ISBN 0-315-55615-3

THE UNIVERSITY OF ALBERTA

SELFHEALING NETWORKS

A Distributed Algorithm for k-shortest link-disjoint paths in a multi-graph with applications in real time network restoration

By

WAYNE D. GROVER M.Sc., P.Eng.

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

FALL 1989

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR :

WAYNE D. GROVER M.Sc., P.Eng.

TITLE OF THESIS :

Selfhealing Networks - A Distributed Algorithm for k-shortest link-disjoint paths in a multi-graph with applications in real time network restoration

DEGREE : DOCTOR OF PHILOSOPHY

YEAR THIS DEGREE GRANTED : 1989

[illegible]

Wayne D. Grover
8944 116 Street
Edmonton, Alberta
T6G 1P8

Date : Sept 4, 1989

"But what if Maine has nothing to say to California ?"

1915 . . . upon opening the first transcontinental
telephone service (unattributed newspaper report)

"At 100,000$ a minute, you can't afford a restoration system that's big, slow and cumbersome"

1988 . . . GTE advertisement for "Restolink" emergency repair
splice kit for fiber optic cables carrying tens of
thousands of transcontinental telephone calls

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled

"SELFHEALING NETWORKS - A Distributed Algorithm for k-shortest link-disjoint paths in a multi-graph with applications in real time network restoration"

submitted by WAYNE D. GROVER in partial fulfillment of the requirements for the degree of DOCTOR OF PHILOSOPHY

[illegible]
(supervisor)

[illegible]
(external)

[illegible]

[illegible]

[illegible]

[illegible]

Date: July 14 , 1989

DEDICATED TO

Jutta K. Preiksaitis,

an applied researcher and an inspirational role model

and to

The Province of Alberta and its noble experiment;
the Alberta Telecommunications Research Centre

for their vision and leadership

ABSTRACT

We introduce a new technique for distributed real time restoration of high capacity telecommunications transport networks. Network restoration is the process of rerouting the digital carrier signals of one transmission span via the spare capacity on diverse facilities, when a failure occurs. The widespread installation of large capacity fiber optic transmission systems (FOTS) has created a need for advanced forms of restoration, because cable-borne FOTS are highly susceptible to cable cuts. Attention has so far been focussed on the idea of *centralized* control of automated digital crossconnect systems (DCS) for future restoration systems but there are outstanding problems in speed of response, central database integrity, and dependence on telemetry during a span failure.

We develop a new method in which the computation of restoration plans is automatically distributed over the DCS's of a network, so that reroutings are computed and executed autonomously *without any recourse to central control or global network knowledge*. Every node derives isolated crosspoint operating decisions which, taken collectively at the network level, form coherent large-scale restoration plans. Our method uses a protocol for the isolated manipulation of quasi-static indications, called *signatures*, which are impressed, transparently, onto every carrier signal of the network. Every node resides in, reacts to and modifies a sea of signatures within its vicinity. Restoration is only the system level manifestation of their massively parallel collective interaction.

This Selfhealing protocol is characterized by the concurrent execution of numerous task instances that interact asynchronously in defined study network models. Results show that complex but efficient multiple path restoration plans are reliably devised and deployed in under 2 seconds. In one example, 10 restoration paths, up to 13 spans long, were established in parallel over 7 distinct routes by the *blind coordination* of 16 different DCS machines. The restoration pattern was as efficient as topologically possible and took 880 milliseconds to deploy. The method finds 100% of all the paths that are topologically possible in the test networks and yields path length efficiencies of over 95% with respect to those obtained by a centralized reference algorithm. The most important advances are the new paradigm for distributed interaction via signatures and the method for simultaneous synthesis of multiple disjoint paths.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the technical support, constructive critique, discussions and other forms of benefit received from the following: **Dr. P. Goud** supervised me and I am grateful for the many hours of his time as a mentor and good counsel in both technical and personal senses. **Brad Vanables**, devoted two undergraduate co-op terms to work as my assistant. His dedication, enthusiasm, and hard work were exceptional and contributed greatly to implementation of the necessary software tools and debugging of the protocol. **Ray Fortune** leads the unique research environment in which this work was possible. Ray's encouragement and the facilities of the Alberta Telecommunications Research Centre (ATRC), made it possible to conduct this work rapidly and to benefit from interactions with staff at AGT, BNR, AT&T Bell Labs, and Digital Equipment Corporation, Telecom Canada and other laboratories. **Mike MacGregor** collaborated on the study of call-attempt transients, part of which is used as introductory material here. Mike is also thanked for bringing some of the most relevant 'shortest path' literature to my attention. **Jim Sandham** raised many points of an operational nature and, with **Bill Edwards** (both of AGT), assisted in obtaining a **Telecom Canada** research contract to pursue studies on this topic starting in Jan. 1989. **Dr. I. MacDonald** is thanked for frequent discussions and particularly for encouraging me to consider my Selfhealing work as a thesis topic. **Keith Wilcox** and **Dave Johnson**, technical support staff at the ATRC, gave first class support in providing computing equipment, troubleshooting it, and introducing and supporting software tools as needed. Dave particularly helped in real-time profiling for protocol optimization on the PC version of the network emulator. **Eric Eid** conducted several thorough electronic literature and patent database searches which saved much time. **Joanne Whalen**, technical secretary at ATRC, has been unfailingly helpful and cheerful. She is thanked particularly for assistance with numerous figures in this work. **Salva Sivaji** of Northern Telecom is thanked for his experimental verification of the feasibility of Selfhealing signature transport via DS-3 F-bit modulation using equipment not available at ATRC. I have also benefitted from presentation and discussion of this work with these people who raised relevant points, triggered further ideas and increased my understanding of the problem : **D. Smith, H. Eslambolchi** and staff - AT&T Bell Laboratories, **H. Yang, R. Ballart** - Bell Communications Research (Bellcore), **G. Dewitte, P.Richards, Peter Leach, Ram Choma** and others, - Bell Northern Research, **M. Bliss, R. DiMeo** and staff - Network Systems, Digital Equipment Corp., **R. Yates, Bob Enos, J. Hillier** and others - Northern Telecom, **R. Wojinski, C. Rohrs** and others - Tellabs Corporation, **Dr. Hussein Mouftah** and his graduate students - Queens University. I am also grateful to members of the supervisory committee, **Dr.'s Goud, MacDonald, Krzymien,** and **Kamal**, and to the external examiner, **Dr. Chan**, each of who contributed valuable improvements, corrections and clarifications to the thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PHOTOGRAPHIC PLATES

LIST OF SYMBOLS, NOMENCLATURE AND ABBREVIATIONS

Symbols

(u,v)	a link in a simple graph between nodes u and v.
$(u_{i,q},u_{i,q+1})$	the span of the network to be used as the q^{th} span in the i^{th} restoration vector.
(u_{n-1},u_n)	a span between two nodes comprising the nth span of a route
$<(u_{ij},u_{ij+1}),x_{ij}>$	the composite specification of an individual link on an individual span for use in the jth span of the ith restoration vector in a restoration plan
[affected-ports]	set of DCS ports which were connected through DCS matrix to members of [alarmed-ports] prior to failure
[alarmed-ports]	set of working ports directly affected by span failure
[emp-set@span]	set of DCS ports with empty transmit signature registers in specified span
[L]	a set of length values taken from a restoration plan, internally ordered by increasing length
[P]	a set of links comprising a path, internally ordered so that $head([P]_i) = tail([P]_{i+1})$ *for every* $i \in (1..\lvert[P]\rvert-1)$
[[P]]	a set whose elements are themselves sets of links comprising paths. Internally ordered so that $\lvert[P]_i\rvert \leq \lvert[P]_{i+1}\rvert$ *for every* $i \in (1..k)$; $k = \lvert[[P]]\rvert$
$[[Pa]]_{uv}$	the *set of actual or experimental paths* found in a restoration event for spancut (u,v).
$[[Pt]]_{uv}$	the *set of theoretically ideal rerouting paths* for spancut (u,v).
$[[P_t]]_{uv}^k$	a set comprised of k paths ordered by increasing length from the ideal restoration plan for spancut (u,v)
$[r_{AB1}..r_{ABk}]$	the set of restoration vectors comprising a *restoration plan* R_{AB}
[rec-set@span]	set of DCS ports with active receive signatures in specified span
[send-set@span]	set of DCS ports with active transmit signatures in specified span
[spans]	the set of all logical spans *at a node*
[S]	the set of all logical spans **in a network**
A[u,v]	adjacency matrix
alarm-span	span on which failure has occurred
C	number (or average number) of spans connected to a node
c(u,v)	the length or cost of one link (u,v)
C([P])	the total length of a path **[P]**

C[u,v]	**cost matrix**
current-class	**a family of signatures emanating from a common spancut, for which the current Selfhealing task was invoked.**
d	**span distance (km)**
fsig	**signalling rate available for signature repetition or transfer (kb/s)**
G = (N, L)	**a simple graph network comprised of a set of nodes and a set links between nodes with at most one symmetric link pair between any two nodes**
G = (N, S, W, R)	**a real transport network multigraph comprised respectively of a set of nodes, spans, working and spare circuit quantities.**
int-port	address of port on DCS machine with event causing the current SH task interrupt
int-span	span in which int-port is found
IRV	initial repeat value (can be a network constant or assigned to each node)
k	a number of working paths between the same end-nodes for which restoration rerouting is required.
maxrepeats	a network wide constant limiting the repeatering of signatures
MTBF	mean time between failure
MTTR	mean time to repair
$\mathcal{M}x(R_{xy})$	end-to-end mapping preservation function at node x for restoration plan R_{xy}
n(i)	the logical length of the i th restoration vector
nbits	number of bits in a signature (including checksum and preamble synch)
NID	node identifier
P	processor speed normalized to a SUN 3/60
PLE	path length efficiency, a measure of the total path length used in a restoration plan
PNE	path number efficiency, a measure of the total number of paths found in a restoration event
R_{AB}	a restoration plan for span (A-B) comprised of a set of k individual restoration vectors of possibly differing lengths
r_{ab}	a simple routing vector between nodes a,b
reach-permission	= (maxrepeats - IRV) ; IRV may be negative
RL	repeat limit (logical spans)
S	signature transfer rate (kb/s)
spancut	the event of a transmission span being completely severed

SPP	shortest path probability
STT	signature transfer time
t_{95}	time by which restoration is complete on 95% of all spancuts in a network
T_{AB} or T(A,B)	number of link disjoint paths topologically possible between nodes A,B in the given network
t_{p1}	time of first path restored in a restoration event
t_{pav}	average path restoration time in a restoration event
TPL	total path length (of a restoration plan) in logical spans
t_R	completion time of last path to be completed in a given restoration event
TRL	threshold repeat limit
v^*	carrier facility propagation velocity
W_{AB} or W(A,B)	the number of working circuits on span (A-B)
$x_{i,q}$	an individual link in the set of parallel links on the span (q, (q+1)) allocated to the ith restoration path of a restoration plan.
mapping$_{uv}$	a relational set indicating paths of an actual restoration plan with length equal to paths in the corresponding ideal restoration plan

Nomenclature

[ ]	a set of items ordered by increasing size
$[\,]_i$	the ith item of an ordered set
[[ ]]	a set of ordered sets, itself ordered by increasing size of its member sets
|[ ]|	the number of elements in a set
|[[ ]]|	the number of subsets in a set of sets
(X-A-B- ....-Y)	denotes one complete route for restoration of (X-Y)
(X-A-B-D or Z-H-Y)	denotes an incomplete segment of a restoration route
(X-Y) or (Y-X)	denotes the span between network nodes X and Y
(x_1@l_1, x_2@l_2, ..)	listing of number of paths (x_i) at each path length (l_i) in a restoration plan
[-]\x	all elements of set [-] except x
\ or NOT IN	set exclusion
already-sending(span,port)	a boolean function which returns true if there exists any TS in span which has its precursor at port
cancel-tx-sigs(*port*)	a function which suspends all signature rebroadcasts from a precursor at *port*.
class (*sigspec1,sigspec2*)	returns a pair of alarm-node names extracted from the specified signatures

complement(*portx, porty*)	tests for complement relationship between RS,TS signatures at specified ports
complement(*port*)	abbrev. form of above to test for complement in RS, TS of same port
DCS n/x	designation of a digital crossconnect system interfacing at hierarchical level n and performing switching at hierarchical level x or higher.
ϵ or IN	set inclusion
index congestion	a phenomenon in which the time for Selfhealing rises due to excessively long-lived remnant signatures
networkxF	designation of a network model generated by uniform scaling of all spans in *network* by the factor F.
nullout(*port*)	a function which resets TS registers at specified port to non-active background state
O(f(n))	time complexity of form f(n)
precursor(*TS@port*)	the receive signature which currently justifies rebroadcast of the specified transmit signature
remnant signature	all signatures temporarily extant in the network after reverse-linking on a given index mesh
sense(*sigspec1,sigspec2*)	returns *same* or *opposite* depending on relative directions of two signatures
sig-spec	the address specification of a signature in some port of a DCS
reg.<field>@port	the structure of a sig-spec
spancut	the occurance of a failure on a span in which all working and spare circuits are failed
topology shift	a change in the topology of a set of restoration paths associated with a change some parameter value
unsatisfied index	a forward flooding mesh of signatures on which reverse-linking has to been triggered
Xpoint(*portx,porty*)	a function which operates bi-directional DCS crosspoint between specified ports
|	such that

Abbreviations

AIE	alarm interrupt enable
APS	automatic protection switching system
BER	bit error ratio
BT	British Telecom
CCITT	International Telephone and Telegraph Consultative Committee

CDT	Call Dropping Threshold
CGA	Carrier Group Alarm
DCS	Digital Crossconnect System
EOC	embedded operations channel
FOTS	Fiber optic transmission system
FSM	finite state machine
HRDP	hypothetical digital reference path
LATA	local access and transport area
MD	MetaDijkstra algorithm
O/S	operating system (of a computer)
OSI	open systems interconnection
PSR	port status register
RA	return alarm
RS	receive signature (register)
RSDEL	receive signature change (interrupt flag)
SCIE	signature change interrupt enable
SH	Selfhealing (task or process)
SIE	Signal Identity interrupt enable
SIGID	signal identifier
SONET	Synchronous Optical Network
STD	state transition diagram
std dev	standard deviation
STS-1	synchronous transport signal - level 1
STT	signature transfer time
TE	transport entity
TIB	temporary index blocking
TLL	transient linked list
TS	transmit signature (register)
VF	voice frequency

CHAPTER 1 INTRODUCTION

1.1 The Restoration Problem

At 11:20 A.M. E.S.T on September 21 1987, a work crew of the Public Service Electric and Gas Company was using a power auger to dig a hole for a utility pole in Trenton N.J., between New York and Philadelphia. The auger hit and severed a 3/4 inch polyethylene jacketed 12-fiber lightwave cable which was a backbone carrier facility of one of the heaviest telecommunications routes in the world. Had only one fiber failed, an automatic span protection switching (APS) system would have switched traffic to a dedicated standby fiber in the same cable, within about 200 msec. However, the protection subsystem was now helpless, with all working and standby systems severed by the cut.

About two seconds after the cut, an estimated 100,000 affected telephone calls and computer data connections between New York, Philadelphia, Washington, Boston and dozens of smaller centres on the Eastern seaboard, plus traffic destined for overseas, went from a state of temporary interruption to being permanently abandoned as numerous telephone switching centres "busied-out" all trunks affected by the carrier system failures. In total, about 125,000 trunks were out of service. Spares on physically diverse routes existed, but it took over two hours to restore manually as much capacity as possible over those routes. Physical repair was finished in seven hours through an almost military emergency mobilization of major equipment and the efforts of hundreds of personnel. The primary event resulted in massively mis-engineered traffic loading on parts of the network not directly affected by the fault. Long distance telephone service over large parts of the network experienced heavy blockage for several hours. An undisclosed number of private leased data lines were out of service for the entire day. Amongst those affected was United Airlines, which lost all of its connections to retail travel agents for the day and a large data processing company lost service to thousands of its customers. [Fole87]

The above incident is a dramatic but increasingly frequent illustration of the need for an advanced method of carrier facility restoration. This is formally called the *restoration problem* in modern telecommunications practice. The objective in restoration is to reroute carrier signals rapidly and accurately via diversely routed spare transmission capacity in a network when a failure takes down both the working and spare links on a given span or, when a partial span cut fails more working systems than are protected by the APS subsystem. This problem is significantly different from the well-studied packet routing and call routing problems and presents very demanding real time computational challenges. To date there has been no published report of a *distributed real time* solution to the restoration problem. Such a technique is the main contribution of the present research.

The recent interest in an automated system for restoring route outages in the inter-office transport network is motivated by two developments: (a) the widespread deployment of fiber optic transmission systems (FOTS) and, (b) the simultaneous development of electronic digital crossconnect systems (DCS). These trends create both a new need and a new opportunity: fiber networks carry higher capacities in a single cable than ever before and they are experiencing outages due to cuts at an alarming rate. At the same time, remotely-controlled electronic DCS machines are being designed to improve restoration times by *centralized automation* of the traditional process of restoration by manual rearrangement of carrier signals at the DSX crossconnect panel.

The hypothesis of this work is that the deployment of DCS machines represents an opportunity for an advance in the speed and simplicity of network restoration which is *well beyond the obvious step of automating today's manual restoration methods*. That is, there is an opportunity to use the new DCS technology to address restoration in a completely new way, rather than simply to automate today's methods. Our idea is to recognize that every DCS machine is also a computer and to consider the entire transport network as one distributed multi-processor. We have then posed the question: *"How can the computation of network restoration plans be distributed over the DCS processors of a network so that reroutings are computed and executed autonomously in real time, without recourse to any centralized control or databases of global network configuration ?"* The hypothesis that DCS machines can autonomously restore the network in real time without reliance on central control or databases we call "the Selfhealing Network". Selfhealing does not replace centralized supervisory control of the network. Rather, it is envisaged as a real time assistant to centralized operations systems which will : (a) restore cable cuts without loss of connections in progress, (b) replace span protection switching subsystems in transmission equipment, (c) reduce speed and database requirements for centralized network control systems, (d) reduce the structural availability and redundant capacity requirements of fiber optic networks.

In the following chapters, we will show how DCS machines can be endowed with the ability to compute isolated crosspoint decisions, rapidly, safely and autonomously, which taken collectively at the network level, synthesize coherent and efficient restoration plans comprised of multiple fully link-disjoint carrier signal paths, even though no DCS node has a global view of the network before, during or after the event.

1.2 Severity and Economic Impact of the Restoration Problem

The rapid deployment of fibre-based networks since 1981 has created the potential for major service outages when fiber cables are damaged. Bit rates in commercial fiber optic transmission systems (FOTS) have risen from 6.3 Mb/s in 1981 to 1.2 Gb/s in 1988. At 1.2 Gb/s, a single fiber carries 16,128 voice circuits (or equivalent data) and there may be up to 48 fibers in a cable. This high capacity plus advantages in cost, bandwidth, payload flexibility, quality of transmission and immunity to electromagnetic interference have given digital fiber optic technology a position as the medium of choice for the future transport network. In fact, high-speed multiplex and transmission standards for the future are being exclusively optimized for lightwave technology [SONET88].

Despite these advantages however, FOTS is a *cable-based* technology and has proven susceptible to frequent damage. From 1986 to 1988, at least a half-dozen events similar to the Trenton event cited above have been reported. These have been due to such hazards as: construction work, lightning strikes, rodent damage, craftsperson error, fires, train derailment, bullets, vandalism, car crashes, ship anchors, trawler nets, and even shark attacks! [Szen86], [Dora86], [Nell86], [Vick87], [Saul86], [Abe88a], [Fole87], [Roec87], [Marr87], [LiCa88], [Meye88], [Eckh88], [Mayo88], [Titc88], [Krei88], [Zorp89]. As Plates 1 and 2 indicate, revenue losses of $US 75,000 to 100,000 per minute of outage have been reported for such events (see also [Nell86]). The large capacity of FOTS, and the fact that they become even more economic when large amounts of traffic are aggregated to load a fiber system, means that there is a tendency towards sparsely connected networks with fewer but more heavily-loaded routes [AbRi86] [RoRo86]. Fiber cable cuts are therefore not only more frequent than facility outages in microwave networks, but each cut generally has more serious consequences in terms of numbers of connections lost and the subsequent blocking performance of the network [RoRo86].

Recognizing these trends, cable vendors and network planners have taken complementary steps to address the problem. Cable equipment vendors are improving methods of cable location, typified in Plate 1, and methods of fast physical repair by splicing, as evidenced in Plate 2. These approaches aim to improve *structural availability* by intrinsic physical means. Network planners are simultaneously pursuing means for synthetically maintaining *service availability* through improved restoration of traffic via spare facility routes. New network planning criteria are being adopted in which physically diverse routes and closed topologies are mandatory, so that no major centre could be totally isolated by a single facility cut and so that relatively high levels of redundancy exist overall. Whereas microwave networks tended topologically towards minimum spanning trees (see [Chri75], [Thom76]) fiber-based networks are evolving to closed topologies in which no single span loss can disconnect part of the network graph ([Grov86], [Farl81], [RoRo86] advocate and describe such topologies). In addition, network planners are pursuing automation of the previously manual 'DSX' cross-connect panels [TelC83] at which the carrier facilities have to date been rearranged when needed. The latter is intended as the means for providing the agility for relatively quick reconfiguration within the span-diverse topologies that are planned.

REMOVED DUE TO COPYRIGHT RESTRICTIONS

Plate 1. Technical advertisement for cable locating equipment emphasizing the economic hazards of fiber cable disruption.

REMOVED DUE TO COPYRIGHT RESTRICTIONS

Plate 2. Technical advertisement emphasizing importance of fast physical cable repair as one form of restoration.

It is generally recognized that, because of fiber's vulnerability, future transport networks will have route structures in which it is *topologically possible* to restore any single span failure. These properties of the network are not at issue in the present work. It is important for the reader to appreciate that, without the *topological possibility of rerouting, all forms* of restoration system are powerless, regardless of their speed.

What is keenly at issue at the present time however, is how *best to use electronic DCS machines* for fast network restoration. Until the present work, one assumption has been ubiquitous within the field : DCS machines would be centrally controlled, with all crossconnection commands downloaded to each DCS to implement restoration plans, based on global knowledge at the central site. Centralized control will provide great improvement in restoration times with respect to manual methods. The 6 to 8 hours required by manual methods will be reduced to perhaps 10 *minutes*. However, we argue that it is feasible and important to leap-frog this level of performance by reducing restoration times even further, to *below 2 seconds*. This is an objective that to date has not been anticipated. We argue that, in addition, it is important to remove the dependence of a restoration system on access to a correct and up-to-date centralized global database of the network in order for it to act safely and effectively. Let us now look at the special significance of achieving restoration times of less than *two seconds*.

1.3 The Call-dropping Threshold: A New Goal for Restoration Speed

The motivation for pursuing the fastest possible restoration times is not just the economic one of reducing total outage. A marked *characteristic change* in network performance happens if restoration times *below two seconds* can be obtained. It is a property of today's network that, if the interruption due to a cable cut lasts longer than two seconds, all calls in progress *will be lost.* [1] In every cable cut in today's network such loss of all connections is ensured. We consider this so-called *"call dropping threshold"* (CDT) to be the ultimate objective of importance for a restoration system. However, this goal is not presently sought or generally believed to be feasible by any centralized control scheme.

