channel repair (or re-establishment) rate, λ_1 and λ_2 are failure rates of the primary and backup channels, respectively, and λ_3 is the failure rate of the shared part of both channels. State 0 is the initial state and state 3 is the absorbing state. Figure 2.4 (b) is a simplified model when the primary and backup channels are of the same length. For example, if both the primary and backup channels of a $\mathcal{D}$-connection are of 4 hops

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

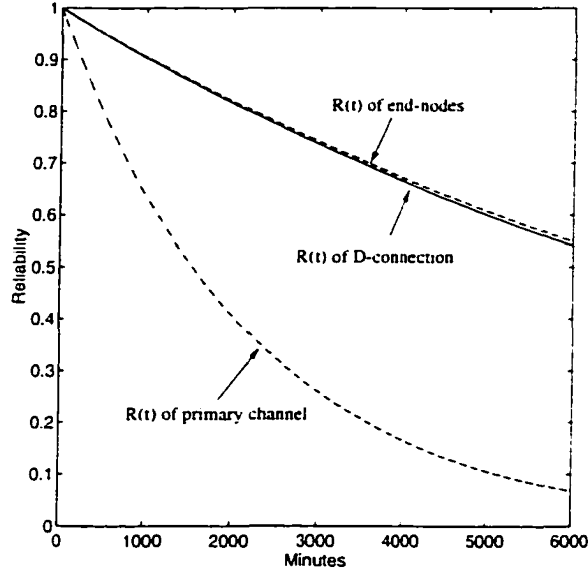


Figure 2.5: $R(t)$ of a $\mathcal{D}$-connection with a single backup

length and are routed disjointly, λ_1 is 9λ and λ_3 is 2λ considering two end nodes shared by both channels. Using the technique in [95], one can calculate $R(t)$ of this $\mathcal{D}$-connection from the Markov model of Figure 2.4 (b):

$$\begin{aligned} R(t) &= 1 - P(\text{the system is in the absorbing state at time } t) \\ &= \frac{238\lambda^2 + \mu\lambda + \lambda\sqrt{49\lambda^2 + 42\mu\lambda + \mu^2}}{(25\lambda + \mu)\sqrt{49\lambda^2 + 42\mu\lambda + \mu^2} - 49\lambda^2 - 42\mu\lambda - \mu^2} \cdot \epsilon^{-(1/2)(25\lambda + \mu - \sqrt{177\lambda^2 + 42\mu\lambda + \mu^2})t} \\ &\quad - \frac{238\lambda^2 + \mu\lambda - \lambda\sqrt{49\lambda^2 + 42\mu\lambda + \mu^2}}{(25\lambda + \mu)\sqrt{49\lambda^2 + 42\mu\lambda + \mu^2} + 49\lambda^2 + 42\mu\lambda + \mu^2} \cdot \epsilon^{-(1/2)(25\lambda + \mu + \sqrt{177\lambda^2 + 42\mu\lambda + \mu^2})t}. \end{aligned}$$

We plot the reliability of this connection in Figure 2.5 by setting λ to 0.00005 ($1/\lambda$ measured in minutes) which results in 332 hours of MTBF and setting μ to 0.1 ($1/\mu$ measured in minutes), which means 10 minutes of channel repair time.

However, representing the QoS parameter as a function of time is unsuitable for the client-interface model. Thus, instead of using Markov models, we use a combinatorial model to approximate the reliability of a $\mathcal{D}$-connection. The approximation is possible because the channel repair rate (μ) is much larger than the channel failure rate — the channel re-establishment time is in the order of seconds or minutes, whereas MTBF is in the order of 100 or 1000 hours. Thus, a $\mathcal{D}$-connection affected by a failure returns to the initial state (state 0) much before the second failure occurs. Figure 2.5 shows that the $\mathcal{D}$-connection's $R(t)$ is very close to that of its end-nodes, which implies that the quick recovery results in a nearly perfect reliability except for the cases of end-node failures.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In the combinatorial approximation, each network component is assigned a probability, λ, of failure occurrence during one time unit, and the $\mathcal{D}$-connection under consideration is assumed to be in the initial state at the start of each time unit. When $\mathcal{P}_r$ represents the $\mathcal{D}$-connection's reliability under this combinatorial model, $\mathcal{P}_r$ is equal to the probability that at least one channel of the $\mathcal{D}$-connection remains healthy during one time unit. For example, the $\mathcal{P}_r$ of a $\mathcal{D}$-connection with a single backup is

$$\mathcal{P}_r = P(\text{primary not fail}) + P(\text{primary fails} \cap \text{backup not fail}).$$

2.2.2 Calculation of $\mathcal{P}_r$

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

Calculation of $\mathcal{P}_r$ for a $\mathcal{D}$-connection with backup multiplexing requires us to consider the possibility of multiplexing failures. The $\mathcal{P}_r$ of a $\mathcal{D}$-connection composed with a primary channel M_i and a backup channel B_i is:

$$\mathcal{P}_r(i) = P(M_i \text{ not fail}) + P(M_i \text{ fails}) \cdot P(B_i \text{ not fail}) \cdot \{1 - P_{muxf}(B_i)\},$$

where $P_{muxf}(B_i)$ is the probability that B_i is not available due to a multiplexing failure. $P_{muxf}(B_i)$ is not greater than

$$\sum_{\ell}^{\text{links of } B_i} P_{muxf}(B_i, \ell),$$

where $P_{muxf}(B_i, \ell)$ is the probability that B_i suffers from a multiplexing failure at ℓ, a link on the path of B_i. The $\mathcal{P}_r$ value associated with more backups can be derived similarly. Presented below are two methods for calculating $P_{muxf}(B_i, \ell)$; Method 1 is for accurate derivation and Method 2 is for quick approximation.

<u>**Method 1**</u>

A backup channel may suffer a multiplexing failure at a link, if the total resource needs by simultaneous backup activations exceed the total spare resources at the link. Suppose the number of backups on link ℓ is Z and the spare resource at ℓ is s_ℓ. Then, there can be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2^{Z-1} different patterns of simultaneous backup activation with B_i. Since we can, without loss of generality, label the k backups activated along with B_i from 1 to k and label the remaining $Z-k-1$ backups from $k+1$ to $Z-1$, the probability associated with each activation pattern is

$$\mathcal{S}(B_i, B_1, \ldots, B_k) \cdot \{1 - \mathcal{S}(B_i, B_{k+1}, \ldots, B_{Z-1})\}.$$

Here, $\mathcal{S}(B_i, B_1, \ldots, B_k)$ indicates the probability of simultaneous activation of $B_i, B_1, \ldots, B_k$. Among the 2^{Z-1} sets, we can tell which requires more resources than s_ℓ, and which does not. $P_{muxf}(B_i, \ell)$ is equal to the sum of the probabilities associated with those cases which require more resources than s_ℓ.

We use an incremental approach to calculate $\mathcal{S}(B_1, \ldots, B_k)$. We first choose a component C_j shared by more than one primary channel of the backups under consideration, and calculate $\mathcal{S}(B_1, \ldots, B_k)$ after removing C_j, which is denoted by $\mathcal{S}_{\{C_j\}}(B_1, \ldots, B_k)$. Then,

$$\mathcal{S}(B_1, \ldots, B_k) \quad = \quad \lambda + (1-\lambda) \cdot \mathcal{S}_{\{C_j\}}(B_1, \ldots, B_k),$$

where the second term represents the probability that all k backups will be activated simultaneously when C_j does not fail. $\mathcal{S}_{\{C_j\}}(B_1, \ldots, B_k)$ can be obtained similarly. Thus, by selecting another shared component C_m,

$$\mathcal{S}_{\{C_j\}}(B_1, \ldots, B_k) \quad = \quad \lambda + (1-\lambda) \cdot \mathcal{S}_{\{C_j, C_m\}}(B_1, \ldots, B_k).$$

The same step is applied recursively until there remains no shared component. The last term

$$\mathcal{S}_{\{C_j, C_m, \ldots\}} \quad = \quad (1-(1-\lambda)^{c(M_1)}) \cdot (1-(1-\lambda)^{c(M_2)}) \ldots (1-(1-\lambda)^{c(M_k)}),$$

where $c(M_i)$ is the component count in M_i.

For example, when M_1 and M_2 share a component C_j and both consist of 3 components, then

$$\begin{aligned} \mathcal{S}(B_1, B_2) &= \lambda + (1-\lambda) \cdot \mathcal{S}_{\{C_j\}}(B_1, B_2) \\ &= \lambda + (1-\lambda) \cdot \{(1-(1-\lambda)^2) \cdot (1-(1-\lambda)^2)\}. \end{aligned}$$

Method 2

Multiplexing failures do not always occur even if multiplexed backups are activated simultaneously. Thus, to capture the exact probability of multiplexing failures, we have to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

compare the total resource demands by simultaneous backup activations against s_ℓ as in Method 1. This method, however, over-estimates $P_{muxf}(B_i, \ell)$ by simply accumulating the probabilities of simultaneous backup activations which are ignored in multiplexing. This method requires much simpler calculation than Method 1 at the cost of accuracy. Note that the over-estimation of $P_{muxf}(B_i)$ leads to the under-estimation of $P_r(i)$, thus erring on the safe side.

B_j is multiplexed with B_i at link ℓ, only if $\mathcal{S}(B_i, B_j)$ is smaller than ν_i, the multiplexing degree of B_i. Thus, the probability that B_i will suffer from a multiplexing failure on link ℓ due to the simultaneous activation of B_j is not greater than $\mathcal{S}(B_i, B_j)$. Let $\Psi_{B_i,\ell}$ denote the set of backup channels which are multiplexed with B_i at ℓ:

$$\Psi_{B_i,\ell} = \{\text{all backups on } \ell\} - \Pi_{B_i,\ell} - B_i.$$

Then, we get

$$\begin{aligned} P_{muxf}(B_i, \ell) &\leq 1 - \prod_{\forall B_j \in \Psi_{B_i,\ell}} (1 - \mathcal{S}(B_i, B_j)) \\ &\leq 1 - (1 - \nu_i)^{|\Psi_{B_i,\ell}|}, \end{aligned}$$

where $|\Psi_{B_i,\ell}|$ is the number of backups multiplexed with B_i on link ℓ. In the rest of this thesis, we assume the use of Method 2.

2.2.3 QoS Negotiation Procedure

As in the case of primary channels, QoS negotiation is a crucial step of backup channel establishment. Here, we present two QoS negotiation methods.

In the first method, the network selects the number of backups and the multiplexing degree by considering the client-specified $\mathcal{P}_r$ requirement and/or the network status. Then, the resultant $\mathcal{P}_r$ of the connection is calculated and notified to the client. The client may or may not be satisfied with the offered fault-tolerance QoS level, and may accept or reject the offer. With this method, the client-specified $\mathcal{P}_r$ requirement is met "loosely" or in a "best-effort" manner.

In the second method, the client's $\mathcal{P}_r$ requirement is met as requested, or the request gets rejected. Assume that the channel establishment is initiated by the source node.[1] A backup channel is established by using a pair of channel-establishment messages: (i) the 'resource reservation message' from source to destination and (ii) the 'resource relaxation message'

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

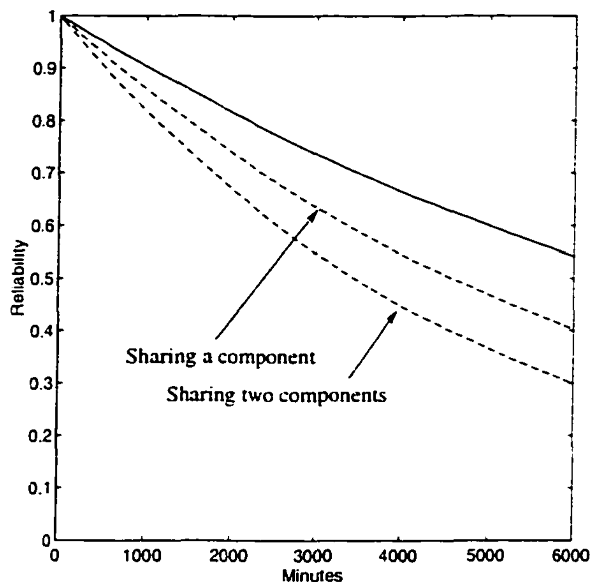


Figure 2.6: The effect of non-disjoint routing on $R(t)$

from destination to source. In the forward pass (reservation message) to the destination, spare resources are reserved for the backup without multiplexing, while $\Psi_{B_i,\ell}$ is calculated on each link ℓ of the channel path with various ν values. The reservation message collects the $\Psi_{B_i,\ell}$ information and passes them to the destination node. Then, the destination node selects the largest ν which satisfies the required $\mathcal{P}_r$ based on the collected information. In the backward pass (relaxation message) from destination to source, the spare resources on the channel path are multiplexed according to the selected ν. If the required $\mathcal{P}_r$ is too high to satisfy, the client's request will be rejected. (The rejected client may opt to retry with a lower $\mathcal{P}_r$ requirement.)

Essentially, we transform the problem of meeting the $\mathcal{P}_r$ requirement to that of deciding the multiplexing degree. Fortunately, we need to try only a couple of different ν values, because the values of $\mathcal{S}(B_i, B_j)$ are distributed around integer multiples of λ, i.e., $\mathcal{S}(B_i, B_j) \approx sc(M_i, M_j) \cdot \lambda$. Thus, the backups on a link are grouped into a certain number of classes according to their multiplexing degrees. The number of classes is not greater than the number of components on the longest possible path in the network. The network can decide the number of backups *a priori* or can establish backups incrementally until the required $\mathcal{P}_r$ is achieved.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.4 The Number and Disjointness of Backup Channels

Unless λ, the component failure rate, is very large, additional backups will not increase $R(t)$ of a $\mathcal{D}$-connection much, because the first backup provides nearly the maximal reliability (i.e., $R(t)$ of its end nodes), as shown in Figure 2.5. Instead, we find the benefit of multiple backups from a different perspective: we can take advantage of multiple backups to reduce resource overhead in conjunction with backup multiplexing. More details on this will be discussed in Chapter 4.

The routing of a backup channel has a significant impact on the reliability of its connection. The links/nodes used by a primary channel should be avoided in routing its backups, because overlapping routes among the channels of the same $\mathcal{D}$-connection will degrade the reliability of the connection. The reliability degradation by non-disjoint routing is illustrated in Figure 2.6 with the same example used for Figure 2.5. Throughout this thesis, we assume disjoint routing of the channels belonging to the same $\mathcal{D}$-connection.

The route selection of backup channels has a significant impact on the amount of spare resources as well. The next section is dedicated to this issue.

2.3 Backup-Route Selection

The shortest-path (i.e., minimum-hop path) algorithm is often used for channel route selection. A node which wants to set up a real-time channel broadcasts route-search messages to find a shortest path. If network topology information is maintained at each node, a path can be found without broadcasting route-search messages. Recently, more elaborate routing algorithms have been developed. For instance, in [100], a smallest-delay path, instead of a minimum-hop path, is selected. In [75], a smallest-delay path is selected among minimum-hop paths. Another popular metric of QoS routing is the residual bandwidth, so as to favor a path with larger available bandwidth. The algorithm presented in [98] uses a metric which aggregates multiple routing parameters such as throughput, delay, and error rate. Unlike the above-cited research, our interest is in backup-route selection.

2.3.1 Optimal Routing Problem

Essentially, we want to minimize the amount of spare resources while providing the required fault-tolerance level. Unfortunately, there doesn't exist any efficient algorithm for 'optimally' routing backup channels; the problem of finding a path set with multiple constraints is known to be NP-complete. The NP-completeness proof of the following decision

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

problem – which is subsumed by the optimal backup-routing problem — can be found in [54]:

> *Is there a feasible set of channel paths such that the sum of traffic flows at each link is smaller than the link capability, when channel traffic demands are given?*

The optimal backup-routing problem is therefore NP-complete, even without considering backup multiplexing.

Integer Programming (IP) can be used for optimal backup routing. We present, as an example, an IP formulation to achieve 100% failure recovery from any single link failure under deterministic multiplexing. The notation used in the IP formulation is as follows:

$\hat{L}$: the set of all links in the network.

c_i: the bandwidth capacity of link i.

s_i: the spare bandwidth (for backup channels) on link i.

a_i: the active bandwidth (for primary channels) on link i.

r_m: the bandwidth requirement of primary channel m.

$\hat{M}$: the set of all primary channels.

$\hat{E}_m$: the set of eligible paths for the backup channel of primary channel m.

$\hat{P}_j$: the set of primary channels which run over link j.

$X_{k,m}$: 1 if k-th path in $\hat{E}_m$ is selected as the backup path of primary channel m, and 0 otherwise.

$Y^i_{k,m}$: 1 if k-th path in $\hat{E}_m$ contains link i, and 0 otherwise.

$Z_{m,j}$: 1 if primary channel m contains link j, and 0 otherwise.

We assume that the information about primary channels is given. Thus, $\hat{P}_j$, a_i, and $Z_{m,j}$ are given, and $\hat{E}_m$ and $Y^i_{k,m}$ can also be derived from the network topology. Since c_i and r_m are constants, the only variables in this IP formulation are s_i and $X_{k,m}$, essential for backup route selection. All values are non-negative integers.

Our goal is to minimize the sum of spare resources at all links, so the objective function is

$$Min \left\{ \sum_{i=1}^{\hat{L}} s_i \right\}.$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The constraint set that should be satisfied to tolerate all single link failures is:

$$s_i \geq 0, s_i + a_i \leq c_i, \quad \forall i \in \hat{L},$$

$$\sum_{k=1}^{\vec{E_m}} Z_{m,j} \cdot X_{k,m} = 1, \quad \forall j \in \hat{L}, \forall m \in \hat{M},$$

$$s_i - \sum_{m=1}^{\hat{P}_j} \sum_{k=1}^{\vec{E_m}} X_{k,m} Y_{k,m}^i \cdot r_m \geq 0, \quad \forall i \in \hat{L}, \forall j \in \hat{L}, i \neq j.$$

The first constraint is straightforward. The second constraint indicates the property that only one backup path should be selected among $\vec{E_m}$ for each primary channel m, which is disconnected by the failure of link j. When all possible failure scenarios are considered, all primary channels will fail at least once, so the number of equalities needed to specify this constraint will be equal to the number of primary channels in the network. Hence, this constraint can be rewritten as $\sum_{k=1}^{\vec{E_m}} X_{k,m} = 1, \quad \forall m \in \hat{M}$.

The third constraint represents the property that the spare resource on link i should be sufficient to meet the resource demands of backup activation caused by the failure of link j.

The computational complexity of the above IP formulation is very high. For instance, the number of inequalities resulting from the third constraint is proportional to $(|\hat{L}| \cdot |\hat{P}_j| \cdot |\vec{E_m}|)$. If the search space includes all possible paths, the dimension of search space is an exponential function of the product of link numbers and channel numbers. The dimension of constraint matrix for the simulation condition used in Section 2.4 easily reaches several thousands.

As a result, we have to resort to heuristics that reduce complexity at the expense of optimality. We adopt a 'two-step' approach in which channels are quickly set up by heuristics, then reconfiguration is performed periodically to optimize resource usage.

2.3.2 Initial Route Selection

The path, which appears best when a backup channel is being established, is selected among a set of "eligible" paths. The eligibility of a backup path can be defined by maximum path length, end-to-end delay, or bandwidth. A shortest-path search algorithm is used to find the minimum-cost path where a cost value is assigned to each network link. When multiple backups are set up for each $\mathcal{D}$-connection, one can use such algorithms as in [99, 87] which find multiple disjoint paths.

The link cost functions considered here are:

$$f_1(\ell, B_i) = 1,$$

$$f_2(\ell, B_i) = \Omega_\ell,$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$f_3(\ell, B_i) = \Delta_{B_i,\ell}.$$
$$f_4(\ell, B_i) = f_1(\ell, B_i) + \omega_1 \cdot f_2(\ell, B_i).$$
$$f_5(\ell, B_i) = f_1(\ell, B_i) + \omega_2 \cdot f_3(\ell, B_i).$$
$$f_6(\ell, B_i) = f_2(\ell, B_i) + \omega_3 \cdot f_1(\ell, B_i).$$
$$f_7(\ell, B_i) = f_2(\ell, B_i) + \omega_4 \cdot f_3(\ell, B_i).$$
$$f_8(\ell, B_i) = f_3(\ell, B_i) + \omega_5 \cdot f_1(\ell, B_i).$$
$$f_9(\ell, B_i) = f_3(\ell, B_i) + \omega_6 \cdot f_2(\ell, B_i).$$

where ℓ is a link identifier and B_i is the backup channel to be routed. f_1 and f_2 are popular cost functions for general real-time channel routing, and will be used as references for performance comparison in Section 2.4. f_3 is a new cost function we devised by exploiting the property of backup multiplexing. The other cost functions are composed by combining these three basic cost functions.

When f_1 is used, all links have an identical cost, and hence, the minimum-hop path will also be the minimum-cost one. This simple cost function differs from the rest in that it does not utilize the knowledge on resource usage. f_2 is an attractive/popular cost function when increasing the network throughput is a main concern. The rationale behind this cost function is to disperse resource reservation for balancing traffic loads. Ω_ℓ in f_2 is equal to the total resources reserved on link ℓ by both primary and backup channels. $\Delta_{B_i,\ell}$ in f_3 is the increment of spare resources at ℓ, if B_i is to be established on ℓ.

The weights in the composite cost functions are selected so that the first terms may become primary factors and the second terms may be used only to break ties among multiple candidate paths. For example, ω_1 is small enough to ensure $\omega_1 \cdot f_2(\ell, B_i) < f_1(\ell, B_i)$. Each routing heuristic is named according to the cost function used, as R_1, R_2, ..., R_9, respectively.

Different kinds of information are needed for different routing heuristics. R_1 requires the information about network topology with the health information of each network component. R_2 requires the information about resource reservation at each link. R_3 requires the path information of primary channels to predict the aftermath of backup multiplexing. There are two options for managing such information; each node can maintain (i) a database of global knowledge (about others) or (ii) a database of local knowledge (about itself). In the former, all information is broadcast in the network whenever any change (e.g., channel setup or teardown) occurs, so that the source node of a channel can decide its path using the information in the node's database. In the latter, there is no information broadcast, but

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

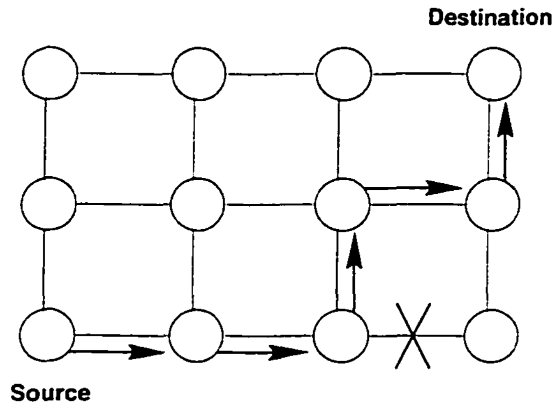


Figure 2.7: Boundary routing

instead, routing messages are broadcast to execute a distributed shortest-path algorithm.

Information outage is inevitable in both options, because the network condition may change during the resource reservation stage when multiple channels are established simultaneously by different nodes. However, even though the information used for routing is out-of-date during resource reservation, dependability QoS guarantees remain unaffected, because backup multiplexing has nothing to do with the stale information used by routing heuristics. It only results in routing a backup channel over a less efficient path.

In addition to these link-state heuristics, we would like to describe a simple routing heuristic based on the topological information. With this routing heuristic, a channel is routed on a topologically shortest path which is closest to the boundary of possible routing area (see Figure 2.7). The cross sign means the unavailability of a link due to failures or resource shortage. This boundary routing allows a wider search space for routing next channels between the same nodes. We name this routing method R_0.

2.3.3 Periodic Route Reconfiguration

In the initial routing, those backup channels which have already been established are not disturbed for routing a new backup channel. Intuitively, if we reroute early backup channels which had been routed without considering later backup channels, we can improve the overall resource usage efficiency. Rerouting primary channels will make further improvement possible, since routing of primary channels plays a key role in backup multiplexing. However, moving primary channels is a complex process and can cause service disruptions. So, we do not consider the rerouting of primary channels. Repair of failed components is another case in which rerouting existing backups is beneficial. Network component failures will

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

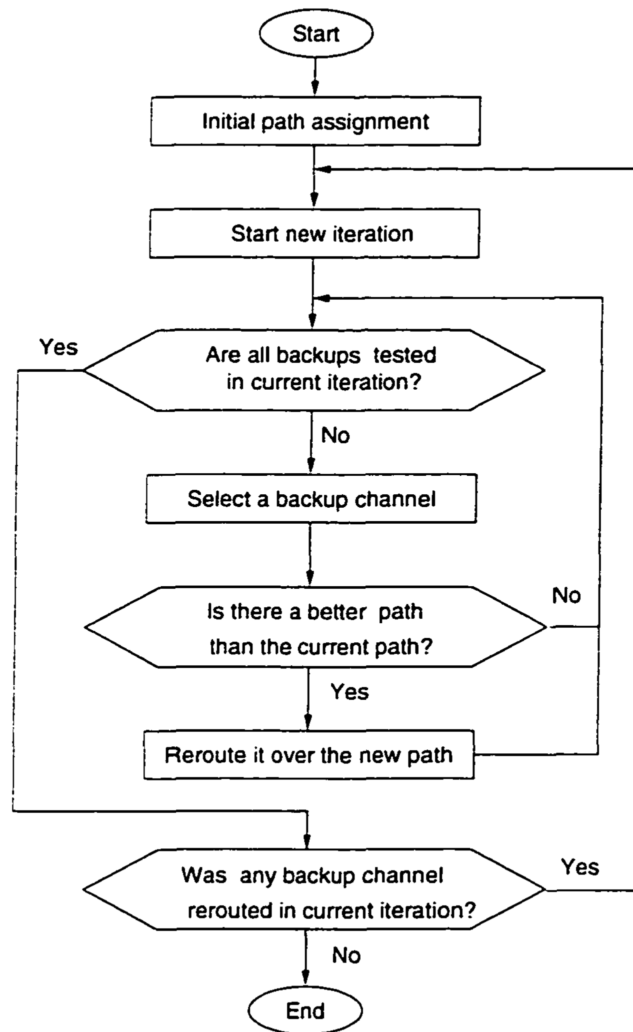


Figure 2.8: The iterative optimization method

disable/activate backups. and new backups will be routed to substitute for the old (i.e.. disabled or promoted) backups by avoiding the failed components. possibly on a longer path than the original backup. When the failed components are repaired. the efficiency of resource usage can be improved by rerouting some backups over the repaired components.

Recalculating all existing backup paths whenever a new backup is established is what we want to avoid in the dynamic channel setup/teardown environment. Instead. in our two-step approach. channels are set up quickly using the initial routing heuristics. and then reconfiguration is performed periodically considering all of the channels which exist at the moment the reconfiguration starts. Though, Integer Programming (IP) can be used for the optimal reconfiguration of existing backups, its computational complexity is high. Long delay in route reconfiguration involves a risk that some reconfiguration decisions may

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

become less beneficial or meaningless, since existing backups can be torn down or new channels can be added during the optimization. IP is not a feasible solution particularly for a large-scale network in the dynamic environment. As a practical alternative, we developed an iterative optimization method.

The flowchart in Figure 2.8 depicts the iterative optimization method. Starting with an initial path assignment, a new minimum-cost path of each backup is searched, one at a time. When there is a new path which requires less spare resources than the original path, the backup is moved to the new path. The iteration is continued until no further improvement is made. This algorithm always converges and the number of iterations is bounded by the square of the total number of backups in the network, as the total cost (i.e., network-wide sum of spare resources) monotonically decreases as channels are rerouted. However, the final result is not necessarily optimal since we do not explore the possibility of rerouting multiple backups simultaneously. Another reason for sub-optimality is the lack of checking local optimality. For instance, rerouting a backup over a path with the same or larger cost can reduce the cost of other backups by a greater margin.

2.4 Evaluation

In this section, we evaluate the performance of backup multiplexing schemes and routing mechanisms via simulations. The metric for performance evaluation is the average spare resource at a link to meet the fault-tolerance requirement. The dependability goal assumed in this section is single-link failure tolerance.

2.4.1 Simulation Setup

Some general simulation setups which will be used throughout this thesis are as follows. The simulation networks are an 8×8 torus (wrapped mesh) network and an 8×8 mesh network (see Figure 2.9). In the simulation networks, neighbor nodes are connected by two simplex links, one for each direction, and all links have an identical bandwidth. To obtain a similar total capacity for both networks, we set the link capacity of the torus network to 200 Mbps and set that of the mesh network to 300 Mbps.

$\mathcal{D}$-connections are established incrementally, one at a time. Channels of each $\mathcal{D}$-connection are routed disjointly by a sequential shortest-path search algorithm. Thus, the primary channel is routed first over a shortest path, then the backup is routed without using the components of the primary channel. Primary channels are always routed using the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

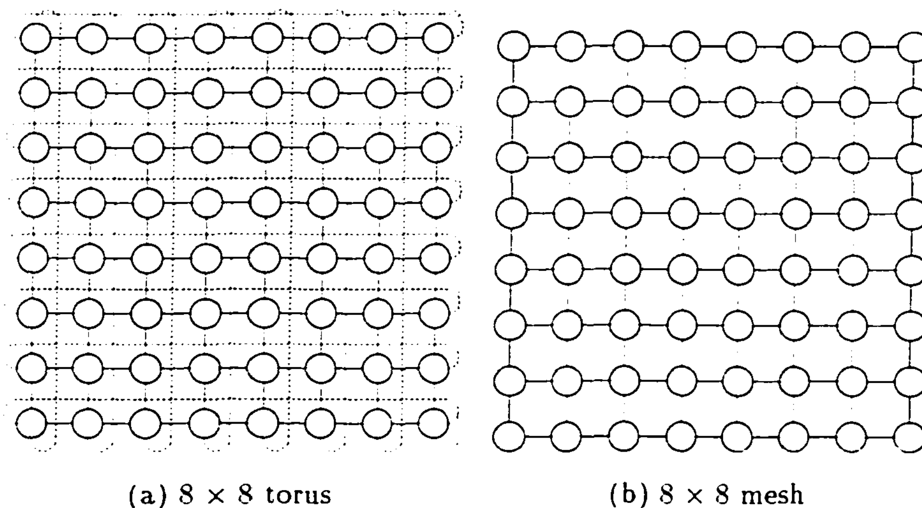

(a) 8 × 8 torus (b) 8 × 8 mesh

Figure 2.9: Simulation networks

boundary-routing method to allow a wider search space for backup routing.

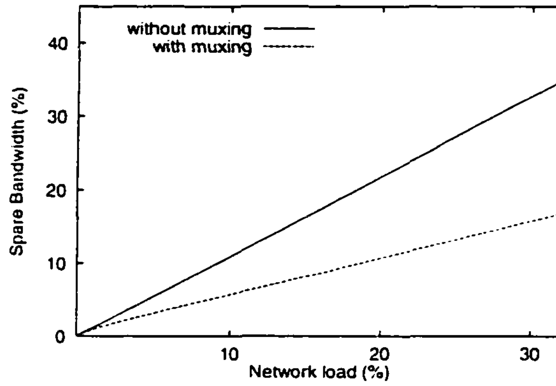
For simplicity, all $\mathcal{D}$-connections are equipped with same number of backups, and the same traffic model is assumed for all channels, so each channel requires 1 Mbps of bandwidth on each link of its path. The end-to-end delay requirement of each channel is assumed to be met if the channel path is not longer than the shortest-possible path by more than 2 hops. A total of 4032 connections are established incrementally, so that there may exist a $\mathcal{D}$-connection between each node pair, i.e., $64 \cdot 63 = 4032$.

Remarks: We simulated a regular topology network with a relatively high connectivity. This is because dependability guarantees are impossible to make on networks with low connectivity, and simulations over a regular topology may reveal the general properties of the scheme under test without being influenced by topological randomness. Note that the future backbone networks are expected to have higher connectivity than the current Internet, so that k-ary n-cube topologies (like meshes) are reasonable to consider.

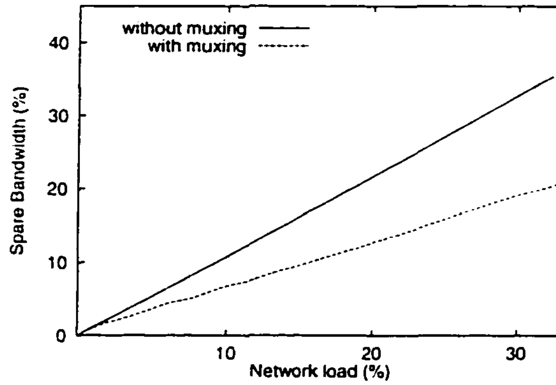
2.4.2 Measurement of Spare Resource Overhead

We first measure the average spare bandwidth for various backup configurations. In this simulation, the boundary-routing method was used for both primary and backup routing. Figure 2.10 and 2.11 show the simulation results. The 'network load' is a metric to indicate the ratio of the total bandwidth consumed by all primary channels to the total network bandwidth capacity. The establishment of 4032 connections resulted in a 33 ~ 34% network

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Single backup in 8×8 torus



(b) Single backup in 8×8 mesh

Figure 2.10: Average spare-bandwidth reservation under deterministic multiplexing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

load in both torus and mesh networks.

For deterministic multiplexing, the single backup configuration is sufficient to to tolerate any single link failure. So, the algorithm in Figure 2.1 (a) was employed.

For probabilistic multiplexing, single and double backup configurations were simulated in the torus network, but only the single backup configuration could be simulated in the mesh network because of its topological limitation. Seven different multiplexing degrees were applied in each backup configuration. In the double backup configuration, the same multiplexing rule was applied to both first and second backups. The notation 'mux=α' means that two backups are multiplexed when their primary channels share less than α network components, i.e., $\nu = \alpha\lambda$. 'mux=0' has the same effect as disabling multiplexing. The results of 'mux=2' and 'mux=4' are not plotted in Figure 2.11, because, due to the nature of channel routing, they were very close to the cases of 'mux=3' and 'mux=5', respectively. Two channel paths are not likely to share two nodes without sharing a link between the nodes, so the results of 'mux=2' and 'mux=3' are very close to each other. The case of sharing two consecutive links (i.e.,'mux=4' and 'mux=5') can be reasoned similarly.

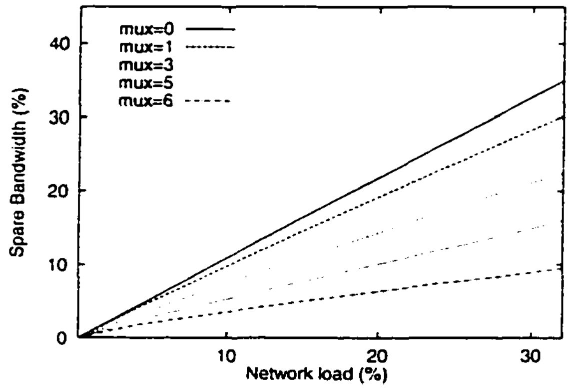
There are several interesting observations to make from the simulation results.

(i) The network capacity is reduced by more than 50% for each backup when backup multiplexing is not applied. It is because some backups are routed over longer paths than their corresponding primary channels and the paths of second backups become longer than those of the first backups. For example, in a torus network, there are usually two shortest disjoint paths between any two nodes that are more than one hop apart. If the source and destination nodes lie on the same principal axis and the distance between the two is not exactly one half of the torus dimension, there exists only one shortest path. Therefore, without backup multiplexing, the use of multiple backups will lower the network utilization to an unacceptably low level.

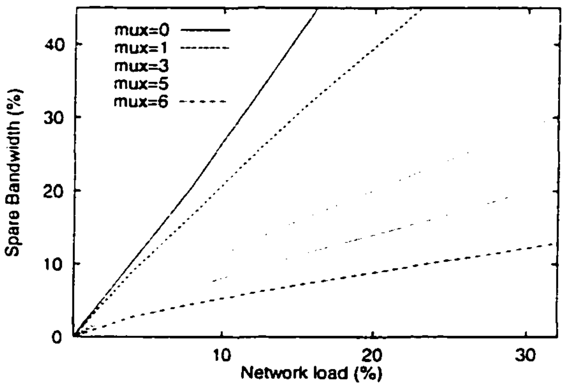
(ii) Comparison between Figure 2.10 and the single backup configuration with 'mux=3' in Figure 2.11 shows that deterministic multiplexing requires smaller spare resources than probabilistic multiplexing, while both guarantee 100 % tolerance to any single link failure. It is expected considering that deterministic multiplexing reserves minimal spare resources to tolerate deterministic failures. The dependability QoS measurement in Chapter 4 will reveal each scheme's capability of tolerating other failure types.

(iii) Under probabilistic multiplexing, the overhead of multiple backups becomes close to that of a single backup, when high multiplexing degrees are used. See the case of 'mux=6' in Figure 2.11 (a) and (b). This result is significant for the dependability QoS comparison

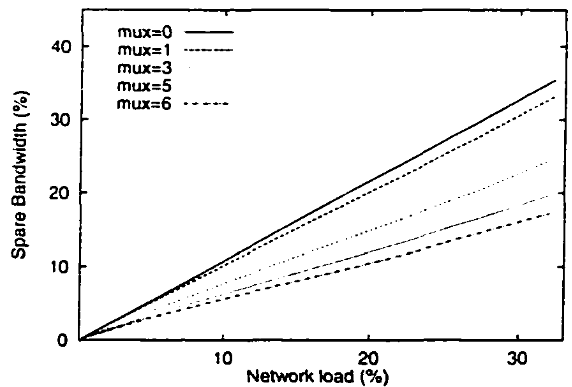
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Single backup in 8 × 8 torus



(b) Double backups in 8 × 8 torus



(c) Single backup in 8 × 8 mesh

Figure 2.11: Average spare-bandwidth reservation under probabilistic multiplexing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

in Chapter 4.

(iv) In the mesh network, the reduction of spare bandwidth by backup multiplexing is not as much as in the torus network. This is because the absence of wrapped links in the mesh network makes the primary-channel paths more concentrated on the central region of the network, thus discouraging multiplexing among their backups.

We performed other simulations with inhomogeneous traffic, such as mixed bandwidth requirements or hot-spots in resource reservation. The results indicate that the efficiency of backup multiplexing is relatively insensitive to network traffic conditions, but is more sensitive to network topology. In general, backup multiplexing is less effective in a sparsely-connected network, because there are a smaller number of possible backup paths, and hence, backups have to share the same links even if they are related, which hinders effective resource sharing among backups. For example, since the routing space of the mesh network is smaller than that of the torus network, the overall performance of backup multiplexing is degraded in the mesh network, as compared to in the torus network.

2.4.3 Comparison of Routing Heuristics

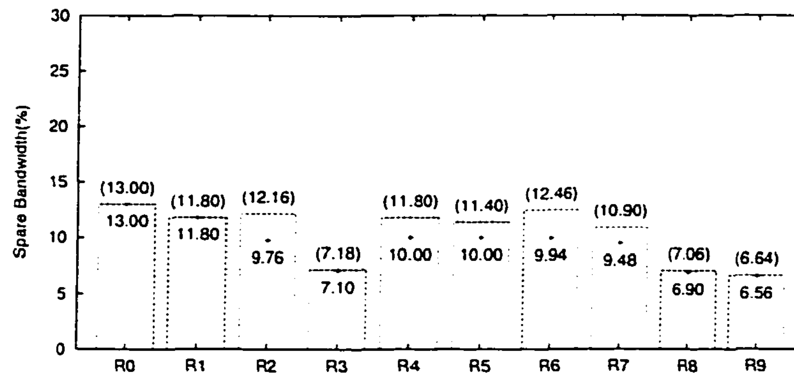
For comparison of routing heuristics, we used smaller simulation networks. In the simulation, the algorithm in Figure 2.1 (a) was used for deterministic multiplexing, and 'mux=3' was applied to the single-backup configuration for probabilistic multiplexing.

The first experiment (Case 1) was conducted in a 5×5 torus network. The bandwidth requirement of each $\mathcal{D}$-connection was identical, so each channel (both primary and backup) required 2% bandwidth on each link of its paths. The end points of each $\mathcal{D}$-connection were evenly distributed across the network. A total of 600 $\mathcal{D}$-connections were established, such that there existed a $\mathcal{D}$-connections between each node pair, i.e., $25 \cdot 24 = 600$.

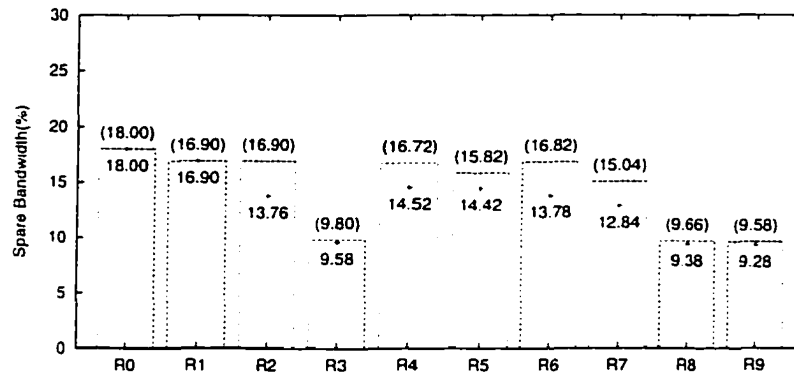
As a result of establishing 600 $\mathcal{D}$-connections, the network load became 30%. Figure 2.12 shows the simulation results for Case 1. The average amount of spare resource required without backup multiplexing was 36% of total network capacity. We call this value 'backup load.' In this case, the overhead of backup channels without multiplexing is $36/30 \times 100 = 120\%$. The average spare bandwidth by the initial routing method is depicted as a bar, while the reduced spare bandwidth after the iterative optimization is labeled with '+'.

Then, we relaxed the homogeneity condition of Case 1 in next experiments. In Case 2, the bandwidth requirement of each $\mathcal{D}$-connection was not identical, but three types of connections were mixed: 1/3 connections of 1% bandwidth, 1/3 connections of 2% bandwidth, and the remaining 1/3 connections of 3% bandwidth among a total of 600 $\mathcal{D}$-connections.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Deterministic multiplexing



(b) Probabilistic multiplexing

Figure 2.12: Case 1 (network load = 30%, backup load = 36%)

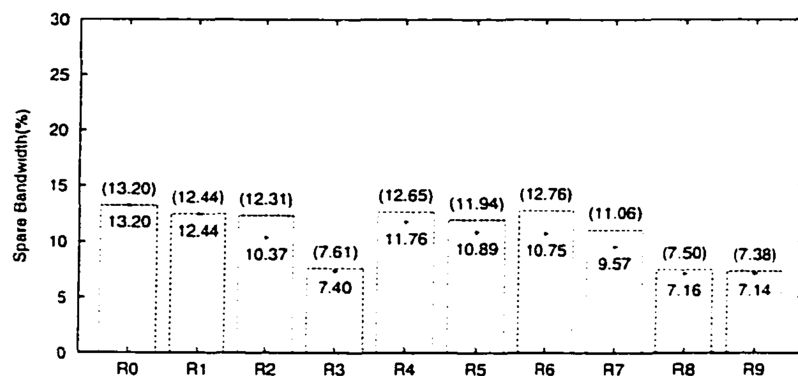
The selection of connection end-points and the network topology were the same as Case 1. Figure 2.13 shows the simulation results.

In Case 3, while keeping other conditions the same as Case 1, the selection of connection source nodes was restricted such that all connections were originated from only 12 randomly-selected nodes among 25 nodes. The destination nodes were evenly distributed as in Case 1, so that the pattern of resource reservation had hot-spots around the selected source nodes. The simulation results are shown in Figure 2.14.

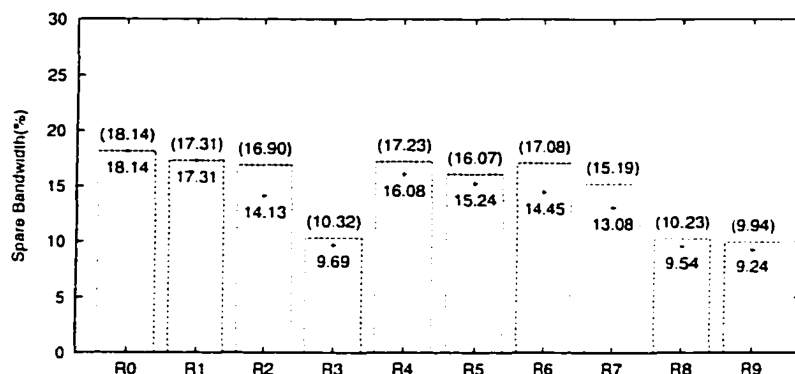
In Case 4, we established 600 $\mathcal{D}$-connections in a 5×5 mesh network. The source/destination nodes were evenly selected, but the bandwidth requirement of each channel was set to 1% instead of 2%, considering the smaller network capacity of the mesh. The resultant network load and backup load were 25% and 30%, respectively. The simulation results are presented in Figure 2.15.

Generally, both deterministic and probabilistic multiplexing reduced spare resources significantly, regardless of the network conditions and routing heuristics. Some observations

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Deterministic multiplexing



(b) Probabilistic multiplexing

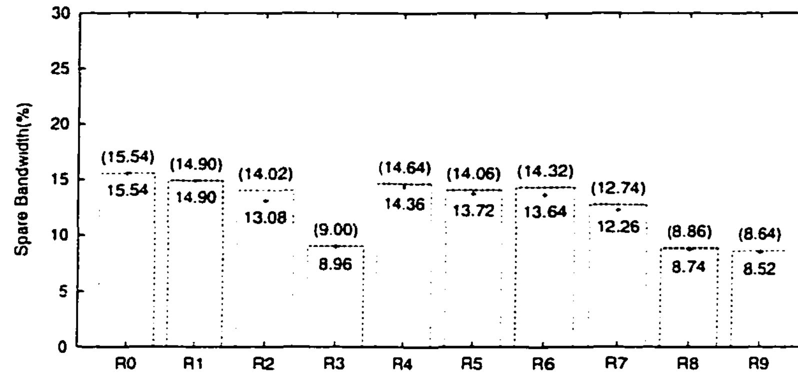
Figure 2.13: Case 2 (network load = 30%, backup load = 36%)

from the simulation results and their implications are summarized below.

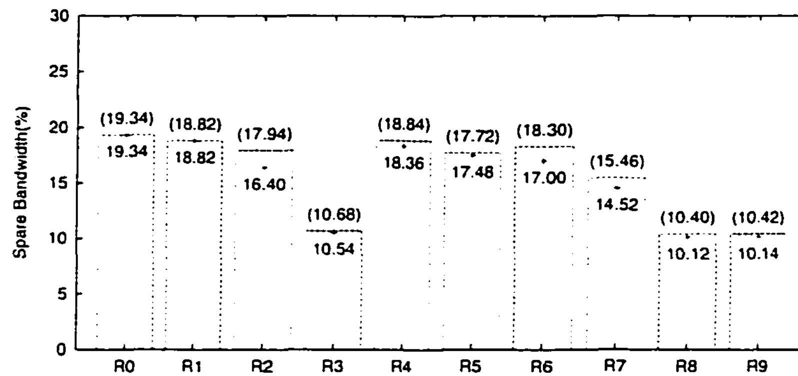
(i) For the initial route selection, those heuristics which include f_3 as a primary component (i.e., R_3, R_8, R_9) outperformed other heuristics by a large margin. Meanwhile the performance of R_2 was not much better than that of R_0 or R_1. These results suggest that the general-purpose heuristics like f_1 and f_2 are less effective for backup routing than those heuristics which directly take advantage of backup multiplexing. In addition, the performance was improved by properly combining basic cost functions. R_5 and R_7 outperformed R_1 and R_2, and both R_8 and R_9 did over R_3.

(ii) Inhomogeneous network conditions degrade the performance of backup multiplexing. With mixed bandwidth requirements (Case 2), the efficiency of backup multiplexing was degraded, though not significantly. Performance loss was also observed in Case 3. It is because channel routes were concentrated on the hot spots, thus discouraging backup multiplexing. The same explanation is possible for Case 4, in which channel routing is more concentrated around the central area of the network than in the torus.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Deterministic multiplexing



(b) Probabilistic multiplexing

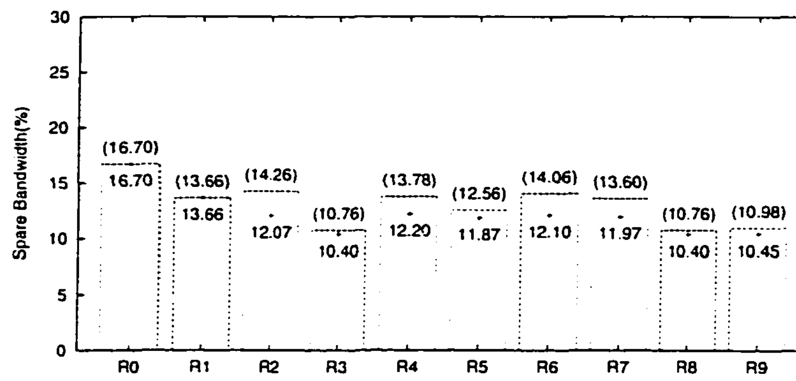
Figure 2.14: Case 3 (network load = 30%, backup load = 36.2%)

(iii) While the impact of iterative optimization was not substantial in the heuristics which use f_3 as a primary cost function, it was more significant in other heuristics. For instance, the spare bandwidth using R_2 yielded a similar level to R_1 before the optimization, but became much smaller than R_1 after the optimization. . The large improvement by iterative optimization implies that the performance of the corresponding heuristic is sensitive to the order of channel establishments. The minor improvement by optimization with f_3-contained heuristics hints that with those heuristics, we can set the interval of periodic optimization to a large value.

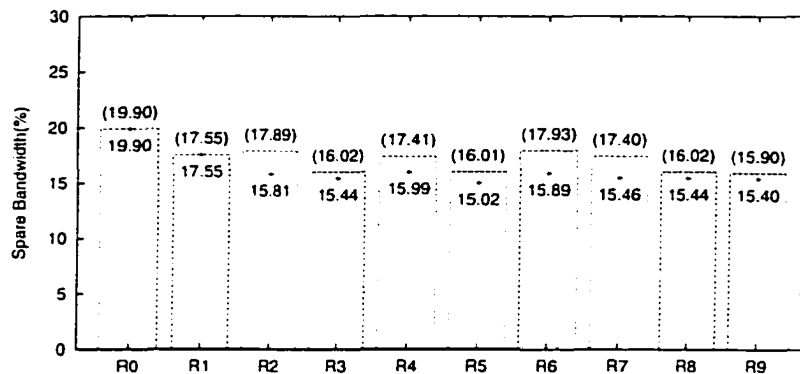
2.5 Summary and Conclusion

In this chapter, we presented the procedure of backup establishment. For each primary channel, backup routes are pre-determined and spare resources are reserved in advance along with the routes for guaranteed failure recovery. To reduce the amount of spare

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Deterministic multiplexing



(b) Probabilistic multiplexing

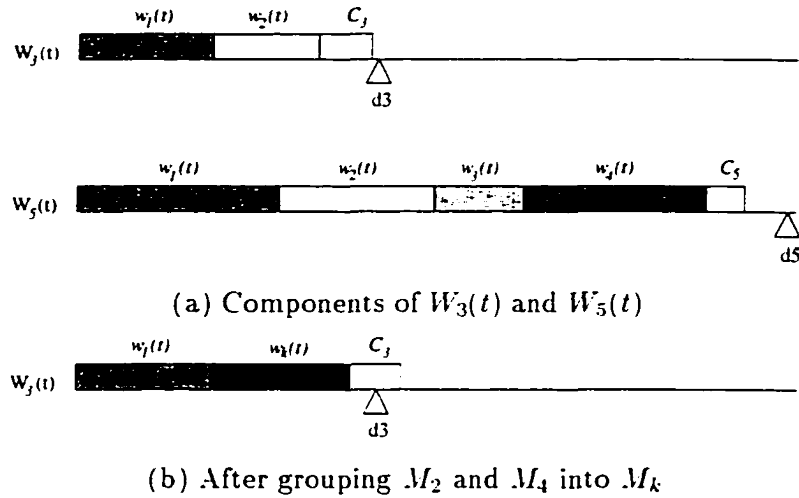
Figure 2.15: Case 4 (network load = 25%, backup load = 30%)

resources, we developed backup multiplexing methods which allow resource sharing among "unrelated" backup channels. By controlling the amount of spare resources, the network can provide different dependability levels to different connections. We also developed the two-step backup routing approach which consists of an initial routing stage and a route optimization stage.

We then evaluated via simulation the effectiveness of the proposed scheme under various network conditions. The simulation results show that the routing heuristics which directly exploit the property of backup multiplexing outperform other more general heuristics like minimum-hop routing or load-balancing routing by a significant margin. With a proper routing heuristic, backup multiplexing enables very efficient resource utilization. For instance, only as low as 20–30% resources for active channels are needed to be reserved as spare resources to guarantee successful recovery from any single link failures in mesh-type networks, while more than 120% spare resources are necessary without backup multiplexing.

There are issues on backup establishment that we have not addressed in this chapter. For

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Components of $W_3(t)$ and $W_5(t)$

(b) After grouping M_2 and M_4 into M_k

Figure 2.16: A counter example

example, when will backup channels be established, together with their primary channels or after establishing primary channel? If backups are established later, will the primary channels start services before backups are established? Another unanswered question is whether backups of the same $\mathcal{D}$-connection are allowed to overlap with each other in their paths. It might be inevitable in a low-connectivity network. Such issues should be dealt with in the stage of actual deployment, considering the application and network characteristics.

Appendix 2.A Inapplicability Proof of Deterministic Multiplexing

To apply deterministic multiplexing, resources should be exchangeable between backup channels. More specifically, the underlying real-time channel scheme should satisfy the following condition:

On a link, multiple channels can be grouped into a single channel and they can be separated into independent channels again if needed.

We will prove that this condition is not satisfied in scheduler-based schemes, by showing a counter example in the RTC scheme [55], a scheduler-based scheme.

In the RTC scheme, a real-time channel M_i on a link ℓ is described by a set of parameters (C_i, d_i, p_i), where C_i is the maximum service time on ℓ for any message of channel M_i, d_i is the maximum permissible delay for messages belonging to M_i at this link, and p_i is the minimum message inter-arrival time. When a set of channels $\{M_i = (C_i, d_i, p_i), i =$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$1, \cdots, m\}$ run on a common link ℓ and the channel set is ordered according to channel priorities with M_1 being the highest-priority, the time required to process a message of M_i in the presence of higher-priority messages is:

$$W_i(t) = \sum_{j=1}^{i-1} C_j \cdot \lceil t/p_j \rceil + C_i.$$

Suppose at $t = 0$ the message arrival of a channel coincides with message arrivals of all other channels of equal or higher priority (i.e., the worst case for M_i). Then, the message deadline of M_i will always be met if $W_i(t) \leq d_i$, where t is the instant of M_i's message arrival.

Assume, for example, there are five channels $M_1, \cdots, M_5$ on ℓ and we want to group M_2 and M_4 into a new channel M_k. M_1's guarantee will not be affected since the new channel M_k need not come before M_1 in the channel-priority list. However, the grouping can be critical to M_3 and M_5. The relations between the system-time requirement and the delay requirement of M_3 and M_5 are illustrated in Figure 2.16 (a), where $w_i(t) = C_i \cdot \lceil t/p_i \rceil$. The priority of M_k should be higher than M_3, since the guarantee for M_2 cannot be achieved regardless of the bandwidth reserved for the channel with lower priority than M_3. The bandwidth reserved for M_k should be greater than that for M_2 and that for M_4. If the bandwidth requirement for M_2 is greater than that for M_4, simply using the parameter set of M_2 as that of M_k will work fine. But in the opposite case, $w_k(t)$ which should be located at the position of M_2 becomes greater than $w_2(t)$ as shown in Figure 2.16 (b). Hence, the QoS guarantee for M_3 will be violated as a result of grouping M_2 and M_4 into M_k.