For telephone callers, loss of a connection during the talking phase of a call is arguably just a nuisance: one has simply to hang up and dial again. However, the impact on the numerous data processing and computer communications networks is much more serious. Hundreds of custom data networks exist for which there *is no simple equivalent to hanging up and dialing again*: these are whole networks which may have taken days or weeks to commission completely and to test and debug. Once these networks are 'up' they assume that their dedicated connections are supposed to stay up all the time. In such cases the impact of an interruption is well beyond nuisance value - it is very costly in direct terms and indirectly in lost productivity. Some widely used types of data processing networks require a complete system regeneration to recover from an outage lasting over two seconds and this can take most of a day in a large corporate data network. In addition, sudden disconnection in the middle of active transactions in networks of point-of-sale transaction terminals, automated tellers, airline flight data networks, personal computers, stock and commodity trading networks, news reporting networks, trucking fleet networks, medical organ tissue-matching networks, etc., can cause an uncertain state from which it may be difficult or impossible to recover the application state. Therefore, as the volume of data communications rises in the network it becomes especially important to avoid the loss of connections in progress.

Another detrimental effect of a major call-dropping event is that the dynamic traffic surges that can result may threaten the stability of large telephony central office switches. This issue is far less documented although anecdotal evidence abounds in the industry. To illustrate this additional principle which motivates our work, we have conducted a side-investigation to assess the magnitude of call-reattempt surges that could occur under plausible conditions of trunk group loss, followed by restoration below and above the CDT.

1.3.1 Traffic-related Dynamic Effects of Fast Restoration

A study was undertaken as an adjunct to the present work in order to investigate the dynamic traffic effects at the call-processing layer of the network which can be associated with interruptions in the transport network.[2] The goal of this study was to define manageable but realistic conditions in which to conduct trunk-group cutting experiments. Simulation was used because the phenomenon involves dynamic non-stationary statistical processes. The traffic engineering principle of prime relevance is the sensitivity of a large group to overload, a phenomenon which is well-known to be dramatically nonlinear (cf. [Bear88], [Hill79]).

In the simulations, 83 minutes of real time were simulated at 1 msec resolution. Individual call arrivals were generated randomly according to standard Poisson birth/death processes. The arrival rate and call holding times were adjusted to create an equilibrium trunk utilization of 0.71 erlang/trunk (a typical trunk utilization in the inter-office network) in a group of 50 outgoing trunks, for a total of 35.3 erlangs offered traffic. After allowing time for the arrival/ departure processes and group occupancy states to reach equilibrium, a facility failure was simulated affecting all 50 of the outgoing trunks at time t= 33 minutes. A hypothetical restoration system then restores 80% of the lost trunks, (a) just *after* the call dropping threshold, (b) *just before* the call dropping threshold.

Throughout the simulation, the normal call arrivals process carries on and any call arrival that encounters blocking (all trunks busy) is shunted into a re-attempt pool where it tries again after another random interarrival time. A uniform random re-attempt waiting time distribution was used as a simple model of subscriber re-attempt behavior. Any call-in-progress that is 'dropped' as a result of a trunk failure longer than the CDT also goes into the reattempt pool and obeys the same rules. These relatively simple but representative circumstances are sufficient to give a dramatic demonstration of the value of a restoration system which can beat the CDT, even if only 80% of total restoration is possible.

One selected result (Fig. 1.1) shows the offered call attempts rate versus time under the simulated conditions. In (a) the restoration system acts to restore 80% of the entire trunk group but does so *just after* the CDT. Consequently all calls *are dropped* and go into the statistical reattempt pool. In (b) the same amount of total restoration occurs but it does so *before* the CDT, so that only calls on the 20% of unrestored trunks are dropped and go into the reattempt pool. The sharp contrast in the dynamics of these two cases has to do with inherent nonlinearities of traffic engineering. In a real spancut, dynamic transients of this nature could be expected to occur on every trunk group affected by the failure, so the complete transient in offered load due to loss of a major carrier facility could generally be much worse in reality. The concern with behavior such as in Fig. 1.1 is that it is simply not known with confidence how large, modern software-controlled switching machines *react* to events like this. Many suspect that more than one total crash has been due to such transients. Very fast restoration can therefore mitigate exposure to this type of stress on the call-processing elements of the network.

1.4 Present Methods and Plans for Centralized Restoration

Methods for the restoration of DS-3 (and DS-1) rate digital facilities have evolved little since the first days of commercial digital transmission. The present method for restoration is manual rearrangement at passive DSX-3 crossconnect panels as outlined by Doran [Dora86]. The sequence of patch operations is determined from stored plans or is developed at the time. Crews are dispatched first to patch the rearrangements and then to proceed with physical fault location and repair. The time to restore traffic by manual patching varies significantly from one administration to another but averages from 6 to 12 hours [Dora86], [Nell86], most of which tends to be travel time to the unmanned transmission hubs where the DSX-3 flexibility point is. Fluery reports a *minimum* of 1 hour for restoration by such means. [Fleu87]. In the UK, where population densities are higher and terminal centres are manned 24 hours a day, it is reported that British Telecom achieved manual patch restoration times of 20 minutes to 3 hours [Schi87].

The major problem with manual restoration is the long time needed to recover the required route cards, to ensure such records are up to date, to develop a rerouting plan and to alert and dispatch the personnel needed. Even then, the chance that a manual error or an error in remote coordination can result in a misrouted DS-3 or further disruption of working circuits is a significant concern. Coordination difficulties are especially a concern when failures affect multiple operating jurisdictions, as in the case of Telecom Canada.

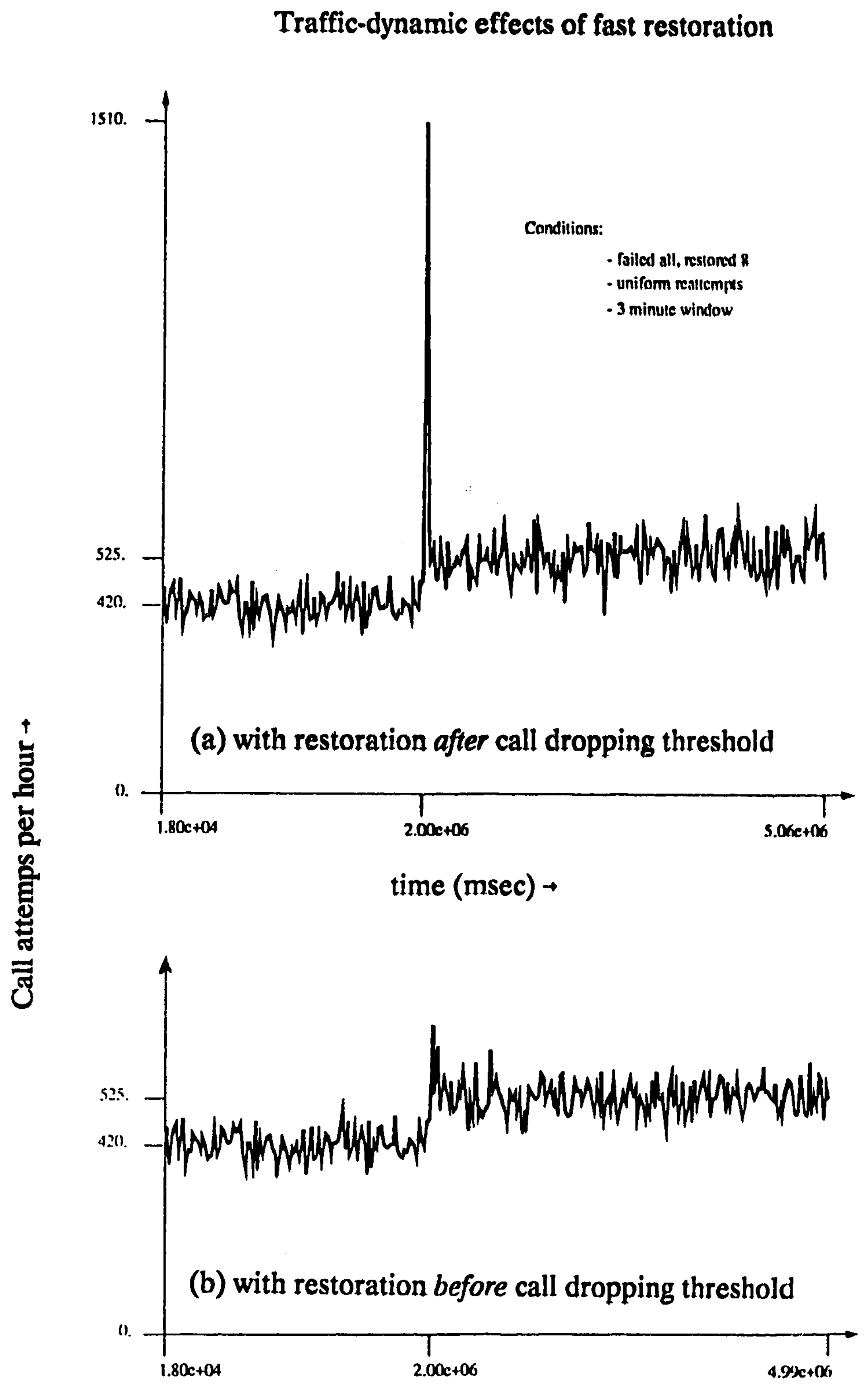


Fig. 1.1 Reattempt transient in offered traffic associated with 80% restoration of a severed trunk group with and without call-dropping on the restorable portion of capacity (50 trunks, 35.3 erlangs)

The predominant view at present is that restoration in the future will be based on DCS machines as a vehicle to automate the above process, without any fundamental change in the basic steps of restoration.[3] Essentially, men in trucks will be replaced by remote telemetry links and the record cards will be replaced by a computer database. Systems of this type are described or proposed by Doran [Dora86], Nellist [Nell86], Vickers [ViVi87], Grover [GrRo85], Fluery [Flue87], Brant [Bran85], Birkwood [BiAl88], Saul [Saul86], Yan [Yan86], Zanella [Zane88], Hasegawa [HaKa87], Sutton [Sutt86], Schickner [Schi87], and Grosvenor [Gros87]. While most are satisfied with the improvement from hours to minutes that centralized automatic restoration will provide, there are still many things left to be desired in such a solution. The centralized approach raises concern about the size, cost, complexity and vulnerability of the surveillance and control complex that will be needed for transport management. In addition, the centralized approach does not ultimately address the hazard of a rerouting error which remains dependent on database accuracy, nor does it reduce the difficulty of multi-administration coordination.[4] A centralized system will be dependent on the ability to maintain a complete, consistent, and accurate database image of the network over years of operation. Eventually one maintenance change that is not immediately and correctly reflected in the database creates the possibility of service-affecting error during a centrally controlled restoration event. Such an event would not only fail to restore the fault but is likely to cause secondary outages from which it could be quite difficult to recover.

With centralized control, all connections will continue to be lost whenever a cable is cut because the outage duration remains much longer than the call-dropping threshold and various data protocol thresholds. This means that *automatic protection switching (APS) systems will continue to be required* in transmission systems to handle single-fiber failures without call dropping, as in current practice. Operating companies will then have to bear the cost of DCS machines *plus* the ongoing use of APS systems. By comparison, a restoration system fast enough to beat the CDT can eliminate the separate requirement for span protection switching and obtain greater benefit out of the existing investment in dedicated spare span lines.

Centralized control of DCS machines requires redundant high-availability telemetry arrangements so that the very facility cuts to be restored do not simultaneously remove all communications with the central control site that will issue restoration commands. This requires redundant communications interfaces on the DCS equipment and special engineering considerations for the operating company to monitor the mapping of circuit provisioning onto the span facility.

Centralized control also runs the risk that a failure in the network will coincide with down-time at the central control site. By comparison, the Selfhealing strategy allows autonomous isolated operation of DCS machines during the emergency. There are no database requirements, and no dependence on telemetry to the central control or availability of the control site. It is however always envisaged that Selfhealing crossconnects would still have telemetry links to a central administrative site for overall supervision and normal provisioning and maintenance. Only in time-critical, restoration events is it necessary for the DCS to function all on its own. The administrative centre still runs the network in an overall sense but it has the help of a field-deployed assistant to deal with the real-time problem of restoration, thereby avoiding the need to specify real-time database intensive operations in the network operations system.

1.5 History of the Present Project and Related Developments

The idea that DCS machines might be able to interact autonomously with each other to solve restoration problems, not necessarily operating under central coordination, began in August 1986 as a response to practical difficulties in the design of a centralized control system for DCS machines. From 1984 to June 1986 the author was involved in the planning and design of DS-3 and SONET-based DCS equipment at BNR. During this time two points became clear:

(1) The surveillance and control system for a DCS-mediated centrally controlled transport network would be very complex, software intensive, and vitally dependent on the integrity of remote telemetry links and of the database image of the network from which restoration plans would have to be derived and downloaded as fast as possible. It was generally recognized that reliable attainment of a ten minutes restoration time would take years of software development for the near-real time control system and surveillance software. In the end, costly APS systems would still be required in addition to DCS machines.

(2) Each DCS machine would be a powerful computer system in its own right. Although DCS's would be relatively isolated and completely without global knowledge of network configuration, each would nonetheless have on board a Motorola 68020 or Intel 386-class 32 bit processor and multitasking operating system as a standard engine for control, maintenance, telemetry, and feature development purposes.

When taken together, (1) and (2) amounted, in the author's view, to the following undesirable and ironic situation: we were trying to develop a cumbersome software-intensive control system for near real time centralized coordination of a distributed network which was itself virtually *littered* with powerful idle processors in every node. An intuitive suspicion arose that there had to be a more natural, more elegant, approach to the restoration problem - one in which DCS machines would efficiently, quickly and reliably solve restoration problems *amongst themselves.*

The ideas behind Selfhealing were however preceeded by a separate idea for the use of *signatures* [Grov87a] simply as an in-service network-level path audit mechanism for DS-3 networks in order to improve the integrity of the network database which is critical for centralized restoration. Signatures provide a *hardware-level* audit trace of individual digital paths wherever they *actually* run in the network, as a check against where the databases *think they run*. The best analogy for this use of signatures is the radioisotope labelling methods used to selectively illuminate certain arteries, etc. in medical diagnostic work.[5] The problem of *carrying* signatures motivated a companion study on methods for auxiliary signalling transport in DS-3 networks. The companion study proposes and analyzes three original approaches to the problem. These are reported in [Grov87a], [GrKr87], [Grov88c] and are reviewed here in Ch.5.

The principal ideas of Selfhealing arose subsequently in 1986-87 as a scheme which relies on the *interaction of signatures*, as opposed to conventional messaging between processors, for distributed determination of restoration plans. This led in early 1987 to the study of example networks by hand to refine the ideas of how signature processing for Selfhealing would work. A PC-based network emulator was then developed which would mechanise the numerous complex signature interactions between nodes in Selfhealing to test the hypothesis that multiple independent restoration routes could be determined in a distributed manner by local processing of signatures. A working Selfhealing protocol was obtained through use of the PC-based emulator and characterized in two representative study networks. These results were published in Nov. 1987 at the Global Communications Conference [Grov87b]. The Selfhealing protocol itself was not fully disclosed then (nor has it otherwise been disclosed prior to this thesis) to preserve ATRC's patent rights. The 1987 publication outlined the approach, the main principles and gave study results. This paper brought a significant response from Bell Labs, Bellcore, DEC, BNR, NEC and others and triggered related studies at those organizations. Exactly a year after the above publication (Nov 1988) Bellcore and NEC Laboratories published papers describing their own approaches to distributed DCS based database-free restoration scheme (and acknowledged them to be Selfhealing-inspired) [YaHa88], [Amir88]. There is however an important difference in both of these other methods which have arisen since this work: they both convert the problem of finding k-shortest independent restoration paths into k separate time-consecutive phases in which one path is found in each phase. This avoids the major difficulty of ensuring *disjointness* between the several paths that are found if they are synthesized simultaneously. However, Selfhealing does find all paths simultaneously in parallel and this remains its largest single advantage over methods of the more recent workers in this area. In addition the Bellcore, NEC, and other workers have yet to report any implementation and characterization studies, such as are presented herein for Selfhealing, for their proposed methods.

Two developments just prior to this writing are the following: (a) Starting in January 1989, Telecom Canada is funding a major research contract at ATRC to study the impact of Selfhealing on the availability, topology, and capacity requirements of the Canadian long haul fiber network as envisaged around 1995. (b) A submission to the US patent office, made in 1987, received official examination and 76 (out of 78) claims have been allowed. An issued patent on this method will therefore be forthcoming.

1.6 Outline of Thesis

The chapter structure of the thesis follows four themes: **Ch's 1, 2, 3, 4** are devoted to *the problem*. This includes background, a review of the literature and definition of new theoretical measures relevant to the performance of a restoration system. **Ch's 5, 6** give *the substance of the new method*, primarily the Selfhealing protocol and the concept and details of event-driven interaction through signatures rather than through messaging. **Ch 7** explains the *research methods* used. **Ch's 8, 9, 10, 11, 12, 13** are all *results* chapters. Within this overall 4 part structure, individual chapters cover the following areas:

Chapter 2 completes the necessary background, begun in this chapter, on transport network technology and terminology and reviews the properties of the real transport networks for which Selfhealing is intended to be used. This serves as a basis for study network models used later in the work.

Chapter 3 is devoted to an original formulation of restoration in terms of a *routing problem* and to establishing the uniqueness of this particular problem in the existing literature on distributed routing algorithms for both data and call-routing applications and in the literature of shortest path problems in graph theory.

Chapter 4 is a necessary preliminary to define quantitative measures of routing performance which are appropriate to the restoration problem and to define the relevant theoretical target against which the performance of Selfhealing (or any other advanced restoration scheme) can be objectively assessed. An original centralized algorithm for finding k-shortest link-disjoint paths in a multigraph is developed and introduced. This is a tool which we will use to determine ideal reference solutions against which the distributed restoration plans derived by Selfhealing can be systematically compared.

Chapter 5 is the first of two chapters which together give the substance of the Selfhealing method. Ch. 5 is devoted to an explanation of signatures and the principles of massively parallel distributed interaction through signatures as opposed to conventional interprocessor messaging. The particular advantage of signature-based interactions to the multigraph routing problem presented by the transport network is explained. DCS based support hardware for Selfhealing signature processing operations and methods for signature transport in the DS-3 signal are also explained.

Chapter 6 is a detailed description of the Selfhealing protocol, ie. a protocol through which restoration plans are derived autonomously in real time by event-driven signature processing.

The net effect of the protocol at the network level is first explained to convey overall concepts. The focus then shifts down into a single node to describe the actions of the protocol in case-by-case terms of signature-manipulations in response to pre-defined signature-related events that can occur at any node. A graphical convention for representing signature relationships and actions is used to aid the explanations. A complete formal specification of the Selfhealing protocol (one that simultaneously puts into effect all of the signature rules described in this Ch.6) accompanies the chapter as Appendix B.

Chapter 7 describes those aspects of the research method which are important to understanding how the method would be implemented in practice and how it was implemented for research purposes in our study network models. Two essential aspects of methodology were (a) the use of truly asynchronous concurrent virtual-time execution of the protocol in order to study network-level behavior from the interaction of all nodes running the protocol and (b) the separate structuring of a protocol module and network emulator so that the protocol is written and executes exactly as it would in a real DCS machine. These methods essentially deliver *experimental results on a prototype implementation*, not functional simulation of an *intended* behavior. An overview is also given of the important mechanised research tools and methods which were developed for timing, visualization, demonstration and debugging. The overall research environment that was developed supported both *synthesis* (to arrive at the desired protocol) and *analysis* activities (to characterize performance).

Chapter 8 presents Selfhealing results obtained for all spancuts in an artificial study network called *Smallnet*. This is a highly connected network with a moderate number of links and nodes that facilitates complete graphical representation of the routing plans derived by Selfhealing showing every link individually. Full graphical representation is an aid to inspection and understanding that is not feasible for the larger study networks and this is Smallnet's primary *raison d'etre*. The results of this chapter are primary evidence that the synthesis objective of the work (ie. a working Selfhealing protocol) has been achieved. Smallnet results are also used as a vehicle for introduction of *restoration trajectory diagrams*. This is a particular type of plot which gives insight into the complex dynamics that occur in Selfhealing and assists in discussion of *index blocking* and other effects which are introduced.

Chapter 9 is devoted to presentation and interpretation of equivalent results obtained in three larger network models which are based on the properties of real transport networks. These are referred to as the *Bellcore*, *US* and *Metro* networks. The data obtained comprise a comprehensive set of 125 spancut experiments in varying conditions of topology, link length, span redundancy, and path finding demand. The results are statistically summarized and selected cases are discussed to show instances of *path spreading*, *index blocking*, and *topology shifting* phenomena.

Chapters 10, 11 and 12 are devoted to parametric studies of important parameters. Ch. 10 explores the effect of the *repeat limit* (RL) which determines the range-of-influence of a Selfhealing event and the free-lifetime of a *remnant signature*. Ch. 11 experimentally characterizes the rate at which the time required for Selfhealing grows with the size of a network. For deployment in real networks it is important to confirm that Selfhealing exhibits polynomial-time, not exponential-time complexity. Ch. 12 explores the relative effects of processor speed and signature transfer time on the speed of Selfhealing and examines the effect that the *ratio* of these two parameters can have on solution topology.

Chapter 13 summarizes the work, identifies the major research findings and outlines areas of further research, some of which are already being undertaken.

CHAPTER 2 DIGITAL CROSSCONNECTS AND TRANSPORT NETWORKS

This chapter gives background on the networks into which Selfhealing will be deployed, and on the intended host machine for Selfhealing; the DS-3 level digital crossconnect system. The topologies, connectivities, sizes and physical extent of the real networks which we survey show the relevance of the network models in which we later report the performance of Selfhealing. Our review of the essential architecture of the DCS machine and the ways in which it is augmented to support Selfhealing is necessary background for later chapters. Some necessary terminology is also introduced.

2.1 Digital Crossconnect Systems

The DCS is a new type of switching machine which automates the functions of the traditional DSX patch panel described in [TelC88]. Unlike the central office telephone switching machine which handles individual telephone calls, a DCS performs switching to manage the configuration of the transport network which carries those calls. A DCS operates directly on the digital carriers of the network, without recognition or processing of individual telephone trunks or other services borne on those carriers. For this reason, and because switching for transport network rearrangements is much less frequent than in normal voice switching, the DCS is also called a *facility switch* or a *slow switch*. Whether manual or electronic, the crossconnect provides a defined interface point at which standard impedance, power levels, digital waveshapes and timing jitter are guaranteed so that all items of equipment that source, sink or transport a digital carrier signal can be interconnected in any manner desired.