Thus, there is no general way to find a parameter set which subsumes any of two parameter sets, while preserving the guarantees for other channels. A similar argument can be applied for other scheduler-based schemes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 3

CHANNEL FAILURE DETECTION

In the previous chapter, we described the procedure of connection establishment prior to the occurrence of a failure. Here we focus on failure detection, the first step of run-time failure recovery. We present two *behavior-based* detection schemes: *neighbor detection* and *end-to-end detection*. We then evaluate their effectiveness through fault-injection experiments in a laboratory testbed. In particular, we measure and analyze the coverage and latency of the proposed failure-detection schemes.

This chapter is organized as follows. Section 3.1 presents the failure-detection schemes under evaluation. Section 3.2 gives an overview of the fault injector used for experimental evaluation. Section 3.3 describes the experimental setup. Section 3.4 analyzes experimental results. The chapter concludes with Section 3.5.

3.1 Channel Failure Detection Schemes

Network reliability can be accounted for with message loss rate and recovery delay. Conventional computer network applications such as electronic mail, file transfer, or network file systems do not mandate fast failure recovery, but require reliable (correct) delivery of all messages even if it takes a long time. By contrast, real-time applications require a different reliability goal from conventional non-real-time applications. In this section, we define the reliability goal for real-time communication and present failure-detection schemes to achieve the goal.

3.1.1 Channel Failure

The usual non-real-time datagram service is often called 'best-effort delivery', implying that the network attempts to deliver messages as quickly as possible by using the avail-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

able resources. Reliable transport protocols for non-real-time communication guarantee the eventual (loss-free) delivery of messages between two end-points. Here, the "grain" of failure detection is a message. Message loss can be detected using *behavior-based* failure-detection schemes. For example, with the 'positive acknowledgment' method, the receiver informs the sender of the reception of each message (or a group of messages), so that the sender can detect delivery failures. 'Negative acknowledgment' is an alternative, in which the receiver detects message losses and requests the retransmission of missed messages to the sender or other servers.

The reliability goal of real-time communication is not loss-free delivery of messages. In contrast to non-real-time messages, the content of a real-time message is meaningful only when it is delivered in time. Retransmitting real-time messages in case of their loss or corruption may be of little use, since their deadline is usually too tight to allow retransmission. Therefore, loss-free real-time communication is hard to achieve without relying on expensive forward-error-correction techniques. However, fortunately, many real-time applications do not require such strict reliability as 'no message loss at all'. For example, loss of a couple of frames in video/voice data streams is acceptable. Temporary message losses are also tolerable in real-time control applications because of the 'system inertia' characterized by the control system deadline [86]. Therefore, the grain of failure detection is *channel failures* instead of individual message losses. A real-time channel is said to have 'failed', if the rate of correct[1] message delivery within a certain time interval is below a threshold specified by the application. The same failure manifestation (i.e., error rate) may be acceptable to some applications while it may not to other applications.

Since a long service disruption will cause application failures, effective failure detection with high coverage and low latency is essential in reliable real-time communication. For example, telephone services require fast failure recovery so that humans may hardly notice the service disruption caused by network failures. Telephone networks employ an expensive failure-detection technique using hardware duplication/comparison to quickly detect switching-node failures [94]. A key challenge is how to achieve fast failure detection without relying on special hardware support for real-time communication in packet-switched computer networks. We use behavior-based failure-detection schemes which are similar to the failure-detection schemes used for reliable datagram communication. In what follows, we present two failure-detection schemes to uncover channel failures in real-time communication. These schemes rely on behavior-based techniques can be applied to *any* network.

[1]In terms of both content and timing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

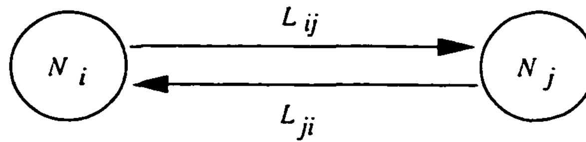


Figure 3.1: Two nodes connected by dual simplex links

3.1.2 Neighbor Detection Scheme

To detect node crash/hang failures or permanent link failures, adjacent nodes periodically exchange *node heartbeats* ("I am alive"). If a node does not receive heartbeats from one of its neighbors for a certain period, it considers the silent neighbor failed and stops sending heartbeats to that node. A heartbeat scheme is generally specified by two parameters: heartbeat interval t_h and a tolerable number, m, of heartbeat misses. When no heartbeat has been received for $(m+1)t_h$ time units, a failure is declared. Heartbeats do not carry any useful information, and regular messages can be used as heartbeats. Explicit heartbeats are sent only if there are no regular messages for a pre-specified period.

In networks where two nodes are connected by dual simplex links, a node cannot tell the difference between the failure of its neighbor nodes and that of the corresponding links by exchanging node heartbeats only. Instead of relying on sophisticated diagnosis, we treat all channels running through the suspected link as faulty. So, when the incoming link from a node fails, the channels on the outgoing link to the node will be considered failed, even if they were healthy. Suppose a link ℓ_{ji} from node N_j to node N_i fails (Figure 3.1). Even though channels on ℓ_{ij}, a link from N_i to N_j, can continue their services, N_i can no longer detect the crash of N_j or ℓ_{ij} without healthy ℓ_{ji}. Such a channel should be torn down even if the channel itself is healthy. This is reasonable because a channel cannot maintain dependable service if the health of the channel cannot be monitored.

3.1.3 End-to-End Detection Scheme

This scheme involves both end nodes of a real-time channel. It works as follows. The source node, whenever necessary, injects a *channel heartbeat* into the channel message stream. A channel heartbeat is a sort of real-time message, and the intermediate nodes on a channel do not discriminate channel heartbeats from data messages. Each channel heartbeat contains the sequence number of the latest data message. In this way, the destination node can monitor the number of data messages lost. If the message-loss rate of the channel exceeds a threshold specified by the application, the destination node declares that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the channel has failed.

For each channel, the source node manages a heartbeat-generation timer which is periodically incremented. The heartbeat-generation timer is reset every time a message (data or heartbeat) is transmitted over the channel. Only when the value of the heartbeat-generation timer reaches the maximum heartbeat interval h_{max}, an explicit channel heartbeat is generated. Therefore, when h_{max} is set to a sufficiently large value relative to the data message interval, explicit heartbeats will seldom be generated due to the (near) periodic nature of real-time messages, thus making the overhead of channel heartbeats small. The h_{max} value of a real-time channel should be chosen to fit the channel's traffic characteristics. In real-time communication, a contract on traffic characteristics and QoS levels is established between an application and the network before messages are actually transferred. The network computes and reserves resources for a real-time channel from this information. Therefore, the h_{max} value of a real-time channel should be larger than the channel's minimum message interval specified in the QoS contract; otherwise, the resources reserved for the channel will not be sufficient to carry both data and heartbeat messages of the channel. The smallest possible heartbeat interval of a channel is therefore determined by the channel's traffic characteristics.

3.2 Fault-Injection Tool Set Development

The use of the proposed failure-detection schemes is not restricted to a particular system/application. However, its effectiveness is intrinsically related to many system-dependent issues, such as basic operating system support, communication protocol implementation, underlying hardware capability, and so on. To fully explore such issues, we have implemented and evaluated the failure-detection schemes on a laboratory testbed.

In this section, the fault-injection tool set that we have developed for experimental evaluation is described. The basic design principles are first explained, and the implementation issues to support the principles are then discussed.

3.2.1 Fault-Injection in Distributed Real-Time Systems

Fault injection has long been viewed as a useful means of testing/evaluating fault-tolerant systems [49].[2] Numerous hardware-implemented fault injectors (HFIs) [5, 38, 67]

[2] Although numerous approaches have been proposed for dependability evaluation, the complexity of distributed real-time systems, due mainly to the intercommunication among nodes, makes most of the techniques intractable except for fault injection into actual prototype systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

have been developed and used for various experiments. However, as the complexity of contemporary computer increases as a result of using highly-integrated VLSI chips, it is becoming more difficult, or nearly impossible, to evaluate dependability with HFIs alone. On the other hand, software-implemented fault injectors (SFIs) [83, 16, 53, 27, 57] have been proposed as less expensive and more controllable alternatives. Although SFI techniques such as overwriting memory or register contents are becoming popular, they still face many difficulties.

Some requirements for fault injection in distributed real-time systems are enumerated below.

1. The fault model should include faults on communication links and communication adaptor circuitry as well as faults inside a processing node such as memory faults, CPU faults, or bus faults.
2. The fault injector should be able to coerce the whole target system to follow a certain intended execution path, which requires it to orchestrate all participants' behaviors. This is not achievable by randomly selecting fault type and injection timing.
3. The operational characteristics of workload should be easily adjustable, especially in terms of the communication activities.
4. Fault injection and data collection must require as little modification to the target system as possible.
5. The intrusion into normal execution by fault injection and data collection should be minimized and isolated to obtain accurate measurements.
6. When clocks are not synchronized in the distributed system under test or clock skews among different processing nodes are unacceptably large, a special time-stamping technique without using system clock should be employed.

To satisfy the above-claimed requirements, we have developed an integrated fault-injection environment called DOCTOR (integrateD sOftware-implemented fault injeCTiOn enviRonment).

3.2.2 Organization of DOCTOR

Figure 3.2 depicts the organization of DOCTOR. In the distributed system architecture assumed, a host computer works as a console node and several processing nodes are connected via a 'system network' and linked to the host node by a 'development network.' Each node can be a bus-based multiprocessor system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

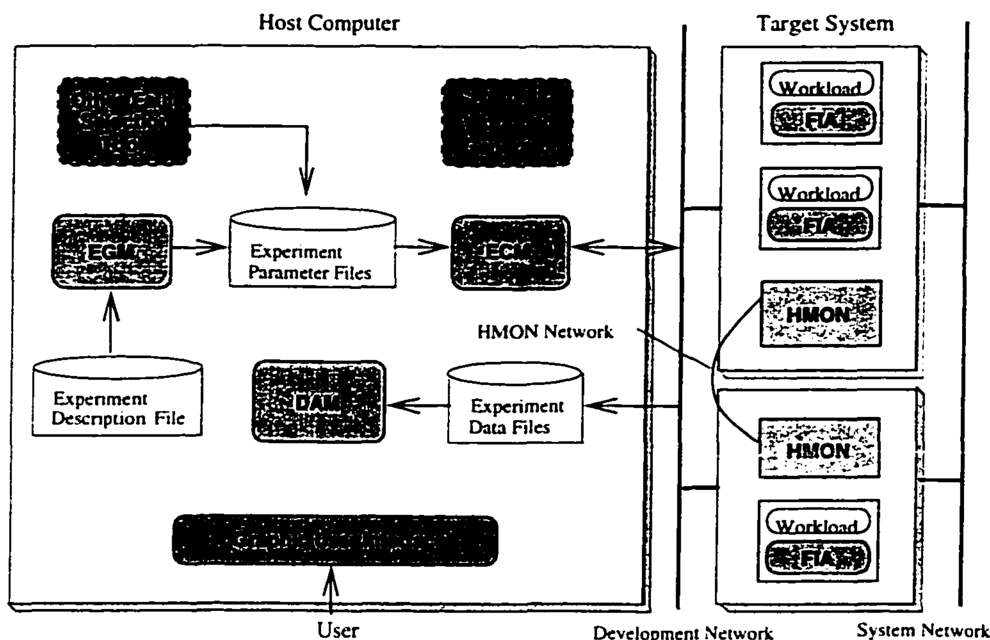


Figure 3.2: The organization of DOCTOR

DOCTOR provides a complete set of tools for automated fault-injection experiments: fault selection tool, fault injector, synthetic workload generator, data monitor, post data analysis tool. The fault-selection tool, EGM (Experiment Generation Module) is responsible for generating a set of fault instances to be injected. ECM (Experiment Control Module) and FIA (Fault Injection Agent) composes the fault injector. ECM is the run-time experiment manager and FIA performs actual fault injections under ECM's control. SWG (Synthetic Workload Generator) is to generate various artificial workloads. The data monitor, HMON, collects the experimental data at run time. DAM (Data Analysis Module) analyzes the collected data off-line after completing the experiment.

The first role of EGM is to analyze the executable images of workloads which will be run on the target system. A workload could be run on a single processing node or be distributed among a number of nodes. The user can use real programs as workloads, or can rely on SWG for artificially-generated workloads. In either case, the symbol-table and object-code information are extracted to be referenced for the purpose of fault-instance generation. The second role of EGM is to parse the 'experiment description file' supplied by the user and to generate the 'experiment parameter file'. EGM generates an experiment parameter file for each node involved in the experiment. Each experiment parameter file contains the experiment plan such as the information about fault types, injection timings, the start/quit of each experiment, and so on.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ECM functions as an external controller of FIA. In a fault-injection experiment, one of the factors that determine the quality of analysis results will be the number of runs.[3] Therefore, it is very useful to automate multi-run experiments. The key problem in experiment automation is the synchronization and re-initialization of several processes involved. The level of re-initialization required depends on the status of the target system after completing each run. In some cases, it may be necessary to reset the whole system, and in some other cases, the restart of workloads may suffice. ECM utilizes the experiment parameter files supplied by EGM and sends proper commands to FIAs about the fault injection plan and the synchronization of each run. ECM also sets up an experiment environment by downloading executable images of the workload, FIAs, and even system software if needed. FIA receives commands from ECM and executes them by injecting faults or making workloads wait/start/stop. It also reports its activities to HMON, such as the injection time, location, type, etc. FIA runs as a separate process (or runs on an interrupt thread depending on fault types to be injected) on the same processor where the workload is running.

One distinct structural feature of DOCTOR is the separation of software components of the host computer from those of the target system. Thus, ECM runs on the host computer while FIA runs on the target system. It has the advantage of reducing the run-time interference with the target system caused by fault injection, because only essential components are executed on the target system. Another advantage is higher portability since the highly system-dependent part is isolated from the rest. It also eases the synchronization of multiple target nodes, i.e., orchestrating the execution of the distributed system under test.

In the remainder of this section, we discuss the implementation of the fault injector and the data monitor in more detail.

3.2.3 Software-implemented Fault Injection

Faults affect various aspects of the system state or operational behavior, such as memory or register contents, program control flow, clock value, the condition of communication links, and so on. Modifying memory contents has been a basic technique of software-implemented fault injectors. Faults are likely to (eventually) contaminate certain parts of memory, so memory faults can represent not only RAM errors but also emulate faults occurring in the other parts of the system. Though the memory fault model is quite powerful, some faults may affect system memory contents in a very subtle and nondeterministic way, and hence, it is very difficult to emulate such a faulty behavior with memory fault injection alone.

[3]Each fault-injection experiment with specific workloads is called a *run*.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For instance, it is difficult to emulate erroneous communication-related functions by just changing memory contents. A more sophisticated fault model is therefore required.

DOCTOR supports the injection of a variety of faults and errors, ranging from low-level faults such as memory or processor faults to high-level errors such as communication errors or system-behavior-level errors (e.g., making processes slow or fast, terminating or suspending processes, corrupting clock/timer services, or corrupting system-call services). In essence, low-level faults are injected into addressable storage space, such as caches, main memory, and CPU registers, while higher-level errors are injected by manipulating message transmission services or system software services. In real-time systems where time is the most precious resource, fault injection must be performed with minimum overhead to the target system. Otherwise, the correctness of the validation itself becomes questionable. To this end, only essential functions are performed on the same processor under test (i.e., by FIA) and relatively simple fault-injection techniques which are described below are employed. (Details on the fault-injector implementation can be found in [44].)

Memory fault: Contents of the cache or main memory are corrupted. The fault injection target can be either explicitly specified by the user, or chosen randomly from the address space using the symbol table and object-code information. For better controllability, DOCTOR allows faults to be injected only into a certain memory section of a particular target task, such as text area, global variable area, or stack/heap area.

Processor fault: CPU faults can occur in data registers, address registers, the data fetching unit, control registers, the op-code decoding unit, ALU, and so on. However, accessibility to hardware components is usually limited. To overcome this difficulty, we inject erroneous effects rather than faults themselves, by emulating the consequences of CPU faults in the architecture-independent level; hence, the contents of CPU registers or instruction codes are used as the targets of fault injection. The timing of fault injection can be randomly selected or can be controlled to be the time when a specific task or instruction is executed. Bus faults can be emulated as well, by corrupting the content of an instruction just before its execution and restoring it after the instruction cycle. Similarly, various types of faults in the processing unit can be emulated, such as ALU errors and instruction fetching unit errors.

Communication error: Errors in communication links can be emulated using a special fault-injection layer inside the protocol stack. The user can define the intended fault behavior, while some pre-defined fault types are supported including message loss,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

corruption, delay, and duplication. Fault injection timing/duration can be specified by either time or message history (e.g., dropping several consecutive messages of a certain type).

Evaluation of a fault-tolerance mechanism needs a systematic fault-injection plan, and thus, the capability of injecting a proper fault instance into a proper location at a proper time is essential. One important aspect of our fault model is its fine controllability of the fault-injection timing. DOCTOR supports three temporal types of faults: transient, intermittent, and permanent. A transient fault is injected only once, and an intermittent fault is injected repeatedly. When injecting an intermittent fault, the user can specify such parameters as the fault recurrence interval by probability distributions. Besides the random injection timing control, DOCTOR allows the user to design fault-injection scenarios with user-specified timing control in either time-based specification or history-based specification.

3.2.4 Non-intrusive Data Monitoring

The data monitoring instrument to trace the run-time behavior of the system under test or to collect various dependability parameters is an important component in fault-injection experiments. Conventional hardware monitors probe a fixed number of physical signals to obtain low-level data. Though such hardware monitors eliminate interference in the monitored system, they lack the ability of catching a wide range of software-level events. Meanwhile, software-implemented monitors can cause unpredictable interferences. Such software monitors usually rely on the operating system service to snapshot the timing of a certain event occurrence. Therefore, the overhead of intensive time-stamping may sometimes be unacceptable for real-time systems. The problem of time-stamping is complicated even further in distributed systems where events can occur asynchronously in different processors. Comparison of time-stamps is meaningful only if the clock services are synchronized with sufficient accuracy. It is not a practical assumption for some experiments since the typical clock skew bound is in the order of millisecond, while the required time-stamp resolution may easily reach the microsecond order.

To overcome the limitation of pure hardware monitors and software monitors, we have developed a hybrid data monitor.[4] We envision the data collection process consisting of three steps: event probing, time-stamping, and event logging. Event probing is done by

[4] Young utilized a hardware/software hybrid monitor [105], where a commercial logic analyzer was used to probe hardware signals and a software monitor checked the system status to assist the control of hardware monitor. However, this approach is more oriented toward hardware monitors, and its major concern lies in the measurement of low-level system activities.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

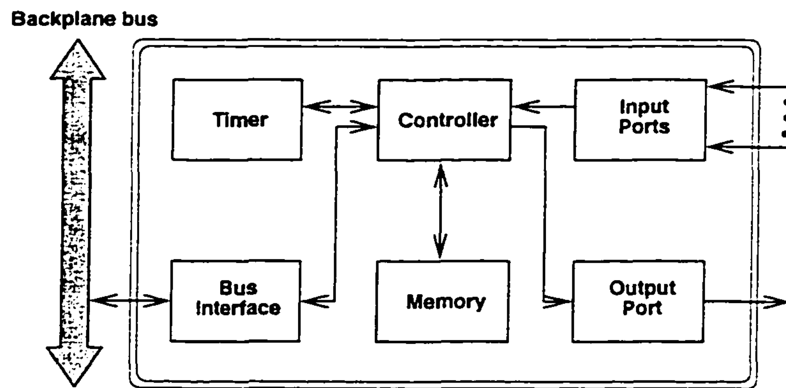


Figure 3.3: Architecture of HMON

the special code inserted into the monitored objects, such as system software, application program, communication protocol stack, or fault-tolerant mechanisms. Events may include system call invocations, context switches, interrupts, fault injections, fault detections, and so on. The probed events are time-stamped and logged by the hardware monitor. A hardware monitor is assigned to each processing node in a distributed environment, and a special time-stamping mechanism is devised to guarantee that the events from distinct processors can be compared as if their time-stamps are generated by a centralized clock.

HMON is implemented as a separate board to reduce the hardware dependency on the processor board and to allow a HMON to be shared by several processor boards on the same backplane bus[5]. The logical architecture of HMON is shown in Figure 3.3. In the current implementation, 1 Mega SRAM is used for data logging purpose, and 25 *nanosecond* resolution time-stamp is generated by internal hardware timer. The function of HMON is passive. It has a memory-map interface, so when a certain address in the memory-map is accessed, it performs a pre-specified action. For example, when a memory write operation is requested at specific addresses, the data content is combined with a time-stamp to make a logging entry, and the new entry is stored into HMON memory. The data collected in HMON memory is dumped to the disk during experiments or after experiments. (Details on the HMON implementation can be found in [39].)

For distributed data collection, a dedicated simple network, called 'HMON network,' is used to connect HMON boards. Each HMON has one output port and 7 input ports to communicate with other HMONs, and two HMONs are linked using a twist-pair cable. Rather than synchronizing the internal timer of each HMON, we chose an indirect way. When a signal comes through an input port, the identifier of the port is written with current time-

[5]Currently a VME bus.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

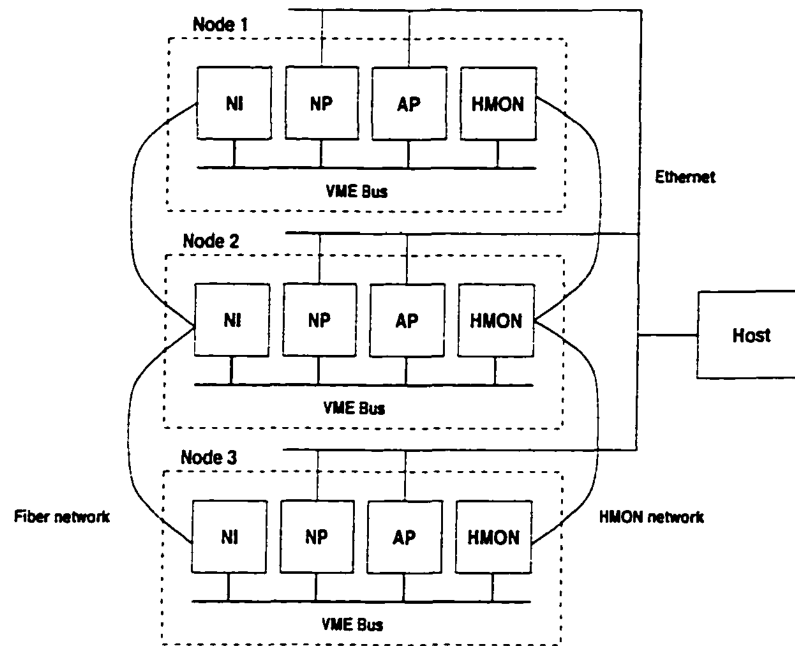


Figure 3.4: Configuration of the experimental platform

stamp into HMON memory. The HMON which generates the signal also logs the event with its time-stamp, so that in the post analysis all the events can be totally ordered by time-stamps.

3.3 Fault-Injection Experiment Setup

In this section, we describe the hardware/software configuration of our experimental platform and the specification of fault-injection experiments which are conducted on the platform.

3.3.1 Testbed

As shown in Figure 3.4, the experimental platform consists of three nodes, Nodes 1–3. Each node is a VME bus-based multiprocessor system with Motorola 68040 microprocessors. In each node, a CPU board (labeled as NP) is dedicated to communication processing, while a separate CPU board (labeled as AP) is used for application processing. As a communication fabric between nodes, a network interface board (NI) featuring the 'ANSI Fiber Channel 3.0 standard'[1] is equipped with each node. In addition, a HMON board is added to each node for data collection. Node 1 and Node 2 are connected by two simplex links (i.e., optical fibers), one for each direction. The same type of connection exists between

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Node 2 and Node 3. The host machine is connected to nodes through an Ethernet (i.e.. development network). Nodes are not equipped with disks. and application/system software is downloaded from the host machine.

An extended version of the pSOS^{+m} real-time OS [93] is used for AP's system software. The AP-side software is not important in our experiment, since APs run very simple applications which request message delivery to the associated NP. and retrieve messages received by the NP. NP employs a derivative of x-kernel 3.1 [77] as a system executive and a substrate for building the protocol stack. Since NPs do not run user tasks. we disabled the virtual address management of x-kernel. Thus. all tasks in NP are executed within a single (kernel) address space. Memory protection of x-kernel was also disabled to minimize the overhead.

Each NP features the real-time communication scheme described in [55]. Figure 3.5 gives an overview of the NP protocol stack. The protocol stack includes protocols for application program interface (API). network management (NM). remote procedure call (RPC).[6] transport-level fragmentation (FRAG).[7] network/data-link layer (HNET). and the device driver for network interface boards (DD). The API protocol exports routines that applications can use to set up/tear down real-time channels and perform data transfer on the channels. The RPC protocol is used by the NM protocol for transmitting channel establishment/teardown messages. The HNET protocol provides the function of the network layer and part of the data-link layer. The run-time message scheduler is also implemented in it.

The NP system software, x-kernel. uses a non-preemptive scheduling policy with 32 priority levels for task scheduling, and its protocol processing is based on the *process-per-message* model. Whenever a message arrives at a network device or needs to be transmitted into the network. a process (or thread) is created to shepherd the message through the protocol stack; this eliminates extraneous context switches encountered in the usual process-per-protocol model. Once a protocol thread is scheduled, it runs without preemption until completion of protocol processing. While the process-per-message model suffices for best-effort messages, it introduces complexity for maintaining QoS guarantees and performing traffic policing. For this reason. we implemented the run-time message scheduler as a special thread that is created at system startup and runs at the highest-priority level. Implementation details can be found in [70].

A node-heartbeat generator also runs as a separate task. It is periodically executed and

[6] A modified version of x-kernel's CHAN protocol.
[7] A modified version of x-kernel's BLAST protocol

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

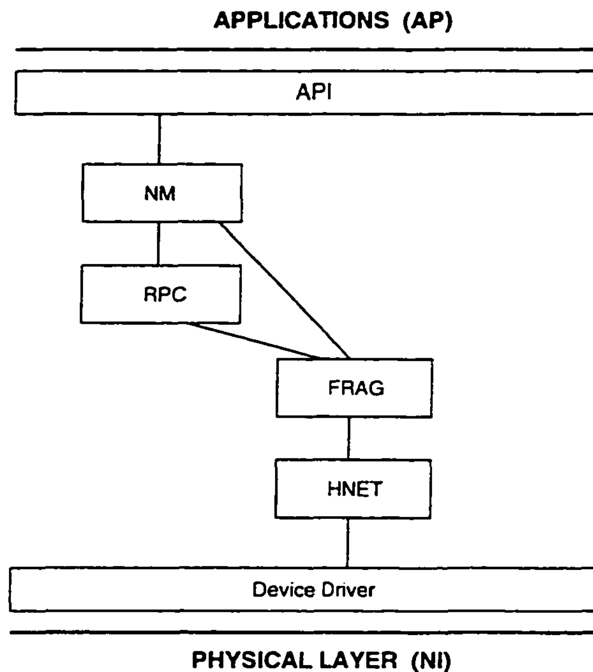


Figure 3.5: The protocol stack in NP

checks the values of special flags, each of which is associated with a link and is set whenever a message is transmitted over the link. If the flag is not set when a heartbeat generator is invoked, a new heartbeat is generated and sent over the corresponding link. The heartbeat checking is done in a similar way. The heartbeat checker regularly checks the set of flags which are set when a message is received over the associated link. The heartbeat checker resets the flags after checking. If the heartbeat checker finds a flag not set for a longer than the specified period, it declares a failure detection. Channel heartbeat generation and checking is done similarly.