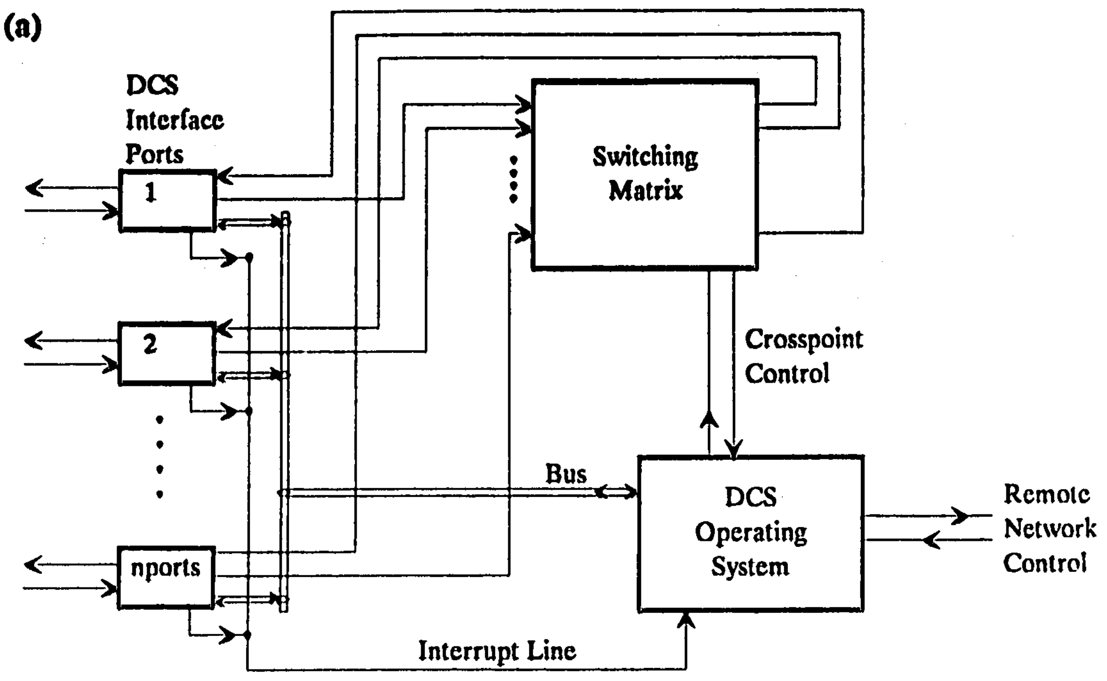
A variety of DCS machines have been developed for various applications and a standardized notation has arisen to describe a DCS. The notation **DCS n/x** denotes two attributes: **n** is the *highest* digital hierarchical transmission level at which the DCS machine can interface to transmission equipment and **x** is the *lowest* digital level at which the DCS can switch sub-multiplexed tributaries. The values of **n** and **x** refer to the standard levels of the North American digital transmission hierarchy, ie: **DS-0** = 64 Kb/s, **DS-1** = 1.544 Mb/s, **DS-2** = 6.312 Mb/s, **DS-3** = 44.736 Mb/s. (see [CCIT85], [AT&T88] and [Smit85] for details of these multiplex levels and for the equivalent European practise.) For example, a DCS 3/1, interfaces at the DS-3 transmission level and can interchange digital tributaries at the DS-1 level or by implication at the DS-3 level. The DCS 3/1 is primarily used to collect DS-1 facilities which share a common destination to load efficiently a single DS-3 to that destination, a role called 'grooming'. DCS 1/0 and 1/1 are used primarily for implementation of private business networks in which dedicated DS-0 and DS-1 connections are provisioned to customers almost on demand. A DCS-3/3 interfaces at the DS-3 level and only performs switching directly at that level. Its primary role is in restoration and reconfiguration of the high capacity, high velocity DS-3 backbone inter-city transport network.

A DCS 3/3 is functionally a simpler machine than a 3/1 and a 3/1 can be used to emulate 3/3 functionality if desired. The pure 3/3 function is nonetheless required in the high capacity transport network because the signal path delay and lower availability of the more complex 3/1 is unacceptable in the DS-3 transport network. While the signal delay through a 3/3 machine is a few hundred nanoseconds, there is a minimum delay of 125 μsec for every equipment stage that has DS-1 processing. Some multiple of this time is typically introduced by a DCS 3/1, sufficient to aggravate the echo-loss-delay planning problem in long distance networks ([HlEv73] or [BTL71]). In the future it is expected that the DCS-3/3 will also be used for direct DS-3 customer networking applications, such as for video conferencing, just as private DS-0 and DS-1 networks are provided today.

In depth descriptions of the design, function and planning for DCS 1/0, DCS 3/0, 3/1 and 3/3 machines can be found in [AvDu86], [ToUe86], [GrLi87], [Tole87], [RoKa88], [DeKa88], [GrMo88a], and [BCor85]. For research purposes we have abstracted from these and other sources, a *representative DCS architecture*, shown in Fig. 2.1(a). The addition of a Selfhealing capability will influence the DCS-3/3 design by adding *signature registers* (TS,RS) and *status registers* (PS) to the port cards and by addition of the Selfhealing protocol as task to the DCS O/S (SH-task), as shown in Fig.2.1(b). Each of these additions is the topic of an entire chapter to follow. The reference architecture is used throughout this work to define a basic computational and interface environment for implementing and specifying Selfhealing. The reference model is not meant to imply that only one DCS 3/3 implementation is possible but rather that for purposes of Selfhealing, these are the essential architectural attributes. The philosophy is that by defining a reference DCS architecture, the output of the work is most generally applicable. Selfhealing can be implemented in any actual DCS-3/3, DCS-3/1, DCS-1/1 and SONET DCS STS-1/1 design, whether using time switching, space switching or a mixture, by translating the specifications for Selfhealing with the reference architecture into corresponding requirements for implementation in any other design. By using this reference architecture we avoid assuming any one manufacturer's DCS design and increase the feasibility of eventual standardization of the protocol and associated signature format.

The reference architecture of Fig. 2.1. is nonetheless most representative of the pure DCS n/n function and we do focus specifically on the *third* level of the digital transmission hierarchy in this work because this is where the prime responsibility for network restoration resides [Dora86]. Notwithstanding the generality which strictly applies, we are in practise most interested in the DCS 3/3 or SONET DCS 1/1 as the preferred vehicles for Selfhealing. Selfhealing in SONET

(a)



(b)

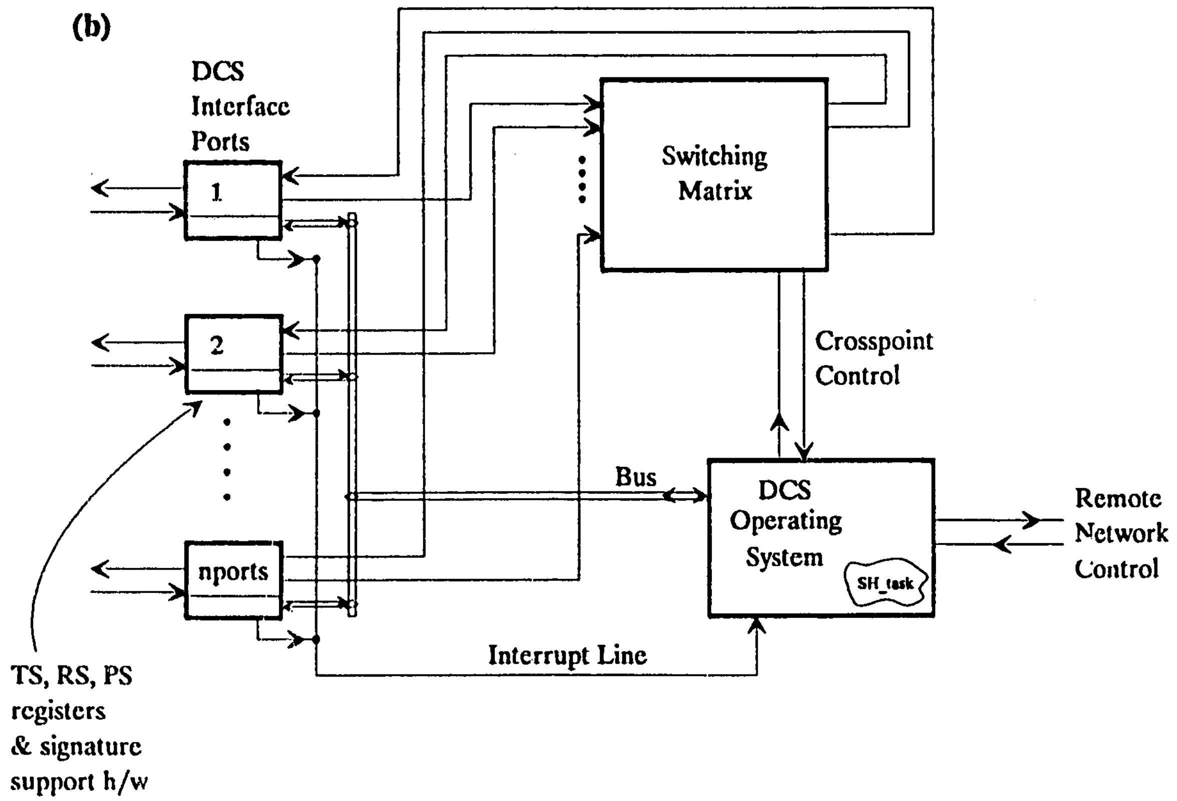


Fig. 2.1 Generic architecture of DCS n/n machine for specification and standardization of Selfhealing. (a) present DCS n/n standard architecture. (b) with enhancements to support Selfhealing.

[SONET88] networks will be functionally identical to that in DS-3 networks except that details of *signature transport* will differ.

2.2 DCS and Transport Network Terminology

Some important aspects of the DCS 3/3 can be highlighted with reference to Fig 2.1. A *port* is a single *bidirectional* interface which meets all relevant DSX-3 interface standards and provides equalization, clock recovery, data regeneration and interface functions to adapt the input signal [CCIT85] to a format compatible with the switch core.[1] Every port on the DCS reference model has receive alarm generation circuitry and access, via data bus or equivalent means, to the DCS O/S. In addition each port in the standard model has access to a common (or vectored) processor interrupt line. To simplify later discussion we assume that each port on the machine has an interrupt vector which identifies that port to the operating system whenever that port is the source of an interrupt. Alternatively, a single interrupt line with subsequent polling can be assumed.

A *transport entity* (TE) is a standard digital multiplex signal defined for the carriage of a certain number of individual voice-equivalents and having a defined framing structure, maintenance overheads etc. The transport entity is the basic unit of transmission capacity which is manipulated by a restoration scheme. eg) a DCS-3/3 crossconnect is designed for manipulation of DS-3 transport entities. The *transport rate* is the bit rate of the corresponding transport entity.

A *facility* is a physical transmission system which conveys a number of transport entities between two restoration switching nodes. The line transmission rate of the facility should not be confused with the rate of the transport entities that are conveyed eg) the FD-565 system uses a 570 Mb/s line rate to transport 12 DS-3 TEs per optical fiber. A facility is characterized by the form of transport entity it employs. So for instance, FD-565 is a DS-3 facility.

In the following work a *link* is a one-way virtual transmission line which conveys one transport entity between two nodes (eg, one fiber of an FD-565 system implements 12 (one-way) links a in a DS-3 transport network) and a *circuit* is the term used for a *bidirectional pair of links* sharing the same end nodes. A *working circuit* has a bidirectional associated pair of in-service transport entities carrying live traffic. The TEs of a working circuit bear a network-wide *path identifier*.[2]

A *spare circuit* is a circuit between DCS nodes that is in a fully equipped and operational condition but is presently not in service and is not part of any end-to-end digital path through the network. A *spare circuit* is in all transmission respects identical to a working circuit and can go into working mode simply by crossconnection to a live signal within the DCS and changing its working/spare status bit. For this work, all circuits of a span are in a working or spare state.[3]

A *span* is the complete set of circuits (working and spare) in parallel between two nodes. A span extends only between crossconnect machines. The *size of a span* is the total number of equipped circuits it includes. eg) the FD-565 facility has a size of 12 (DS-3 links) in each direction. If only eight of the 12 possible circuits from the facility are presently terminated on a DCS machine we would say the size of the span is eight.

A *path* is a series of individual circuits concatenated by connection through DCS machines. Every *working path* (or just path) originates at a node in the network where an individual DS-3 transport entity is first created by a multiplexer or other equipment that originates a DS-3 signal (eg. video codec) and terminates at the node where the DS-3 is finally demultiplexed or otherwise terminated. A *restoration path* (or replacement or rerouting path) is a series of spare circuits concatenated by connection through DCS machines for the purpose of restoring end-to-end continuity of a working path that was disrupted by failure at some span in the route of that path. A restoration path terminates at the two nodes in the network which terminate the span on which the failure occurred whereas a working path terminates where the TE it conveys is actually terminated.

A *path* is specified by its *route* and, if necessary, by stipulation of an individual path number when several paths sharing the same route (ie. sequence of spans) end-to-end. The *logical length* of a path is the number of links (or spans) in series along the route of the path. Unless stated otherwise, the length of a path in this work is its logical length, not its geographical length.

A *route* is the series of spans over which one or more paths are realized. A route may support several paths but every path has only one route.

A *signal identifier* is a unique code assigned to every working transport entity in the network when created by an originating multiplexer or other DS-3 signal *source*. (Source is here meant in the sense of the primary *information source*, not in the sense that a DS-3 signal physically emerges from each repeater or DCS in the signal path.) Every working path conveys one live transport entity and the identifier of the TE carried by a working path is also called the *path identifier*.

The *size of a node* is the total number of ports in service at the node. The *span degree of a node* is the total number of distinct spans emanating from that node, directly connecting it to other nodes in the network.

All *spare* circuits are assumed to have a *null* value in their *signal identifiers* and to continually transmit a keep-alive self-test signal such as framed all-ones or blue signal as defined in [ECSA87b]. Spare circuits (as well as working circuits) are constantly monitored for BER and framing integrity by the receive DCS port function. Any spare circuit without an alarm of its own is available for restoration purposes. If a spare has an alarm indication it will not be used in restoration nor will it trigger restoration.

Span restoration is the problem of restoring end-to-end integrity to the *working paths* affected by any particular *facility* failure between two nodes x,y. In span restoration no knowledge of the ultimate destination of any individual sub-rate payload is assumed or taken into account. For example a DS-3 restoration between nodes x,y may restore some working path via node z, without knowing that some of the DS-1s within the affected DS-3 are ultimately routed to node z. Such payload destination-based restoration is not the goal of span restoration.

Restorability (of a network) is the average, over all spans in a network, of the fraction of working circuits on a spancut that can be restored if limited only by the inherent topology and spare circuit quantities in the network. The *redundancy (of a network)* is the ratio of the total number of spare links to working links in a network (Σspare/ Σworking).[4]

2.3 Representative Transport Networks

2.3.1 Telecom Canada

Telecom Canada's fiber network is based on 565 Mb/s line rate FOTS and interfaces to terminal equipment at the DS-3 level. At 565 Mb/s each fiber carries 12 DS-3s if fully loaded. Three diverse transcontinental facility routes are planned, one of which is the existing 8 GHz digital radio system, which is also uses the DS-3 transport entity. The other two are new fiber routes which are presently in construction. The three main routes will be strategically cross-linked to create route diversity. Each fiber route is based on an 8-fiber cable design, providing 4 bidirectional pairs. Each facility route requires about 110 individual regenerator sections and 22 switching sections which define the spans of a DCS-based restoration plan. [Saul86]

Saul [Saul86] reports that with today's manual DSX-3 patching method in Telecom Canada, cable cuts typically require 4 to 12 hours to restore. The 1986 plan for the Telecom network described by Saul was to achieve a 10 minute restoration time is based on DCS machines and a FOTS surveillance system under centralized control of the whole network from a 'war-room' in Ottawa. As mentioned, Telecom Canada is now in 1989 studying the feasibility of much faster restoration through Selfhealing methods. Saul points out that care is being taken to ensure that in no cases do any two of the same routes share the same physical duct or conduit. The 1995 network plan has a fully closed (restorable) topology with an average of 3.4 spans terminating at every node.[5] (We mean by a 'closed' topologogy that every node is connected to the network by at least two different spans.)

Some observations helpful to the present work, can be made from [Saul86] : (a) The basic transport entity is the DS-3 signal and restoration is to be based on DCS 3/3 functionality, (b) About 66 DCS nodes can be expected from the description in [Saul86] (3 routes x 22 spans/route). The maximum size of any single span is 8/2 x 12 = 36 DS-3 circuits, (c) A

REMOVED DUE TO COPYRIGHT RESTRICTIONS

Plate 3. Long haul US fiber systems planned or in place (from Kessler 1987)

MCI

FUTURE

maximum of 20 span cuts per year is assumed and facilities avoid shared ductwork. Therefore it is realistic to assume single independent span cuts.

2.3.2 AT&T Fibre Network

Nagel and Gray [NaGr86] give a description of AT&T's high capacity DS-3 fiber optic network planned for the early 1990's. AT&T's network is also one of the networks described in Kessler's system map, reproduced here as Plate 3. AT&T uses a mixture of 565 Mb/s (12 DS-3) and 1.7 Gb/s (36 DS-3) FOTS technology, requiring some DCS nodes with hundreds of ports. To a large extent the network is closed and comprised of a minimum spanning tree-like core with peripheral route diversity provided by the lower capacity radio systems.

The AT&T 1990 network will apparently still have 15 nodes which are connected to the remainder of the network by one span. Such nodes are intrinsically subject to isolation by a single span cut and because they are only restorable by physical repair. However, these outlying centres correspond largely to the remaining radio routes which have intrinsically adequate structure availability. Eventually diverse routes must be established to each of these outlying nodes if they are to be automatically restorable by any technique, but this may not be necessary as long as continued growth on these routes is taken up on radio. This leaves a smaller total number of nodes which comprise a core fiber network that is nearly completely closed, even in 1990. The average span degree of the closed (restorable) portion of this network is 3.1.

Another writer on AT&Ts network [Hans87], confirms this intent stating that multiple route connectivity will be eventually established for the 100 largest communities in the U.S., eliminating the risk of isolation from any single-point failure. Because of the adverse effects of delay on voice quality, data throughput, and on the performance of dynamic non-hierarchical call routing (DNHR) algorithms, satellite circuits have been progressively transferred to the terrestrial network and now comprise only 2 percent of the total circuit-miles in the AT&T network, mostly to Puerto Rico and Hawaii.

2.3.3 US Sprint

Almie, Dyer et al [AlDy88] and White [Whit88] have both written on US Sprint's network which employs 9 DS-3 and 12 DS-3 FOTS facilities and is topologically similar to AT&Ts network although it is shown as already being fully closed in its backbone portion in [AlDy88]. An older map of this network is shown in Plate 3. Sprint claims 24,000 miles of cable installed with 41 switching hubs presently established. Network survivability is addressed with a mixture of DCS machines at larger hub sites and specialized Reverse Direction Protection switches which can reverse-direct up to 24 DS-3 circuits so that traffic accessing the network at a position intermediate to a span between DCSs is routed back to the nearest DCS hub for restoration.

When a fiber cut occurs, a mid-span reverse switch is commanded if needed and then a reroute loop through DCS machines is established under central control. This strategy effectively allows a decrease in the length of switching section spans while making diverse route restoration possible for spans whose terminal nodes only have span degree two. This permits an equipment simplification in these circumstances because a full DCS-3/3 system is not needed for locations of span degree two. The full DCS 3/3 function of course remains fully acceptable in this context. These authors report rerouting lengths from typically 3 spans to as many as 6 or 7 but give no data on the time taken for restoration.

2.3.4 Other Commercial Networks and Utility Networks

Several other commercial transport network ventures are shown in the route maps of Plate 3. and some further descriptions are provided by Siperko, Nellist, and Yukawa. Siperko [Sipe88] describes a mid-west US based commercial DS-3 network (Litel) which presently has 3000 miles of FOTS and 1000 miles of digital radio backbone network. Nellist [Nell86] also describes the Litel network. In addition both Nellist and Yukawa et al [YuNo86] describe some extensive networks being developed by railway companies and electric power utilities in the US and Japan. Both of the latter utilities have significant needs of their own for high availability communications, particularly the nuclear electric power business. Both are also interested in DS-3 based video conferencing to reduce otherwise heavy travel loads for management meetings and the ability to involve distant experts in a more effective way in critical times. These utilities have a natural advantage in deploying such networks for their own use; they already own diverse networks of right-of-way, in which to place the new fiber optic cables. Other writers describing or proposing diverse route fiber networks include: Kojima [Koji87], Yamamoto et al. [Yalw86], Abel [Abe88b] and DeMartino [DeMa87].

2.3.5 UK Network

British Telecom's entire network is based on the 140 Mb/s transport entity, which they call a digital block. Fig. 2.2 shows an approximate model of the UK network based on [Sutt86] and [IEE87]. It is the most highly connected of any network found in this survey with an average span degree of 4.0. Schickner [Schi87] and Sutton [Sutt86] both discuss British Telecom's 140 Mb/s Digital Service Protection Network. This system has a hierarchy of centralized control and an in-depth defense against outage comprised of dedicated span protection switching, pre-computed 'fast-make-good' plans and tertiary 'slow-make-good' plans. The high-density highly-meshed structure of the BT network is ideal for exploiting connectivity-capacity tradeoffs and consequently their normal approach is to *pool* protection systems, as also advocated in [Bran85]. Reconfiguration of the pooled protection spans are managed under central control with remote-controlled switching equipment which is functionally similar to the generic DCS architecture of Fig. 2.1.

2.4 Summary

In this chapter we defined a reference architecture for the DCS machine on which Selfhealing will be based and explained why the DCS 3/3 is the most likely vehicle for Selfhealing in practise in today's networks. We considered the properties of several real fiber networks with an interest in their characteristic levels of connectivity and topological closure as input to our later studies. Fig.2.3 summarizes the statistical frequencies of nodal span degree of three of these networks. The average level of span connectivity varies between 3 to 4, a range is reflected in our later study network models. Several other conclusions about these networks are essential background to this work: (1) All the N. American transport networks are based on the DS-3 transport entity and are essentially all terrestrial. (2) All have fully closed or nearly closed span topologies, so restoration by rerouting *is topologically possible* for all spancuts in these networks, either now or in the future. (3) Special efforts are made to keep facility routings physically separate at all times. This, plus the fact that failure frequencies are still low in an absolute sense, means that it is realistic to consider only one span failure at a time.[6]

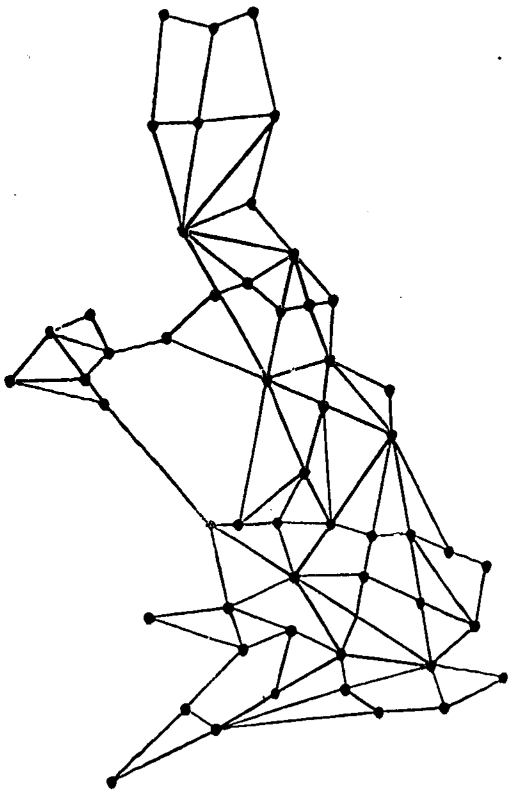


Fig. 2.2 United Kingdom 140 Mb/s transport network showing typically high connectivity and closed topology of a mature fiber-based transport network.