3.3.2 Experiment Goal

The end-to-end detection scheme will uncover *all* channel failures, so the main concern is its detection latency. Under this scheme the destination node has to wait for the loss of a specified number of messages before declaring a failure occurrence. Thus, the failure detection latency is tightly coupled with the definition of channel failure.

The neighbor detection scheme has a much weaker dependency on the underlying failure definition than the end-to-end scheme, because it monitors the behavior of a neighbor node, rather than that of a real-time channel. Therefore, one can achieve smaller detection latencies with the neighbor scheme than with the end-to-end scheme. However, in the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

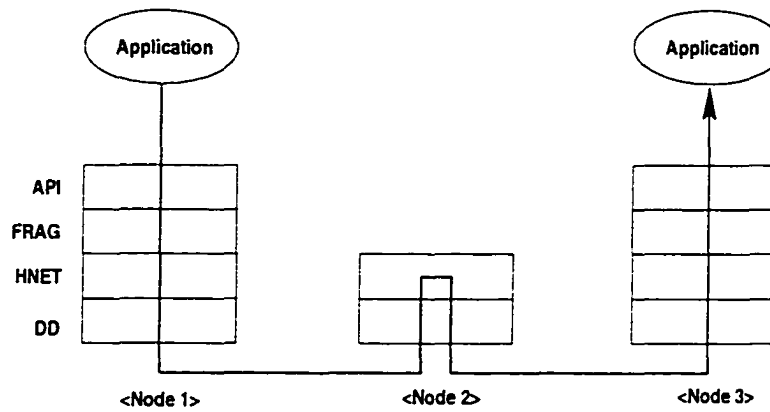


Figure 3.6: Real-time message passing at run-time

neighbor scheme. there is a possibility that a node is not operating correctly in terms of message forwarding. but still generates node heartbeats or propagates part of regular messages. In such a case. the neighbor scheme will result in a less than perfect detection coverage. Even though the faulty node becomes silent eventually. the detection latency may be larger than that of the end-to-end scheme.

The implementation of the neighbor detection scheme may influence its performance. For instance. instead of running as a separate task. the heartbeat generator can run on top of the clock interrupt thread. However. we have not used this implementation option because of the following two drawbacks. First. execution time of the clock interrupt handler is extended. during which other interrupts are disabled. Second. even when the OS task scheduler or the message scheduler hangs, heartbeats will still be sent out. which will lower the detection coverage. Not only the mechanism of generating heartbeats but also the transmission path of heartbeats is important. In general. the closer the transmission path of heartbeats is to that of regular real-time messages. the higher the detection coverage will be. In the experiment. we tested two heartbeat-transmission mechanisms:

(i) Sending heartbeats as best-effort messages (option 1).

(ii) Sending heartbeats as real-time messages (option 2).

Figure 3.7 depicts the transmission path of heartbeats in two implementations. At an intermediate node, only the bottom two protocols. HNET and DD. are executed for message passing, as illustrated in Figure 3.6. NM and RPC protocols are used for channel establishment and API and FRAG protocols are executed only at end-nodes.

The overall experimental goal is to measure the performance of the neighbor detection scheme (i.e.. its detection coverage and latency) and to examine if and how much sending

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

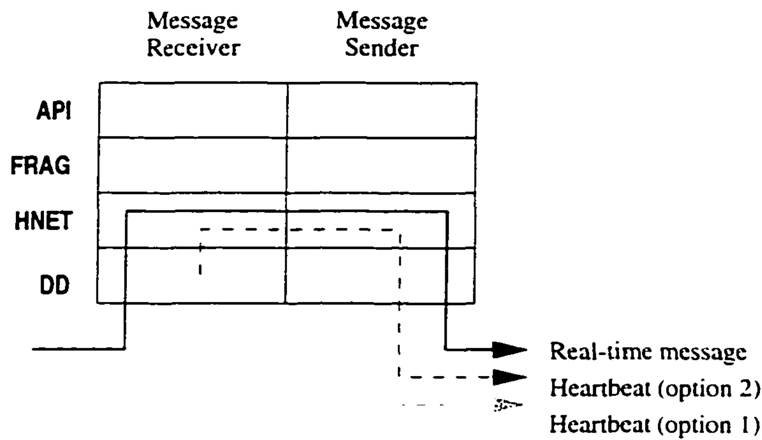


Figure 3.7: Two implementations for heartbeat transmission

node-heartbeats as real-time messages can enhance the detection coverage of the neighbor detection scheme.

3.3.3 Experiment Specification

In this section, we formulate the fault-injection experiment using the ***FARM*** specification model [5]. There are four major attributes of a fault-injection experiment: a set of faults $\boldsymbol{F}$, a set of activations $\boldsymbol{A}$, a set of readouts $\boldsymbol{R}$, and a set of derived measures $\boldsymbol{M}$.

As the $\boldsymbol{A}$ attribute which specify the workloads used to exercise the system, real-time channels were established from Node 1 to Node 3 through Node 2 on the testbed. The end-to-end delay requirement of the real-time channels was set to 30 msec and the application program generated real-time messages regularly once every 50 msec without any burst. Since messages were generated periodically, no 'channel heartbeat' needed to be injected into the message stream of real-time channels. A channel failure is said to occur if no message is delivered for more than 400 msec. The interval of 'node heartbeats' was set to 60 msec and the tolerable number of (node) heartbeat misses was set to 2. Hence, if heartbeats are not received for three consecutive heartbeat intervals (i.e., 180 msec), a failure is declared (detected).

As the $\boldsymbol{F}$ attribute, three classes of faults were injected: memory faults, CPU register faults, and communication faults. Memory faults were injected only into the text area of the target programs at randomly-selected times. The effects of memory faults in the data area can be covered largely by CPU register faults, since memory variables are typically loaded into registers. To emulate CPU faults, the values of data/address registers were corrupted. Time-driven triggering was not very effective in injecting CPU register faults in our platform.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

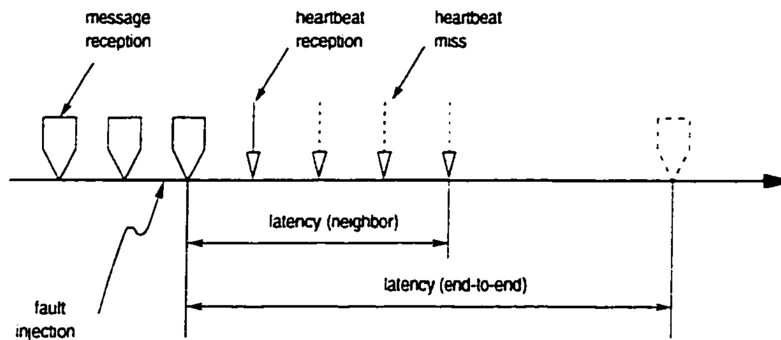


Figure 3.8: Failure-detection latency

It is because message threads are created and destroyed very quickly and thus, CPU is idle for a large portion of time, unlike usual fault-injection experiments in which application programs run continuously. To increase the fault-activation rate, a different method was used to trigger a CPU fault-injection: i.e., when a certain instruction is executed, a fault is injected into the register used by the instruction. In addition, we injected faults into CCR to study the effects of faults in control registers. To maximize the chance of fault activation, CCR faults were injected when conditional branch instructions were executed. We also emulated the faults in (physical) communication links. The fault-injection layer was inserted between DD and HNET. Since the results of such communication errors as message drop or message data corruption are straightforward, we injected corruption errors into the message header part.

All of the injected faults were transient single-bit toggle faults. According to the common practice in software-implemented fault injection, we use the term 'transient' to mean the opposite to 'permanent'. For example, register faults are transient in that the corrupted register contents can be overwritten by the subsequent instructions. A communication fault corrupts the header of only one message, so it is also transient. The faults injected into memory are also transient, but, since the program text area is corrupted, they will have properties similar to permanent memory faults. Since we are interested in detecting failures of real-time channels after their establishment, faults were injected only into the NP of the intermediate node, Node 2, in which only HNET and DD protocols are executed. In addition to HNET and DD protocols, two OS modules, the task manager (TM) and the clock service (CS), were included in fault-injection targets.

As the ***R*** attribute, HMON collects time-stamped data of such events as message generation, message relay, message reception, fault injection, failure detection, and heartbeat generation/reception. Note that we need the distributed monitoring feature, since these

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

events are generated by different processing nodes.

After each experiment, we calculated (i) the channel-failure rate, and (ii) failure-detection coverage/latency of two detection schemes. These correspond to the M attribute. First, the result of each fault injection is classified into 3 categories: *no error, tolerable error, channel failure*. The channel-failure rate is then computed as the ratio of the third case to all cases. The failure-detection coverage is the percentage of detections among the runs in which channel failures had occurred. When calculating the coverage of the neighbor scheme, we excluded the case when the neighbor scheme detects a failure which had already been detected by the end-to-end scheme. The failure-detection latency is computed as the duration between the time of the last message delivered correctly and the time of failure detection. (Figure 3.8 illustrates the failure-detection latency of two detection schemes.)

3.3.4 Fault-Injection Experiment Sequence

The experiment controller, ECM, on the host machine communicates with FIAs in the target nodes through an Ethernet. To avoid the interference caused by fault injection, the function of the FIA in NP is minimized; the communication with ECM is done by a FIA proxy in AP, and the FIA in NP communicates with the FIA proxy through the VME bus. The FIA proxy is also responsible for controlling HMON and uploading the collected data by HMON.

To synchronize the start and the end of each run, dummy FIAs and FIA proxies were also executed at Node 1 and 3. The experiment was fully automated, so that multi-run experiments were done without human intervention. Each run consists of 6 sequential steps:

Step 1: EGM generates a fault-injection script for the run. (Scripts for multiple runs can be generated at once.)

Step 2: ECM downloads system software (including communication subsystem) and fault-injector software to NPs and APs, and remotely boots the target system.

Step 3: When the connection between ECM and FIA proxy is ready, ECM sends the current fault-injection script to FIA proxy.

Step 4: FIA waits until applications establish real-time channels.

Step 5: After the message transmission is started, FIA injects a fault (or multiple faults) and HMON collects data.

Step 6: When the pre-specified experiment duration is reached, the collected data is uploaded to ECM, and FIA proxies reset all nodes for the next run.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.4 Analysis of Experimental Results

In this section, we analyze the experimental results collected from more than 15,000 runs. Each run took about 70 seconds: 40 seconds for experiment setup + 30 seconds for executing the experiment. The same experiments were conducted with two different implementations of the neighbor detection scheme. In the first set of experiments (Experiment-1), node-heartbeats are transmitted as best-effort messages. In the second set of experiments (Experiment-2), node-heartbeats are transmitted as real-time messages over a special-purpose real-time channel.

3.4.1 Fault Manifestations

Tables 3.1 shows the composition of fault manifestations when node heartbeats are sent as best-effort messages. The fault manifestations differed by fault types and locations. For example, fault injection into address registers resulted in a higher channel-failure rate, while the fault injection into data registers resulted in a lower channel-failure rate. Faults injected into CS had a relatively low activation rate, as compared to TM. The fault manifestations when node heartbeats are sent as real-time messages are summarized in Table 3.2. The fault instances injected in Experiment-2 might be different from Experiment-1, because fault instances were randomly generated for each run. As a result, though the overall trend of fault manifestations is similar, the values in Table 3.2 do not exactly match the values in Table 3.1. Even if the same fault instances were injected, we might not get the same results. In general, the repeatability of fault-injection experiments is not very high, because the granularity of controlling the injection timing is usually too coarse to inject faults at the exactly same moment repeatedly. Moreover, the target system may follow slightly different execution paths in different experiments, particularly in distributed systems. Thus, the obtained composition of fault manifestations may not be a representative result. More meaningful implication of this experimental result lies in the pattern of system behaviors caused by faults.

Interestingly, while many of the tolerable errors were due to 'message deadline violations', some of them were due to 'message losses' and very few 'message data corruption' errors were found. This is because the message data part is stored by the device driver when the message arrives and is not copied or modified by other protocols in x-kernel, so that the chance of message corruption is low. In some cases, the delayed messages were eventually delivered, and the system behavior returned to normal. In some other cases, the abnormal

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Channel failure (cf); tolerable error (ok); no error (no)

		HNET	DD	TM	CS
Memory	runs	392	387	393	387
	cf	28.1%	9.3%	17.8%	15.8%
	ok	0.8%	0.3%	0.0%	0.3%
	no	71.2%	90.4%	82.2%	84.0%
Dx	runs	387	373	343	373
	cf	17.1%	13.1%	11.4%	2.4%
	ok	3.1%	8.8%	0.0%	0.0%
	no	79.8%	78.0%	88.6%	97.6%
Ax	runs	389	359	379	304
	cf	68.1%	59.9%	61.5%	45.5%
	ok	6.2%	6.4%	0.0%	17.1%
	no	25.7%	33.7%	38.5%	37.5%
CCR	runs	307	289	340	345
	cf	7.2%	13.8%	44.1%	0.0%
	ok	18.6%	17.3%	9.4%	0.9%
	no	74.3%	68.9%	46.5%	99.1%
Msg	runs	386			
	cf	16.8%			
	ok	0.0%			
	no	83.2%			

Table 3.1: Fault manifestations (Experiment-1)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Channel failure (cf); tolerable error (ok); no error (no)

		HNET	DD	TM	CS
Memory	runs	396	398	398	399
	cf	31.3%	10.3%	16.8%	18.5%
	ok	2.3%	0.0%	0.0%	0.8%
	no	66.4%	89.7%	83.2%	80.7%
Dx	runs	398	385	327	382
	cf	13.3%	11.7%	13.1%	2.6%
	ok	3.5%	3.4%	0.6%	0.0%
	no	83.2%	84.9%	86.2%	97.4%
Ax	runs	396	388	391	318
	cf	74.2%	62.9%	70.6%	41.5%
	ok	2.3%	4.9%	2.3%	15.7%
	no	23.5%	32.2%	70.6%	42.8%
CCR	runs	340	318	334	357
	cf	12.6%	20.4%	52.7%	0.6%
	ok	10.9%	6.0%	11.7%	0.8%
	no	76.5%	73.6%	35.6%	98.6%
Msg	runs	397			
	cf	17.4%			
	ok	0.0%			
	no	82.6%			

Table 3.2: Fault manifestations (Experiment-2)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

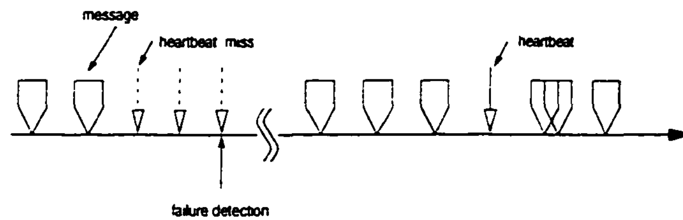


Figure 3.9: An example of false alarm

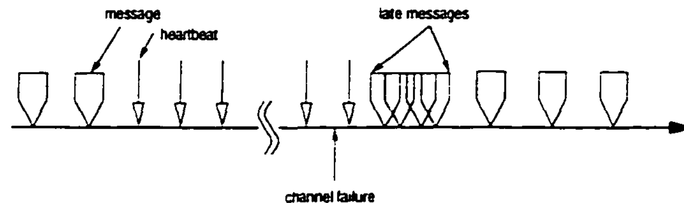


Figure 3.10: An example of channel failures undetected by the neighbor scheme

behavior keeps on occurring and vanishing without causing a channel failure or heartbeat interruption. When such an abnormal behavior continues for longer than three consecutive heartbeat intervals but less than 400 msec. the neighbor scheme signals a failure detection. even if no channel failure actually occurs. We call such a situation *false alarm*. An example of a false alarm case is depicted in Figure 3.10. According to our experimental results. false alarms are rare but do occur. If we reduce the number of missing heartbeats before declaring a failure in the neighbor scheme. the likelihood of false alarm will increase. The possibility of false alarm bars the neighbor scheme from quickly declaring a failure upon miss of a single heartbeat. especially when the heartbeat interval is small.

3.4.2 Failure Detection Coverage

We are most interested in the failure-detection coverage and latency of the neighbor scheme. because the end-to-end scheme has always a perfect coverage. Sometimes channel failures went undetected by the neighbor scheme. The measured detection coverage and latency of the neighbor scheme from Experiment 1 are summarized in Table 3.3. A typical failure symptom which was not detected by the neighbor scheme is illustrated in Figure 3.10. where faults cause only data messages to be delayed or dropped.

It would be interesting to compare our results with the experimental results about 'fail-silent' behavior of computer systems. For example. $(72.6 + 0.1)/(100 - 18.2) = 88.8$ % of failures were reported in [80] to have been detected by the CPU-inherent detection mechanism. The discrepancy of the detection coverage can be explained as follows. First,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Coverage (c): latency mean (l_m) in msec

		HNET	DD	TM	CS
Memory	c	87.3%	94.4%	95.7%	85.5%
	l_m	198.3	192.8	201.7	193.0
Dx	c	72.7%	100%	97.4%	44.4%
	l_m	192.5	200.4	194.5	191.5
Ax	c	98.9%	100%	91.8%	92.0%
	l_m	193.7	193.2	193.4	194.6
CCR	c	68.2%	92.5%	71.3%	N/A
	l_m	196.3	196.8	194.3	N/A
Msg	c	100%			
	l_m	193.5			

Table 3.3: Detection coverage and latency of the neighbor scheme (Experiment-1)

the two used different fault-injection methods. In [80], a hardware-implemented (pin-level) fault injector was used, so the fault injected at a CPU pin for two memory cycles can be manifested as several errors at the level which a software-implemented fault injector deals with. Moreover, the faults forced into control signal pins will make more pronounced impacts on the target system than the data-level errors. For instance, even in our experiments, faults in address registers were detected well. By contrast, CCR errors resulted in a low coverage. It is because CCR errors cause incorrect control flow which is difficult to detect without special hardware support (e.g., a watchdog processor), while errors in address registers can be detected by CPU-intrinsic fault-detection mechanisms like bus error, unaligned memory access, etc. Second, the underlying workload (i.e., system software and application program) was different. Computationally-intensive workloads (e.g., sorting, searching, matrix multiplication, etc.) were executed in the experiments of [80]. The dependency of fault-tolerance measures on workload has been reported by several researchers, e.g., [14, 17]. In addition, our target system, like most other real-time systems, is not equipped with memory-protection capability. The experimental result in [67] indicates that memory protection can enhance detection coverage up to 15%. Third, the two used different fault-detection schemes. Though the neighbor detection scheme will detect all crash failures, we excluded the late detections by the neighbor scheme from the coverage calculation. Thus, the failures which are detected by the neighbor scheme later than the end-to-end scheme

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Coverage (c); latency mean (l_m) in msec

		HNET	DD	TM	CS
Memory	c	96.8%	100%	100%	98.6%
	l_m	197.1	198.0	196.5	197.5
Dx	c	79.2%	97.8%	100%	70.0%
	l_m	201.5	204.1	201.3	203.3
Ax	c	100%	100%	100%	96.2%
	l_m	204.1	204.6	203.6	203.6
CCR	c	90.7%	100%	100%	100%
	l_m	206.9	209.7	203.4	205.5
Msg	c	100%			
	l_m	196.8			

Table 3.4: Detection coverage and latency of the neighbor scheme (Experiment-2)

were treated as undetected. The implementation of heartbeat generation and transmission may also affect the detection coverage.

In fact, the difference in the heartbeat transmission mechanism affects the performance of the neighbor detection scheme. Comparison between Table 3.4 and Table 3.3 indicates that the detection coverage observed in Experiment-2 is significantly higher than that of Experiment-1. A near-perfect coverage was observed for most cases in Table 3.4. It was possible because, while heartbeats shared only part of execution path with real-time data messages in Experiment-1, they went through the same execution path as real-time data messages in Experiment-2 (see Figure 3.7). In Experiment-2, a special-purpose real-time channel was dedicated to heartbeat transmission. Considering the sharing of program codes in processing real-time messages belonging to different real-time channels, if a fault affects a channel, it is very likely to affect other channels as well. Recall that in x-kernel, whenever necessary, a shepherd thread is spawned to process a new message, and all shepherd threads execute the same (protocol-processing) program code. The property of program code sharing in message processing is not limited to our platform and is common in most conventional protocol implementations such as in BSD Unix.

Generally, if the execution of a thread is faulty because of faults in the local data of the thread, only the message associated with the thread will be affected, not all messages of a channel. This will be most likely to end up with a transient error. If the source of the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

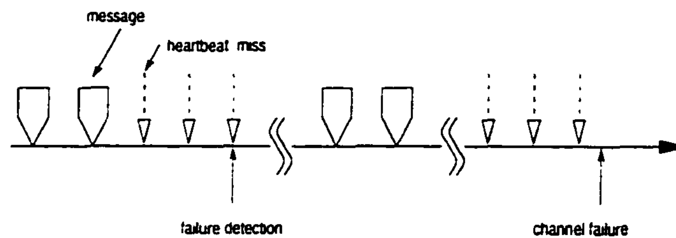


Figure 3.11: An example of early failure detection

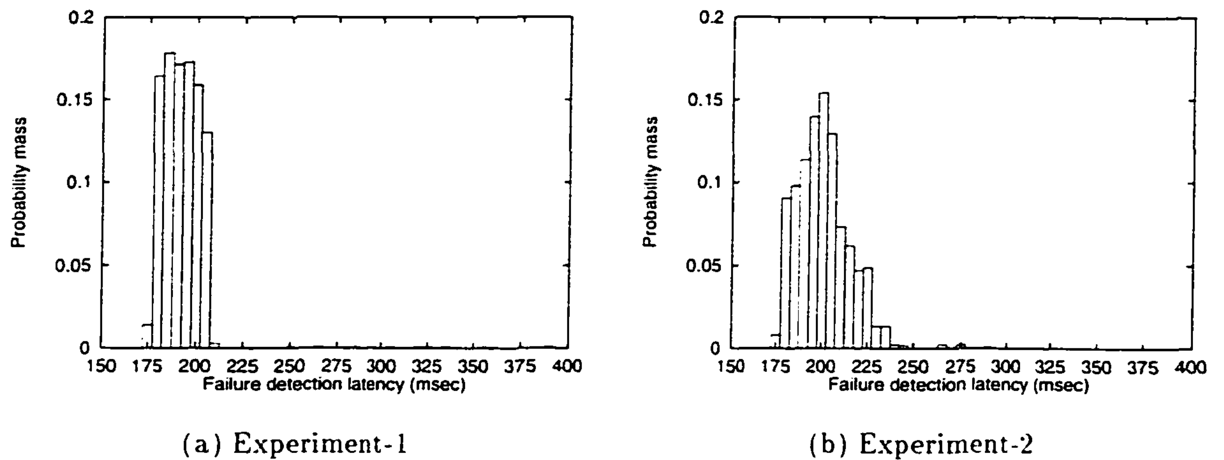
incorrect execution is in a globally-shared component like program code or system software. all messages of all channels will be affected. which will not hurt the detection coverage of the neighbor scheme. Only when the channel-specific data (i.e., a link-deadline) is corrupted. only the corresponding channels will fail while other channels will not. However. according to our experimental results. the chance of such occurrences is very low.

3.4.3 Failure Detection Latency

In the current experimental setup. the end-to-end scheme will always result in a constant latency. 400 msec. The expected detection latency of the neighbor scheme is 180 msec. i.e.. duration of three consecutive heartbeat losses. However. the actual detection latency measured was about 190–200 msec on average. where some latencies were over 250 msec. There are three reasons for this. First. the heartbeat generator and the heartbeat checker are periodically invoked at every 30 msec. As a result, heartbeats may not be generated immediately after there is no traffic for 60 msec. and a failure may not be declared immediately after the passage of 180 msec without any traffic. Second. faults may delay or drop messages for some time before a complete channel failure. Late real-time messages are the same as message losses from the application's point of view. while they are considered as implicit heartbeats from the heartbeat checker's perspective. Thus. delayed messages may extend the detection latency. Third. the fault-propagation delay can be another reason for a long detection latency. if the fault-propagation delay is different for real-time messages and heartbeats. In an extreme case. faults affect real-time messages quickly and affect heartbeats slowly. which will result in a long detection latency. The second and third reasons can explain the occasional occurrences of long detection latency over 250 msec.

Meanwhile. negative detection latencies were observed, though rarely. The negative detection latency means that a failure is detected before it actually occurs. It happens when message transmission is stopped for longer than 180 msec (i.e.. failure detection) and then message transmission is resumed but the channel eventually fails. In such a case. the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Experiment-1

(b) Experiment-2

Figure 3.12: Comparison of detection latency distribution

detection precedes the channel-failure occurrence. Figure 3.11 illustrates this case. Note that the situation is the same as false alarm except the eventual failure.

The comparison of fault detection latencies shows that Experiment-2 resulted in a slightly larger average than Experiment-1. It is because the population of long latencies was increased in Experiment-2 as compared to that of Experiment-1. Figure 3.12 compares the overall distribution of detection latencies by two experiment setups.

3.4.4 Workload Dependency

We also conducted fault-injection experiments by setting up multiple real-time channels. The existence of peer channels did not have a significant impact on the performance of the failure detection schemes. When a channel suffered an abnormal behavior, other channels experienced similar symptoms in most cases.

3.5 Summary and Conclusion

In this chapter, we investigated the effectiveness of two failure-detection schemes — neighbor and end-to-end detection — for real-time communication services. The idea of the neighbor scheme has been used widely in (non-real-time) computer networks to monitor the health of network nodes, and the end-to-end scheme resembles the conventional reliable transport protocols. Our experimental results on a laboratory testbed, which was designed without any particular consideration for fault-tolerance, have indicated that the neighbor scheme detects a significant portion of failures very quickly, although a long latency occa-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sionally results. The end-to-end scheme can detect the failures which were missed by the neighbor scheme. Despite its perfect coverage, the end-to-end scheme has such weaknesses as high overhead, long latency, and the inability to locate failures. Thus, improving the detection coverage of the neighbor scheme is indispensable for effective failure recovery. To meet this requirement, we have developed a new heartbeat-transmission technique in which heartbeats are sent along a dedicated real-time channel. By applying this technique, we could achieve very high detection coverage with the neighbor scheme.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 4

RUN-TIME FAILURE RECOVERY

Depending on the failure semantics and the application characteristics, all channels on the failed component may fail or only part of them may fail. At run-time, failed channels should be detected and recovered as quickly as possible. In this chapter we focus on the procedure necessary after failure detection, which consists of failure signaling (i.e., reporting), channel switching and resource reconfiguration. The key principle of our failure-recovery process is *localization*, so that the traffic on non-faulty parts of the network remains unaffected by failure recovery. The delay associated with the failure-recovery process is also discussed.

This chapter is organized as follows. Section 4.1 describes the procedure of replacing a faulty primary channel by a healthy backup channel. Section 4.2 deals with resource reconfiguration after the disrupted service is recovered by channel switching. Section 4.3 presents the condition and result of bounding the failure recovery delay. Section 4.4 presents the simulation results to measure dependability QoS. The chapter concludes with Section 4.5.