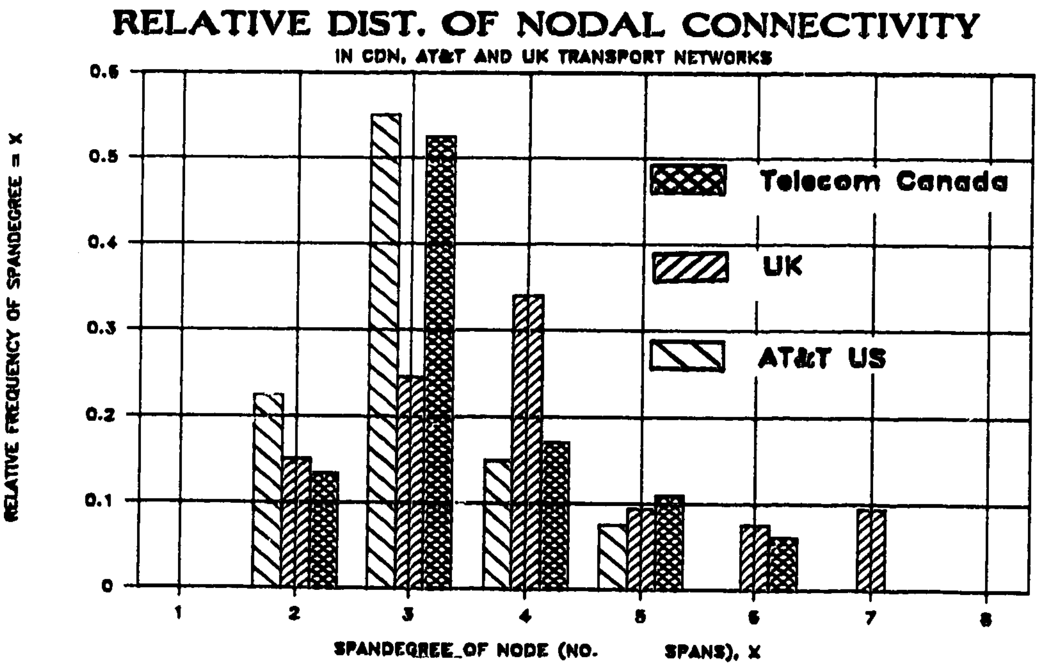


Fig. 2.3 Statistical frequencies of nodal span connectivity in representative US, UK, and Canadian fiber transport networks.

CHAPTER 3 RESTORATION AS A ROUTING PROBLEM

In this chapter we review the literature on routing and graph algorithms to support our finding that restoration is an unaddressed problem in the field of communications routing and in the related field of shortest path graph algorithms. To see the position occupied by Selfhealing in the literature we need to consider it both in terms of the theoretical problem it addresses and the computational model to which it applies. We therefore precede the literature review with a graph-theoretical formulation of the restoration problem and a review of the various computational models found in the literature of routing and path finding problems.

Packet routing in computer data networks and call routing in telephone trunk networks, are the predominant routing problems in the minds of communications engineers today. It is therefore essential for us to explain Selfhealing with respect to those predominant ideas. When formulated in comparison to such schemes, restoration is a clearly different problem. It involves more variables, more simultaneous constraints, and a higher dimension solution-space than either of the two well known routing problems. This is primarily an outcome of the need in restoration to find a multiplicity of fully link-disjoint paths in a multigraph network representation whereas both former problems are treated with simple graph concepts and require single path routings which are not mutually exclusive in any way.

The finding that restoration has apparently not been investigated either as a problem in routing, or in graph algorithms, seems to need some secondary check. The absence of distributed restoration algorithms, or even centralized algorithms explicitly suited to restoration, may be due to the following. The restoration problem historically arose *very* infrequently in previous, microwave radio, transport networks. When it did, it was dealt with by manual rearrangements developed on an ad-hoc basis, perhaps with previously determined guidelines as to which facilities to use in an emergency. Few questioned the adequacy of these methods. Prior to telecommunications deregulation in the US there was also no *real* revenue loss from service outages (everyone places their call again on the same network), and there was little pressure from business customers who can today force the carriers to compete on the basis of *measured* service quality and reliability. In addition, unlike packet routing and alternative call routing for which equipment exists to *implement* the algorithms (ie. X.25 switches, DMS-100's), there was, until very recently, no equipment to implement algorithms for automatic restoration routing, even if such algorithms had been researched. Only since 1986/87 has the restoration problem been given its equivalent implementation engine: the DCS machine.

In addition, the industry has been largely preoccupied with the assumption of *centralized control* of DCS machines. The analogy has not been widely seen that just as the advent of call switchers with powerful embedded processors (eg. DMS-100) permitted distributed adaptive call so could the advent of DCS machines lead to a fast and elegant means for distributed network restoration. Failure to pursue the analogy is probably due in part to the fact that packet and call routing technology is encompassed within a view of the network as seen at its *voice circuit* level. The properties and problems of the network viewed at the DS-3 transport level are very different. To date the traditions and ideas of the call and data routing communities overlap little with the high speed transmission community who parented DCS technology. It seems plausible therefore that there has been little prior reason to study restoration as a problem for *distributed automated* solution. In addition, the *combination* of interests which leads to the present synthesis (distributed routing and DCS hardware design) seems to be found in no single design community today.

3.1 Graph Theory

The meaning of various technical terms in graph theory have not yet acquired universally accepted meanings and so we define the important terms we need for this work. A directed graph **G** = (N,L) consists of a finite set **N** of nodes and a finite set L of links. The more usual graph theoretical terms *vertices* and *arcs* are equivalent to our use of *nodes* and *links*. The notation $|N| = n$, $|L| = e$ denotes the number of elements in the sets N and L respectively. The common term *edge* in graph theory is analogous to the term *circuit* in this work. An *edge* is a pair of arcs between nodes u,v which have opposite directions. A *circuit* is an anti-symmetric pair of *links*.

A link, *a*, in L is described as an ordered pair (u,v) where u and v are nodes directly connected by the link. *a* = (u,v) is *directed* from u to v and *begins* at u and *ends* at v. A common graph theoretical equivalent is that *a* has its *tail* at u and its *head* at v. A *network* is a directed graph **G** plus a set of weights mapped one-to-one onto the set of links, or a function *d* defined on the set of links. For a link *a*, c(*a*) is the *length*, *distance*, *cost* or *weight* of the link *a* = (u,v). A link is said to be *positive* or *negative* based on the sign of its weight. We are presently interested only in networks of *positive* links and we (primarily) consider all links to have *unit weight*. i.e. we will be measuring routing lengths in terms of logical number of links rather than geographical or other real-valued measures.

The *cost matrix*, or link length matrix, **C**[u,v] is an (n x n) matrix whose (u,v)th entry stores c(u,v) if a link connecting u and v exists. By convention when no such link exists, the corresponding entry in **C**[u,v] is left blank, indicating infinity. The term *adjacency matrix* is used by some as the cost matrix but this term is more properly reserved for a matrix **A**[u,v] which simply indicates logical connectedness (1) or non-connectedness (0) when all links have unit weight. In our work these are equivalent because we measure distance only in terms of logical number of links. An *undirected network* is one that has a symmetric link length matrix. The transmission networks we are interested in are inherently symmetric and it is helpful at various times to consider them as undirected networks of circuits between nodes or as symmetric directed networks with *pairs of links* between nodes. Either point of view is valid but the methods of Selfhealing take significant advantage of the *symmetric directed network* view.

3.1.1 Simple Graphs, p-graphs and Multigraphs

A *reflexive* graph has a self-loop at every node; an *antireflexive* graph has no self-loops. In a *symmetric* graph, every (u,v) link has a return link (v,u). An *anti-symmetric* graph has no two nodes sharing such a pair of links. All of the transport networks which we are interested in are antireflexive and symmetric. We say a graph is *simple* if it is *anti-reflexive* and *symmetric* and has *at most one pair of links between any two nodes*, each pair having the same endpoints but opposite orientations. A graph is a *p-graph* if it allows nodes to be joined by *several distinct arcs with the same orientation*, where p is some maximum number of arcs having the same head and tail. The specialized form of graph needed for representation of the restoration problem is called a *multigraph*. We define a *multigraph* to be an *anti-reflexive, symmetric* network in which *any pair of nodes can be joined by any number of symmetric link pairs in parallel.*

3.1.2 Properties of Paths

A *path* is an ordered set of concatenated links that may represented by the ordered set of links present in the path: $[P] = [a_1,a_2,...a_{t-1}] = [(h_1,h_2)(h_2,h_3)...(h_{t-1},h_t)]$ (3.1)

or as a vector $\underline{P} = (h_1,h_2,...h_t)$ where h_1 is referred to as the source (or origin) node of the path and h_t is called the target (or destination) node. For a set of links to form a *path* between nodes x,y every link a_i in [P] satisfies *head* (a_i) = *tail* (a_{i+1}) for every i:= 1 to $(|[P]|-1)$.

The *length of path* P is obtained by the operator $C([P])$:

$$C([P]) = \sum_{i=1}^{|[P]|} c(h_i,h_{i+1}) \tag{3.2}$$

A path whose endpoints are distinct is said to be *open* whereas a path whose endpoints coincide is a *closed path* or a *cycle*. A *simple path* does not use any *link* more than once and an *elementary path* does not include any *node* more than once. Every elementary path is by implication simple. Some workers use the term *loopfree* to mean elementary. In restoration, each of the several paths which may be found must be individually open and elementary.

3.1.3 Properties of a Set of Paths

The restoration problem requires some particular notions of *path disjointness* amongst *sets of paths* which share the same start and end nodes. In a *node-disjoint set of paths*, [P], no node appears in more than one of the path descriptors, other than the start and end nodes. In a *link-disjoint set of paths*, no individual link appears in more than one path. We will later see that *link* *disjointedness* is a necessary condition amongst any set of paths which are established to restore a given span failure. Node disjoint paths are also link disjoint but node disjointedness is an unnecessarily strict condition to place on the paths of a restoration plan. The requirement that *individual* paths be elementary (ie. must never visit any node more than once) should not be confused with the simultaneous requirement that a *set of restoration paths* must be *mutually link-disjoint* but may share nodes in common amongst the *different* rerouting paths established.

3.2 Computational Models

Four computational models are found in the literature on routing and shortest path problems:

Centralized: This is the common single-processor / single-process computing environment seen by the user of a personal computer or a time-sharing mainframe applications programming environment. In problems formulated for this model, all data structures, such as network topologies, are assumed to be current, accurate, complete and resident in local memory.

Parallel: This is the environment of multiple large-grained or small-grained identical processors which are synchronized, physically co-located, and have a fast local communications network between processors. All necessary data for a problem is again assumed to be complete and available either in shared memory or in memory associated with particular processors but available to any other processor. Examples are the Connection Machine^{TM} and the Myrias^{TM} parallel machine which are multiple-instruction multiple-data (MIMD) architectures of varying granularity.

Systolic: This model is very similar to *parallel* in terms of data structure locality and synchronism but the processor model is a very fine-grained and deterministically interconnected for efficient execution of calculations which benefit from pipeline oriented methods.

Distributed : Each processor of a *parallel* machine is not a standalone *computer* in its own right. The latter implies the additional resources of a *multi-computer.* If the elements of a multi-computer are not co-located or synchronized then we refer to the system as a distributed multi-computer. In most such distributed computer systems however, direct communications between any two processors is possible, however they are physically interconnected, by message relay services at every node. We define a *distributed* *embedded* *multi-computer* environment as having the additional property that only processors which happen to be physically interconnected directly can communicate with each other. The distributed multicomputer environment is created by the deployment of modern equipment for switching telephone traffic or packet data. The deployment of DCS machines however creates the *distributed* *embedded* *multi-computer* environment. Each DCS node has a stand-alone computer system with large

RAM, multi-tasking operating system, disk storage, I/O, and subsystem for communications to the central control site. During DCS-based restoration however, we propose not to rely at all on communications to the central site, for both speed reasons and because the failure may have taken out the telemetry channel. The only communication links between processors that are assumed are those which the arbitrary topology of the communication network provides. This creates the environment of a an asynchronous distributed multi-computer in which processors are constrained to communicate only with adjacent nodes via the TE's of the actual transport network. Overall knowledge of network topology is not available at any node and propagation delays between nodes are high in comparison to those between elements of a parallel computer.

The distributed embedded computational model is the one in which Selfhealing will operate. It is similar to the model for distributed packet data and call routing applications but has the additional constraint of adjacent node communications only. The centralized model is the historical workhorse of algorithm development and is the model assumed in conventional proposals for centralized restoration and in most of the literature on paths algorithms. The parallel and systolic computational models are not directly relevant to restoration or routing but they are included because we are interested in algorithms developed for these machines that may have relevance to the implicitly parallel Selfhealing technique which we propose.

3.3 Formulation as a Routing Problem

A formal statement of the restoration problem will show the differences between restoration and the packet routing and call routing problems. Let us first consider the latter problems as typified by survey papers such as [ScSt80] on packet data routing, and [HuSe87] on call routing.

3.3.1 The Packet Data and Call Routing Problems

A particularly simple way to show the difference between the problems of packet data routing, call routing and restoration routing is not to attempt to present all various solutions to the former problems in detail, but rather to ask of each *class of problem*: *what is the form of the required solution?* That is to say, consider the information which must be specified in the solution to any problem of that category, however obtained. For example, many sophisticated techniques have been developed to determine routings which minimize various objective functions in packet data routing, but ultimately the solution to any such problem is, specifiable for any pair of nodes, in the form of a single *routing vector in a simple network graph*:

$$r_{ab} = [(u_1,u_2)(u_2,u_3)\ldots(u_{n-1},u_n)] \tag{3.3}$$

where a,b is any node pair in the network. We call r_{ab} the *routing vector* between nodes a,b. In packet routing r_{ab} specifies the sequence of nodes through which packets will be relayed. We have *a solution* to the packet routing problem for one node pair whenever we have a recommended series of links forming a path between those nodes. The usual embodiment of the such routing vectors is in routing tables at each node, giving instructions as to which link to send traffic out on depending on its final destination and its arrival link. Great art goes into algorithms to specify such routes so as to minimize delay or other metrics globally, or to do so in a distributed manner, perhaps updating it very frequently, and so on. But at all times the solution is completely specified for nodes a,b by the information in *one routing vector*. And, there is no direct constraint on the simultaneous use of any link by different routing vectors: all routings can re-use any one link as desired.

In dynamic adaptive call routing [HuSe87], the offered call attempt replaces the data packet. Although many aspects change in detail, the overall situation is not fundamentally different from packet routing in terms of the information required to specify a solution. In this case r_{ab} specifies the *sequence of trunk groups* over which a new call attempt should be successively offered to complete the circuit establishment process. Routings may be updated hourly, or even for each call, but the form of the solution is the same. Although call routing is part of the so called "circuit switching" process, the *routing* part of the call establishment problem is essentially similar to the packet routing problem: packets are routed through shared links just as call attempts are routed through trunk groups. A circuit establishment will follow a successful routing of the call attempt, by seizing one trunk arbitrarily in each trunk group transited by the call attempt, but the call attempt itself is routed, simultaneously with many other call attempts, through a *simple graph* comprising all the logical trunk groups in the network. Just as packets following many different routings can share one link, call attempts similarly share multiple access to trunk groups.

In call routing we try to minimize blocking by wise choice of routing vector whereas in packet routing we try to minimize delay but regardless of the variety and sophistication of the means employed to derive and update optimal r_{ab} for every relation in the network, the facts relevant to the present work are the following: (a) Packet routing and call routing are problems of route-finding in simple graphs, (b) The routing solution is specifiable by *one* routing vector r_{ab} for every node pair ab, (c) Any given "link" in the graph theoretic network may appear in any or all of the routing vectors between node pairs in the network, (d) r_{ab} is not necessarily symmetric with r_{ba} ; each direction is treated as a separate unidirectional routing problem (more so in packet data routing than call routing methods). (e) Solution of either of these routing problems takes place in a two-dimensional space, ie. the 2-D space of the connectivity matrix which completely specifies a simple graph network representation. Also in both problems, aside from *indirect* influences between paths effected by the algorithms that seek various global optimizations of network performance, no *hard* constraints apply on the composite set of routing vectors that are determined. By this we mean that there is no rigid cap to the number of routings that can use a certain link. If too many routings share one link in common and overload that link, the implication is a relatively soft degradation in terms of the engineered multiple-access parameter, delay or blocking.

3.3.2 The Restoration Problem

Let us now consider the problem of span restoration. The first difference is that the scalar link entities in packet and call routing (scalar because their only attribute is length) transform into dimensional objects called *spans*. Each span comprises a variable-sized set of link pairs. Restoration of a spancut between two nodes requires not *one shortest path* as above, but *many* simultaneous, collectively shortest, unique paths between the two nodes. The target number of paths is equal to the number of working circuits lost. Each restoration path must be a completely closed transmission path dedicated to one TE. When routing a carrier signal, as opposed to data packets or call attempts, there can be no multiple access to transmission links. Each TE completely fills one digital restoration path in space, time and frequency. Sharing links with other routings is a physical impossibility. This property is relatively rare in routing problems because such total physical exclusion is unusual in the majority of routing problems that arise in the physical world. Consider its equivalent for instance in a transportation routing problem: it means that once you have routed a truck (or a packet, a call) over streets x,y, z, *none of those streets can be used again* by any other truck for other routings. Most routed entities do not *fill* the medium of the routing path in physical length, breadth, time and frequency.[1] The restoration problem must accordingly deal with a multigraph representation of the network and, while a slightly sub-optimal routing solution in the prior problems means some excess delay or blocking, the penalty for any imperfection in restoration is complete outage of some number of carried services. An attempt to summarize all these comparative aspects of the call routing, packet routing, and restoration routing problems is given in Table 3.1.

The solution to a restoration problem requires specification of *a set of restoration vectors* for any physically connected node pair in the network, as opposed to one *routing* vector. This *set* of vectors (not an *array* because they may be of differing lengths) has to be found in a 3-D problem space subject to hard constraints of link disjointness, span capacities, mapping preservation and shortest path length. Specifically, restoration of a span between two nodes A and B, with W_{AB} working circuits, requires correct specification of the following information, subject to the additional constraints which follow:

$$k = \min(W_{AB}, T_{AB}) \tag{3.4}$$

$$R_{AB} = [\, r_{AB1} .. r_{ABK} \,] \tag{3.5}$$

where W_{AB} = number of working circuits on span A-B,

T_{AB} = number of link disjoint paths topologically possible in the given network,

R_{AB} = a set of k restoration vectors, of possibly differing lengths, comprised as follows :

$$r_{AB1} = [<(u_{1,1},u_{1,2}),x_{1,1}>,<(u_{1,2},u_{1,3}),x_{1,2}>\ldots<(u_{1,n(1)-1},u_{1,n(1)}),x_{n,1})>]$$

$$\vdots \qquad (3.6)$$

$$r_{ABK} = [<(u_{K,1},u_{K,2}),x_{K,1}>,<(u_{K,2},u_{K,3}),x_{K,2}>\ldots<(u_{K,n(K)-1},u_{K,n(K)}),x_{n(K)})>]$$

where $(u_{i,q},u_{i,q+1})$ = the span of the network to be used as the q^{th} span in the i^{th} restoration vector.

$x_{i,q}$ = an individual link in the set of parallel links on the span (q, (q+1)) allocated to the ith restoration path.

$n(i)$ = the logical length of the i^{th} restoration vector

The combined notation $<(u_{i,q},u_{i,q+1}),x_{i,q}>$ is a specification for an individual link of the network in terms of span identity, $(u_{i,q},u_{i,q+1})$, and the specific link number, $x_{i,q}$, within the span. R_{AB} , the full set of k restoration vectors, is called the *restoration plan* for nodes AB. The k *restoration vectors* of a restoration plan have to be found in the 3-D space of connectivity-capacity, as opposed to the 2-D connectivity space which describes a simple graph. The extra dimension is the difference between a *routing vector* and a single *restoration vector.* While a *routing vector* specifies a sequence of links in a simple graph, a *restoration vector* specifies a sequence of spans and individual links within each span in a multigraph. A *restoration plan* specifies a set of k mutually link-disjoint restoration vectors, ranked by increasing length. In each restoration vector *twice as many variables* must be specified as in a routing vector of the same logical length. In addition, the following constraints must be simultaneously satisfied by the set of restoration vectors collectively comprising any restoration plan R_{AB}:

3.3.3 Constraint 1: Link Disjointness

Every link used in one path of the restoration plan, must appear nowhere else in any other restoration paths of the plan:

if $<(u_q,u_p),x_i> \in R_{XY,j}$ *then* $<(u_q,u_p),x_i> \notin R_{XY,z} \ \forall \ z \neq j$ (3.7)

where $R_{XY,j}$ denotes the jth path of the restoration plan for nodes X,Y. The usual communications routing problems have no such constraint because packets and calls do not fill their corresponding link entities completely in space, time and frequency. But in restoration, links must absolutely not be shared amongst different paths. The penalty for failure to find enough link-disjoint restoration paths is total outage of all the services that were borne by the lost TE.

Table 3.1 <u>CONCEPTUAL SCHEMA OF COMMUNICATIONS ROUTING PROBLEMS</u>

	Routing in Packet Networks	**Alternative Traffic (Call) Routing**	**Span Restoration**
"Link" Resource	logical "pipe" between nodes	logical trunk group	digital carrier signal eg) DS-3 or STS-1
# "Links" between nodes (type of network graph)	1 (simple)	1 (simple)	$k \geq 1$ (multigraph)
mode of access to links	queued (delay engineered) (time shared)	full availability (traffic engineered) (space shared)	non-shared (dedicated in time and space)
sharing of links among paths?	yes	yes	no (fully link disjoint paths)
grade of service metric	path delay $P(d \geq x)$, d	call blocking P(B)	network availability $A = MTTR/(MTTR+MTBF)$
application	data communication	voice telephony	physical bearer network for all voice and data services
impact of failure	increased delay	increased blocking of originating calls	total loss of all calls in progress plus service outage

3.3.4 Constraint 2: Span Capacity Constraints

Any restoration routing plan must satisfy $|[S]|$ simultaneous constraints on the number of paths routed over any one span: If $\Omega\,[S]$ is an operator that produces a set of the *spare capacities* corresponding to each unique span of $[S]$, then any set of restoration vectors comprising a valid restoration plan $[R_{xy}]$ must also satisfy the following $|[S]|$ inequalities:

for every $m = 1 \,..\, |[S]|$

$$\sum_{i=1}^{k} \sum_{j=1}^{n(i)} \left[(S_m \in [S]) \in (r_j \in [R_{xy}]) \right] \leq |[\Omega_m\,[S]]| \qquad (3.8)$$

where $\Omega_m\,[S]$ denotes the mth element of the set produced by the operator $\Omega[\text{-}]$. Each inequality simply expresses that the sum of all restoration paths in a restoration plan which transit any given span must be less than or equal to the number of (spare) links on that span.

3.3.5 Constraint 3: End-to-End Mapping Preservation

This constraint corresponds physically to the requirement that it is not enough just to find k paths between X,Y. Both end nodes must share *a common scheme of ordering the paths* to avoid transposition errors in the substituted traffic streams. This is another aspect totally foreign to the usual routing problems where only one route is considered between nodes. This constraint can be formulated as follows: let $M(R_{xy}) \dashrightarrow [O]$ represent an arbitrary function which provides a one-to-one mapping of each restoration vector of a restoration plan onto one element of an arbitrary but unique set of integers $[O]$. Then for any set of restoration vectors on the relation XY, a necessary part of a solution is the independent determination of two mapping functions $Mx(R_{xy})$ and $My(R_{xy})$ computable at the nodes X and Y respectively. The constraint that must be satisfied is:

$$Mx(R_{xy}) = My(R_{xy}). \qquad (3.9)$$

The three constraints so far introduced are essential for the basic function of restoration: Constraint 1 says the paths created must be fully link-disjoint, Constraint 2 says they must respect the finite span capacities present and, Constraint 3 says they must share a common end-to-end identifying scheme for correct traffic substitution. While it is essential to satisfy Constraints 1 to 3 inclusive, it is also *an objective* that the shortest possible set of paths is found.