4.1 Connection Restoration Procedure

The restoration of a primary channel from component failures is accomplished in two steps. First, if the node which detects a channel failure is different from the node which is responsible for channel switching, the failure is reported to the latter node. Then, the latter node selects a backup channel and switches the faulty channel to the selected backup. To determine the health of backups, failures of backup channels are monitored and reported to their end-nodes in the same way as primary channel failures. The difference of handling backup failures from primary failures is the lack of channel switching.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

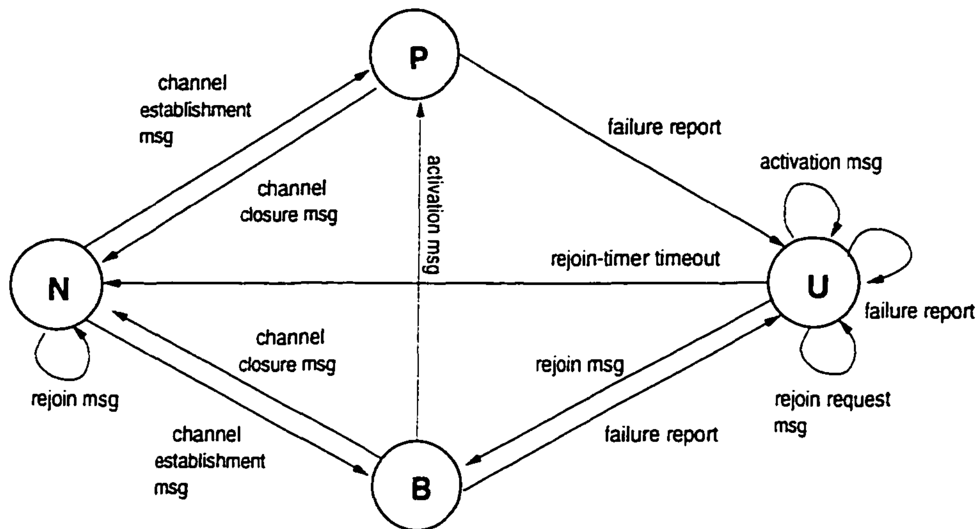


Figure 4.1: Channel state transition

There are three important issues in failure reporting. First, who will need to receive failure reports ? Second, which path will be used for failure reporting? Third, what information needs to be carried in a failure report? Our approach to these issues is:

(i) Failure reports are sent from the failure-detecting nodes to the end nodes of failed channels.

(ii) Failure reports are delivered through healthy segments of the failed channels' paths.

(iii) Each failure report contains the 'channel-id' of the failure channel.

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

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

The failure-recovery process sketched above is elaborated on with a state transition diagram in Figure 4.1. At each node, a channel can be in one of four states: non-existent state

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(N). healthy primary channel state (P). healthy backup channel state (B). and unhealthy channel state (U). The initial state is N. Upon reception of a '*channel-establishment message*,' the state machine enters state P or B. When a node receives a failure report (or detects a failure) in state P or B. the state machine enters U and the failure report is forwarded to the appropriate node. Additional failure reports received in state U are ignored. When an activation message is received in state B. the state machine enters P. The activation messages received in state U are ignored. The state transition for resource reconfiguration (e.g.. from U to N. or from U to B) will be explained in Section 4.2.

4.1.1 Failure Reporting and Channel Switching

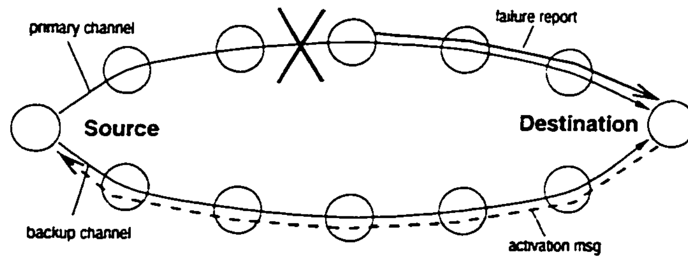
Now. we compare three possible schemes for failure reporting and backup activation. Figure 4.2 illustrates them. The main distinction among these schemes is where the failure reports and activation messages are generated and destined for. In Scheme 1 (Figure 4.2 (a)). the downstream node of the failed component generates a failure report and sends it to the destination node of the failed channel. Then. the destination node initiates an activation message. which travels in the opposite direction of the backup channel to be activated. By contrast. in Scheme 2 (Figure 4.2 (b)). the upstream node generates the failure report. the channel source node receives the failure report. and the activation message is sent to the channel destination node. Scheme 3 (Figure 4.2 (c)) is a hybrid of the first two schemes. Both end-nodes of a failed channel receive failure reports. and backup-channel activation is done in two ways. If an activation message reaches a node on which the backup channel has already been activated by the activation message from the other end-node. the activation message is discarded by the node.

The above descriptions are valid only when a failure is detected by the neighbor detection method. In case a failure evades the neighbor detection method and is detected by the end-to-end detection method, no failure reporting will occur and the destination node which directly detects the failure will initiate an activation message similar to Scheme 1.

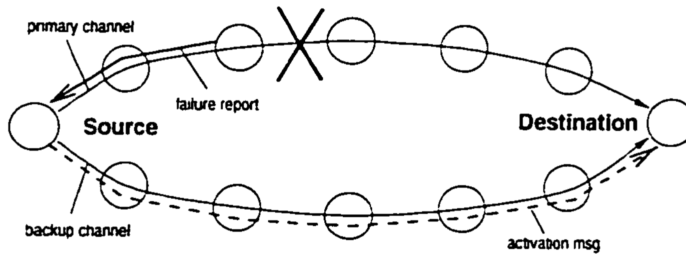
Scheme 2 and Scheme 3 have an advantage over Scheme 1 in terms of recovery delay, because data transfer through the new primary channel (i.e., the backup channel being activated) can be resumed by its source node immediately after sending the activation message, while in Scheme 1 the data transfer has to wait until the activation message arrives at the source node.[1] If a failure occurs near the destination node or is detected by

[1] Because the source node has no way to recognize the failure until it receives the activation message generated by the destination node.

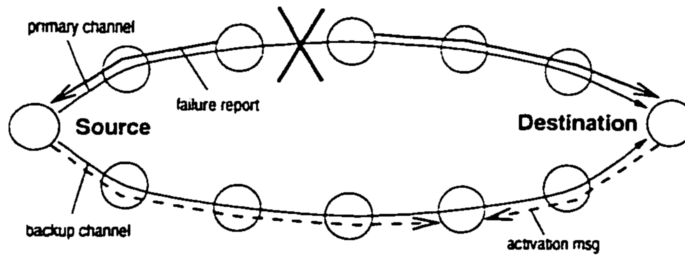
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Scheme 1



(b) Scheme 2



(c) Scheme 3

Figure 4.2: Three channel-switching schemes

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the end-to-end method, this advantage will be minimal. Albeit unlikely, in Scheme 2 or 3, if a data message arrives at intermediate nodes of the new primary channel before the channel is activated, the data message will be discarded with no harm. Scheme 3 has an edge over Scheme 2 in that the channel destination node can prepare early for channel switching, and the activation delay will be reduced by the bi-directional activation.

In Scheme 3, if a $\mathcal{D}$-connection is equipped with multiple backups, it is necessary that both end-nodes activate the same backup. If the destination node activates a different backup from the one selected by the source node, the backup need to be deactivated, since data messages have already been transmitted over the backup activated by the source node. One way to accomplish this synchronization is to allocate serial numbers to the backups of each $\mathcal{D}$-connection, and select a backup according to the serial number.

4.1.2 Priority-based Backup Activation

Connection priorities can be considered in the activation of backup channels. The idea is to activate the backups belonging to higher-priority $\mathcal{D}$-connections ahead of those of lower-priority $\mathcal{D}$-connections, if there are not enough resources to grant all activation requests. This is important when backups with different dependability QoS requirements are coexisting at a link. Remind that only backups with no greater multiplexing degrees (i.e., backups with the same or higher-priority) than that of each backup are considered in calculating the amount of spare resources. Therefore, without priority-based activation, it is possible that backups with high dependability QoS may suffer resource shortage because backups with low dependability QoS have already claimed spare resources. In such a situation, the dependability of a connection may be decided by the temporal order of backup activation instead of their dependability QoS, so that the connections with low dependability QoS may receive higher fault-tolerance levels than contracted, while the connections with high dependability QoS may be treated in the opposite way.

The priority-based activation can be achieved by delaying the activation of lower-priority backups. Thus, an activation message for a backup channel is sent after a certain delay determined by the dependability QoS of the corresponding connection. The activation of backups with a large multiplexing degree (i.e., lower-priority backups) is delayed so as to allow the backups with small multiplexing degrees (i.e., higher-priority backups) to be activated first.[2] The main drawback of this method is that the 'activation wait delay' is always imposed on lower-priority backups. To completely avoid priority inversion, this

[2] Recall that the importance of a backup channel is represented by its multiplexing degree.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

delay should be longer than the transmission delay of the activation message over the longest channel path in the network. In a large-scale network, the recovery delay incurred to lower-priority backups could be unacceptably long.

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

4.1.3 Recovery from Multiplexing Failures

Another type of failures which must be dealt with is multiplexing failures. A multiplexing failure happens when the activation of some backups causes the complete depletion of spare resources. If the spare resources at a link are exhausted by the activation of backups, the remaining backup channels on the link cannot function as standby channels. In such a case, the remaining backups are said to suffer multiplexing failures. Multiplexing failures are reported and treated in the same way as component failures.

4.2 Resource Reconfiguration Procedure

After inflicted connections are restored, the faulty channels are torn down and, if necessary, new backup channels are established. Activating backup channels may necessitate the reconfiguration of the spare-resource reservation, because the resources shared with other backups are now dedicated to the activated backups (or new primary channels) and the remaining backups may not receive their original dependability QoS with the reduced spare resources.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2.1 Channel Closure or Repair

To tear down a channel, a '*channel-closure message*' is usually sent over the channel's path, so that resources for the channel may be released. However, if failures disconnect a channel's path or disable the channel end-nodes, the resource-release process becomes complicated. The concept of "soft-state connections" of RSVP [111] is useful for reclaiming the resources reserved for failed channels.

In addition to easy release of resources, we support optional repairing of channels which are temporarily out of services. To this end, each node on a channel regularly sets a *rejoin timer* whose expiration automatically triggers the channel tear-down at the node. The purpose of the rejoin timer is to give the unhealthy channels (i.e., in U state) a chance to repair themselves. Channel repair has an advantage of eliminating the need of new channel establishment, in case the unhealthy channels become usable again soon.

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

If the channel destination node receives the rejoin-request message, the channel can be considered healthy (repaired). The destination node then sends a '*rejoin message*' back to the source node over the same path, and the channel state is changed from U to B, meaning that a repaired channel becomes a backup channel. If the rejoin timer had already expired when the rejoin message arrives at a node (i.e., in N state), the channel should be torn down as the resources for the channel had already been released. To undo the rejoin operations which have already done for the channel, a channel-closure message is generated by that node and is sent toward the channel destination. Figure 4.3 illustrates such a case.

The initial value of the rejoin timer should be chosen carefully. While it should be small for a quick tear-down of unhealthy channels, it should also be large enough to allow their repair, including (i) the failure reporting delay, (ii) the round-trip time of the rejoin-request message and the rejoin message, (iii) the time for additional resource allocation.

If all channels of a $\mathcal{D}$-connection fail simultaneously, a new primary channel has to be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

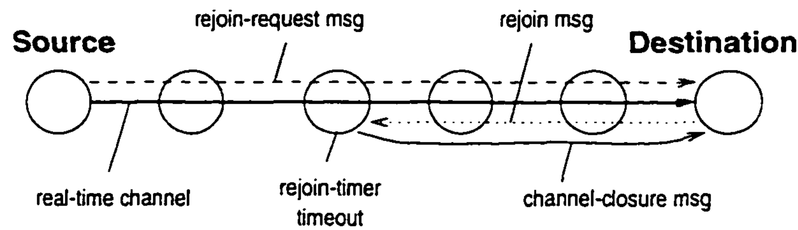


Figure 4.3: Unsuccessful channel repair

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

Table 4.1: Cases requiring resource reconfiguration

established from scratch. When there is no route which can meet the QoS requirement of the $\mathcal{D}$-connection, its client will be informed of the unrecoverable failure. Similarly, if any channel end-node fails or the network is partitioned, all attempts of channel re-establishment will be unsuccessful and the client will be informed of the unrecoverable failure. In any of these cases, all the resources reserved for the connection will be released, when the rejoin timer expires.

4.2.2 QoS Maintenance

After a $\mathcal{D}$-connection is established, in a normal (failure-free) situation, its dependability QoS is maintained by controlling the admission of new connections not to impair the QoS of existing connections, just as for preservation of performance QoS. Upon occurrence of a failure, more explicit actions are taken to maintain the "signed" terms of QoS. Thus, each connection inflicted by failures is fixed through resource reconfiguration. There are three cases which require resource reconfiguration for QoS maintenance as shown in Table 4.1 where × denotes failure and ○ denotes non-failure (thus healthy).

In Case 1, both primary and backup channels should be re-established. This can happen even due to a single component failure, when a primary channel is disabled by the component failure and its backup channel is disabled by a multiplexing failure due to a "side effect" of the component failure. Since the primary channel has to be established from scratch, time-bounded failure recovery is not guaranteed in this case. In Case 2, the faulty backup is torn down and a new backup is established to maintain the dependability QoS of the injured

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

connection. The network tears down the old backup before the new backup is established, so that the new backup can be routed with the healthy components on the old backup path, if necessary. In Case 3, the healthy backup is promoted to a new primary channel (i.e., by backup activation and channel switching), while the faulty primary is torn down. After channel switching, a new backup is necessary since the original backup has been activated, thus ceasing its backup role. The network tears down the faulty primary before the new backup is set up.

Even when a connection is not directly inflicted by failures, its dependability QoS can be affected by the failure recovery for other connections. That is, spare resources are shared by multiple backups, and activation of a backup will deplete the spare resources on its backup path. As a result, the remaining backups on the links of this path may not receive their original P_r. At such links, the network has to allocate more spare resources to maintain the same dependability QoS for the remaining backups. The amount of additional spare resources is calculated by the same method used for initial backup establishment. In the process of QoS maintenance, the network has to take care of a situation when there are not enough resources available at a link to match the additionally-needed spare resources. If the required spare resources are not available, some of the remaining backups have to be moved to different paths. When proper paths for the backups to be moved do not exist, such backups may have to be closed, resulting the degradation of dependability QoS of the associated $\mathcal{D}$-connections. Here, one has to determine which backups to close or move. To minimize the overhead of moving or rerouting backups, the network may want to choose a minimal set of backups whose teardown can resolve the resource-shortage problem at that link. This issue is detailed in the following section.

4.2.3 QoS Degrade & Upgrade

If the network fails in establishing new backups to replace their failed predecessors or rerouting the present backups due to resource shortage, the degradation of dependability QoS is inevitable. In such a situation, the network may give up on finding new backups, hence degrading the dependability QoS of the corresponding connections. In this scenario, QoS degradation is applied to only the connections which are directly affected by failures (option 1). Spare resource shortage due to failures can be dealt with differently. For example, instead of completely closing certain backups at a link where spare resources fall short, the network can keep as many backups as possible and degrade the dependability QoS of their corresponding connections until the available spare resources at the link can accom-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

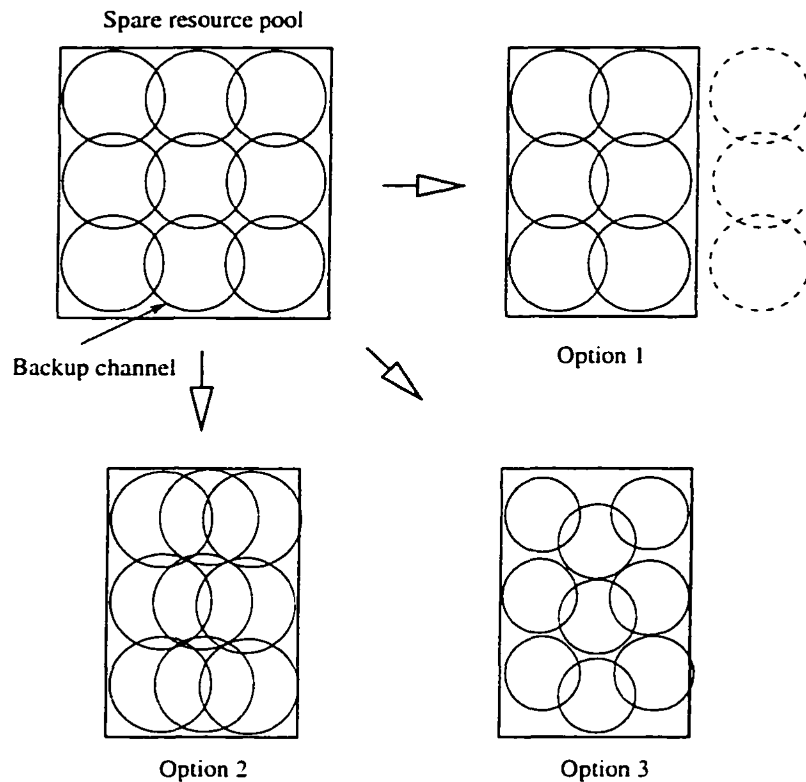


Figure 4.4: Three options of QoS degradation

modate all backups (option 2). Another way is to degrade the performance QoS of backups instead of dependability QoS (option 3). While dependability QoS degradation means the decrease of the probability of successful failure recovery, performance QoS degradation of a backup means the actual degradation of the connection's performance guarantee when the next failure happens. Therefore, option 3 is applicable only if the application allows performance degradation.

Figure 4.4 illustrates the difference among these options. The degree of overlap between backups represents the degree of backup multiplexing, where the size of the rectangles reflects the amount of available spare resources at a link. A combination of these options allows the network to adopt different policies, depending on the criticality or importance of each connection. Thus, the network will neither degrade the QoS of important connections nor close them to save less important connections from running out of their backups, while allowing the opposite to be feasible.

When failures occur far more often than the network can handle, severe QoS degradation including connection closures is inevitable. However, in practice, the network will be able to tolerate most failures without causing a large number of connections to be terminated, as it

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can usually recover from component failures in a much shorter time than the components' MTBF (mean time between failures). Thus, most QoS-degraded connections are likely to be restored to their original state before the next/another failure occurs. Repairing failed components forces the network to move channels on the (congested) links over which QoS-degraded channels run, back to a route that runs through repaired components. Backups are preferably migrated first due to the simplicity of their re-routing, as compared to primary channels. After moving some channels, the QoS of the remaining channels on the congested links can be upgraded. The channels with degraded performance will be given higher priority for QoS-upgrade than the channels with degraded dependability.

4.3 Bounded-Time Failure Recovery

Most resource-reconfiguration operations, especially channel re-establishment, are time-consuming. Fortunately, however, unlike failure detection, failure reporting, or channel switching, resource reconfiguration is not a time-critical action, because its delay does not directly affect the service-disruption time except for the case of loss of all channels of a $\mathcal{D}$-connection. But resource-reconfiguration delay can influence the recovery capability/delay in handling future failures.

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

4.3.1 The RCC Network

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

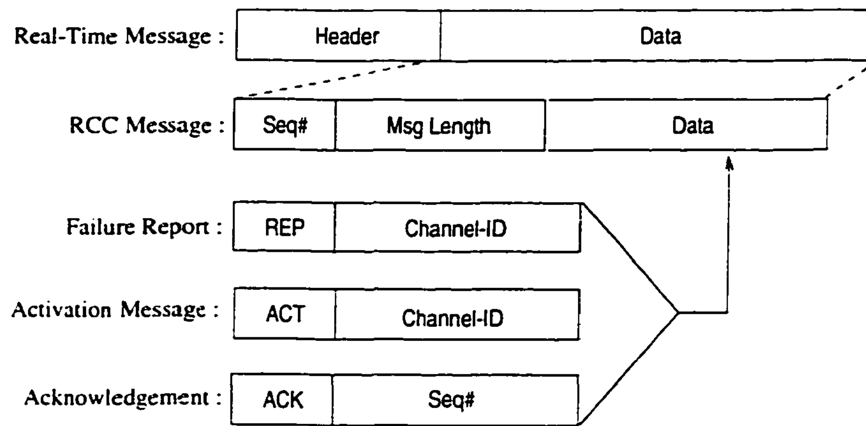


Figure 4.5: The RCC message format

The format of an RCC message is shown in Figure 4.5. Basically, an RCC message contains a combination of failure reports, activation messages, and acknowledgments. The control messages related to resource reconfiguration are excluded, since their delays are not time-critical. In addition, an RCC is a possible vehicle to convey node-heartbeats for failure detection. An interesting component of the RCC message format is acknowledgments, which are used to ensure reliable transmission of control messages. Generally, occasional losses of real-time (data) messages are tolerable in many applications. However, the loss of control messages is critical even in these applications. For loss-less transfer, each RCC message is acknowledged hop-by-hop between two nodes, and if a node does not receive an acknowledgment of the RCC message which it sent, it resends the unacknowledged RCC message. Each RCC message contains a sequence number, so that duplicated messages may be easily detected and discarded.

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

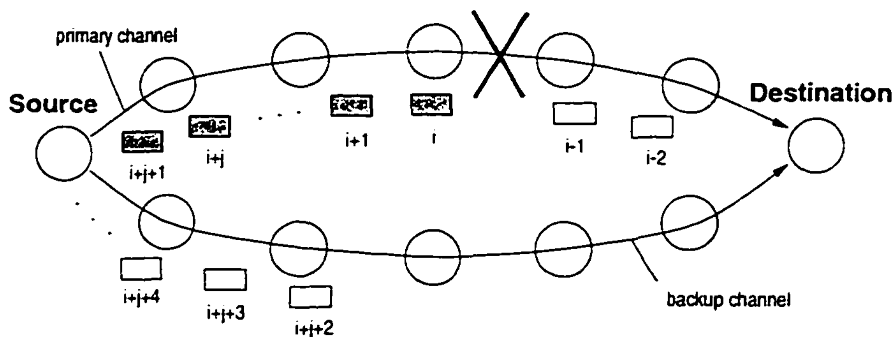


Figure 4.6: Message loss during failure recovery

The collection of RCCs on all links forms a virtual network,[3] called the *RCC network*, of the same topology as the underlying physical network. One can consider a (physical) network as a composition of three logically-separated networks — the primary-channel network, the backup-channel network, and the RCC network.

4.3.2 RCC Message Delay

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

There is an upper bound on the RCC message traffic, for the reasons given below. The number of failure reports on a link ℓ cannot exceed the number of primary/backup channels on a pair of links between two nodes incident to ℓ. We have to consider both links, because in Scheme 3 (of Section 4.1) failure reports for a channel can travel in both forward and backward directions of the channel, depending on the failure location. Similarly, the number of activation messages on link ℓ is bounded by the number of backup channels on the pair of links between the two nodes incident to ℓ. Since both the failure report and the activation message for the same channel cannot be transported over the same link at the same time, the maximum RCC traffic is determined by the largest number of channels on a link pair among all link pairs. The RCC message delay on any link is bounded by D_{max}^{RCC}, if S_{max}^{RCC} is greater than the maximum RCC traffic. If the maximum RCC traffic on a certain link exceeds S_{max}^{RCC}, some RCC messages may experience a longer delay than D_{max}^{RCC} at that link.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.3 Failure-Recovery Delay Bound

Now, let's consider the failure-recovery delay of a $\mathcal{D}$-connection. We assume that at least one of its backup survives failures, RCC messages are delivered without loss/retransmission, and the computational delays for recovery operations are negligible compared to the control message delays. Then, the failure-recovery delay, Γ, is the sum of

- failure detection delay,
- failure reporting delay,
- activation retrial delay,
- backup activation delay.

With Scheme 2 or 3, the backup activation delay is zero, because services are resumed immediately after sending the activation message by the source node. If the RCC message delay on each link is bounded by D_{max}^{RCC}, we can derive an upper bound of Γ for a $\mathcal{D}$-connection as follows.

The 'failure reporting delay' is less than $(\mathcal{L}_m - 1)D_{max}^{RCC}$, where $\mathcal{L}_m$ is the number of hops of the primary channel of the $\mathcal{D}$-connection. If the failed component is located close to the source node, the reporting delay will be very short. The 'activation retrial delay' needs to be considered in case the connection has multiple backups. When the activation message for a backup encounters failures during its journey, one additional round-trip delay is added to the recovery delay — the transfer delay of the unsuccessful activation message itself and the delay for reporting the new failure. It is bounded by $2(b-1)(\mathcal{L}_b - 1)D_{max}^{RCC}$ where b is the number of backups and $\mathcal{L}_b$ is the length of the longest backup of the $\mathcal{D}$-connection. With a single backup, the activation retrial delay is none and Γ is equal to the sum of the failure detection delay and the failure reporting delay.

Figure 4.6 illustrates the message loss during failure recovery (shaded messages are lost). The amount of actual message losses (or the service disruption time) is determined by the sum of (i) messages which are sent over the failed primary channel during failure recovery and (ii) messages which are already on the failed primary channel but do not pass the failed component yet, when failure is detected. Thus, the worst-case service disruption time is the message round-trip delay plus the failure detection latency.

[3] A separate network in terms of resource reservation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	8×8 torus	8×8 mesh
Spare bandwidth	16.88%	20.16%
1 link failure	100%	100%
1 node failure	92.67%	93.29%
2 node failures	86.49%	83.04%

Table 4.2: R_{fast} with deterministic multiplexing

4.4 Dependability QoS Measurement

We measured the fault-tolerance capability provided by various backup configurations through simulations.

The same simulation setup as in Chapter 1 was used. The simulation networks were an 8×8 torus with 200 Mbps link capacity and an 8×8 mesh with 300 Mbps link capacity. A total of 4032 connections were established. Unless explicitly stated otherwise, the source and destination of each connection were evenly distributed so that there may exist a $\mathcal{D}$-connection between each node pair. Channels of each $\mathcal{D}$-connection were routed disjointly with the boundary-routing method (no route-optimization was performed). For simplicity, the same traffic model was assumed for all channels, so each channel required 1 Mbps of bandwidth on each link of its path.

Three failure models were simulated: single link failure, single node failure, and double node failures. When failures were injected into the network after establishing 4032 connections, each single link failure disabled about 64 primary channels in the torus network, and about 85 primary channels in the mesh network. By injecting a single node failure, about 139 and 276 primary channels were disabled in the torus and mesh network, respectively. Each double node failure caused the disconnection of about 365 and 512 primary channels, respectively.

4.4.1 Fault-Tolerance Level of Various Backup Configurations

At first, we assessed the overall fault-tolerance capability of various backup configurations. R_{fast} was used as a metric for measuring the fault-tolerance level achieved by each backup configuration. R_{fast} is the ratio of fast recovery by using backup channels to the number of failed primary channels. Thus, the $(1 - R_{fast})$ of $\mathcal{D}$-connections, whose primary fails, require the establishment of new channels for failure recovery. Note that R_{fast}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

(a) Single backup in 8 × 8 torus

Muxing degree	mux=0	mux=1	mux=3	mux=5	mux=6
Spare bandwidth	N/A	N/A	30.25%	21.25%	12.88%
1 link failure	N/A	N/A	100%	100%	100%
1 node failure	N/A	N/A	100%	100%	97.68%
2 node failures	N/A	N/A	100%	99.82%	93.28%

(b) Double backups in 8 × 8 torus

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

(c) Single backup in 8 × 8 mesh

Table 4.3: R_{fast} with probabilistic multiplexing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

accounts for only those connections whose primary fails.[4]

The simulation result when deterministic multiplexing was applied is shown in Table 4.2. (The algorithm for single-link failure tolerance was used.) For probabilistic multiplexing, the results with five multiplexing degrees are summarized in Table 4.3. To investigate the benefit of the multiple backup configuration under probabilistic multiplexing, the single and double backup configuration were compared. In Table 4.3 (b), N/A indicates that the total bandwidth requirement had exceeded the network capacity before establishing all connections.

For probabilistic multiplexing, 'mux=1' guaranteed a perfect recovery coverage from all single failures, and 'mux=3' did from all single link failures. The spare-bandwidth overhead and the fault-tolerance capability of deterministic multiplexing came between 'mux=3' and 'mux=5' of probabilistic multiplexing.

Under probabilistic multiplexing, the use of a smaller multiplexing degree results in higher fault-tolerance (a larger R_{fast} value) with an exception between 'mux=0' and 'mux=1'. 'mux=0' is the same as disabling backup multiplexing, since no backup will be multiplexed. Interestingly, the R_{fast} in case of double node failures was not improved by disabling backup multiplexing (i.e., 'mux=0') as compared to 'mux=1'. The fault-tolerance level was even decreased by disabling backup multiplexing in the mesh network (see the bottom line of Table 4.3 (c)). It can be explained by the impact of backup multiplexing on the channel route selection, so that primary or backup channels are routed over different paths depending on whether backup multiplexing is enabled or disabled.

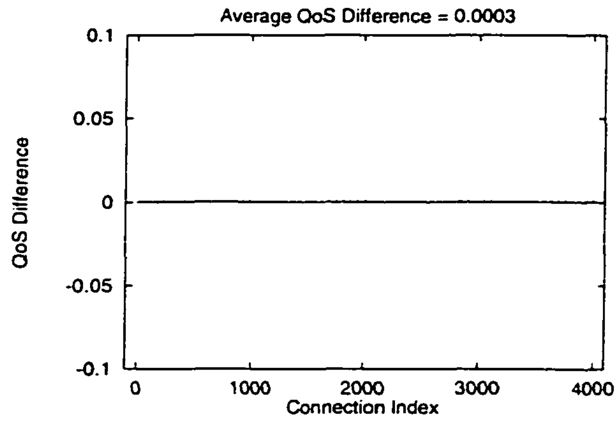
Another interesting observation is that a similar level of fault-tolerance was achievable with significantly less spare resources in the double backup configuration. For example, let's compare the case of single backup with 'mux=3' with the case of double backups with 'mux=6' in the torus network. Using a much smaller spare bandwidth, we achieved comparable R_{fast}, demonstrating the usefulness of multiple backup channels with effective resource sharing. The comparison between double backups with 'mux=6' and a single backup with 'mux=5' more clearly reveals the benefit of the multiple backup configuration.

4.4.2 Per-Connection QoS Management

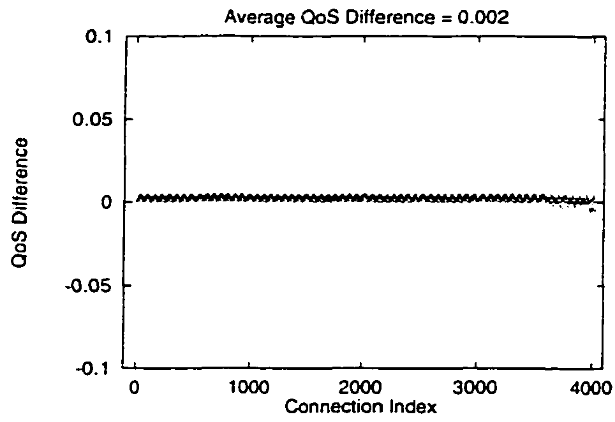
Deterministic multiplexing supports a very simple type of dependability QoS, e.g., 100% tolerance to all single link failures. In contrast, probabilistic multiplexing offers more versatile QoS supports. Though the R_{fast} data in Table 4.3 shows the overall fault-tolerance level

[4] We exclude from consideration the connections whose end nodes fail.

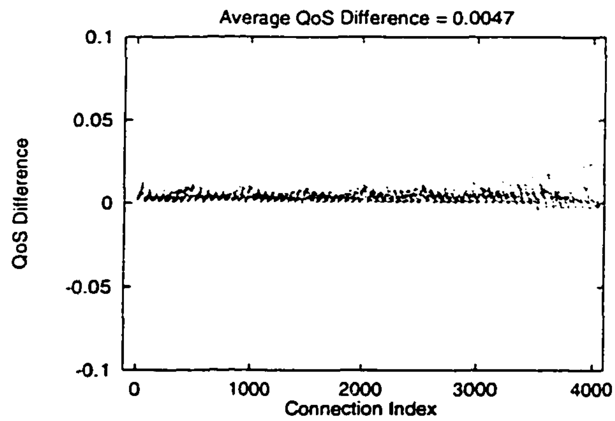
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) 8 × 8 torus (mux=3)



(b) 8 × 8 torus (mux=5)



(c) 8 × 8 mesh (mux=5)

Figure 4.7: Distribution of QoS differences

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

provided by probabilistic multiplexing, they do not reflect the QoS level each connection actually receives, since the data in Table 4.3 are average values. To examine if the required QoS is actually provided for each connection under probabilistic multiplexing, we measured the QoS each connection experiences. Then, this measured QoS was compared with the QoS negotiated at connection establishment.

Let $P_r^{msr}(i)$ denote the actual QoS received by the i-th connection and $P_r^{est}(i)$ be its estimated QoS. We compared $P_r^{msr}(i)$ with $P_r^{est}(i)$ under the 'single-link failure hypothesis'. The single-link failure hypothesis was used in the calculation of P_r^{est}: λ for links was set to $1/(64 \cdot 4)$ while λ for nodes was set to 0. The Method 2 presented in Chapter 2 was used for the calculation of P_r^{est}. $P_r^{msr}(i)$ was derived from the simulation results, so that $P_r^{msr}(i)$ is the ratio of the number of cases of the i-th connection not suffering from failures or recovering from failures with its backup, to all simulation runs.

For comparison, we calculated the QoS-difference ($P_r^{msr} - P_r^{est}$) for each connection in the single-backup configuration. Figure 4.7 (a) and (b) show the distribution of QoS-differences of 4032 connections in case of 'mux=3' and 'mux=5', respectively, in the torus network. The average QoS-differences were 0.0003 for 'mux=3' and 0.002 for 'mux=5', showing very accurate QoS estimation. As explained before, Method 2 under-estimates (i.e. positive QoS differences) the QoS level of each connection. We get negative QoS-differences for some high-indexed connections, because we attempted to recover low-indexed connections first in the simulation. The margin of QoS-difference is increased (i.e., less accurate estimation) as the multiplexing degree gets higher, because the QoS under-estimation by Method 2 gets worse. The simulation results of the mesh network are plotted in Figure 4.7 (c). Comparing this with Figure 4.7 (b), one can observe a slightly larger QoS-difference.

4.4.3 QoS Support for Heterogeneous Connections

So far, we have assumed that all $\mathcal{D}$-connections require the same level of fault-tolerance. We now show how the fault-tolerance level of *each* $\mathcal{D}$-connection is maintained when different connections require different levels of fault-tolerance. To this end, we simulated a combination of four types of connections: 1/4 of connections with 'mux=1', 1/4 of connections with 'mux=3', 1/4 of connections with 'mux=5', and the remaining 1/4 of connections with 'mux=6'. The number of backups was the same for all connections.

Table 4.4 shows that the fault-tolerance level of each class of $\mathcal{D}$-connections can be readily controlled, while the overhead remains to be around the average of all the classes. From the simulations measuring QoS-difference, we obtained similar results to the case

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

(a) Single backup in 8 × 8 torus

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

(b) Double backups in 8 × 8 torus

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

(c) Single backup in 8 × 8 mesh

Table 4.4: R_{fast} with mixed multiplexing degrees

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

when the same level fault-tolerance was required for all connections.

4.4.4 Comparison with Brute-Force Multiplexing

We compared the efficiency of our backup-multiplexing schemes with a simple multiplexing method. called *brute-force multiplexing*. In the brute-force multiplexing method. the same amount of spare resources is reserved for all links without considering the network status. For comparison. the amount of spare resources of brute-force multiplexing was set to the average amount of spare resources resulted by each backup configuration of our schemes.

The resultant R_{fast} values of brute-force multiplexing are compared with our schemes in Table 4.5. The comparison reveals some interesting points. For example. our schemes are sometimes only marginally better than the brute-force scheme. We attribute this to the homogeneity of the simulated network in terms of network topology. channel traffic model. and the distribution of channel end-nodes. The resource demands for backup activations are therefore evenly distributed throughout the network. Hence. the performance difference between brute-force multiplexing and our methods may not be substantial.

However. when any inhomogeneity exists. our schemes outperform the brute-force scheme by a larger margin. The simulation results of the mesh network supports this observation (see Table 4.5 (b)). Furthermore. if the channel end-nodes are not evenly distributed or the required bandwidths of all channels are not identical. hot-spots (in term of the spare resource demands) occur. and the efficiency of the brute-force scheme degrades significantly unlike the proposed scheme. As an example. simulation results when connections were rooted from only 32 nodes among the total of 64 nodes are summarized in Table 4.6. The gap between our schemes and the brute-force scheme was doubled as compared to Table 4.5.

4.4.5 Graceful QoS Degradation

We also investigated how our scheme maintains connection dependability in under-loaded and over-loaded networks. To simulate an over-loaded network. we set the bandwidth requirement of each connection to 2 Mbps. Under-loaded and over-loaded networks were generated by establishing 2016 $\mathcal{D}$-connections and 4032 $\mathcal{D}$-connections. respectively. in a 8×8 torus with each link of 200 Mbps bandwidth. We set the bandwidth requirement of each connection to 2 Mbps and equipped each connection with a single backup whose multiplexing degree was set to 'mux=3'. Under this simulation setup. 32.0% and 64.5% of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	(Brute-force.Probablistic)				(Brute-force. Deterministic)
	mux=1	mux=3	mux=5	mux=6	
Spare bandwidth	30.25%	22.5%	16%	9.5%	16.88%
1 link failure	(100%. 100%)	(98.05%. 100%)	(92.19%. 97.27%)	(76.31%. 74.11%)	(92.97%. 100%)
1 node failure	(100%. 100%)	(95.34% 100%)	(87.98%. 89.99%)	(68.87%. 64.72%)	(88.7%. 92.67%)
2 node failures	(93.11%. 93.11%)	(89.82%. 92.98%)	(82.23%. 84.05%)	(63.53%. 58.36%)	(82.98%. 86.49%)

(a) 8 × 8 torus

	(Brute-force.Probablistic)				(Brute-force. Deterministic)
	mux=1	mux=3	mux=5	mux=6	
Spare bandwidth	33.11%	24.47%	19.69%	17.22%	20.16%
1 link failure	(96.18%. 100%)	(89.74%. 100%)	(83.18%, 97.63%)	(78.18%. 90.39%)	(83.72%. 100%)
1 node failure	(96.56%. 100%	(88.31%,) 99.94%)	(79.49%, 91.74%)	(72.86%. 84.08%)	(80.24%. 93.29%)
2 node failures	(86.78%, 89.22%)	(79.62%. 88.83%)	(71.88%, 81.82%)	(66.03%. 75.32%)	(72.53%. 83.04%)

(b) 8 × 8 mesh

Table 4.5: R_{fast} comparison with brute-force multiplexing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	(Brute-force.Probablistic)				(Brute-force. Deterministic)
	mux=1	mux=3	mux=5	mux=6	
Spare bandwidth	31.37%	24%	16.88%	11.5%	18.37%
1 link failure	(98.97%. 100%)	(95.36%. 100%)	(84.99%. 94.14%)	(73.39%. 76.64%)	(87.8%. 100%)
1 node failure	(98.12%. 100%)	(93.63%. 100%)	(80.63%. 85.41%)	(66.6%. 69.01%)	(84.02%. 93.78%)
2 node failures	(91.69%. 93.11%)	(87.49%. 92.15%)	(75.62%. 78.18%)	(61.66%. 62.58%)	(78.09%. 86.6%)

(a) 8 × 8 torus

	(Brute-force.Probablistic)				(Brute-force. Deterministic)
	mux=1	mux=3	mux=5	mux=6	
Spare bandwidth	33.21%	26.33%	21.09%	18.35%	21.82%
1 link failure	(91.40%. 100%)	(84.95%. 100%)	(77.55%. 97.02%)	(72.68%, 89.08%)	(78.67%. 100%)
1 node failure	(90.24%. 100%)	(81.85%. 99.78%)	(72.77%. 91.45%)	(66.96%, 82.87%)	(74.16%. 93.91%)
2 node failures	(80.66%. 89.35%)	(73.33%, 88.60%)	(65.44%. 81.26%)	(60.31%. 73.82%)	(66.62%. 83.41%)

(b) 8 × 8 mesh

Table 4.6: R_{fast} comparison with brute-force multiplexing in case of hot spots

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

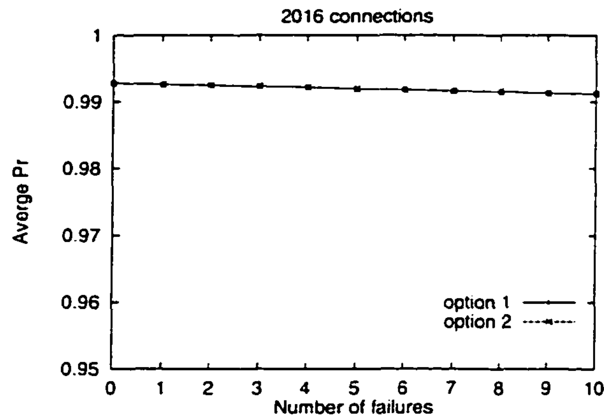


Figure 4.8: QoS maintenance in the under-loaded network

network loads were generated for under-loaded and over-load networks, while backup loads were 15.1% and 27.3%, respectively.

Ten consecutive link failures were injected with no repair of failed links throughout the simulation. After each failure, we measured the number of connections terminated and the average dependability QoS of the surviving connections.[5] We assumed that the interval between failures were sufficiently large for resource reconfiguration, while only the option 1 and 2 of QoS-degradation were simulated.

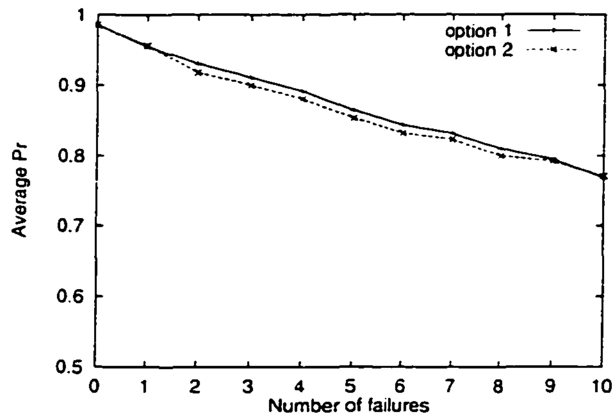
The change of average P_r for the option 1 and 2 in the under-loaded network is plotted in Figure 4.8. All primary channel failures were recovered by switching to their backups, and nearly the same average P_r was maintained throughout the simulation. Since there happened no QoS degradation, both options 1 and 2 produced the same result. The slight decrease of average P_r as the number of failures increases is due to the extended path lengths of both primary and backup channels. (Failures forced channels to be routed over a longer path than before.)

The simulation results in the over-loaded network are plotted in Figure 4.9. First, in a failure-free situation, the average P_r of the over-loaded network is lower than that of the under-loaded network — the margin is 0.9928 vs. 0.9852. This is because in the over-loaded network more backups are multiplexed together, and as a result, the probability of multiplexing failures, a dominant factor of P_r, increases.

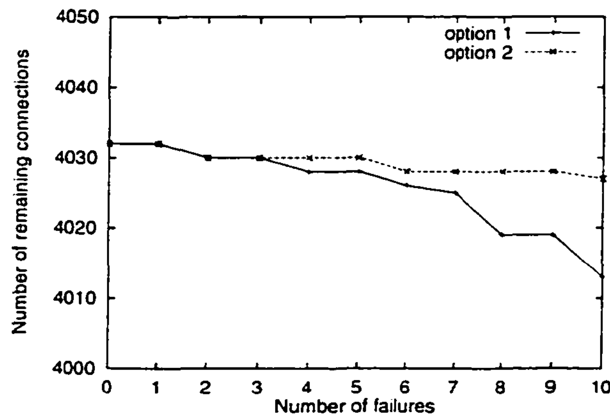
A high multiplexing failure probability causes the network to fail to re-establish primary channels in the over-loaded network. Thus, when there is no unreserved resources available,

[5]In calculating P_r, we assume that each link fails independently with rate 1/(64*4), i.e., a single link-failure model.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) The change of average P_r



(b) The number of connections in service

Figure 4.9: Graceful QoS degradation in the over-loaded network.

the re-establishment of primary channels for failure recovery will not be successful. In such a case, the network can either close such connections or re-establish primary channels by reducing spare resources on their channel paths. In this simulation, we adopted the former, which results in the closure of some connections. The rationale is that we would like to preserve the contracted QoS of other connections. If the network allows the re-establishment of primary channels by decreasing spare resources without any restriction, the dependability QoS of other connections may be unpredictably compromised.

The decrease of the number of connections in service is plotted in Figure 4.9 (b), showing that option 2 can preserve more connections in service than option 1, because option 2 degraded the dependability QoS of more connections than option 1. Nevertheless, option 2 still provided a comparable level of dependability QoS to option 1 (see Figure 4.9 (a)).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5 Summary and Conclusion

In this chapter, we presented the run-time failure recovery procedure after failure detection. Two goals in run-time failure recovery are minimizing the service-disruption time and minimizing the effect on the non-faulty connections. To satisfy these goals, the detected failures are reported only to the affected connection's end-nodes instead of broadcasting to the whole network. Failure-report messages are transmitted over a special-purpose real-time channel for time-bounded and robust data transfer. We also addressed the issues of backup activation, channel switching, and resource reconfiguration. The dependability achievable with various backup configurations was measured by simulating various failure models. The simulation result shows that dependability QoS requirements are supported on a per-connection basis. To demonstrate the effectiveness of our scheme, we compared the performance of our scheme with that of a brute-force scheme which reserves the same amount of spare resources at all links. Finally, we demonstrated how the resource shortage is dealt with by graceful QoS degradation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 5

ADAPTIVE RESOURCE MANAGEMENT

The reservation of spare resources is the essential cost of backup channels. It reduces the network's capacity of accommodating more connections and as a result, degrades network utilization. The focus of this chapter is on how to utilize the spare resources in a failure-free situation, so as to eliminate the "dependability cost" in terms of network utilization. To this end, we developed an adaptive QoS-control scheme. In a failure-free situation, spare resources are adaptively allocated to active channels for performance QoS, depending on the network-load condition or application requests. This allocation is carefully made so as not to unpredictably compromise existing connections' dependability QoS even when spare resources need to be used for recovery from randomly-occurring failures. The adaptive QoS-control enables "seamless" utilization of spare resources for both performance QoS and dependability QoS, so that the network can operate without incurring any dependability cost in a failure-free situation, while being able to *predictably* respond to failures.

This chapter is organized as follows. Section 5.1 presents a new QoS negotiation model for adaptive QoS control. Section 5.2 describes a QoS adaptation method that responds to changes in the network-load condition. Section 5.3 describes a QoS adaptation method that responds to application's requests for changing the QoS requirements. Evaluation results are presented in Sections 5.2 and 5.3, along with the corresponding adaptation methods. The chapter concludes with Section 5.4.

5.1 Elastic QoS Control

Negotiation on QoS parameters between the network and applications is essential to real-time communication, because QoS guarantees require resource reservation, and the availability of resources necessary to provide certain QoS guarantees should be checked in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

advance. So far, we have assumed that the performance QoS requirement of a connection is specified as a single value.[1] The single-value QoS model has commonly been used in QoS negotiation of many real-time communication schemes. An application specifies its QoS requirement as a value, then the network either accepts or rejects the request [9]. In some schemes, the network determines currently possible QoS and notifies it to the application. Under the single-value QoS model, a QoS value is not changed in the connection's life-time once it is negotiated at connection establishment. In contrast, we consider an 'elastic' QoS control scheme which allows the change of the performance QoS of a connection at run time.

5.1.1 Range-QoS Model

For QoS negotiation under the elastic QoS control scheme, we use a range-QoS model, in which QoS is expressed in the form of [min-QoS, max-QoS].[2] The network accepts an application's request for a real-time connection if there are resources enough to satisfy its min-QoS requirement, and the resources required to provide min-QoS are 'firmly' reserved for the connection. In addition, each connection can claim *excess* resources which are not firmly reserved by other connections. Thus, a connection, once admitted, is guaranteed to receive its min-QoS, and probably more, up to its max-QoS, depending on the availability of excess resources. The spare resources are reserved by considering only min-QoS, of connections, so that a backup, upon its activation, is guaranteed to receive the min-QoS of the corresponding connection.

In the range-QoS model, an application can optionally specify its *utility* of additional QoS beyond its min-QoS. Utility expresses the value of addition QoS levels (e.g., bandwidth) for an application. For example, under a network pricing architecture based on resource usage, utility can be interpreted as the price the application/client will pay for additional QoS (or resources). Utility is used to determine how excess resources are allocated among connections. In our model, there is no utility associated with the min-QoS, because we give a priority to the min-QoS guarantee of existing connections over accepting new connection requests or increasing the QoS of other on-going connections. Therefore, no utility comparison is necessary against min-QoS. An application can specify its utility for each additional QoS level as a function, i.e., utility function. Figure 5.1 shows an example utility function.

[1] One value for each QoS parameter if there are multiple kinds of parameters.

[2] Throughout this chapter, min-QoS and max-QoS are used to mean performance QoS. Dependability QoS parameters are negotiated as single values but implicitly have a range form: P_r is a lower bound of connection reliability whose upper bound is 100% and Γ is an upper bound of the connection-disruption time whose lower bound is zero.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

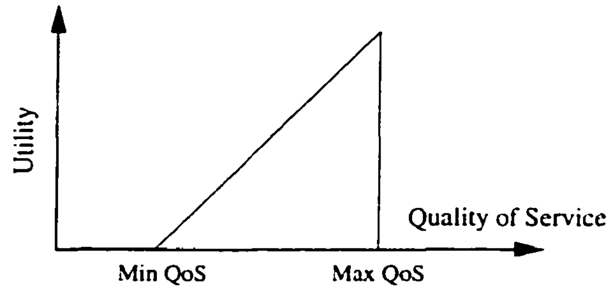


Figure 5.1: Example QoS/utility specification

5.1.2 Excess-Resource Allocation

The goal of excess-resource allocation is to maximize the total utility (revenue) the network will receive. To achieve this goal, we consider two allocation policies: *equal-share policy* and *local-max policy*. Their basic algorithms are presented in Figure 5.2, where the notation is:

Φ_ℓ: the set of primary channels at ℓ.

E_ℓ: total excess resources at link ℓ.

ϵ_i: the amount of excess resources allocated to P_i.

δ: the resource allocation unit.

$u(i)$: the utility of a primary channel P_i.

Under the equal-share policy, excess resources are fairly allocated to each channel regardless of its utility, until each channel's current QoS reaches its max-QoS. By contrast, under the local-max policy, excess resources are allocated to the channel which advertises the highest utility value at that moment. In case of tie, excess resources are fairly distributed among the channels with the same utility value. The channels which receive max-QoS are not considered any further for excess resource allocation under both policies.

The QoS of a channel is determined by its bottleneck link that allocates the least amount of excess resources to the channel on its path. Let us call the amount of excess resources which will be assigned for the channel by the underlying allocation policy if there is no bottleneck constraint, as '*fair share*.' Then, at non-bottleneck links of a channel, it is possible that a channel does not receive its fair share, while some other channels may obtain more than their fair share. If many channels claim lower excess resources than their fair share at a link and other channels cannot fully utilize the remaining excess resources,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

01 **loop**
02   **loop** for each primary channel $P_i \in \Phi_\ell$
03         $\epsilon_i \leftarrow \epsilon_i + \delta$; $E_\ell \leftarrow E_\ell - \delta$
04   **if** $\epsilon_i \geq$ max-QoS of P_i **then**
05             $\Phi_\ell \leftarrow \Phi_\ell - P_i$
06   **endif**
07   **if** $E_\ell \leq 0$ **then** quit **endif**
08   **endloop**
09 **endloop**

(a) Equal-share policy

01 **loop**
02   find a set of P_j, $u(j) = max\{u(i)\}, \forall P_i \in \Phi_\ell$
03   **loop** for each P_j
04         $\epsilon_j \leftarrow \epsilon_j + \delta$; $E_\ell \leftarrow E_\ell - \delta$
05   **if** $\epsilon_i \geq$ max-QoS of P_i **then**
06             $\Phi_\ell \leftarrow \Phi_\ell - P_i$
07   **endif**
08   **if** $E_\ell \leq 0$ **then** quit **endif**
09   **endloop**
10 **endloop**

(b) Local-max policy

Figure 5.2: Algorithms for excess-resource allocation at a link

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

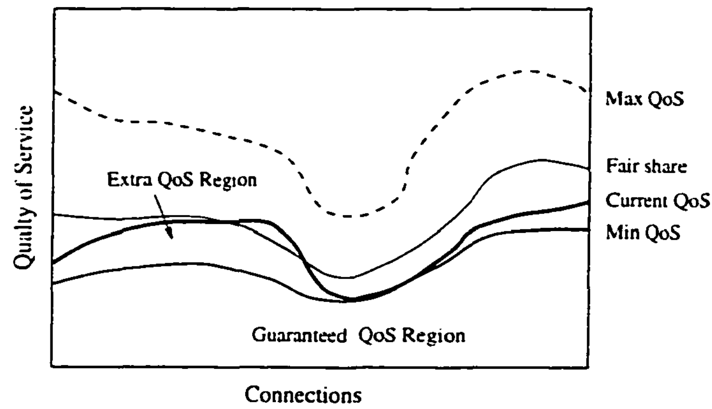


Figure 5.3: Excess-resource allocation at a under-utilized link

the link will suffer 'under-utilization'. An extreme case of under-utilization happens when all channels run through the same link, which becomes a bottleneck for all channels, and all other links in the network are under-utilized. Figure 5.3 illustrates excess-resource allocation at an under-utilized link, where the extra QoS region is smaller than the region between the fair-share curve and the min-QoS curve. Of course, the sum of resource requirements for the guaranteed QoS region and the extra QoS region should be equal to, or less than, the total capacity of the link.

In our elastic QoS control scheme, spare resources are included in the pool of excess resources. The difference between spare resources and other excess resources lies in their use. Spare resources can be used only for enhancing QoS beyond min-QoS, *not* for min-QoS guarantees themselves, where other excess resources can be used to establish a new channel (i.e., min-QoS guarantee) as well as enhancing QoS of existing connections. It is because, if spare resources are used for min-QoS guarantees, those min-QoS guarantees can be violated when failures occur and spare resources are diverted to failure recovery. By differentiating spare resources from other excess resources, the network can preserve the original QoS contract even in case of failures, while not degrading network utilization in a failure-free situation. Elimination of the dependability cost in a normal situation is important, since it is the "real" penalty without any benefit; the benefit of spare resource reservation comes only when failures occur and are handled with the reserved spare resources.

5.2 Network-Triggered Performance QoS Adaptation

In this section, we present a QoS adaptation method which changes the performance QoS of a connection according to the current network-load condition. We assume that the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

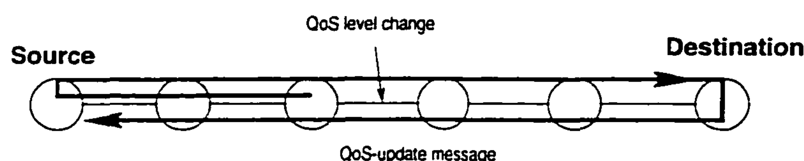


Figure 5.4: QoS-update procedure

source host of a real-time connection can adjust the traffic rate to its current QoS level using such techniques as those in [52, 84].

5.2.1 Run-time QoS Adaptation

When a connection is first established, it receives its min-QoS. At run-time, each connection claims excess resources and enhances its QoS level, so that if the network is under-loaded and excess resources are abundant, most connections will receive their max-QoS. The QoS of an existing channel may be upgraded, degraded, or preserved, as other channels are added or torn down. When a new channel is established, no QoS degradation will occur to existing channels, if there are enough excess resources to provide max-QoS to all channels including the newly-established channel. Otherwise, the excess resources allocated to each channel will be reduced, thus degrading the QoS of existing channels. As the network load increases (i.e., more channels are established), the amount of excess resources at each link decreases until it becomes equal to the amount of spare resources at the link. Thus, at a fully-reserved link, only spare resources can be used as excess resources, while all other resources are reserved for min-QoS guarantees. Backup channel activation can cause QoS degradation as well, since a newly-activated backup competes for excess resources in the same way as a newly-established primary channel. The difference between these two is that the network has already reserved resources for the min-QoS of the former, while the latter needs to pass an admission test to reserve the needed resources. When a primary channel is closed, its resources will be released and returned to the pool of excess resources, thus allowing the QoS upgrade of some of the remaining channels. The closure of a backup channel does not directly increase excess resources, because spare resources are already included in excess resources.[3]

The procedure of deciding the current QoS of a channel is depicted in Figure 5.4. When a QoS-level change is possible (in case of QoS-upgrade) or enforced (in case of QoS-degradation) at a link, a QoS-update message is generated carrying the new desired QoS

[3]The network's capability of accommodating more channels will be enhanced by closure of backups, because more resources become available for min-QoS guarantees.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and sent to the source node of the channel. Then, the channel source sends this message to the channel destination over the channel path. Upon arrival of this message, each link (controller) checks if it can support the new QoS. If it can, the message is unchanged and forwarded to the next hop. If not, the content of the QoS-update message is replaced by the best-possible QoS (i.e., lower than the original value) at the link before forwarding it. In either case, excess resources necessary for new QoS are reserved at each link. When the message arrives at the channel destination, it will contain the bottleneck QoS of the channel. The QoS-update message is then sent toward the channel source, during which the new QoS is confirmed and over-reserved excess resources, if any, are released.