3.3.6 Constraint 4: Shortest Paths

This objective is an aspect that restoration and the packet and call routing problems do share. If the operator $|[P]|$ obtains the number of spans in a path $[P]$, then for any restoration plan R_{XY} comprising k restoration vectors the objective is:

$$\sum_{j=1}^{k} |r_{XY_j}| = \text{MINIMUM} \qquad (3.10)$$

In summary, we can state the problem of restoration between two nodes in graph theoretical language as: *find k-shortest link-disjoint paths in a multigraph subject to hard span capacity constraints* with mapping preservation. By comparison the equivalent problem statement for the classical packet switching and call routing problems are: *find a shortest path in a simple graph subject to overall optimization* of delay (or blocking). Compared to the latter problems restoration involves: (a) 2*K as many variables to specify, (b) one higher dimension problem space, (c) strict link disjointness constraints, (d) strict span utilization constraints, (e) strict mapping preservation constraint. In addition two-way symmetric routing solutions are required in restoration, not independent one-way routings.

3.4 Literature on Graph and Routing Problems

Now that we have a formal definition of the problem which we seek to solve in a distributed manner we are in a position to survey the extensive literature on routing algorithms and path problems in networks. Three survey papers [DePa84], [SaSt80], [QuDe84], amongst those retrieved[2] tend to represent three clearly delineated groups of investigators in this area: (a) the community interested in shortest path graph algorithms for centralized computation, exemplified by [DePa84], (b) the distributed computer communications and call routing community represented by [SaSt80] and, (c) the parallel computation community with an emphasis on graph problems, as surveyed in [QuDe84].

We find that the dominant interest in common amongst these groups is in single shortest path problems without path disjointness for many applications in operations research, transportation, packet data and telephone traffic routing. Group (a) has a particular emphasis on efficient algorithms for exact solutions of shortest path problems assuming centralized computation. Group (b) emphasizes distributed heuristic solutions to the usual shortest path routing problem. Group (c), the smallest and most recent, focuses on adapting existing graph algorithms for execution on parallel processors and developing new intrinsically parallel algorithms for graph problems. The subspecialty of the shortest paths literature that is most *theoretically relevant* to restoration is broadly called the *k-shortest paths* problem, but it appears to be exclusively formulated for *centralized* computation in the existing literature. The work with the most similar *motivation* to Selfhealing tends to be the literature on distributed routing algorithms, but all of these are devoted to simple-graph, non-disjoint single shortest path

problems and assume a distributed rather than *distributed embedded* computational environment.

A classification system was developed to categorize and summarize the papers on graph and routing algorithms that have been examined in this study. Selected papers are discussed here and all are categorized in Appendix A according to the following system of attributes. Those classification terms not already discussed will be introduced in the discussion which follows:

problem type	**= {single, all pairs, k-shortest, other}**
network type	**= {simple graph, multigraph }**
path type	**= {elementary, node disjoint, link disjoint, distinct, other}**
computational model	**= {centralized, parallel, systolic, distributed}**

The major finding of the literature review was the apparent absence of any *distributed* algorithm for k-shortest paths problems and specifically none involving k *link disjoint* paths in either distributed or centralized contexts. In the classification system we used, a distributed solution to the restoration problem would be represented by the attributes (**k, m, ld, d**). For comparison, the call routing and packet data routing problems and their contemporary solutions generally have the attributes (**s** (or **a**), **s**, **e**, **c** (or **d**)). The classification term, *single*, refers to the *single-source problem* which is the most frequent class of shortest-path problem. The single source problem is that of finding the shortest path between one specified source node and all other nodes. The most well-known algorithms for this problem are from Dijkstra [Dijk59], Bellman [Bell58], and Moore [Moor59] using the temporary label setting methods first proposed by Dijkstra [Dijk59]. Extensive treatments of the one-to-all and one-pair algorithms are found in [DeFo79], [DGKK79], [John77], [Vlie78], and [YenJ75]. Many efficient algorithms (Dijkstra for example) intrinsically solve the one to all problem, not the single-pair problem. That is to say even if you ask for one specific path, such algorithms internally derive all information needed for the one-to-all problem as the most direct means to in fact find any one-pair shortest route. Many single source algorithms also find just the *distance* of the shortest path and recovery of actual paths require reconstruction by retracing internal states in the algorithm. A problem closely related to the single source problem is the *all-pairs* as denoted in the above classification. In *all pairs* problems the algorithm intrinsically computes the single shortest path from every node to every other node.

One recent paper in which a centralized algorithm *for routing with DCS machines* is the direct topic of concern is Hasegawa et al. [HaKa87]. They focus on the problem of an algorithm for computation of crossconnection maps for the *initial configuration* of path routings between every source-destination pair using a shortest path criteria. The algorithm they report for these applications deals with the network in a simple graph form; every *span* in the real network is a *link* in their simple graph. They find single shortest paths between all pairs to establish the initial network routing plan and can also find single-shortest paths for addition of new paths to the network. However, no link-disjointness constraint is included as the level of routing specifies span sequences only. Consequently, they explain that after deriving an all-pairs shortest routing plan a secondary check is required to see if the total routings on any span has *exceeded the actual number* of carrier circuits on that span. The final mapping of paths into unique circuit assignments in the *depth* of each real span is also not included in the algorithm.

The work in [HaKa87] amounts to adaptation of a simple-graph shortest paths algorithm to the address the all-pairs initial configuration routing problem for optimal loading of spans of the network. This does not however address the needs of restoration in which k link-disjoint sets of composite span and link specifications are required, not just the sequence of spans along the one shortest route, and it must be assured that span capacity constraints are incorporated directly in the solution method. A point arising implicitly from the *intentions* of [HaKa87] is however that if one has a solution to the k-shortest link-disjoint paths problem in a multigraph then one also has a tool for determining the optimal initial loading of the entire network, without the drawbacks of the above algorithm. In fact we think that Selfhealing can be used for applications of this type as well as for restoration. In Ch.13. we briefly report some additional results obtained in which Selfhealing is adapted to provide an advanced in-service provisioning tool we call *Capacity Scavenging*. The Scavenging application of Selfhealing is in fact a distributed solution to the problems pursued by [HaKa87], without the limitations they encountered.

3.4.1 The k-shortest Paths Literature

In classical shortest path problems on graphs it is sometimes desirable to have knowledge of the *nearly optimal* paths as well as the *one* shortest path. This is the origin of the *k-shortest paths problem* in graph theory. It refers to finding first *the* shortest, then the *second shortest* path, then the *third shortest* path, and so on, to the kth shortest path. k-shortest paths knowledge is typically used to study the sensitivity of a pure shortest path solution, or to have k nearly equal routes for time dispersive transportation dispatching or to find fixed alternative routes for data routing or call routing applications. In such cases the k paths are *ranked by their total path length* without requirements for link disjointness amongst the alternative routings. In operations research, a listing of all paths in order from the shortest onward is a general technique to allow inspection of solutions to determine the first path that satisfies possibly complex constraints for which a custom algorithm is unknown or unwarranted.

The general relevance of *k-shortest paths* to restoration is obvious but there are important variations of the problem amongst the few authors who are concerned with such problems. Most consider simple graphs only and do not pursue any form of link disjointness in the set of k-paths.

The most frequent algorithms seek k paths which simply have *different* total lengths and the same end nodes. Such paths are referred to as *distinct* (term used in the classification scheme above). Distinct paths may share links in common while two paths which might be completely link-disjoint may not be recognized as distinct because their lengths are equal, especially when path length is measured by logical link count. The most common k-shortest path problem formulations also allows paths with loops.

[Brad75], [Drey69], [Elma70], [Lawl76], [Mini78], [Yen75], [FoxB73], [LeeC62], [Shie74,76,79] and [Wein73] all treat problems of the type where k length-distinct paths are found in a simple graph, with looping allowed. A smaller set of authors consider finding the k-shortest *loopless* paths [ClKR63], [Lawl72,76], [Mars73], [Poll61], [Saka68], [Wong76], [Yen71], [Topk88], but in a simple graph context and with distinct, not link disjoint, paths. Still smaller is the group that consider any form of disjointness between paths [RoPe84], [Suur74], [Topk88], [Lawl76], [MoMe81] but they focus predominantly on *node* disjointness. Only two writers [Suur74] and [SuTa84] even mention *link*-disjointedness or multigraphs, in passing. A few of the k-shortest paths papers warrant discussion however, although they all assume a centralized computational model.

Lawler [Lawl76] gives a method to compute the k shortest paths from an origin node to each other node in $O(K\ n\ \log n)$ time after a preliminary $O(n^2)$ stage to determine shortest paths using Dijkstra's method. In this method two paths are considered distinct if they do not visit precisely the same *nodes, in the same order*. Lawler also considers the problem of finding k paths, ranked by length, where the paths are not permitted to contain *repeated nodes*. This requires $O(Kn)$ applications of a basic shortest path algorithm for $O(Kn^3)$ time complexity. The algorithm is formulated only for simple graph problems and enforces *node* disjointedness. Enforcing node disjointedness has the effect of ensuring *looplessness* (because a path cannot revisit any node twice). Node disjointedness is however quite an inefficient means through which to ensure *link* disjointness. Lawler's algorithms seem to be motivated by path problems in management planning such as when using critical path chart methods.

In 1976 Shier published a description of three methods for centralized computation of the single source k-shortest paths problem and reports computational experiments to determine one that is most effective [Shie79]. Shier's meaning of k-shortest paths is again distinction by virtue of total path length and he allows paths that may loop through a node more than once. The algorithms developed actually find the *lengths* of the k shortest paths and subsequent processing is needed if the paths themselves are to be recovered. Shier's methods are also based on the assumption that k is fixed and known. If k is unknown (as it is in restoration when topology limits restoration path number to less than the number of working circuits lost), then any of Shier's three methods require repetition, successively assuming a higher k, until the method fails. It is then known that (k_{last} -1) is the solution to use. The use of iterative linear equation solving methods as part of the recommended algorithm also requires an initial approximation and convergence time can depend on the goodness of first approximation.

Shier's research method is of relevance. He uses experimental testing to estimate algorithmic time complexity by defining an arbitrary 400 node network in which 1520 randomly connected arcs are placed. Arc weights are generated uniform randomly on [1..100]. He chooses k = 5 and selects six arbitrary source nodes for study. He finds all the methods studied to be much faster in practise than their theoretical bounds. Perko [Perk86] uses essentially similar experimental methods to assess algorithmic time complexity. These aspects of research method are similar in principle to some of the methodology to follow in this work.

In [Shie79] another comprehensive computational study of five (centralized) algorithms for k-shortest paths is reported. Each of these algorithms produces k-shortest paths which are distinct in terms of total length, may contain embedded routing loops, and require subsequent trace processing to recover the paths themselves. All of the methods inherently calculate the k-shortest paths from one source to all other nodes. For the restoration problem the possibility of paths with loops is obviously unacceptable. Yen's algorithm [Yen71] finds k-shortest paths without loops and Perko has studied implementation alternatives for this algorithm [Perk86]. Yen's approach is based on first finding a number of paths that may include loops and then filtering these to find a required number of loopless paths ranked by total length. The worst case time complexity is $O(kn^3)$ but storage for a large number of intermediate path candidates can be required and the number is difficult to estimate. Although loopless paths result there is no form of disjointness constraint introduced, either node or arc.

Weintraub [Wein73] and Wongseelashote [Wong76] are concerned with k shortest loopless paths but again consider paths distinct by virtue of total length only. Wongseelashote applies the techniques of *schedule algebra* to represent the operations frequently required in path algorithms and he presents a k-shortest paths algorithm framed in the terms of his proposed algebra. Weintraub's interest is in viewing the same k-shortest paths problem as an optimal assignment problem to which linear programming methods can be applied. He concludes that the k-shortest loopless paths can be obtained in at most $O(kn^3)$ operations. Weintraub also uses experimental characterization with arbitrary study networks to verify and characterize his algorithm.

Topkis [Topk88] recently published an algorithm for centralized computation of k shortest loopless paths in a simple graph in which *at least the <u>first</u> links of every path* are disjoint. The motivation for this is in distributed sequential call routing or delta-routing packet data applications. Topkis's problem arises as follows: Each node originating a call has a routing table that specifies the set of outgoing trunk groups (links) which should be tested in order for their ability to accept the offered call. The goal is to have routing tables at every node which offer k diverse choices for initially launching the routing of a call from any node. To achieve this *the first links in each of the k routes* determined by Topkis's method are assured to be node disjoint. Thereafter, the set of k different routings may merge on their way to the destination. Topkis's algorithm is intended to run at a network control site where the network topology is known and local delay or blocking estimates are received from every site. The algorithm runs in $O(kn^2)$ time to compute the k-paths from one node to all others.[3]

Other investigators who deal with some form of disjoint paths constraint include Suurballe [Suur74], Ronen and Peri [RoPe84], Suurballe and Tarjan [SuTa84] and Moss and Merlin [MoMe81]. [Suur74] considers paths which are *node-disjoint* (except at the terminals) and gives an algorithm for finding k *node-disjoint* paths with minimum total length in k iterations of a simple-network single shortest path algorithm. He mentions the difference between node and link disjointness but does not consider multigraph applications.

In [SuTa84] the same author considers the problem of finding *one pair of node disjoint paths* between two nodes. The principle is repeated application of a single shortest-paths algorithm but the advance is in an efficient way of combining the Dijkstra calculations for one source and all destinations to obtain an $O(m\log_{(1+m/n)}n)$ algorithm for finding a shortest pair of edge disjoint paths from one node to all others where m is the number of arcs in the network.

In [MoMe81] Moss and Merlin present a theoretical treatment deriving conditions for the *existence of exactly k multiple paths* between every pair of nodes in a given simple graph. Their formal requirements for such paths are that they are loopfree and *different* from one another in at least one node or link. The paths are not necessarily shortest paths or disjoint.

[RoPe84] considers the problem of finding *the maximum number of node disjoint paths* between two nodes such that *the length of every path is bounded by some constant r*. Paths are not required to be of a k-shortest family but are bounded in their maximum length instead. This problem is also called the maximum length-bounded disjoint paths problem in [GaJo79] where it is cited in a compendium of problems known to be NP-complete (it is strictly NP-complete for $r > 4$ and polynomial time solvable for $r <= 4$). This problem is closely related to the maximum number of node disjoint paths (in a simple graph) with minimum total length (with no constraint on individual path length) for which there is a polynomial time algorithm by Suurballe [Suur74]. In [RoPe84] a heuristic polynomial time algorithm is given for approximate solution of the former problem for general r. The time complexity is $O(k^2\ |[N]|\ |[L]|)$. Experimental methods are used in which random graphs are generated and sample node pairs are used to test performance of the heuristics.

By virtue of the problem it sets, [RoPe84] raises an issue relevant to this work regarding the target criteria for the path length properties of an ideal restoration scheme. The point is that *k*-

***shortest* link-disjoint paths** is not strictly the same as *k link-disjoint paths with minimum total length*. The apparently small change in problem statement makes a major difference in practise: the latter problem is NP-complete. This distinction will be taken up in Ch.4 where we treat the problem of defining and obtaining ideal reference solutions for use in the study of distributed restoration schemes.

3.4.2 Distributed Routing Algorithms

There are a number of routing algorithms for packet data networks and alternative traffic routing networks that effect distributed routing but still use essentially centralized *computational methods* to derive the routing plan. These schemes *either* (a) perform routing calculations in a central site using link information obtained by telemetry from nodes of the network and download results to each node [PeLe83], [ChFr72] [SoAg81], [SaSt80], [Topk88] or, (b) they include a scheme of mutual information exchange through which all nodes of the network *first build a global view* of the network and then locally execute an instance of a centralized algorithm, extracting the part of the global routing plan that applies to their site [SaSt80], [Rose80]. We dismiss these approaches because they do not achieve our ambition of a truly *distributed computational* result, although the routing policy that they derive is indeed *put into effect* in a distributed manner.

There is therefore a distinction to be made between distributed routing and distributed computation of the routing plan. The real time demands of the restoration problem motivate us to seek a *truly distributed routing scheme* in which *both* computation and implementation of the routing plan is an inherently distributed process without global knowledge at any node. Our specific notion of a *truly* distributed scheme has these properties: (a) no one node has or ever obtains a global or even partial network description; (b) every node acts autonomously and has no dependency on external telemetry to a supervisory or coordination centre; (c) every node computes only the part of the composite network routing strategy which it is required to put into effect; (d) each node uses only data that is local at the node and nodes do not maintain any database; (e) Routing decisions computed in isolation at every node coordinate with the routing actions also taken autonomously by all other nodes, collectively synthesizing a complete and coherent routing plan at the network level; (f) there are no network-wide messaging relay services available to nodes of the network.

There are few papers on routing systems of such a truly distributed nature. [MoBh86] however recently described an algorithm which has some of these attributes in an asynchronous distributed solution to the single source shortest paths problems. The algorithm is analyzed as having $O(n^2)$ messaging requirements but message queuing and processor times are not taken into account. The algorithm finds non-disjoint paths and is formulated for a simple network graph assuming normal interprocessor messaging facilities. Every node initially requires knowledge only of the identity of its adjacent nodes. Each node then goes through a number of iterations where it learns of successively farther away nodes and sends messages directly to them to learn of routings to still farther nodes and so on until the diameter of the network is traversed. The source node builds a shortest path tree (SPT) as it learns, through returning messages which carry their route with them, of the shortest route seen to each other node. For the problem of single shortest paths in a simple graph it is a potentially fast distributed algorithm but would not function in our distributed embedded computational environment. Selfhealing differs from [MoBh86] in terms of the problem solved and by virtue of not relying on explicit inter-processor messaging across the network, interacting instead by the method of signature exchanges only with its direct neighbours.

[Chau87] presents a collection of general utility algorithms for simple graph operations in a distributed computation model using principles similar to [MoBh86]. These are (from any one node): breadth and depth first searches of the graph, recognition of acyclic conditions and strong connectedness, single source shortest paths, and detecting slack in project activity planning graphs. Other distributed routing algorithms for the single shortest path in a simple graph use time-averaging convergence methods. These methods are unnecessarily slow and inappropriate from our viewpoint because of the transient speed and one-time *first-time* precision needed for restoration. Such convergence methods are intended for continuous gradual adaptation of routing in networks with slowly varying nearly stationary statistical load patterns in systems that are relatively forgiving of slight sub-optimalities in the overall control trajectory. Algorithms of this type are described in [AbRh82], [Gar88a], [Gar88b], [MaNa87], and [Anis85]. [Anis85] treats 'nondeterministic' local algorithms for this problem. [MaNa87] reports an experimental implementation of learning automata with various reinforcement and penalty functions to adapt the nodal routing tables within a network by means of generating probabilistic perturbations in routing decisions and arranging for feedback on packet delay (measured at the destination) to close the learning feedback loop.

3.4.5 The Literature on Parallel Graph Algorithms

There is just beginning to be some activity in graph algorithms specifically for parallel computation [DePa80], [QuDe84], [QuYo84], [PaRa85] but the motivation of these investigators is to harness the power of parallel processing architectures with an ongoing presumption of centralized non-realtime applications and global network knowledge for the algorithms. [MaDe82] presents parallelized versions of Dijkstra and Moore's algorithms for the single-source single shortest path problem on MIMD and array processing architectures. [QuDe84] surveys parallel graph algorithms for maximal clique, maximum cardinality matching, minimum spanning tree, shortest single path and travelling salesman problems on various parallel architectures including SIMD, MIMD, array, associative and systolic types. [QuDe84] states that the only 'parallel' shortest path algorithms found in his 1984 survey were for the single source and all-pairs single shortest path problems. The single shortest path problem in a simple graph is also studied for MIMD implementation using a version of Moore's algorithm and the Warshall-Floyd algorithm in [DePa80]. Other parallel algorithm implementation studies on the same problem are reported in [QuYo84], and [PaRa85] and in [ScTh86], the latter being for a systolic architecture.

3.5 Summary and Conclusions

We have formulated span restoration as multiple-path routing problem subject to constraints of link disjointness, finite span capacities and mapping preservation with the objective of shortest-paths. As such, restoration appears to present a unique problem in routing. It is significantly different from the well-known problems of packet and call routing by primarily virtue of its *k-shortest link-disjoint paths in a multigraph* formulation, rather than a single shortest path in a simple graph. Table 3.1 is a summary of the important distinctions between these well-known routing problems and the restoration problem. We have also surveyed the literature of both graph algorithmic path problems and distributed communications routing problems and found that the particular area of k-shortest paths seems not to have been addressed at all for distributed computation. In addition it appears that the more specific problem of k-shortest *link-disjoint* paths in a multigraph has similarly not been directly addressed even for solution by centralized computational means.

CHAPTER 4

PERFORMANCE METRICS AND THEORETICAL OBJECTIVES

This chapter is devoted to developing appropriate figures of merit through which the performance and efficiency of Selfhealing, or any other distributed automated restoration scheme that may be proposed, can be assessed systematically and quantitatively. One performance measure of obvious interest is speed of restoration. But as a routing mechanism we are also concerned with topological performance measures - in particular: does the method find all paths that are topologically possible?, are the lengths of the paths found as short as theoretically possible? The latter questions require a criterion for what constitutes an ideal restoration plan and, in practice, some reference algorithm to provide such theoretically ideal solutions. The chapter proceeds as follows: first a necessary distinction is made between two classes of performance measures, those of interest in an operational context and those of relevance in a research context. Next, the proposed performance measures are introduced in order of perceived importance in practice. These are : path number efficiency, speed of operation, path length efficiency and shortest path probability. Three of these measures are defined with respect to properties of the ideal restoration path sets and so we also define the properties of the ideal solutions and introduce a new centralized algorithm to find such reference solutions.

4.1 Operational and Intrinsic Path Metrics

In any real transport network, each span has a finite number of working and spare circuits. When a span is cut in such a network, the prime concern is to restore all of the lost working capacity on the given span. Obviously the ability to do this depends on the amount and placement of redundancy in the network as well as on the method used to find the restoration paths. Full restoration of working circuits may or may not require all of the paths that are *topologically possible* between the end nodes of the failed span. In an operational context therefore, the relevant measures of performance are different from the measures of interest from a research viewpoint. In the latter, one wishes to characterize the intrinsic ability of a method to find efficiently *all the paths that are topologically possible*, as if there were an unlimited number of working circuits to be restored.

The results given in the first published report of Selfhealing [Grov87b] were of the operational restorability type. In that paper, the US and metro study networks were arbitrarily provisioned with *finite* working circuit quantities on each span and when a given span was cut, the Selfhealing mechanism sought a number of reroutings that equals the lost capacity, *whether this fully exhausted the route-finding ability of the mechanism or not.* In most cases it did not. Results in the form of operational measures were relevant in that paper which aimed to show the high fault coverage and speed obtainable with DCS-based Selfhealing in networks with quite minimal redundancies. However such results depend on the arbitrarily assumed number of working circuits on each span, which determines the target number of paths sought by the restoration system and are therefore not the most general form of results.

Our present objectives are different: we now want to assess the intrinisic efficiency of a proposed restoration method in the most general manner possible. This means *finding all possible restoration paths* for each span cut, subject only to purely topological limits in a given network. Results of this type, when compared to the ideal reference solutions, are intrinsic measures of a given method's ability to make maximum use of the resources given to it, ie. the sub-graph of spare links in a network. In three of the four experimental network models which follow there are accordingly no working circuit quantities given. In these networks the target number of restoration routes for any given spancut is considered infinite. We refer to this as conditions of *maximum path finding stress*. A fourth network to be used is treated in operational terms (finite path stress) because in this one case, actual working circuit quantities were available and this particular network model (Bellcore) could come to serve as a standard comparative trial network.

Operational metrics and the theoretical metrics now proposed both have their place in practice. If there were soon to be several competing Selfhealing-like restoration proposals, a network planner would do best first to compare the candidates in terms of their theoretical efficiency in a benchmark network, under maximum path finding stress. The benchmark network should be of representative connectivity and relative number of spares per span but may be a conveniently smaller than the real network of ultimate interest. Having selected the best performer from the all-out path finding tests in the benchmark network, the planner would then switch to the use of operational measures. In this latter phase, optimization of network restorability is the goal and this involves detailed manipulation of the exact working and spare circuit quantities on each span, in conjunction with the selected restoration method, to reach some target restoration coverage in the real network of interest.

4.2 Path Finding Performance Metrics

We define four figures of merit which reflect different path-related performance measures. These can be applied in both operational (finite path finding) and intrinsic (maximum path finding) variants. To define these measures we begin with the following:

Consider a multigraph representation of a real network **G** = **(N, S, W, R)** where **S** is an ordered set of spans connecting nodes in **N**, and the similarly ordered sets **W** and **R** respectively give the working and spare (redundant) circuit quantities on each span of **G**. Let some span between nodes *(u, v)* in **S** suffer a spancut that severs all working and spare circuits, and let there be some postulated method which finds a set of restoration paths between *(u,v,)* using only spare circuits on non-severed spans. Then define:

$[[Pa]]_{uv}$ = the *set of actual restoration paths* $[Pa]_i$; i := 1 ..k found for spancut (u,v).

$[[Pt]]_{uv}$ = the *set of ideal restoration paths* $[Pt]_i$; i := 1 ..T(u,v) for spancut (u,v).

W(u,v) = the number of working circuits on span (u,v) before the cut. $W(u,v) = \infty$ implies maximum path finding stress.

$[[Pa]]_{uv}$ and $[[Pt]]_{uv}$ are internally ordered by ascending path length. $[[Pt]]_{uv}$ always includes all the paths that are topologically possible. Determination of the reference path set $[[Pt]]_{uv}$ for use in these relations is a problem in itself to which we shall return. For now let us assume $[[Pt]]_{uv}$ can be obtained for research purposes by a non-real time centralized algorithm.

4.2.1 Path Number Efficiency (PNE)

PNE is a measure of the ability of the routing method under test to find the highest possible number of link-disjoint paths in a network. PNE = 1.0 implies ideal topologically-limited performance in conditions of maximum path finding stress, or implies 100% restorability in an operational context. PNE < 1.0 implies less than ideal path finding success or less than 100% restorability, respectively.

$$PNE_{uv} = |[[P_a]]_{uv}| \, / \, |[[P_t]]_{uv}| \quad ; W(u,v) = \infty \qquad (4.1a)$$

$$|[[P_a]]_{uv}| \, / \, W(u,v) \quad ; W(u,v) < \infty \qquad (4.1b)$$

(4.1a) represents PNE for the intrinsic case, (4.1b) is the corresponding PNE for operational contexts. PNE is defined in the sense shown so that failure to create the maximum possible (or required) number or paths appears as a PNE value *less than* 1.0. PNE > 1.0 is impossible by the definition of $[[P_t]]_{uv}$ for the intrinsic measure. PNE > 1.0 can conceivably result in the operational context but would represent an error of the restoration system, which should always hold at 100% restoration. Note that a theoretical PNE of 1.0 can occur at the same time that less than 100% operational restoration occurred: this would mean that all topologically possible paths were found but the number of paths was less than the required W(u,v). (4.1) defines PNE for one span (u,v). A corresponding network average PNE of either intrinsic or operational contexts follows as:

$$PNE = E_{(i=1..|[S]|)}\{PNE_i\} \qquad (4.2)$$

where $E_{(i=|[S]|)}\{.\}$ denotes the expectation over all individual spans of the network. The operational or intrinsic meaning of (4.2) depends of course on the version of (4.1) that is used within it. PNE is by far the most important measure of restoration performance because this determines the extent of working capacity protection. Next in priority is speed.

4.2.2 Speed-Related Performance Measures

When considering an advanced restoration scheme, four speed-related figures are of interest. The first three of these can be applied either to an individual spancut or to a network as a whole by averaging the corresponding measure for every spancut in the network:

1. **First path time:** (t_{p1}) This is the time of completing the first path of a restoration plan. This is of interest primarily to assess the impact on high priority circuits. One may arrange to grant such circuits use of the first path(s) found in restoration so as to have the lowest risk of call-dropping. The first path time indicates the minimum interruption that a priority circuit would see and gives an immediate indication if any portion of the affected traffic was able to beat the CDT with for any restoration event in general.
2. **Restoration time:** (t_R) This is the time of completing *the last path that can be completed* for a given spancut. This is sometimes also called the time of *complete* restoration. This use of "complete" should not be taken to imply 100% restoration however. Rather it is the time at which the transient restoration event is complete, however the outcome.
3. **Mean path time:** (t_{pav}) This is the average of the times at which individual paths in the restoration plan are completed. This is *not* the same as the restoration time divided by the number of paths found, because of the parallelism involved in the process of Selfhealing.
4. **95% confidence time, t_{95} :** This measure is defined for a whole network based on the restoration times of all individual spancuts in the network. This is the time by which 95% of all observed spancuts are completely restored. It is the value at which the CDF of restoration time equals 0.95 :

$$\int_0^{t_{95}} p(t)\, dt = 0.95 \tag{4.3}$$

where $p(t)$ is the probability density function of restoration time. In practice we have only a limited number of spancuts in any given network and have to approximate the above integral with the corresponding discrete summation of a histogram of observed statistical frequencies.

4.2.3 Path Length Efficiency (PLE)

Next in order of priority we are interested in the efficiency of a restoration scheme as a k-shortest paths routing method. In practice we want to ensure that additional signal path delay is minimized in rerouting and we want to ensure reasonably efficient use of network spares. PLE measures the total distance of all links used in an actual restoration plan, with respect to the total number required in the corresponding ideal set of restoration paths.

The path length efficiency of a given spancut in an intrinsic context is:

$$\mathrm{PLE}_{uv} = TPL\ [[\mathrm{P}_t]]_{uv} \,/\, TPL\ [[\mathrm{P}_a]]_{uv} \tag{4.4a}$$

and in an operational context the PLE of a single spancut is defined as:

$$PLE_{uv} = TPL\ [[P_t]]_{uv}{}^{k}\ /\ TPL\ [[P_a]]_{uv} \qquad ; k = W(u,v) \tag{4.4b}$$

where the *TPL* [[P]] is the *total path length* of all restoration paths in a given restoration plan (set of restoration paths) ie :

$$TPL\ [[P]] = \sum_{i}^{|[[P]]|} \mathcal{C}([P]_i \in [[P]]) \tag{4.5}$$

$[P]_i$ is a path in the set of paths [[P]] and $\mathcal{C}([P]_i)$ is the path length operator defined in (3.2). $[[P_t]]^n$ denotes the subset of $[[P_t]]$ formed by taking the first n paths from the ordered set $[[P_t]]$. These are the n shortest paths given the internal ordering of $[[P_t]]$ by ascending $\mathcal{C}([P]_i)$. If all links have only logical length then every c(u,v) = 1 in forming $\mathcal{C}([P]_i)$ and PLE is then the ratio of the total *number of individual links* used in an actual restoration plan to the theoretical minimum total number of links. If links are characterized by geographical distance, then c(u,v) in (3.2) is real-valued and in that case PLE measures the ratio of total path lengths and the ideal plan may possibly use more links in total than the actual plan.

The corresponding network-average PLE measure is analogous to the network PNE measure:

$$PLE = E_{(i=1..|[S]|)}\,(PLE_i) \tag{4.6}$$

It should be stated that PLE is only defined, for our purposes, when the *actual* restoration plan satisfies the condition PNE = 100%. Otherwise if the actual restoration plan has fewer paths in total than required for 100% PNE then it may also have a TPL that is *less than* that of the reference solution, which implicitly has 100% PNE. This appears to produce a PLE greater than 100% but in fact is a condition under which PLE is not defined.

4.2.4 Shortest Path Probability (SPP)

While PLE measures the routing length efficiency in terms of total number of links used to provide all the paths in a given restoration plan, it is also of interest to measure the *probability that any given path is shortest path,* ie. a member of the set of ideal-length paths from the reference solution for the same spancut. To form this measure it is necessary to construct a mapping between the ordered set of actual path lengths $[L_a]_i = \mathcal{C}([P]_i \in [[P_a]])$ and the corresponding ordered set of path lengths from the ideal solution, $[L_t]$. The mapping is rooted in the set of actual path lengths and constructed as follows:

```
1. Both sets are ordered by increasing path length.
2. for i := 1 to |$[L_t]$| mapping[i] := nul
3. for i := 1 to |$[L_a]$| begin
4         j:= i-1
5.        repeat
6.        j:= j+1
7.        if (($[L_a]_i = [L_t]_j$) and (mapping[i] = nul)) then mapping[i] := j
8.        until (mapping[i] <> nul) or j = |$[L_t]$|;          end
```

The SPP is defined for a given spancut (u,v) using the mapping set constructed above as:

$$\text{SPP} = \frac{|\text{mapping}_{uv}|}{\min\{|[[P_t]]_{uv}|, W(u,v)\}} \tag{4.7}$$

where |mapping| gives the number of non-nul elements in the mapping set constructed as above to relate path lengths in the actual solution to those with equivalent lengths in the ideal solution. The respective number of ideal paths to be considered is given by min $\{|[[P_t]_{uv}|, W(u,v)\}$. If $W(u,v) < \infty$ then the SPP measure takes on an operational context, otherwise an intrinsic context is implied for SPP by normalization to the number of topologically possible paths.

Note that the exact routes of two paths, one in the actual solution, one in the ideal solution do not have to be identical to be associated together by the above mapping function. Rather, the mapping seeks to construct one-to-one matchings between paths based on their lengths. Such mapping for determination of SPP is defined only for logical or integer path lengths. The procedure can however be adapted easily to construct an equivalent association between real-valued paths with a defined band of approximate equivalence. The SPP of an entire network is correspondingly defined as the average of the individual SPP values obtained from every possible spancut in the network.

$$\text{SPP} = E_{(i=1..|[S]|)}(\text{SPP}_i) \tag{4.8}$$

4.3 Defining the Theoretically Ideal Solutions

To compute any of the topological performance measures above we require knowledge of some theoretically ideal restoration plan for each spancut in a network. Specifically, we need to know $[[P_t]]$ for use in the above relations. But how exactly do we define $[[P_t]]$ and how can we obtain such reference solutions in practice?

4.3.1 k-Shortest Paths or k-paths of minimum total length ?

In Chapter 3 we discussed the so-called k-shortest paths literature and found that most investigators seek k paths with no disjointness constraint amongst them. Paths are ranked only by differences in total length of each path. Under these circumstances there is no difference between k-shortest paths and k paths with minimum total length. However, when disjointness amongst paths is introduced we have to be aware that *k-shortest link-disjoint paths* (meaning a set of paths comprised of first the absolute shortest path, then the shortest path remaining without reusing any links of the first, then the shortest path remaining without reusing any links of the first or second paths, etc.) is not necessarily the same as the set of *k link-disjoint paths having minimum total length*. We adopt the former criterion for an ideal reference model but

some discussion is warranted to explain this issue and to give the rationale for the choice which we have adopted.

The two criteria which are candidate properties for a hypothetically ideal restoration scheme are: (a) restore via k-shortest link-disjoint paths, (b) restore via k link-disjoint paths which have *minimum* total length. Two variations on (b) are : (c) find paths which have *bounded total length*, (d) find paths which have *bounded individual lengths*.

The difference between (a) and (b) is very significant for the complexity of the problem that is set as the formal objective. Criterion (a) implicitly requires that *the* shortest path is one of the paths in the solution. Criterion (b) requires *any* k (link-disjoint) paths having minimal total length and does not necessarily include *the* shortest path in the solution. A large complexity difference arises because (a) stipulates a length constraint on *each of the k paths individually*, whereas (b) stipulates a length constraint on *the k paths collectively*. The latter can be satisfied only by a *global optimization* whereas the former can be satisfied by *k local optimizations*. Equivalently the former is a combinatorial problem formulation while the latter is amenable to a "greedy algorithm" approach [HoSa78].[1] In the particular case this difference is a choice between an NP-complete problem formulation, which inevitably requires an exponential-time combinatorial search algorithm on the one hand, or a problem solvable by a polynomial-time algorithm on the other hand. The important difference between (a) and (b) only arises when the constraint of link-disjointness is applied because the selection of successively shortest individual paths introduces the possibility that, with two paths $\underline{P}_1$ and $\underline{P}_2$,($C(\underline{P}_1) < C(\underline{P}_2)$), the routing of $\underline{P}_1$ can influence the routing of $\underline{P}_2$ by siezing links which would have been present in $\underline{P}_2$ had all paths been collectively chosen with a global view to minimum total path length.

Criterion (d) above is clearly not appropriate for the restoration problem because it can introduce a compromise in path number. In (d) the path length bound is a primary factor which, if it has effect at all, can *only prohibit* the creation of one or more paths if the network topology does not support k paths *each* with $C(\underline{P}) < B$. That is to say that k', the topologically-limited number of paths under (d) can only be less than or equal to the topology-limited k under either problem statements (a) or (b). Now let us consider the complexity of (c) and (b). (c) is already known to be NP-complete by virtue of the work of [RoPe84]. Regarding (b), it is also fairly easy to deduce that it too is an NP-complete problem by virtue of decomposition of the problem statement and comparison to known NP-complete problem formulations in [GaJo79]. For the application of NP-completeness theory it is necessary to convert the problem which is actually faced, into a related *decision problem* formulation (see [GaJo79] sec. 2.1). We can do this for (b), decomposing it into a set of decision problems which are inherent parts of (b) as follows: Q0(a) Do any k mutually link-disjoint paths exist between u and v in **G** ? If so, let B equal the total path length of the set of paths found in satisfying Q0(a). Now to address (b), we also need to

know that B is a minimum so it is next necessary to ask : Q0(b) Does G contain k disjoint paths between u and v having total length bounded by constant B? We know by this decomposition that (b) is an NP-complete problem because Q0(b) is a decision problem which is listed in a compendium of NP-complete problems produced in [GaJo79].[2]

We can now summarize on this point, justifying our selection of the *k-shortest paths* formulation (a) as the most advisable problem statement for our approach to the restoration problem based on the following considerations: Path number efficiency and speed are paramount for a restoration scheme. The issue between (a) and (b) pertains only to path length efficiency which is of secondary concern in practice but problem formulation (b) is NP-complete. It is therefore not a realistic target if we are to have any chance of coming up with a truly distributed local-data-only method to solve restoration in real time. Formulation (a) has the particular advantages of *progressiveness* and *stability* in that individual path results can be put into effect as soon as they are realized. There is no need to wait for computation of all possible path sets to ensure globally minimum TPL before committing to any paths, or alternatively no need to subject paths, once established, to possible revision for global length reducing trade-offs between individual paths. Because our goal is a fully distributed, fast, single-pass method, k-shortest link-disjoint paths will be our chosen ideal reference model. In practice, Selfhealing deviates very little from perfect k-shortest path properties and these properties also seem to differ quite infrequently from globally minimum TPL in any case. By accepting k-shortest paths behavior rather than minimum total length behavior however, we obtain the desired properties that polynomial-time complexity and stable progressive path accumulation are possible for Selfhealing but such a restoration scheme will not strictly ensure globally minimal-length path sets.

4.4 A Reference Algorithm for k-shortest Paths

Having decided that *k-shortest link-disjoint paths* is the appropriate definition of the *ideal* path properties for the present problem, we need a way to find such solutions. We found in the course of this work that in relatively sparse networks or when few routes are required, it seems relatively easy for a human being to identify, visually, ideal sets of k-shortest paths in a graphical network model when logical path length is used and every individual link is graphically visible. This way of checking the Selfhealing solutions was in fact used for the first year of the project. However, the relatively powerful visual ability to spot these patterns fails when network connectivity is high, when maximum path finding stress applies, when span sizes do not permit graphical resolution of every link or, when real-valued link distances are used.[3]

Therefore to determine, reliably, the theoretically ideal solution sets, a centralized algorithm for the problem of k-shortest link-disjoint paths in a multi-graph was designed and implemented as a tool for this research. The algorithm is called *MetaDijkstra* because the approach it takes is to create one higher dimension than the space within which the normal *Dijkstra* algorithm [Dijk59] works and, from that higher dimension, manipulate the Dijkstra procedure by feeding it a succession of appropriately modified simple-graph network models, abstracted from the actual multi-graph network. An outline of the algorithm is as follows:

An initial database of the entire *multi-graph* network indicates each span in the network, the number of spare circuits on each span, the span distance (and the number of working circuits if in an operational context) in span-list format (see [AHU83] for graph representation formats). Successive calls to an implementation of Dijkstra's algorithm for the single-source shortest paths problem in simple graphs are used to obtain one path at a time until the k required shortest link-disjoint paths are found in the multi-graph, or no more paths are possible. The paths are found by tracing through the $\underline{P}$ vector (not same P as we use for path vector) resulting from the Dijkstra search for the single shortest path in each modified network (see the following or [Dijk59]). At each iteration one path is found. In the nth iteration, one link on each span along the route of the nth path is removed from the *multi-graph* network representation. For the (n+1)st iteration a new *simple-graph* representation is abstracted from the modified multigraph network and presented to Dijkstra to find the shortest path in this dummy network. A link in the simple-graph representation that Dijkstra sees is eliminated from all subsequent simple graph abstractions when all links of the corresponding *span* in the multi-graph network have been utilized.[4] We will now review Dijkstra's shortest path algorithm and then show how it is extended for the present purposes.

4.4.1 Dijkstra's Algorithm

The most common path-finding problem is on a directed graph **G=(N,L)** in which all links have a nonnegative weight (ie. distance, cost, delay, risk etc.) and one particular node (or vertex) is specified as the source. Dijkstra's algorithm solves what is called the single-source all shortest paths problem in this type of network. Our implementation of Dijkstra's algorithm uses a two-dimensional adjacency matrix representation of the network in which directed link (i,j) between the nodes i,j is represented by the finite cost C[i,j] of the link between them. C[i,j] has units of whatever cost metric applies - we assume logical distance. Entries in C[i,j] which correspond to a link that does not exist in the network are given some suitably large numerical value to effect infinitely long (ie.unconnected) links.

The Dijkstra algorithm works by maintaining a set [S] of nodes whose shortest distance from the source is known. Initially [S] contains only the source node *s*. In each iteration another node, *v*, is found which is at the next shortest possible distance from *s* on a path going through one of the nodes already in [S]. Node *v* is then added to the set S (strictly, *v* is concatenated to

[illegible]

[S] because the set order matters in the final interpretation). Once [S] includes all nodes of the network, the algorithm halts. During execution an array D is used to record the minimum length path to each vertex found above. At the end, D holds the magnitude of the shortest path to each destination node. The algorithm does *not* directly produce the actual paths corresponding to each shortest route but it is a simple extension to construct this information during execution and this is included in the following implementation. The notation [-] denotes set operations and \[ ] is the not function for set inclusion.

```
procedure Dijkstra;
begin
S:=[s]; {initialize}
for every i in N \[s] do begin  D[i]:=C[s,i];
                                P[i]:=nul;
                                end;
{main loop}
for i := 1 to (n-1) do begin
{A. find the node outside of the current S that has the minimum distance from s}
        minval:=infin; wmin := 1;
        for every w in N\[S] do
                        begin
                        if D[w] < minval then begin
                                                            minval:= D[w];
                                                            wmin := w
                                                      end;
                        end;
{B. add the selected node to S}
                        [S]:=[S]+[w];
{C. update minimum distances to the next tier of nodes reachable through the selected node w}
                        for every v in N \[S] do
                                if (D[w]+C[w,v] < D[v]) then begin
                                                      D[v]:= D[w]+ C[w,v];
                                                      P[v]:= w;
                                                      end;
        end;
{D. produce a table of the actual shortest path routings from P[v] with destination vertex at left-hand side}
for i:= n downto 1 do begin
        write(n);
        repeat
                j:= P[n];
                write(j);
                n:=j;
         until (j = s);
        writeln;
        end;
end;{ Dijkstra}
```

4.4.2 The *MetaDijkstra* Algorithm

To arrive at the *MetaDijkstra* algorithm, the above procedure is iterated with a successively modified dummy network matrix as a means to obtain k-shortest fully disjoint paths between two specific nodes. While *Dijkstra* (as shown for illustration above) prints out a final table of (non-disjoint) shortest paths from the source node to every other node, we modify that part of the procedure for use in *MetaDijkstra* so that it reports after each call in terms only of the routing chain from source to destination for the node pair of interest. The *MetaDijkstra* algorithm is:

```
{s= source node, d= destination node,
C[i,j] = network span distance matrix, T[i,j] = span depth matrix,
D[v] = final shortest distances from source node to other nodes produced by Dijkstra,
P[v] = predecessor routing vector produced by Dijkstra.
P[v] is initialized to a predefined nul value in Dijkstra,
infin is a constant arbitrarily larger than possible path length}

MetaDijkstra
begin
while not eof(input) do begin {initialize C[ ], T[ ] matrices}
        readln(input, u, v, dist, nlinks)
        C[u,v]:=dist ; C[v,u] := dist;
        T[u,v]:=nlinks; T[v,u] := nlinks
        end;
T[s,d]:=0;  C[s,d]:= infin;
k:=0;
repeat
k:=k+1;
Dijkstra (s,d,C[]);{call modified Dijkstra with specified source, destination and dummy network graph}
if D[d] < infin then begin
        write(k 'th shortest path is :');
        j:=d ;
        repeat
                T[P[j],j]:= T[P[j],j] - 1;
                write(j,'-');
                if( T[P[j],j] <= 0 then begin C[P[j],j] := infin; C[j,P[j]]:= infin; end;
                j:=P[j];
        until (j:=s);
        end
until (D[d] = infin);
end; {MetaDijkstra}
```

An executable program for *MetaDijkstra* with other features such as logical / geographical distances and operational / intrinsic mode choices plus TPL and path number summaries etc. are available from the author but, for brevity, are not included here. Henceforth in this work we treat the output of *MetaDijkstra* as defining the theoretically ideal path sets by the criterion of k-shortest link-disjoint paths in a multigraph for purposes of quantifying the performance of Selfhealing as a truly distributed solution for the restoration problem. Several points of further research involving the *MetaDijkstra* algorithm are discussed in Ch. 13.

4.5 Summary

The field of distributed automatic restoration routing is a new area in which recognized performance measures and agreed objectives do not exist. Consequently a necessary part of this work was to develop and define a quantitative framework within which advanced restoration systems can be systematically measured and performances compared. In order of their practical importance, we have defined path number efficiency (PNE), several speed measures, path length efficiency (PLE) and shortest path probability (SPP) as the quantitative measures of restoration system performance which we will use to characterize Selfhealing. These metrics are defined for both operational and theoretical study contexts and are dependent on knowledge of the ideal k-shortest link-disjoint paths solution. To obtain these reference solutions the centralized *MetaDijkstra* algorithm was developed.

CHAPTER 5

DISTRIBUTED INTERACTION VIA SIGNATURES

As far as the author is aware, all distributed routing algorithms in use or proposed for communications networks today rely on *interprocessor messaging* as the basic paradigm for interaction between nodes. Even the few distributed algorithms mentioned in Ch. 3 assume all interactions take place by means of messages explicitly addressed to neighbouring nodes via some reserved data link which is separate from the normal traffic-carrying signals of the network and that such data links are general purpose, ie. they carry messages of all types pertaining to all transactions between those nodes. Within this paradigm any node in the network can send messages addressed to any other node, not only directly adjacent nodes, via well-known X.25 type packet relay protocols. This approach to distributed interaction is so ubiquitous that many would ask what other paradigm is there?