The above QoS-update procedure has two implications. First, only the bottleneck links can initiate QoS-upgrade, because QoS-upgrade at a link is meaningless unless at least the same level of QoS is possible at all other links of the channel. Each link can tell whether it is a bottleneck link by comparing the current QoS with its fair-share (i.e., if both are same, the link is a bottleneck). Second, there may exist an unstable period after starting QoS adaptation, especially when releasing the over-reserved excess resources triggers the QoS-adaptation procedure of other channels. QoS adaptation may also cause an oscillation. That is, the QoS of a channel may fluctuate during a short time span, which can hurt the overall performance of the network. Considering the nature of QoS adaptation, QoS-degrade should be done quickly, while QoS-upgrade can be done slowly.

5.2.2 Evaluation

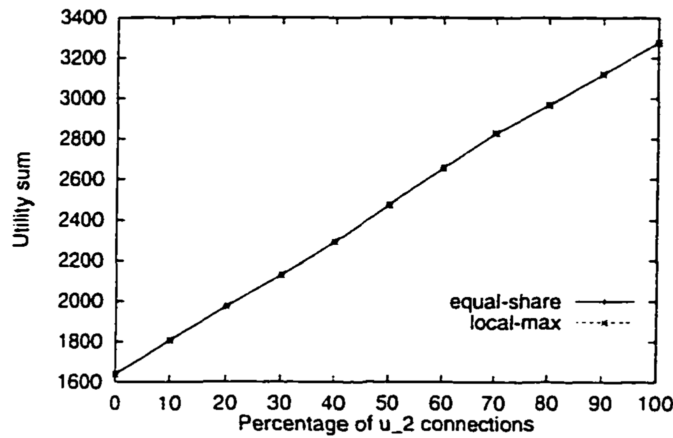
We first compared the performance of two excess-resource allocation policies through simulation. The simulation network was an 8×8 torus with 200 Mbps link bandwidth. For all connections, the single-backup configuration with multiplexing degree of 'mux=3' was used for simplicity.

In the first simulation, QoS range was set to [1 Mbps, 2 Mbps] for all connections, but two utility functions, u_1 and u_2, were mixed.[4] Both u_1 and u_2 are linear utility functions with different slopes. When the resource allocation unit δ is 100 Kbps, the utility for 100 Kbps with u_1 is 0.01, and the utility for 100 Kbps with u_2 is 0.02. In Figure 5.5 (a), the total utility sum after setting up 4032 connections is compared. Both allocation policies yielded the same result because max-QoS could be provided to all channels under both policies.

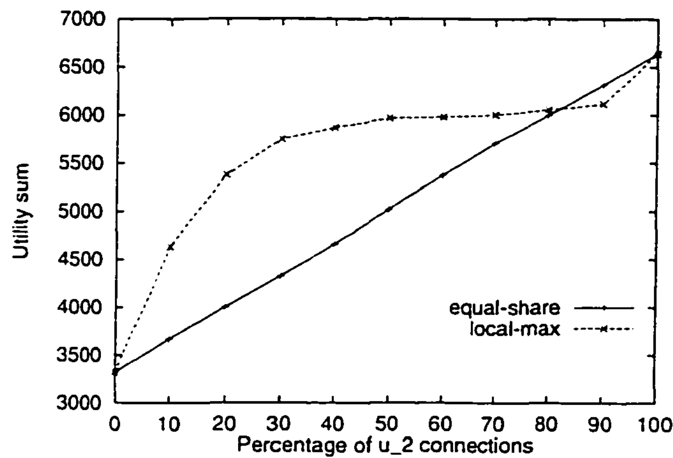
In the second simulation, the QoS range was set to [1 Mbps, 10 Mbps] for all connections.

[4] If all connections use the same utility function, both policies will always generate the same result.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a)



(b)

Figure 5.5: Comparison of excess-resource allocation policies

Now, the network cannot provide max-QoS for all connections, and each policy produced a different result. Figure 5.5 (b) shows the simulation result. Essentially, the local-max policy quickly accrued utilities and hit a plateau as the proportion of u_2 connections increased. This can be explained as follows. When some channels use a steeper-slope utility function than others, they will receive max-QoS under the local-max policy, and channels with a flatter-slope utility function will get a very low bottleneck QoS, which can cause under-utilization of other links. Thus unbalanced resource allocation can cause under-utilization by giving more resources to high-utility channels to maximize the overall utility. The total utility is determined by the gains from high-utility channels and the loss due to under-utilization. As the proportion of u_2 connections increases, the gains by high-utility channels were offset by under-utilization of the network.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

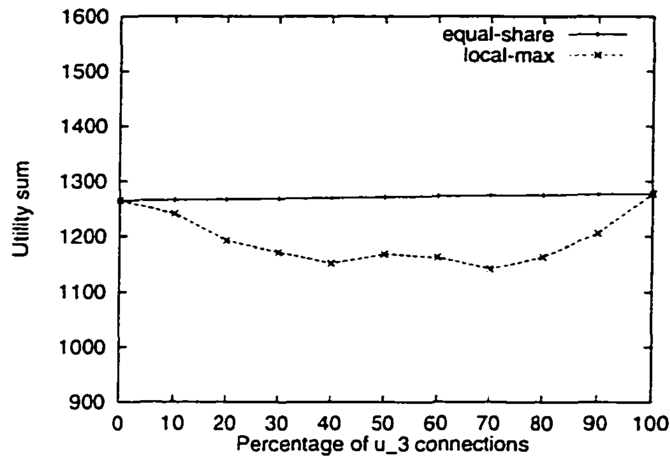


Figure 5.6: A case when the local-max policy flops

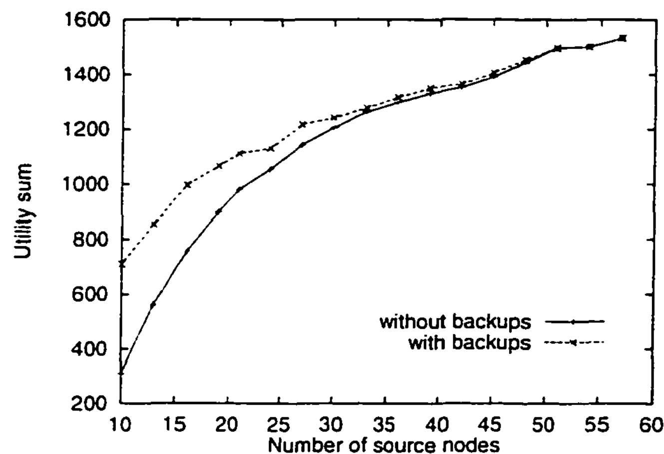


Figure 5.7: The impact of spare resource reservation on network utilization

Nevertheless, the local-max policy outperformed the equal-share policy in most cases, and when the proportion of higher-utility connections was over 30%, more than 90% of maximal utility sum was always achieved with the local-max policy.[5]

However, the local-max policy is not always better than the equal-share policy. An exception occurs when the difference between utility function slopes is very small. In such a case, the factor of network under-utilization dominates the gain by the higher-utility function. Figure 5.6 shows the simulation result when u_1 and u_3 connections are mixed. u_3 is a linear utility function mapping 100 Kbps to the utility of 0.0101. Thus, u_3 offers only 1 % higher utility than u_1 for each resource unit. Even in such an extreme case, the local-max policy is still within 90% of the maximal utility sum.

[5]These numbers are meaningful only under the simulation setting used.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Finally, we analyzed how spare resource reservation affects network utilization. The QoS range was set to [1 Mbps, 5 Mbps] and u_1 was commonly used as the utility function for all connections. As in the previous simulations, a total of 4032 $\mathcal{D}$-connections were established using the single-backup configuration with 'mux=3'. The simulation objective is to compare the network utilization with and without backup channels.

When connections were evenly routed over the network, max-QoS was possible for all channels, both with and without backups. This is because 4032 primary channels with max-QoS consumed only, on average, 64% bandwidth of each link, so that no link suffered under-utilization. To create bottleneck links, we restricted channel sources to be selected among a certain set of nodes, while channel destinations were randomly chosen. Figure 5.7 plotted the result of this simulation, showing that the network with backups earned a larger utility sum (thus, higher network utilization) than the network without backups. This is because spare resource reservation reduces the degree of under-utilization at each link. Even at the most congested link, there exist spare resources available as excess resources, while there are no excess resources in the network without backups.

5.2.3 Discussion

The main idea is to make the reserved spare resources available to real-time traffic in addition to their original intended use (i.e., fast failure recovery) or for non-real-time traffic which do not require resource reservation. This is achieved by including spare resources in the pool of excess resources, thus making all resources available for real-time traffic. With the range-QoS model, even if a lower min-QoS was used by the admission test (in which spare resources are not accounted for the min-QoS guarantee), a real-time connection can receive up to its max-QoS by utilizing spare resources. It is equivalent to setting its QoS requirement to the max-QoS in the admission test, except that applications should agree on possible QoS degradation in case of failures. Associating utility with QoS allows each connection to control the certainty of obtaining its max-QoS. In this way, real-time traffic can fully utilize the network capacity in a failure-free situation.

5.3 Application-Triggered Performance QoS Adaptation

Now, we present a QoS adaptation method which changes the performance QoS of a connection according to its application's demand. The capability of modifying existing QoS at run-time is beneficial for some applications. For example, in many video applications,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

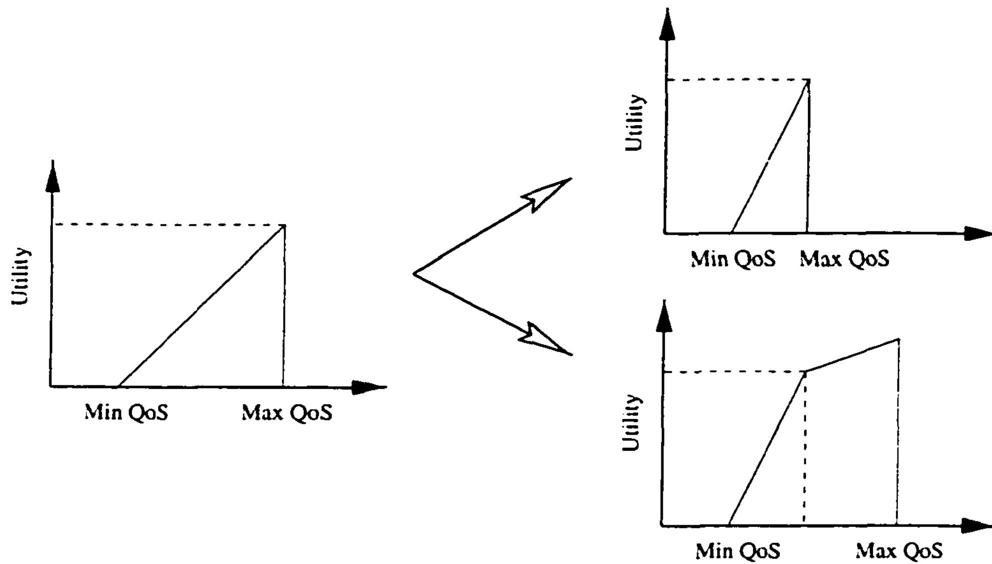


Figure 5.8: Utility function change for QoS-upgrade re-negotiation

the source rate changes significantly with scene changes (e.g., a still scene to a quick-motion scene, and vice versa) and they cannot be handled efficiently by smoothing or buffering alone. It is because unlike short bursts, long bursts lasting for tens of seconds or even minutes. Such long bursts are handled better[6] by using QoS re-negotiation at run-time [18, 109, 76, 35]. For instance, it is shown in [35] that a piecewise CBR stream with QoS re-negotiation requires 3 – 4 times less resources than the corresponding fixed-rate CBR stream.

While the Qos re-negotiation approach can achieve high resource efficiency, a QoS re-negotiation attempt for higher QoS may fail. If this chance (quantified as the *re-negotiation-blocking probability*, RBP) is not maintained at a satisfactory level, the benefit of re-negotiation will be seriously diminished. To bound this probability, the network has to set aside some resources by restricting the admission of new connections even if there are available resources.

In this section, we explore the conjunction of our failure-recovery scheme with QoS re-negotiation, so that the shortcomings of both (i.e., the re-negotiation blocking probability and network capacity reduction for guaranteeing dependability) can be compensated for by each other.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.3.1 Run-time QoS Re-negotiation

Under the range-QoS model. the network should change the min-QoS of a connection to respond to the QoS re-negotiation request by the connection. While re-negotiation for lowering min-QoS is straightforward. re-negotiation for increasing min-QoS requires a new admission control: check the availability of resources for QoS enhancement. This admission control is the same as that for the initial QoS negotiation. except that failure in this admission control means the continuation of the original QoS. We do not consider the option of changing an existing channel's route to overcome the admission test failure. because such a change may generate a domino effect. causing changes of other existing channels' routes. Instead. when QoS re-negotiation for enhancing min-QoS fails. a steeper-slope utility function can be used as a secondary means of QoS re-negotiation. Under the local-max excess-resource allocation policy. applications can receive higher QoS even with the same min-QoS by using a steeper utility function. as more excess resources will be allocated to them. Applications can control the margin of QoS-upgrade by setting the max-QoS to the desired QoS. or by using a tiered utility function (see Figure 5.8).[7]

To prevent unsuccessful QoS-upgrade re-negotiation for existing connections. the network should sometimes reject new connection requests even when enough resources are available to accept them. In general. it is impossible to derive an optimal admission control for this purpose due to the inherent uncertainty in the generation of QoS-upgrade requests. Only for playback applications with all changes of QoS requirements known in advance. an optimal admission control will be possible. For interactive applications. only statistical estimation based on a traffic-generation history has been used in [109. 35]. To avoid the uncertainty and complexity of such statistical estimation, we exploit the spare resources. Thus. RBP is maintained by relying on spare resource reservation instead of a separate admission control.

In QoS-upgrade re-negotiation. the network keeps a record of the original min-QoS value of a channel to be upgraded, while modifying its min-QoS. In a failure-free situation, spare resources can be used for QoS-upgrade re-negotiation by increasing min-QoS. When failures occur and backups are to be activated, the spare resources used for QoS-upgrade are reclaimed for failure recovery. and the min-QoS of the channels affected by this reclaiming is reset to their original min-QoS. The original min-QoS is also used for admission tests for new connection requests. The reason for this is that the dependability of existing connections

[6] higher resource efficiency

[7] However. this method does not guarantee QoS enhancement. For example. there will be no QoS enhancement when many connections opt to use the same steeper-slope utility function.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

may otherwise be compromised upon occurrence of failures.

5.3.2 Evaluation

We evaluated the effectiveness of our approach via simulation with real data.

The first factor which decides RBP is the characteristics of input source. For instance, if there are only minor fluctuations in the input rate, QoS re-negotiation will be rarely needed and will result in a low RBP. In the other extreme case, spare resource reservation may not handle all QoS re-negotiation requests and a high RBP will result. In the simulation experiment, we used a one-hour MPEG trace extracted from the Star Wars movie [31]. The movie stream was decomposed into multiple segments with different data rates, so that QoS re-negotiation may occur at the boundary of each segment.

We assume that the only knowledge the network has is the average rate of the original data, which is used to determine the initial min-QoS of each channel during its setup. Both the segmentation schedule (or time of re-negotiation) and the min-QoS associated with each segment are heuristically determined at run-time by monitoring the input source. A variant of the heuristic presented in [35] was used for this purpose. The QoS level of the next segment is estimated as:

$$\hat{r}_{i+1} = (1 - T^{-1})\hat{r}_i + T^{-1}(r_i + max\{b_i - B_h, 0\})$$

where r_i is the actual traffic generation rate during time slot i of length T, and b_i is the buffer size at the end of slot i, and B_h is a high buffer threshold. The term, $max\{b_i - B_h, 0\}$, ensures that buffer build-up more than B_h should be flushed by the end of the next slot. Re-negotiation is triggered if

$$\{b_i > B_h \cap (\hat{r}_{i+1} - \hat{r}_i) > \alpha\} \cup \{b_i < B_l \cap (\hat{r}_i - \hat{r}_{i+1}) > \beta\}$$

where B_l is a low buffer threshold, and α and β are QoS upgrade and degrade thresholds, respectively. A segmentation process S is denoted by $\{T, B_h, B_l, \alpha, \beta\}$.

The segmentation process has a significant impact on the resource efficiency of QoS re-negotiation. We used two segmentation processes, $S^1 = \{1/6$ sec, 250 Kbits, 0 Kbits, 50 Kbps, 50 Kbps$\}$ and $S^2 = \{1/12$ sec, 250 Kbits, 0 Kbits, 50 Kbps, 50 Kbps$\}$, whose statistics are given in Table 5.1. The *bandwidth efficiency* is the ratio of the average rate of the original data to that of segmentation results. Generally, a higher re-negotiation frequency allows a higher bandwidth efficiency, but induces a larger re-negotiation overhead. Both segmentations resulted in a data-loss rate less than 10^{-5} due to buffer overrun, when each

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	average rate	bw-efficiency	re-negotiations/sec
S^1	367 Kbps	0.909	0.36
S^2	346 Kbps	0.964	0.84

Table 5.1: Statistics of two segmentations

channel source was assumed to have a dedicated buffer of 500 Kbps for traffic smoothing.[8]

A total of 2016 or 4032 $\mathcal{D}$-connections were established in an 8×8 torus simulation network, where each channel used a randomly time-shifted version of the movie data. The initial min-QoS of each channel was set to 335 Kbps, the average rate of the original data before segmentation. The network link capacity was set to 50 Mbps considering the relatively small bandwidth requirement of each channel.

We performed simulations while varying several factors:

- the input data segmentation process (S),
- the number of connections (N),
- the distribution of channel sources (D),
- the degree of backup multiplexing (BM).

S represents the segmentation characteristics, N and D portray the network-load condition, and BM determines the amount of spare resources reserved. The metrics used are the probability (P_{rf}) that a re-negotiation attempt fails[9] and the ratio (R_{rf}) of the duration that the desired QoS was not provided to the total service duration. We measured these metrics while allowing channels to use only spare resources for QoS re-negotiation, as opposed to using all excess resources. In this way, we could simulate the worst-case behavior of our scheme. In other words, the pure contribution of spare resources to RBP was measured.

The simulation results of five cases are summarized in Table 5.2. 'Primary-load' indicates the ratio of the bandwidth consumed by primary channels to the total network capacity at the initial setup. 'Backup-load' is the average amount of spare resources at each link at the initial setup. 'Uniform' means that channel sources and destinations were randomly selected, where 'congested' means that channel sources were selected among a half of total network nodes in order to generate bottleneck links. Two backup multiplexing degrees, BM_1 (mux=3) and BM_2 (mux=5), were simulated.

[8] Usually, the larger buffer space is, the higher bandwidth efficiency is possible.
[9] In such a case, the previous QoS is maintained.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(S, N, D, BM)	primary-load	backup-load	P_{rf}	R_{rf}
Case 1: (S^1,4032,uniform,BM_1)	42.88%	21.44%	0	0
Case 2: (S^2,4032,uniform,BM_1)	42.88%	21.44%	4.85e-04	1.99e-05
Case 3: (S^1,4032,uniform,BM_2)	42.88%	17.25%	2.79e-05	9.42e-07
Case 4: (S^1,4032,congested,BM_1)	42.88%	20.58%	2.62e-02	1.91e-03
Case 5: (S^1,2016,uniform,BM_1)	21.44%	10.14%	3.89e-03	1.90e-04

Table 5.2: QoS re-negotiation results

The first observation comes from the comparison between Case 1 and Case 2; S^2 resulted in a higher P_{rf} than S^1. With the same amount of spare resources, the segmentation process with a lower bandwidth efficiency offered a lower re-negotiation blocking rate than that with a higher bandwidth efficiency. This somewhat counter-intuitive result comes from the fact that the fluctuation of QoS requests by S^2 was larger than that by S^1, while, on average, S^2 consumed less resources than S^1.

Second, less spare resources led to a higher P_{rf} (compare Case 1 and Case 3). Obviously, more spare resources can deal better with QoS re-negotiation requests.

Third, the network with bottleneck links (Case 4) resulted in a higher P_{rf} than the balanced network (Case 1), while the amount of spare resources was similar for the following reason. In Case 4, backups were routed by avoiding the bottleneck links on which primary channels were concentrated. As a result, the ratio of spare resources to active resources at bottleneck links decreased. Smaller spare resources compared to re-negotiation requests means a larger P_{rf} at bottleneck links.

Fourth, a smaller number of connections caused a higher P_{rf} (compare Case 1 and Case 5), because the multiplexing effect between QoS re-negotiation requests in Case 5 was smaller than in Case 1. If there are more channels on a link, their QoS re-negotiation requests will be more likely to be averaged out, i.e., a higher multiplexing effect. Overall, the duration of degraded services due to re-negotiation failures was kept below 1.9×10^{-3} for all cases of simulations, meaning that for 99.8% of total service time the applications received the full QoS.

5.3.3 Discussion

When a sufficient number of connections are established in the network, the network can keep a low RBP by using only spare resources. Even if there are only a small number of connections in the network, RBP will not be high, because there will be abundant excess

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

resources to supplement the small amount of spare resources. (In our simulation, other excess resources than spare resources were not accounted for in QoS re-negotiation.) The risk of a high RBP arises at only those links with many active channels which initiate QoS re-negotiation but reserve only a small amount of spare resources (as in case 4 of Table 5.2). To get around this problem, the degree of backup multiplexing may need to be determined by considering the dependency between the amount of active and spare resources.

5.4 Conclusion

In this chapter, we presented an elastic QoS-control scheme which greatly enhances the feasibility of deploying our scheme by eliminating the dependability cost in a failure-free situation. We combined our failure-recovery scheme with two promising adaptive QoS-control methods: network-triggered and application-triggered QoS adaption. The former resembles the controlled-load service extended with the concept of utility. The latter is close to the re-negotiation-based approach which is considered cost-effective for multimedia applications. The common goal of both adaptive methods is to make the dependability scheme transparent in a normal situation, while still providing predictable dependability guarantees upon occurrence of a failure.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

Future networks are expected to provide dependable real-time communication services for many emerging business- and mission-critical applications. Considering the innate heterogeneity and large scale of the network, any dependability scheme should have a simple and distributed architecture which is independent of the underlying communication services. In addition, since only a subset of applications will require dependable services and different applications/users will require different levels of dependability, the dependability level/cost should be "customizable," depending on the criticality/importance of applications.

In this thesis, a new approach is presented to make reservation-based real-time communication services dependable. Our approach is to cost-effectively provide a reasonable level of fault-tolerance. Though it does not mask network failures so that failures are completely transparent to the applications, its quick recovery allows the approach to be viable in many real-time applications. In fact, there are many real-time applications which can tolerate a short recovery time and do not require continuous availability. However, in a real-time environment, unrecovered failures for a longer period than a tolerable time window can result in application failures. Hence, it is essential to put a certainty on the recovery latency from failures. From the real-time communication perspective, pre-establishing a standby connection is the only way to achieve fast recovery, because the re-establishment latency of a real-time connection is unpredictable and usually long. Besides the predictable recovery latency, other characteristics of our approach such as low overhead, implementation neutrality, and flexible topological requirement will broaden its applicability.

6.1 Research Contributions

This thesis has made several important contributions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

First, we developed a client interface model for fault-tolerant real-time communication. The model provides two dependability QoS parameters: the probability of fast failure recovery and the estimated failure-recovery delay. These parameters are general enough to be understandable to most applications, but if the direct exposure of these parameters to end-applications is not appropriate, translation to application-specific QoS parameters may be necessary. The definition of a failure in real-time communication is also included in the client interface model. We allow each connection to choose a different semantic of failure.

Second, we devised a mechanism which guarantees the required dependability QoS with a minimal fault-tolerance overhead. The essence of this mechanism is the establishment of backup channels before failures actually occur. To minimize the resource overhead of backup channels (i.e., the low network resource utilization by reserved but unused resources for backups), two types of backup multiplexing methods are developed, each of which provides a different grain of dependability guarantees. The dependability for each connection is controlled by adjusting the parameters of the backup establishment procedure. A two-step routing method is also developed for backup route selection. It is shown that this routing method is more effective than conventional routing methods. We evaluated the efficiency of the backup-channel scheme through simulations and showed that with minor degradation of the network's capability of accommodating channels, a desired dependability QoS level can be achieved.

Third, we developed a robust mechanism for run-time failure recovery. A special emphasis is placed on 'fast' recovery and 'insulation' of healthy connections from the recovery operation for faulty connections. The first step of run-time failure recovery is the detection of failures. Our failure-detection protocol uses two behavior-based techniques for low-cost, quick, and perfect failure detection. Since these techniques do not require any special hardware support, it is applicable to any network technology. The efficiency of our failure-detection protocol was experimentally evaluated on a laboratory testbed. The next recovery steps are failure reporting and channel switching. A special-purpose real-time channel is used for robust and timely delivery of control messages associated with these recovery operations. Once an injured connection is restored, resources are reconfigured to maintain the dependability QoS of the connections affected by failures. This procedure is transparent to applications unless network failures make it impossible to preserve all of the existing QoS contracts. In such a case, the QoS of existing connections is gracefully degraded.

Finally, we presented an elastic QoS-control scheme which can virtually eliminate the dependability overhead in a failure-free situation. In this new QoS control scheme, we

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

combined our failure-recovery scheme with two adaptive QoS-control methods: network-triggered and application-triggered QoS adaption. Essentially, the resources reserved for backup channels are utilized by active channels in a normal situation, so that active channels can utilize the entire network resources. Through simulations, it is shown that our scheme does not degrade the network utilization in a failure-free situation, while still providing predictable dependability guarantees upon failure occurrences.

6.2 Future Work

This thesis presents an integrated solution which deals with almost all aspects of dependable real-time communication in multi-hop networks. While our work can be extended in various directions for each aspect, here we discuss two issues which have not been elaborated on in this thesis.

One is the implementation issue. The major challenge of our work is not in developing a new real-time message scheduling discipline. We assume the presence of communication subsystems which can provide performance QoS guarantees deterministically, statistically, or in some other manners. The role of the underlying communication subsystem is to deliver messages within a specific time limit with some level of certainty. Therefore, the performance-related nature of resultant dependable real-time communication service is bound to the nature of the underlying communication subsystem.

A real-time channel service is usually implemented with two protocols: *Real-time Network Management Protocol* (RNMP) and *Real-time Message Transmission Protocol* (RMTP). The main function of RNMP is channel establishment and teardown, while that of RMTP is run-time control such as traffic shaping and message scheduling. When a client requests a real-time channel to be established, it has to specify its traffic parameters (e.g., maximum message rate) and QoS requirements (e.g., message delay bound). Using this information, RNMP performs an 'admission test,' which checks the availability of the resources necessary to meet the channel's QoS requirement. RNMP reserves resources if the admission test is positive. In RMTP, a traffic regulator is used to smooth (oftentimes bursty) packet arrivals, and one or multiple output queues are serviced for message scheduling and transmission. RMTP is closely related to RNMP, because the admission control of RNMP assumes a certain message-scheduling policy used by RMTP.

Our scheme consists of two protocols which are *Backup Channel Protocol* (BCP) and *Failure Detection Protocol* (FDP). Figure 6.1 depicts a general protocol configuration in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

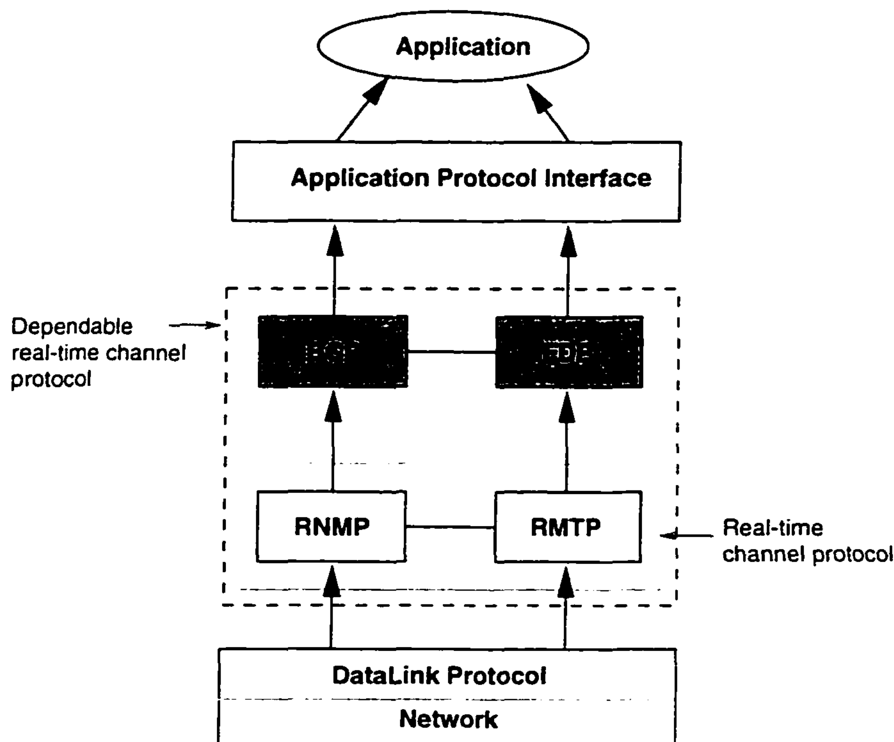


Figure 6.1: General protocol configuration

which the existing (unreliable) real-time channel protocol is augmented by our protocols. The augmentation requires some modification in API (Application Protocol Interface). The API of RNMP should be modified to include dependability QoS parameters which are exported by BCP. The API of RMTP may also need to be changed to support seam-less channel switching, i.e., to make the switch of primary channels invisible to applications. The design of our protocols does not assume a particular real-time communication scheme, so this thesis does not address the details about the integration effort of BCP/FDP with existing real-time communication protocols. In principle, our protocols can be placed on top of any existing (possibly independently developed) real-time channel protocols.

The implementation of our scheme requires support for resource management from the underlying communication subsystem. For instance, backup multiplexing and adaptive QoS control will require the access and control of various resources (e.g., computing bandwidth, transmission bandwidth, or message buffer). A clean interface of the resource manager which hides the system-dependent details is crucial for the efficient implementation of our scheme. Thus, the resource manager should perform 'resource abstraction', by providing an insulating layer between the (low-level) resources and the abstracted resources dealt with by the upper-level protocols. This layer may or may not be straightforward depending on the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

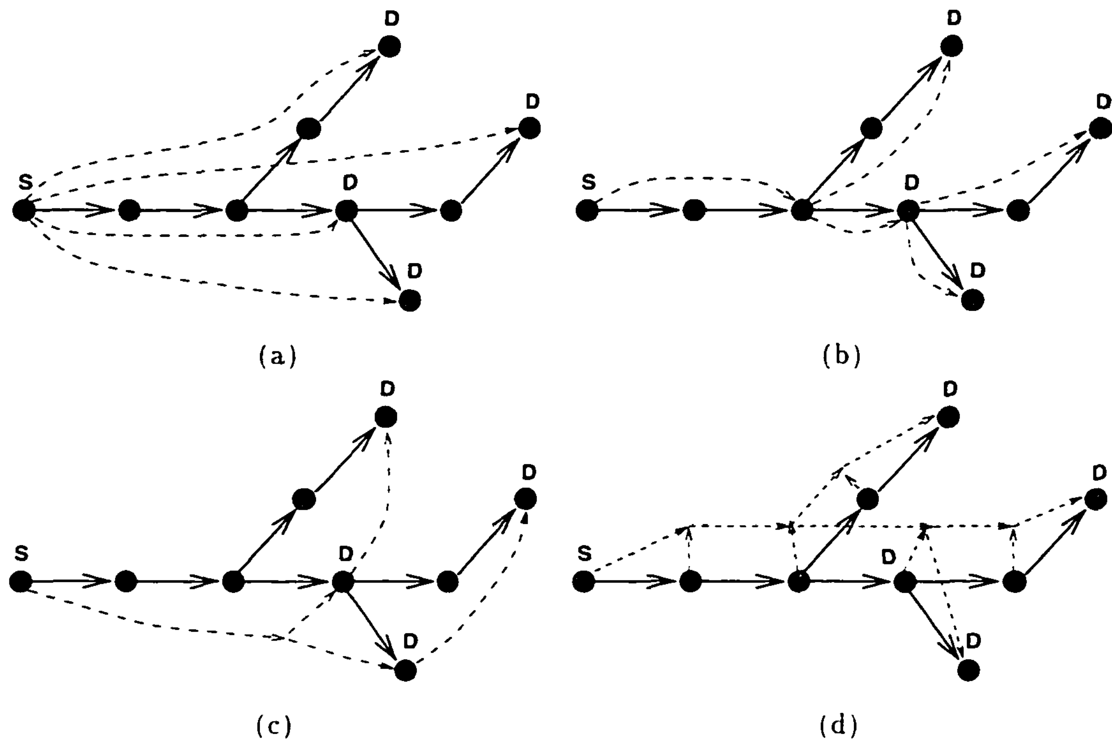


Figure 6.2: Fault-tolerant multicast using backup channels

OS kernel's resource-management policy and the underlying real-time message scheduling policy. Many interesting problems are expected to be found in the course of implementation on various systems.

Another is the group communication issue. While we assumed one-to-one communication throughout this thesis, fault-tolerant one-to-many/many-to-many real-time connections will be necessary and beneficial, particularly in multimedia networking. The key in applying the backup channel approach to multicast communication is the problem of determining where to reserve resources for the fault-tolerance purpose. Some alternatives in establishing backup channels for a multicast tree are illustrated in Figure 6.2. One way is to establish an independent backup channel for each communicating peer as in Figure 6.2 (a). Another way is to add a backup channel to each segment of multicast tree as in Figure 6.2 (b). The third way is to build a different multicast tree of a backup connection as in Figure 6.2 (c). Or, one can combine the second and third ways to build a SFI-like multicast tree as in Figure 6.2 (d). Solid lines represent primary real-time channels and dotted lines backup channels. Each method has advantages and shortcomings, and their performance may be affected by the network topology. The compatibility with existing reservation protocols like RSVP may be an interesting avenue of future research.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BIBLIOGRAPHY

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BIBLIOGRAPHY

[1] *Fibre Channel Physical and Signalling Interface (FC-PH)*. American National Standards Institute, rev. 3.0 edition, June 1992. Working draft.

[2] Y. Amir, P. Melliar-Smith, D. Agarwal, and P. Ciarfella, "Fast message ordering and memebership using a logical token-passing ring," in *Proc. Int. Conf. on Distributed Computer Systems*, pp. 551–560, 1993.

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

[4] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 122–139, January 1994.

[5] J. Arlat, Y. Crouzet, and J.-C. Laprie, "Fault injection for dependability validation of fault-tolerant computing systems.," in *Proc. IEEE FTCS*, pp. 348–355, 1989.

[6] A. Avizienis and G. Gilley, "The STAR computer: An investigation of theory and practice of fault-tolerant computer design," *IEEE Trans. Computers*, vol. 20, no. 11, pp. 1312–1321, November 1971.

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

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

[9] A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma, and H. Zhang, "The Tenet real-time protocol suite: Design, implementation, and experiences," *IEEE/ACM Trans Networking*, vol. 4, no. 1, pp. 1–10, 1996.

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

[11] P. A. Barrett, A. M. Hilborne, P. Verissimo, L. Rodrigues, P. G. Bond, D. T. Seaton, and N. A. Speirs, "The Delta-4 extra performance architecture (XPA)," in *Proc. IEEE FTCS*, pp. 481–488, 1990.

[12] A. Bestavro and G. Kim, "TCP Boston: A fragmentation-tolerant TCP protocol for ATM networks," in *Proc. IEEE INFOCOM*, 1997.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[13] K. P. Birman, "Replication and fault-tolerance in the ISIS system," in *Proc. ACM Symp. on Operating Systems Principles*, Orcas Island WA (USA), December 1985.

[14] S. Butner and R. Iyer, "A statistical study of reliability and system load at SLAC," in *Proc. IEEE FTCS*, pp. 207–209, 1980.

[15] B. Chen, S. Kamat, and W. Zhao, "Fault-tolerant real-time communication in FDDI-based networks," in *Proc. IEEE RTSS*, pp. 141–150, 1995.

[16] R. Chillarege and N. S. Bowen, "Understanding large system failures — a fault injection experiment," in *Proc. IEEE FTCS*, pp. 356–363, June 1989.

[17] R. Chillarege and R. Iyer, "Measurement-based analysis of error latency," *IEEE Trans. Computers*, vol. 36, no. 5, pp. 529–537, May 1987.

[18] S. Chong, S. Q. Li, and J. Ghosh, "Predictive dynamic bandwidth allocation for efficient transport of real-time VBR video over ATM," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 1, pp. 12–23, January 1995.

[19] D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet neworkl architecture and mechanism," in *Proc. ACM SIGCOMM*, 1992.

[20] D. Comer, *Internetworking with TCP/IP*, Prentice-Hall, 1995.

[21] F. Cristian, "Synchronous atomic broadcast for redundant broadcast channels," *The Journal of Real-time Systems*, vol. 2, no. 3, pp. 195–212, 1990.

[22] F. Cristian, B. Dancy, and J. Dehn, "Fault-tolerance in the advanced automation system," in *Proc. IEEE FTCS*, pp. 6–17, June 1990.

[23] S. Deering and R. Hinden, "Internet protocol, version 6 (IPv6) specification," Technical Report Internet RFC 1883, December 1995.

[24] L. Delgrossi, C. Halstrick, D. Hehmann, R. Herrtwich, O. Krone, J. Sandvoss, and C. Vogt, "Media scaling for audiovisual communication with Heidelberg transport system," in *Proc. ACM Multimedia*, pp. 99–104, 1993.

[25] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorthm," *Proc. ACM SIGCOMM*, pp. 3–12, 1989.

[26] B. Dempsey, *Retransmission-Based Error Control for Continuous Media Traffic in Packet-Switched Networks*, PhD thesis, University of Virginia, 1994.

[27] K. Echtle and M. Leu, "The EFA fault injector for fault-tolerant distributed system testing," in *Workshop on Fault-Tolerant Parallel and Distributed Systems*, pp. 28–35, IEEE, 1992.

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

[29] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 368–379, April 1990.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[30] H. Fujii and N. Yoshikai. "Restoration message transfer mechanism and restration characteristics of double-search self-healing ATM networks." *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 1, pp. 149–158, January 1994.

[31] M. W. Garrett, *Contributions Toward Real-Time Services on Packet Switched Networks*, PhD thesis, Columbia University, 1993.

[32] A. Gersht and S. Kheradpir, "Real-time bandwidth allocation and path restorations in sonet-based self-healing mesh networks," in *Proc. IEEE ICC*, pp. 250–255, 1993.

[33] J. Goldberg, M. W. Green, W. H. Kautz, K. N. Levitt, P. M. Melliar-Smith, R. L. Schwartz, and C. B. Weinstock, "Development and analysis of the software implemented fault-tolerance (SIFT) computer," Contractor Report 172146, NASA Langley Research Center, February 1984.

[34] S. J. Golestani, "A stop-and-go queueing framework for congestion management," in *Proc. ACM SIGCOMM*, pp. 8–18, 1990.

[35] M. Grossglauser, S. Keshav, and D. Tse, "RCBR: A simple and efficient service for multiple time-scale traffic," in *Proc. ACM SIGCOMM*, pp. 219–230, 1995.

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

[37] 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*, vol. 9, no. 7, pp. 968–981, September 1991.

[38] U. Gunneflo, J. Karlsson, and J. Torin, "Evaluation of error detection schemes using fault injection by heavy-ion radiation," in *Proc. IEEE FTCS*, pp. 340–347, 1989.

[39] S. Han and K. G. Shin, "A non-intrusive distributed monitoring support in fault injection experiments," in *IEEE International Workshop on Evaluation Techniques for Dependable Systems*, October 1995.

[40] S. Han and K. G. Shin, "Efficient spare-resource allocation for fast restoration of real-time channels from network component failures," in *Proc. IEEE RTSS*, pp. 99–108, 1997.

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

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

[43] S. Han and K. G. Shin, "On providing low-cost dependability by adaptive QoS control in real-time communication services,", 1998. Submitted for publication.

[44] S. Han, K. G. Shin, and H. Rosenberg, "DOCTOR: An integrateD sOftware fault injeCTiOn enviRonment for distributed real-time systems," in *Proc. IEEE IPDS*, pp. 204–213, 1995.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[45] S. Han and K. G. Shin, "A primary-backup channel approach to dependable real-time communication in multi-hop networks," *IEEE Trans. Computers*, vol. 47, no. 1, pp. 46–67, January 1998.

[46] M. Herzberg and S. Bye, "An optimal spare-capacity assignment model for survivable networks with hop limits," in *Proc. IEEE GLOBECOM*, pp. 1601–1606, 1994.

[47] A. L. Hopkins, Jr., T. B. Smith, III, and J. H. Lala, "FTMP-A highly reliable fault-tolerant multiprocessor for aircraft," *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1221–1239, October 1978.

[48] C. Hou, "Design of a fast restoration mechanism for virtual path-based ATM networks," in *Proc. IEEE INFOCOM*, 1997.

[49] M. Hsueh, T. Tsai, and R. Iyer, "Fault injection techniques and tools," *IEEE Computer*, pp. 75–82, April 1997.

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

[51] S. Jamin, P. Danzig, S. Shenker, and L. Zhang, "A measurement-based admission control algorithm for integrated services packet networks," in *Proc. ACM SIGCOMM*, pp. 2–13, 1995.

[52] H. Kanakia, P. Mishra, and A. Reibman, "An adaptive congestion control scheme for real-time packet video transport," in *Proc. ACM SIGCOMM*, pp. 20–31, 1993.

[53] G. Kanawati, N. Kanawati, and J. Abraham, "FERRARI: A tool for the validation of system dependability properties," in *Proc. IEEE FTCS*, pp. 336–344, IEEE, 1992.

[54] D. Kandlur and K. G. Shin, "Traffic routing for multicomputer networks with virtual cut-through capability," *IEEE Trans. Computers*, vol. 41, no. 10, pp. 1257–1270, October 1992.

[55] D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044–1056, October 1994.

[56] B. Kao, H. Garcia-Molina, and D. Barbara, "Aggressive transmissions of short messages over redundant paths," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 1, pp. 102–109, January 1994.

[57] W. Kao, R. Iyer, and D. Tang, "FINE: A fault injection and monitoring environment for tracing the UNIX system behavior under faults," *IEEE Trans. Software Engineering*, vol. 19, no. 11, pp. 1105–1118, November 1993.

[58] R. Kawamura, K. Sato, and I. Tokizawa, "Self-healing ATM networks based on virtual path concept," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 1, pp. 120–127, January 1994.

[59] D. Khun, "Source of failures in the public swithced telephone network," *IEEE Computer*, vol. 30, no. 4, pp. 31–36, April 1997.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[60] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidurai, "The MAFT architecture for distributed fault tolerance," *IEEE Trans. Computers*, vol. 37, no. 4, pp. 398–405, April 1988.

[61] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger, "Distributed fault-tolerant real-time systems: The MARS approach," *IEEE Micro*, pp. 25–40, February 1989.

[62] H. Kopetz and G. Grunsteidl, "TTP – a protocol for fault-tolerant real-time systems," *IEEE Computer*, vol. 27, no. 1, pp. 14–23, January 1994.

[63] J. Kurose, "On computing per-session performance bounds in high-speed multi-hop computer networks," in *Proc. ACM SIGMETRICS*, pp. 128–139, 1992.

[64] J. Lala and L. Alger, "Harware and software fault tolerance: A unified architectural approach," in *Proc. IEEE FTCS*, pp. 240–245, 1988.

[65] J. H. Lala, R. E. Harper, and L. S. Alger, "A design approach for ultrareliable real-time systems," *IEEE Computer*, vol. 24, no. 5, pp. 12–22, May 1991.

[66] H. lawson, "Cy-clone: An approach to the engineering of resource adequate cyclic real-time systems," *Journal of Real-Time Systems*, vol. 4, no. 1, , 1992.

[67] H. Madeira and J. Silva, "Experimental evaluation of the fail-silent behavior in computers without error masking," in *Proc. IEEE FTCS*, pp. 350–359, 1994.

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

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

[70] A. Mehra, A. Indiresan, and K. G. Shin, "Resource management for real-time communication: Making theory meet practice," in *Proc. IEEE RTAS*, pp. 130–138, 1996.

[71] S. Mishra, L. L. Peterson, and R. D. Schlichting, "Consul: A communication substrate for fault-tolerant distributed programs," Technical Report 91-32, University of Arizona, November 1991.

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

[73] K. Murakami and H. Kim, "Comparative study on restoration schemes of survivable ATM networks," in *Proc. IEEE INFOCOM*, 1997.

[74] A. K. J. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*, PhD thesis, MIT, February 1992.

[75] C. Parris and D. Ferrari, "A dynamic connection management scheme for guaranteed performance services in packet-switching integrated services networks," Technical Report TR-93-005, UC Berkeley, 1993.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[76] C. Parris, H. Zhang, and D. Ferrari, "Dynamic management of guaranteed performance multimedia connections," *Multimedia Systems Journal*, vol. 1, no. 6, pp. 267–283, 1994.

[77] L. L. Peterson, N. C. Hutchinson, S. W. O'Malley, and H. C. Rao, "The x-Kernel: A platform for accessing internet resources," *IEEE Computer*, vol. 23, no. 5, pp. 23–33, May 1990.

[78] D. Powell, G. Bonn, D. Seaton, P. Verissimo, and F. Waeselynck, "The Delta-4 approach to dependability in open disributed computing systems," in *Proc. IEEE FTCS*, pp. 246–251, 1988.

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

[80] M. Rela, H. Madeira, and J. Silva, "Experimental evaluation of the fail-silent behavior in programs with consistency checks," in *Proc. IEEE FTCS*, pp. 394–403, 1996.

[81] H. Sakauchi, Y. Nishimura, and S. Hasegawa, "A self-healing network with an economical spare-channel assignment," in *Proc. IEEE GLOBECOM*, pp. 438–443, 1990.

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

[83] Z. Segall et al., "FIAT – fault injection based automated testing environment," in *Proc. IEEE FTCS*, pp. 102–107, 1988.

[84] N. Shacham, "Multipoint communication by hierarchically encoded data," in *Proc. IEEE INFOCOM*, pp. 2107–2114, 1992.

[85] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service," Technical Report INTERNET-DRAFT draft-iet-intserv-guaranteed-svc-05.txt, July 1996.

[86] K. G. Shin and H. Kim, "Derivation and application of hard deadlines for real-time control systems," *IEEE Trans. on System, Man, and Cybernetics*, vol. 22, no. 6, pp. 1403–1413, November 1992.

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

[88] D. Siewiorek, V. Kini, and H. Mashburn, "A case study of C.mmp, Cm*, and C.vmp: Part i - experiences with fault-tolerance in mulitprocessor systems," *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1178–1199, October 1978.

[89] J. Simmons and R. Gallager, "Design of error detection scheme for Class C service in ATM," *IEEE/ACM Trans Networking*, vol. 2, no. 1, pp. 80–88, 1994.

[90] J. Stankovic, D. Niehaus, and K. Ramamritham, "Springnet: A scalable architecture for high performance, predictable, and distributed real-time computing," Technical Report TR-91-74, University of Messachusetts, 1991.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[91] W. Stevens. *TCP/IP Illustrated Volume 1*. Addison-Wesley. 1994.

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

[93] L. M. Thompson. "Using pSOS+ for embedded real-time computing." in *Proc. COMPCON*. pp. 282–288. 1990.

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

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

[96] B. Venables. W. Grover. and M. MacGregor. "Two strategies for spare capacity placement in mesh restorable networks." in *Proc. IEEE ICC*. pp. 267–271. 1993.

[97] A. Vogel. B. Kerherve. G. Bochmann. and J. Gecsei. "Distributed multimedia and qos: A survey." *IEEE Multimedia*. vol. 2. no. 2. pp. 10–19. 1995.

[98] R. Vogel. R. Herrtwich. W. Kalfa. H. Wittig. and L. Wolf. "QoS-Based routing of multimedia streams in computer networks." *IEEE Journal on Selected Areas in Communications*. vol. 14. no. 7. pp. 1235–1244. September 1996.

[99] J. Whalen and J. Kenney. "Finding maximal link disjoint paths in a multigraph." in *Proc. IEEE GLOBECOM*. 1990.

[100] Z. Whang and J. Crowcroft. "Quality-of-Service routing for supporting multimedia applications." *IEEE Journal on Selected Areas in Communications*. vol. 14. no. 7. pp. 1228–1234. September 1996.

[101] J. Wroclawski. "Specification of controlled-load network element service." Technical Report INTERNET-DRAFT draft-iet-intserv-ctrl-load-svc-02.txt. June 1996.

[102] Y. Xiong and L. Mason. "Restoration strategies and spare capacity requirements in self-healing ATM networks." in *Proc. IEEE INFOCOM*. 1997.

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

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

[105] L. Young and R. Iyer. "A hybrid monitor assisted fault injection environment." Technical Report CRHC-92-04. University of Illinois. Urbana. March 1992.

[106] H. Zhang and D. Ferrari. "Rate-controlled static-priority queueing." in *Proc. IEEE INFOCOM*. pp. 227 - 236. 1993.

[107] H. Zhang and S. Keshav. "Comparison of rate-based service disciplines." in *Proc. ACM SIGCOMM*. pp. 113–121. 1991.

[108] H. Zhang and E. Knightly. "Providing end-to-end statistical performance guarantee with bounding interval dependent stochastic models." in *Proc. ACM SIGMETRICS*. pp. 211–220. 1994.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[109] H. Zhang and E. W. Knightly, "RED-VBR: A renegotiation-based approach to support delay-sensative VBR video," *Multimedia Systems Journal*, vol. 5, no. 3, pp. 164–176, 1997.

[110] L. Zhang, "Virtual Clock: A new traffic control algorithm for packet-switched networks," *ACM Trans. Computer Systems*, vol. 9, no. 2, pp. 101–124, May 1991.

[111] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Network*, pp. 8–18, September 1993.

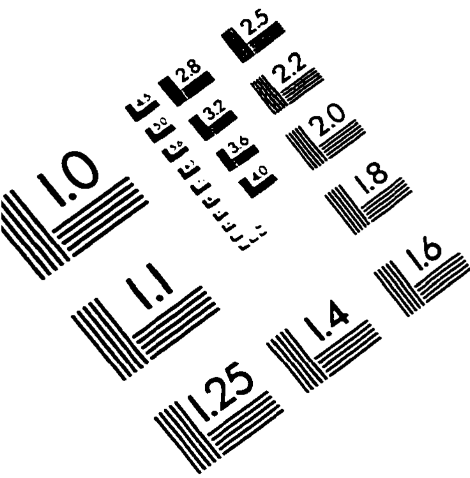
[112] Q. Zheng and K. G. Shin, "Fault-tolerant real-time communication in distributed computing systems," in *Proc. IEEE FTCS*, pp. 86 – 93, 1992.

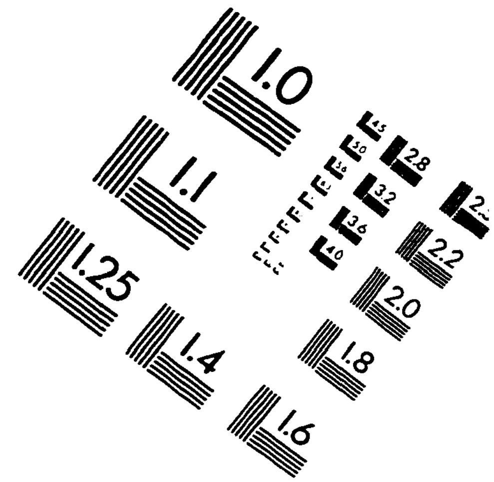
[113] Q. Zheng and K. G. Shin, "Establishment of isolated failure immune real-time channels in HARTS," *IEEE Trans. on Parallel and Distributed Systems*, vol. 6, no. 2, pp. 113–119, February 1995.

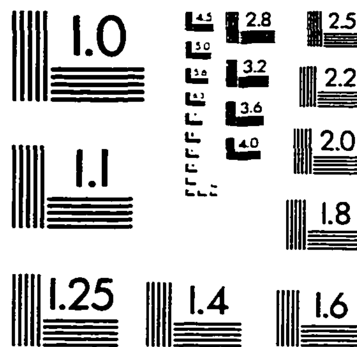
[114] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Trans. Communications*, pp. 1096–1105, 1994.

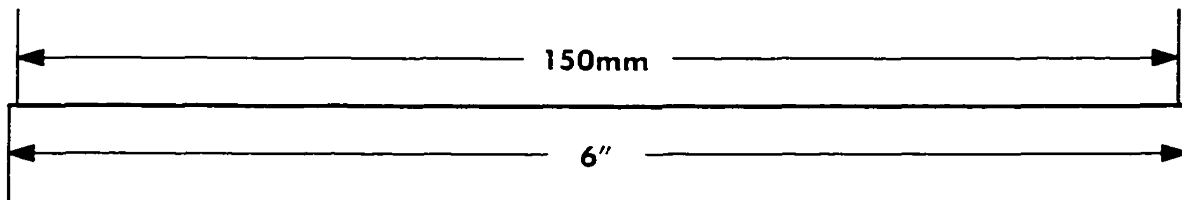
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

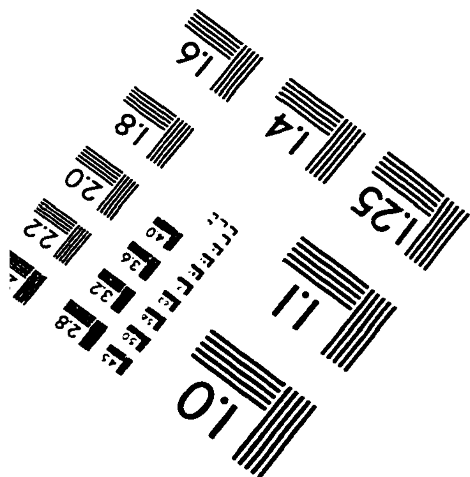
IMAGE EVALUATION TEST TARGET (QA-3)

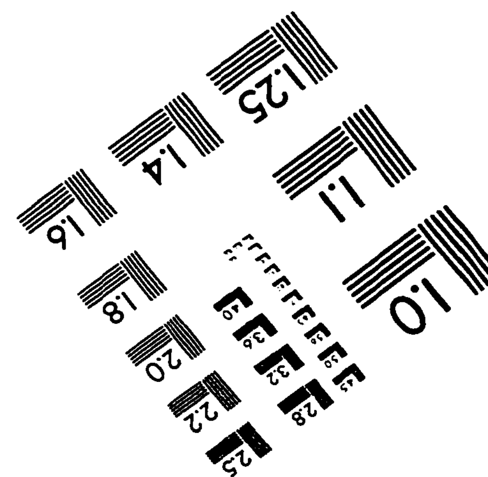












APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.