Selfhealing uses a very different interaction paradigm - that of isolated event driven "signature" processing, rather than interaction through explicit messages addressed between the processors of the network. Selfhealing interaction is much more indirect. It is more like one imagines the workings of a biological system: numerous, isolated, primitive cells simply react to the local environment at their boundaries and change their local environment slightly. Coherent large-scale actions of the organism (ie. the network) occur only as the collective by-product of each node's local interaction with its environment. In Selfhealing, that environment is comprised of *signatures*, one statically embedded on each transport entity sent and received by every DCS node. This chapter describes signatures, and the philosophy of interaction through signatures, rather than messages. This sets the stage for the next chapter which gives the core description of how the restoration problem can be solved by means of a protocol for the distributed isolated processing of these signatures at every node of a network, ie. the Selfhealing protocol. This will involve going into the specific logical manipulations and a hardware support environment for implementation of Selfhealing on a real DCS machine.

How, in practise, to apply signatures transparently to their respective carrier signals is an allied research question that the author has addressed. Particularly in existing DS-3 based networks, there is, at first inspection, no apparent way to attach any ancillary information such as a signature to the DS-3 transport entity. In closing this chapter we return to the problem of applying signatures to the DS-3 signal with a review of three papers by the author which address this important adjunct problem to the present study.

This chapter and the next should be taken as a companion pair. Only when taken together do they completely explain the technique of Selfhealing. In discussing signatures in this chapter it is inevitable to delve partially into how the protocol works, properly the topic of the next

chapter. The reader is asked to view such forward allusions to the protocol itself as a form of introduction to concepts which are revisited in depth in the next chapter. The two chapters will then stand complete : this one describing the objects which Selfhealing manipulates, the next giving the rules for manipulation of these objects.

5.1 Signatures

The various nodes of a Selfhealing network interact solely through *restoration signatures* on the links between them. As a paradigm through which to coordinate the distributed action of many nodes, signatures are different in several aspects from conventional sending of messages between processors. A signature is not an interprocessor messaging channel or even a one-time message - it is a dedicated *attribute of a transmission link*. A signature is invisible to the traffic on its link but is physically inseparable from that link and contains only the fields which will be defined in this chapter, not a general-purpose messaging capability. Unlike a messaging link, a signature is *not addressed to any particular other node*. A node originating a signature does not necessarily know who will receive that signature or who will be influenced by its secondary effects. If the path of a transport entity carrying a signature is suddenly switched anywhere in the network, its signature intrinsically goes with it.

The prior radioisotope analogy best describes the original use of one or a few unique signatures at a time for path audit trace (Ch.1) but does not apply in exactly that manner when there are hundreds of signatures in existence at once in Selfhealing. The radioisotope notion of *labelling* is still relevant to the Selfhealing application however; each signature is a unique label of sorts, but a further analogy is needed to best describe the role of signatures in Selfhealing.

5.1.1 Distributed Interaction Through Signatures

Although Selfhealing nodes all have a processor, no effort is made to communicate processor-to-processor in a direct attempt to co-ordinate efforts in working out the restoration problem. Selfhealing restoration instead occurs as a network-level by product of the *isolated* actions of each node. The Selfhealing task inside each node sees the outside world *only through the signatures arriving on transport entities at its site* and (when Selfhealing is running) the nodal processors act only as specialized signature-processing engines. Processors influence each other indirectly however, because each Selfhealing node reacts to the signatures arriving at its periphery and, according to the Selfhealing protocol, may make changes in the number, content and link association of all the signatures emitted from its site in response to the signatures arriving at its site. (The rules for such changes are the topic of Ch.6) During Selfhealing, the processor of the host DCS effectively becomes isolated from all other nodes in the conventional sense of data communications and acts only as an isolated processor of signatures sitting in and reacting to a sea of changing signatures.

One analogy for the node-based Selfhealing protocol is that it is a set of rules for playing a game. This "game" is defined by rules of operation on signatures and by a goal of seeking certain combinations of signatures amongst several families of signature types. The rules define a set of legal operations and required actions for priority amongst signatures, for the origination, duplication and elimination of signatures, for changing ones own state, and for dropping out of the game, or for declaring success. Each node is a player of this game and seeks a certain goal condition that is specified in terms of pairs of *matched* signatures. From the viewpoint of any one node there is also the apparent element of chance (as to exactly which signatures will come its way). When a Selfhealing node achieves instances of the goal, there is a reward; permission to operate a crosspoint.[1] Information as to *which* crosspoint to operate is not encoded in the signatures but rather, by the port interfaces which happen to hold the matched signature indications when they are found.

This is clearly a major shift in paradigm for distributed interaction. Traditional messaging-based distributed system design methods are replaced in Selfhealing by collective large-scale action which is only a by-product of locally sought goals pursued according to a set of rules. This alternate paradigm of distributed interaction is fundamental to the speed, autonomy and database-freedom that has been achieved in Selfhealing. In further chapters it will become increasingly apparent that the desirable attributes of the Selfhealing method trace back in great measure to this choice of nodal interaction paradigms. In particular, the unique ability to construct k link-disjoint paths *simultaneously* in parallel, is due to this approach. Recent Bellcore [YaHa88] and NEC [Yamir88] attempts at a Selfhealing-like restoration scheme use the conventional messaging based interaction paradigm and, not coincidentally, lack this most advantageous ability.

5.1.2 The Static Nature of Signatures

Unlike messages, which have an essentially one-time occupancy of the data channel, signatures are static quasi-permanent fields repeatedly *impressed* in a transparent way on the individual transport links of a network. The static persistent nature of signatures has benefits in simplicity and robustness when used in Selfhealing, primarily because signature-state information (and implicitly *network* state information) is stored *on the links of the network*. This state information is available to the SH task in the receive signature register of the respective port card so that the network itself becomes the database Selfhealing uses. If the node originating that signature changes it, the relevant 'memory locations' at another node are immediately updated because *the link itself is the memory medium*. To achieve the equivalent real time state information management with one-time messages on a shared data link would require an enormous volume of message traffic and the SH task would have to use memory and time intensive methods to construct and maintain a continually correct synthetic *link signature state image* of the overall node. If one of the conventional messages (being used in this hypothetical case to convey signatures) was missed or corrupted, the whole memory construct loses integrity. This drives the need for X.25 type layered communications protocols to manage retransmission of an errored message when needed and so on, rapidly eroding real time performance if messaging attempts to perform the same functions achieved by signatures.

In the static approach of signatures, no such time-consuming communications protocols is needed: if any one instance of a signature repetition is errored (and fails its Checksum test), the receive hardware register always holds its last valid value and accepts the next repetition of the signature when correctly received and checksum validated in hardware. Quite simple dedicated hardware in the DCS port card is all that is required to send and receive (or rather '*apply*' and '*detect*') signatures reliably. The static space-divided (rather than time divided) assignment of signatures onto individual links is the basic 'signalling method' of Selfhealing. It is a major contributor to overall speed because signatures never queue for transmission, they propagate at carrier signal velocities and are received and verified in hardware. There is no receive buffering in the usual sense: a new signature *overwrites*, rather than *follows*, any previous signature that was applied to a TE. All that matters at any time in Selfhealing is the *current* signature state of a link, not the series of signatures that may have been applied. In messaging of course every message in a series must be preserved, usually through buffering. A final property, at the heart of the method, is that signatures can obviously only be placed on links that exist. The events that occur in a Selfhealing restoration are inherently driven to an outcome that is ultimately determined by the network's own topology at the time of the event. The network topology becomes like a spatially encoded program that is indirectly read and followed by the processors of the network in their collective role acting as a single distributed multi-computer.

5.2 Content and Uses of Signature Information Fields

We now describe the information fields that are present in a Selfhealing signature. A description of the protocol for processing such signatures follows in the next chapter and depends on this section as background. This section aims only to introduce each field and outline the *principles* behind its use, deferring some essential details until the following chapter. The heavily interlinked descriptions of this chapter and the next can then be seen as a single body.

Fig.5.1 shows the information fields in a Selfhealing signature. The number of bits for each of these fields can be set for the size of network in which it is deployed, both in terms of number of nodes and total number of transport entities for which unique signal IDs are required. A *mode 0* signature is the type of signature which is present in the dynamic phase of a restoration event.

In normal times when there is no failure, mode 0 signatures are present on all links but in a defined *null state*. A *mode 1* signature is used after a rerouting path is synthesized to solve the problem of end-to-end mapping (Ch.3, Constraint 3). The mode 1 signature forwards a signal identifier to ensure that an identical traffic substitution assignment is made at both end nodes.

Mode Ø

NID	SOURCE	TARGET	INDEX	REPEAT	RA	MODE	CHECKSUM
						= Ø	

Mode 1

NID	SOURCE	TARGET	SIGNAL ID	MODE	CHECKSUM
				= 1	

Figure 5.1 - Information Fields in a Signature

It is not essential to have two separate signature modes. One composite signature type could be defined in which the contents of *mode 0* plus a *Signal ID* field are present. During the active phases of restoration, the *Signal ID* field would be present in every signature but would be *null* and ignored. The protocol would switch from *null* to a valid *Signal ID* just as it now switches to a *mode 1* signature. Two signature modes are used at present however because Selfhealing speed benefits from use of the shorter *mode 0* signature during the time-critical dynamic phase of path synthesis. In DS-3 networks there may be only 8 to 23 kb/s available for signature transport [Grov87a], [GrKr87]. In these conditions a mode 0 signature, roughly halves the signature length, giving a worthwhile reduction in signature transfer time.

The main development of the Selfhealing method has to do with *mode 0* signature processing. Therefore, unless specifically indicated as mode 1, the word "signature" implies *mode 0* in the following. Also in all of the following, the *mode bit* and *checksum* fields will be omitted from discussion, as their roles are obvious. It is assumed that the hardware functions of the port card include a checksum test and mode determination before the signature contents are made available to the DCS processor.

5.2.1 Fields Common to Mode 0 and Mode 1 Signatures

NID : NID stands for Node IDentifier. This field is always asserted by any node when it applies a signature to any transport entity. The NID field is written with the unique network-wide identifier of the node which physically applied the signature. The NID field is not used for any explicit routing purpose, rather, its role is in deducing link-to-span set associations so that each DCS can group links arriving at its site into logical spans. This uses the property that all signatures arriving with the same NID must have come from the same adjacent node, and therefore form a span. Span entities are a necessary logical construct in the Selfhealing protocol and by this use of NID the requirement for stored tables describing the composition of spans at each node is removed. The SH protocol will deduce link-to-span associations for itself using NID. Although spans are immediately apparent to a human being looking at a network map, recall that all the Selfhealing protocol sees from its position in the node is an unstructured array of port appearances. Span information could be stored in tables in the node but then the usual problem of maintaining accurate distributed network configuration databases arises. Instead with the NID field of a signature, links carry the identity of their adjacent physical originations so that whenever any normal maintenance, or service rearrangements are made, the network is self-updating for Selfhealing purposes. In normal operating conditions a *null* mode 0 signature is placed on every transmit link (working and spare). In a null signature all fields *except the NID* are set to predefined *null code* values. In both normal and emergency times NID is always the ID of the immediate physical applier of the signature.

Source, Target : The SOURCE and TARGET fields are always the identities of the two nodes directly adjacent to the relevant span fault, called Sender and Chooser nodes in the protocol. These fields are only assigned by the Sender and Chooser nodes when they emit original non-null signatures into the network. Other (Tandem) nodes may be required to repeat these fields but they never overwrite them. Source and Target fields are not used for explicit routing purposes as their names may tend to imply. In fact, if only one span fault were to exist in a network at one time, then Selfhealing works *without* any Source, Target fields whatsoever.

In the existing protocol however, Source and Target fields have two purposes: (1) They provide a *problem instance stamp*. For any signature in the network, source and target fields constitute a *unique label* identifying the span failure to which the signature pertains. If there should happen to be two spancuts in the network at once, source-target fields provide a means of keeping all processing for one fault independent of simultaneous involvement in another Selfhealing problem.[2] (2) Given that Source, Target fields exist to provide independence amongst the processing of multiple faults, the SH protocol exploits them further by applying a *signature direction* convention to the name order in Source-Target fields as a means of distinguishing *forward* from *reverse* signatures (If Source-Target fields were not employed a single *direction bit* would be added to the mode 0 signature.). A *forward* signature is distinguished from a *reverse* signature by having SOURCE = (Sender ID), TARGET = (Chooser ID) while a reverse signature has SOURCE = (Chooser ID) and TARGET = (Sender ID). The NID of the Chooser is known by the Sender (and vice-versa) because this is the NID field of the last valid signature received (and transparently latched) in the receive signature register(s) of the working ports that were put into an alarm state by the span failure.

5.2.2 Mode 0 Specific Signature Fields

Index: The INDEX field is an arbitrary but unique *serial number* assigned to each primary signature originated by the Sender at commencement of the forward flooding phase. INDEX is never altered by any other node and comes to represent a sub-family of signatures in the network. The protocol recognizes the index sub-family to which every signature belongs and treats signatures in certain ways within a family and different ways across index families. Briefly, the role of index in the protocol is the management of simultaneous parallel contention amongst the several path construction efforts that occur concurrently. Each family of signatures stemming from one index value assigned by the Sender comes to effect an independent parallel simultaneous path finding experiment which competes with all other index families, each having the potential to result in one link-disjoint path. By the composite action of the protocol at Tandem nodes, some indices will succeed and yield successful restoration paths; others will fail in the competition for the required spare links. An index which succeeds in path creation releases excess links from its family and these are taken up according to rules of competition by other as yet unsuccessful index families and so on.

Each tandem node of the protocol is simultaneously observing relationships between signatures at its site and is empowered to shuffle resources (ie. links) from the mesh of one index into the mesh of another index, always enforcing consistency with the rules for relationships between signatures at any one node. The rules by which the Tandem shuffles links amongst the competing parallel meshes or families of signatures on each index is what causes k shortest disjoint paths to be synthesized. The notions of competition amongst indexes trying to expand the total interlocking mesh of signatures on their index will be important in the next chapter.

Repeat: The REPEAT field is a logical distance counter which is part of an overall mechanism for control of the range (in space *and time*) of signature propagation in a Selfhealing event. Associated with the repeat field of a *mode 0* signature is system-wide constant (or node-specific constant) called the *repeat-limit* (or maxrepeats). When the Sender initiates a forward signature with index X, the numerical difference between the *repeat-limit* and the *initial repeat value* (IRV) that the Sender puts in the repeat field of that signature determines the *reach permission* of the family of signatures on index X. The reach permission is the maximum distance (in logical spans) that any set of related signatures on the given index will be allowed to propagate away from the Sender node. For a network-wide *constant reach permission* plan, all nodes use the same initial repeat value for all signatures they originate. Each Tandem node increments the REPEAT field of any signature that repeats a signature. Tandem nodes enforce

the reach permission by ignoring any signature that arrives at its periphery with a REPEAT value greater than or equal to the repeat limit. *Variable reach permission* network plans are based on specially selected IRV values for each node.

RA bit: RA stands for Return Alarm. The RA bit is used to ensure that an alarm is seen at *both ends* of a failed span without necessarily waiting for ordinary alarm detection processes to raise the alarm. This measure is particularly needed for the special case where Selfhealing acts like a conventional protection switching system. In protection switching the problem is usually a *one-way* single-regenerator, splice, fiber or connector failure. The whole span is not severed and the fault may not be bidirectional. The RA bit ensures that in such circumstances the nodes at the ends of the span are *both* triggered into Selfhealing operation, not just the end physically on the receiving end of the failed link. Whenever the receive circuits of any DCS port raise an alarm, the RA hardware immediately echoes that alarm state in the other direction of the circuit which that port terminates, thereby ensuring an alarm state at *both* ends of any circuit which experiences a failure. When the whole span is cut real alarms in both directions assure the same effect and RA is redundant is this case.

The SH protocol never directly sees or writes the RA bit because all operations on the RA bit can be completely delegated to the DCS port hardware. The effect seen by the Selfhealing protocol is that there is no difference between a primary alarm in a port card or an alarm raised as a result of the RA mechanism. The hardware supplied RA function is as follows: (The notations RS.(), TS.() stand for receive signature and transmit signature registers which will be defined in the next section.)

```
RA Logic {own-alarm is the internal alarm state generated by hardware monitors in this port, alarm is
the output alarm indication shown to the DCS processor. }
If (own-alarm) then TS.RA := true {cause remote alarm} else TS.RA := false;
If (RS.RA or own-alarm) then alarm := true {cause local alarm } else alarm := false;
```

5.3 Signature-related DCS Port Hardware Functions

Fig. 5.2 shows the transmit and receive signature registers, port status register and interrupt logic that constitutes a DCS portcard from the viewpoint of the Selfhealing protocol. All Selfhealing signature processing rules are defined in terms of operations on the set of data structures represented by all the ports of a given DCS node. Fig. 5.2 shows only the programmer's view of a port card, not the circuit functions that lie between this interface and the serial bidirectional DS-3 transmission interface (see [Grov87c] for the latter). The functions and properties of the portcard registers are as follows.

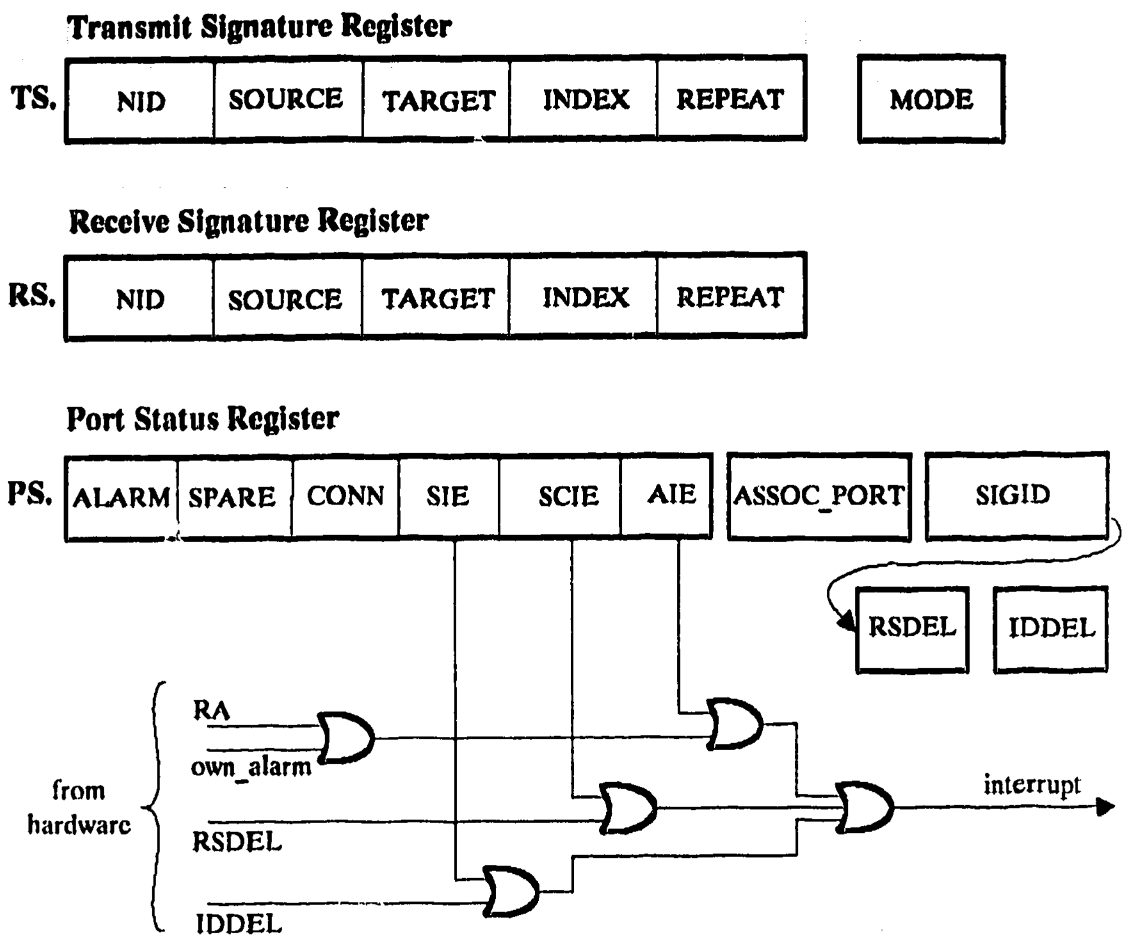


Figure 5.2 - Processor View of Port Card Hardware

5.3.1 Signature Transmission and Reception Registers

Each DCS interface port has provision for storage and transmission of one transmit signature (TS) and reception and storage of one receive signature (RS). From the DCS processor point of view, the TS register is a static memory-mapped port to which it can write the fields of a transmit signature for application to the TE transmitted from that port. The fields written to the TS register are thereafter continually repeated on the outgoing transport entity by special circuits in the transmit side of the port card, according to whatever technique is used for signature transport (sec. 5.6). The TS register has read/write status, as seen by the processor, and is used by the protocol as the memory location which stores the current transmit signature for that port.

On the receive side of a port card, a corresponding read-only receive signature (RS) register exists and is comprised of two main parts. One is a serial-in parallel-out (SIPO) register for signature extraction from the line signal and alignment and verification. The other is a parallel-in parallel-out (PIPO) signature-holding output register. The second register always holds the last valid signature received from the link for presentation to the DCS processor. It is this parallel output register that is shown in the view of the RS function in Fig.5.2. The SIPO register (not shown) accumulates individual signature bits from the incoming transport entity and serially shifts them past circuits that recognize signature start/end orientation. Whenever a complete signature is received in the first register and verified through hardware checksum calculation (or simple repeated persistence checking) and that signature *is different from* the current contents of the *output* register, then the new signature is latched into the output register and a *signature-delta interrupt* (RSDEL) is raised to inform the processor that a signature event has occurred in that port. When an alarm occurs in a port, the last valid receive signature is frozen in the RS output register. Complete descriptions of the transmit and receive signature circuit functions for DS-3 operation using the framing bit modulation are given in a patent on the method [Grov87c].

5.3.2 Port Status Register

The DCS port also provides hardware support for Selfhealing in the form of a Port Status Register (denoted PSR or just PS in some contexts). The PSR includes ALARM, SPARE, CONNECTED, Alarm Interrupt Enable (AIE), SIGID Interrupt Enable (SIE) and Signature Change Interrupt Enable (SCIE) status and control bits plus an ASSOC-PORT field and a SIGID field as shown in Fig. 5.2. The port status register has read/write status, as seen by the SH task, with the exception of ALARM and SIGID registers which are generated by hardware and so are read-only to the processor.

ALARM is generated either within the port itself or via the RS.RA mechanism in response to a remote alarm. ALARM may also include a wired connection to the alarm state of transmission terminal equipment external to the DCS. The **SIGID** register is loaded by line receiving hardware similar to the RS SIPO whenever a mode 1 signature is received carrying a valid SIGID. At any time that a *non-null* SIGID is recorded in the PSR of a port it is known that port is in the path of live traffic. Subject to interrupt masking, a vectored interrupt for the SH task can be generated by the port card in the case of three events : (a) a transition in ALARM from false to true, (b) a transition from null SIGID contents to non-null SIGID contents, and (c) a change in the *output* receive signature register. The vectored interrupt automatically identifies the interrupting port to the processor. Each interrupt can be masked from a port by setting the AIE , SIE or SCIE bits to zero, respectively disabling the ALARM, SIGID, and signature change interrupt functions.

Role of SIGID: The hardware function of the PSR is such that any time the protocol enables the SIGID interrupt (from a masked state) the SIGID register is simultaneously cleared for the receipt of a new SIGID. This arms the port to receive the SIGID of a live transport entity when the

[illegible]
Chooser has initiated reverse linking and is waiting for the traffic identifier to arrive to know which port to connect to in order to restore the one of the several failed live traffic TE's. Unique network-wide SIGIDs are planned for every transport entity in future SONET-based networks. In such a network, SIGIDs are unique identifiers issued to each piece of network equipment which is able to source a transport entity into the network (eg, a multiplexer, a video codec). Each transport entity is impressed with this SIGID when created at its sourcing equipment. In this class of network, when the Selfhealing Sender node substitutes live traffic into a restoration path it need not explicitly also forward a SIGID, because the act of substituting the signal intrinsically also forwards the SIGID. In this case no mode 1 signatures are needed in the method.

The **SPARE** bit is written and maintained by the DCS O/S in normal ongoing provisioning and commissioning activities. If a spancut occurs, SPARE indicates which ports are in traffic-bearing use and which are available (spares) for use in restoration at the time of the actual fault. If SPARE is true, and ALARM is false, the port is available for use in the Selfhealing method. If SPARE is false and ALARM is true, then the port represents one of the *working circuits* that must be restored. As soon as Selfhealing DCS *uses* a SPARE port in a restoration path, it sets SPARE = false so that immediately after restoration, each restoration path will automatically enjoy the revised status of a working circuit if another fault occurs. If a second fault affects a 'working' circuit that in fact is a spare that has already been pressed into restoration service, Selfhealing will intrinsically try to restore again, within whatever further redundant capacity may be available. (Leaving restored routings up so long that a secondary fault has any significant likelihood would not be expected practise however.). Several alternatives are possible regarding reversion after the span failure is physically repaired. Simple extensions to the protocol can permit automatic reversion if desired.[3]

CONNECTED is a bit which explicitly logs whether the port is currently connected via a crosspoint of the DCS matrix to any other port of the DCS. When CONNECTED is true, the allied ASSOC-PORT field is used to store the identity of the port (on the same DCS) to which this port is connected. CONNECTED will ordinarily be true only for a port in working mode or a spare port which Selfhealing has used in a restoration path and is now carrying traffic. When CONNECTED is *false*, ASSOC-PORT has an alternate use as a logical pointer in the Selfhealing protocol. In this role ASSOC-PORT is used by the protocol to store the ID of some other port on the same DCS which has a receive signature that is the current *precursor* for the TS presently emitted by this port (discussion follows). In the normal course of day-to-day operations the ASSOC-PORT register and CONNECTED bit are written by the DCS O/S whenever it makes new connections. Selfhealing may write to both these attributes of the PSR during a restoration event but these actions are compatible with normal provisioning changes to working circuits because Selfhealing only operates on spare unconnected ports.

5.4 Notation for Signature Operations

The set of TS, RS and PSR registers on every port of the DCS collectively form a *'smart' memory space* in which all data needed for the Selfhealing protocol are present and automatically updated by hardware in response to external events. To the processor this memory space appears to be completely self-updating. The entire memory space required by Selfhealing is kept current by events outside the node itself. The SH task is therefore relieved of large overhead efforts which would otherwise be required to maintain a current memory image of the total node signature state and the protocol enjoys the unusual situation that what is in memory is current with the outside world *by definition* - an ideal sought after in all real time systems programming problems. We now define a logical view of this structured memory space and a formal notation for addressing and operations within this space. These notations and operations are a foundation for describing the SH protocol in detail.

From the viewpoint of the processor in a DCS, every port card is a structured memory type with four attributes:

```
type portcard =        port: portnumber        {1..maxnoports}
                       RS  : RSregister  {receive signature register}
                       TS  : TSregister  {transmit signature register}
                       PS  : Port-Status-Register
```

Port is the identity of a port on its host DCS. It is of scalar type **portnumber** which enumerates the set of all valid portcard addresses on the host DCS machine, plus an explicit *null* identifier. The other three elements of the portcard are structured objects:

```
type RSregister = type TS register =   NID, source, target: nodeid
                                       index, repeat : {integer, null}
                                       RA, mode : boolean

type Port-Status-Register =   SigID : integer
                              Assoc-port: portnumber
                              alarm, spare, AIE, SIE, SCIE, Connected : boolean
```

nodeid is a scalar type with range equal to the set of valid node names in the network plus a defined *null* node name. Valid ranges for the index and repeat fields are the integers but their type of definition also includes a defined *null* value.

The memory structures of a port card are addressed by *signature specifications* as follows:

sigspec = register <.field> @ port

This signature addressing format indicates the register (TS, RS, or PSR), the field (any of the variables defined for the type addressed in *register*) and the identity of the DCS port card at which the preceding data is addressed. *<.field>* is an optional specification of an individual variable within the selected register. If *<field>* is not present, the entire addressed register is treated as one object. For example: (a) TS.REPEAT @ 12 specifies the repeat field of the

transmit signature register at port # 12 on the host DCS. (b) RS @ *int-port* specifies the entire receive signature found at the port number given by current value of a variable *int-port*. Often when a signature register it treated as one entity it is for copying a whole signature from one port to another which, for example, might take the form: (c) TS @ port1 := RS @ port2. TS and RS objects have the same subset of fields and are therefore compatible types for operations such as (c). It is necessary in general to specify TS or RS whenever a single field is addressed in one of these registers because fields of the same name exist in both TS and RS structures. In contrast, each sub-field of the PSR is unique to the port card and so, for brevity, specification of the PSR register can be implicit. For example: (d) ALARM @ scanport is equivalent to (e) PSR.ALARM @ scanport.

The ASSOC-PORT field of the PSR is used by the SH protocol for indirect addressing operations. In this use, the contents of Assoc-port is the identity of another port on the DCS, either relating physical connection to another port or logical precursor relationship to another port. The latter usage occurs only when a port is SPARE and not CONNECTED (ie. of use in restoration). An example of the usages that will arise is the following:

(e) TS.INDEX @ port*x* := RS.INDEX @ (ASSOC-PORT @ port*y*)

In this example the index field of the transmit signature register at port*x* would be assigned the value of the index field of the receive signature register which is found at the port *whose address is given in the ASSOC-PORT* register of port*y*.

RS and TS types have a common information structure, but an RS *register is by nature read-only*, whereas TS registers are read/write by the SH protocol. An assignment *to* an RS register is always invalid because the contents of an RS register are determined by the port card receive hardware which interfaces to the world outside the node.

5.5 Special Relationships Between Signatures

5.5.1 Class and Sense

The *class* operator is used to test whether two signatures pertain to the same network fault. The *sense* operator is used to determine whether two signatures of the same class have the same *direction*. Let [name1,name2] be an ordered couplet of valid nodenames and let "=" denote an operator that tests for (unordered) set equivalence. *class* (.) is as an operator which extracts a couplet of node names from the source and target fields of a specified TS or RS signature register. That is: *class* (*RS* @ *portid*) := [*RS*.SOURCE @ portid, RS.TARGET @ portid].

class (reg1 @ port1) "=" *class (reg2 @ port2)* implies that SOURCE, TARGET fields of the two specified signatures are *either* identical or symmetric. A common usage in the protocol is to check every signature received for relevance to the SOURCE, TARGET fault instance for which the SH task is currently executing. This is conveniently done by defining a *current-class* couplet and preconditioning all processing in response to a new signature with

if (class (RS @ int-port) "=" current-class) then....

otherwise the signature is spurious and should be ignored or processed in the context of another concurrent Selfhealing task instance (ie.there are multiple simultaneous faults). It is in fact implicit in the protocol description to follow that the above test to ensure that any signature received pertains to the same fault currently being processed and this will not be explicitly repeated.

The *sense* function is used to determine a notion of direction which is used by the protocol. It is defined as follows:

```
sense ( reg1 @ port1, reg2 @ port2);
begin          if (class (reg1 @ port1) "=" class (reg2 @ port2) then begin
               if (reg1.SOURCE @ port1 = reg2.SOURCE @ port2) sense := same
               else sense := opposite  end
     else sense := null          end
```

5.5.2 The Precursor Relationship

A node in the Selfhealing Tandem state (to follow) is frequently required to rebroadcast a received signature according to specific rules. When such a selective rebroadcast occurs each of the TS registers written by the Tandem node are said to have a *precursor relationship* to the receive signature which they are logically extending by such rebroadcast. If the precursor of the TS at port2 is an RS at port1 we denote this as:

precursor(TS @ port2) = RS @ port1

The precursor relationship always implies the following relationships between one RS and possibly many TSs.

TS.INDEX @ port2 = RS.INDEX @ port1
TS.REPEAT @ port2 = (RS.REPEAT @ port1) + 1
class (TS @ port2) = *class* (RS @ port1)
sense (TS @ port2) = *sense* (RS @ port1)
ASSOC-PORT @ port2 = port1

Precursor is a directed one-to-many relationship. One RS can be the precursor for many TSs, but every TS has only one precursor. The precursor relationship strictly applies between *signatures*, not ports. Because TS and RS signatures within one port may have different *index* fields, the following relationships can both be true, underlining that it is *signatures* not *ports* for which the precursor relationship is really defined:

precursor(TS @ port2) = RS @ port1

precursor(TS @ port1) = RS @ port2

ASSOC-PORT is a convenient way to avoid long searches to find the precursor of a given TS, a fairly frequent requirement in the protocol. Precursor identification can otherwise require scanning the entire node to identify the precursor RS by deduction from the above properties alone. The Assoc-port register converts the $O(n)$ search to an $O(1)$ access by pointing to the port where the precursor RS is found for any active TS at a Tandem node. To obtain this benefit, the protocol must maintain Assoc-port correctly through all its operations. When this is done however, a simple construct serves to implement *precursor*(-) as a function which returns the sigspec of the received signature that is precursor to a given TS:

Precursor (TS @ portx) = RS @ (ASSOC-PORT @ portx)

5.5.3 Complement Signatures

A *complement signature pair* is two signatures found at *reg1* @ port1 and *reg2* @ port2 which satisfy the following symmetry conditions:

class (reg1 @ port1) = *class* (reg2 @ port2)
sense (reg1.@ port1, reg2 @ port2) = *opposite*
reg1.INDEX @ port1 = reg2.INDEX @ port2

Complement(sigspec1,sigspec2) is an operator which tests two specified signature registers for this relationship. In application the complement condition is of most significance when it is found between the RS and TS registers of *the same port*, ie: *Complement* (RS @ portx, TS @ portx) is true. NID and REPEAT are 'don't care' fields in the complement condition. When testing for the complement condition *within one port*, we may use the simplified notation *Complement(port)*.

5.6 Methods for Signature Transport

We have so far described the content and structure of signatures and defined some key relationships and operations for manipulations on signatures. But how in practise are these signatures to be transported over the TEs of the existing network so that they are applied from TS registers and detected in RS registers as we have assumed, without any effect on existing payloads?

The requirement for signature transport is easily met in a SONET network by reserving one of the existing unallocated signalling fields for this purpose [SONE88], [MoGr87]. This would yield 64 kb/s for signature transport. Another possibility for SONET is to use the Embedded Operations Channel (EOC) which is a defined part of the STS-1 standard. This would yield an instantaneous signature transfer rate of 192 kb/s although the EOC is a common shared messaging channel subject to data protocols, buffering, and does not provide the desired properties obtained with static signatures on every TE. In either case, no special transmission methods have to be devised for SONET. In today's DS-3 network however, there is no overhead to add signature indications to the DS-3 line signal. A separate series of investigations has therefore been conducted to address this important issue. This includes the proposal and analysis of three new techniques for transparent carriage of auxiliary information in the DS-3 signal format. This section summarizes the principal ideas and findings of that companion study.

5.6.1 F-bit Modulation

[Grov87a,c] introduces and treats a technique called framing bit (F-bit) modulation. The principle is to exploit the significant resilience and hysteresis of commonly used standard stochastic frame-finding circuit designs to convey auxiliary signaling information as single errors forced on the primary framing bit pattern of the DS-3 format. This can be done in a manner that is compatible with both asynchronous DS-3 and SYNTRAN [SYNT87] DS-3 formats and has only a trivial effect on reframing time statistics and no impact on in-frame DS-3 performance. A logical one in the signaling stream causes a complement operation on the next outgoing F-bit which is defined as a candidate for error modulation. A logical zero would cause no F-bit errors. Signaling-bit and signaling-word synchronization and resistance to errors in the auxiliary information is achieved in relatively simple circuits which process the F-bit error output of an existing DS3 framing circuit. With *gated* F-bit modulation the extension of reframe times is controlled to a maximum of one excess frame-hunting iteration for modulation spacings as low as every tenth F-bit. The limiting factor becomes the need to not trigger a false reframe in a correctly framed system. In gated F-bit modulation with every 13th F-bit subject to modulation during an ON period of 280 F-bits repeating cyclically every 700 F-bits, the capacity of the signaling channel obtained is $(280/700)* 44.736x10^{+6}/(170*13) = 8.097$ Kbs. This is the primary method of DS-3 signature transport assumed in the Selfhealing DS-3 network model which was implemented for research purposes.

5.6.2 Signalling via Randomly Occurring Stuff Opportunity Bits

[GrKr87] pursues the idea of identifying and harnessing the randomly occurring stuff opportunity bits in the DS-3 pulse-stuffing tributary justification scheme [see BTL71 or Smit85]. This method can provide a higher auxiliary signaling rate than F-bit modulation but it is not compatible with the SYNTRAN DS-3 format. It works by substituting auxiliary information bits for the stochastically arising dummy bits present in subframe 8 of the asynchronous DS-3 format whenever positive pulse stuffing occurs for tributary frequency adjustment.

The normal DS-3 signal is obtained by multiplexing seven plesiochronous DS-2 signals. The alignment (called justification) of tributary rates before synchronous multiplexing is achieved through dynamic positive pulse stuffing. Each frame of 680 DS-3 time slots contains one time slot (stuffing slots denoted S1-S7) which can optionally be used to transmit either information from an appropriate DS-2 tributary or to pad-out the DS-3 format as needed to mop up tributary asynchronism. *Stuffing* occurs when the tributary builds up a sufficient phase lag that bit duplication would result unless a dummy bit is substituted. When this occurs, the stuffing slot carries a dummy bit that is presently discarded at the demultiplexer. We propose that such redundant randomly occuring time slots can be used to carry auxiliary signalling information. Such signalling opportunities will arise unpredictably in time but with appropriate buffering we have shown that a minimum average capacity of 23 kb/s is available. This capacity is completely transparent to the existing payloads and to existing or new equipment which is not intended to process the newly-derived signalling channel.

5.6.3 DS-3 C-bit Liberation via DS-2 Stage Synchronization

[Grov88] is an analysis and extension of a recent proposal by AT&T [Tatu87] to free the C-bits (stuff indication bits) of the DS-3 format by artificially synchronizing and frequency depressing the DS-2 intermediate stages of the multiplexers which create the DS-3 signal. As proposed by Tatulis [Tatu87] this results in a jump-level M13 multiplex terminal in which full-time (100%) pulse-stuffing replaces the previous randomly required pulse stuffing operations. This differs from the above in that the bits normally used for *stuff indication* (not the stuff opportunity bits (S1-S7) themselves) are the focus of interest. These so-called C-bits are freed for new uses because explicit indication of which frames contain pulse stuffing is no longer required once it is known that every stuffing opportunity is seized.

Our analysis of C-bit liberation by DS-2 synchronization and rate adjustment to cause *100% DS-3 stuffing* shows that the principle can be advantageously generalized to solutions which free the DS-3 C-bits through deterministic sequences of *ganged stuff events* that *do not require 100%* DS-3 stuffing. This is desirable because we find that depression of the DS-2 rate to cause 100% DS-3 stuffing also depresses the nominal stuff ratio of the dependent M12 stage to 0.07 threatening to compromise DS-1 jitter performance.

Faced with this and other difficulties arising with [Tatu87] we refine the proposal by formulating a generalized method for C-bit release using ganged deterministic stuffing and we exploit an additional degree of design freedom to minimize, simultaneously, the change in M12 conditions and achieve the simplest circuit design for a freed C-bit multiplex. In the generalized approach C-bits are freed not by forcing *all* stuffing slots to be seized but rather by DS-2 stage synchronization and frequency manipulation to cause *wholly predictable sequences* of stuffing patterns to occur. Criteria for an optimal choice of a new (offset) DS-2 rate are formulated and a computer search of 130,816 candidate DS-2 frequencies was performed resulting in two design solutions that simultaneously satisfy all objectives. These solutions each provide a 188 kb/s C-bit channel and either a 32.9 or 21.9 kb/s S-bit channel, while leaving the nominal M12 stuff ratio at 0.334 +/- 0.05 and 0.02 respectively.

5.7 Summary

This chapter was devoted to *signatures*, as defined for Selfhealing. We outlined how signatures constitute a new paradigm for massively parallel event-driven interaction in distributed systems where the signatures can be placed on the links of the transport network. The content of signatures for Selfhealing purposes was outlined and the important attributes of the hardware support environment for Selfhealing operations was defined. In the next chapter we shall see how the Selfhealing protocol operates on the signature registers of its host DCS to derive isolated crosspoint operate decisions that synthesize complete restoration plans at the network level.

Regarding the problem of signature transport, any of the three methods above could be used in Selfhealing DS-3 networks. Each of the methods requires relatively little circuitry additional to an existing DS-3 interface port design and all are completely invisible to existing payloads and network equipment. Therefore none of these methods is expected to create a major obstacle to hardware support of Selfhealing in new DCS-3 designs. A particular advantage of F-bit modulation is however that it is the only method which works equally well in asynchronous DS-3 [CCIT85], SYNTRAN DS-3 [SYNT87], and the newly proposed C-bit parity DS-3 format [Tatu87]. This particular method would then allow a DCS 3/3 machine to be designed for Selfhealing with any mixture of DS-3 format types without further special considerations.

CHAPTER 6

THE SELFHEALING PROTOCOL

In Ch. 5 we were introduced to the concept, structure and principles of interaction through signatures. This chapter now describes in detail how the Selfhealing protocol operates on these signatures in a distributed process that results in rapid simultaneous synthesis of a required number of link disjoint parallel paths. All processing by the SH protocol is of an event-driven nature using Finite State Machine techniques to encode the behavioral rules. Readers unfamiliar with FSM or concurrent processing concepts may find it helpful to first read Ch. 7 (Research and Implementation Methods) before returning to this chapter.

The present description of the Selfhealing protocol has three stages of refinement and formality. Our first discussion will focus at the network level to convey an overall conceptual view of how Selfhealing works. The second stage of refinement goes down to the level of signature manipulations within a single node, focusing on the signature-defined events that occur and the rules for how the node reacts to these events. This description is couched in terms of the hardware model and logical environment of TS, RS and PSR register structures established in Ch. 5. We introduce *signature diagrams* which are a form of graphical language developed in the course of this work to represent the signature processing operations of the protocol.

The third level of refinement is a complete pseudo-code specification of the protocol, included as Appendix B. This level of formality is primarily intended as a reference work for those who wish to pursue implementation or repetition of our results. Appendix B and portions of this chapter use a *pseudo-code language* that is a combination of Pascal and 'C' language constructs, set algebraic constructs, and informal English statements where conciseness benefits without loss of explicitness. This follows the approach of [AHU74] and others where exact but concise procedural specification is required.

6.1 Network-level Overview of Selfhealing

At its highest level of abstraction the Selfhealing (SH) protocol is implemented as a finite state machine (FSM) shown in Fig. 6.1. There are four states and three major processing blocks. The overall action of a SH restoration event can be introduced at the network level with two conceptual stages, *selective forward flooding* and *reverse linking*, and the three states *Sender, Chooser and Tandem.* The phases are not truly exclusive in time or space during a SH event, although forward flooding always precedes any reverse linking. For initial purposes of description we will however describe these phases as if they were non-overlapping in time and space. In Fig. 6.1 there are only two primary events that drive the FSM: a signature arrival (RS in arrow) and the alarm event. After describing the whole protocol in detail this FSM diagram will be revisited summarizing many of the details that will be discussed .

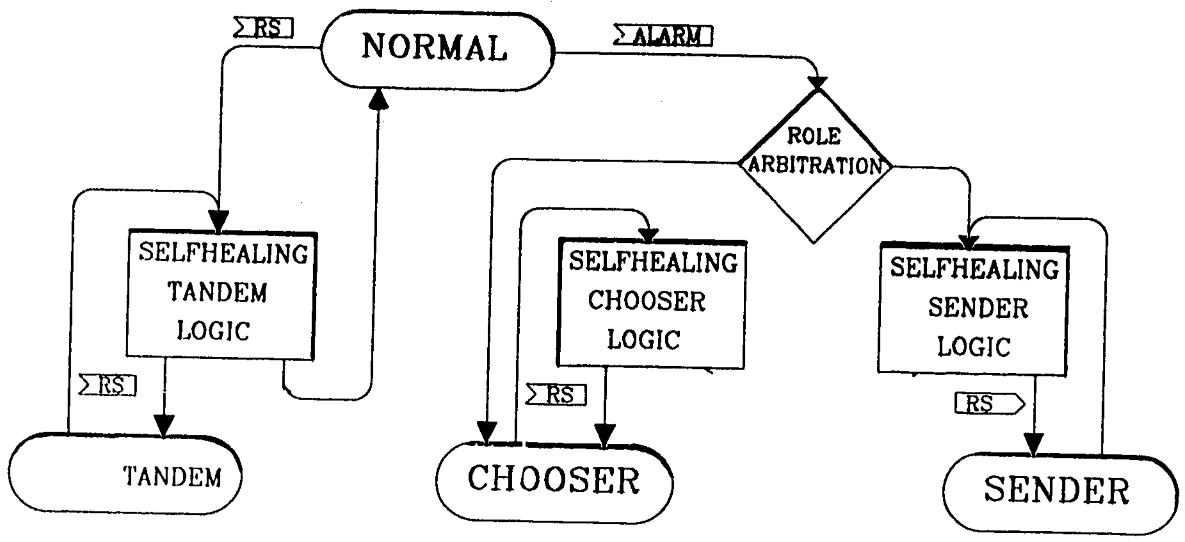


Fig. 6.1 Basic structure of Selfhealing finite state machine.

6.1.1 Selective Forward Flooding

A SH event begins when a failure occurs on a transmission span and causes an alarm. The DCS machines in the two nodes adjacent to the failed span are aware of such traffic-affecting major alarms either (a) by virtue of jumpering transmission bay alarms over to the DCS equipment or preferably, (b) by alarm activation directly in the port cards of the DCS machines through loss of signal, loss of frame, loss of clock and/or BER. The detection of an alarm on any *working* circuit raises an interrupt which invokes the SH task. In the most likely case of one failure at a time, the SH task will be initially in the *Normal* state. When invoked in the *Normal* state by an alarm, the SH task first identifies all the working ports that have an alarm state due to the failure. The individual port alarm detections may be distributed in time. The protocol therefore has an alarm holdoff time (100 msec used) for alarm collection. It can also incorporate *late alarms* arriving after its initial holdoff and primary scan for alarm pickup.

Sender-Chooser Arbitration: Immediately after the spancut, only two SH tasks are 'awake' in the network. These are the two SH tasks invoked by alarms in the end nodes of the failed span. After the above alarm pickup action, each of these SH tasks reads the last valid contents of RS.NID @ every port where (ALARM = true) and (SPARE = false) and thereby learns the identity of the node to which connectivity has been lost. When the SH task finds the RS.NID that is common to all failed ports, it performs an ordinal-rank test on the remote node NID with respect to its own NID to determine whether to act as *Sender* or *Chooser* in the remaining protocol events. For instance, if span (B-C) is cut *ord*(C) > *ord*(B) so both nodes implicitly reach the determination that C will act as Sender and B will act as *Chooser*.

The actual outcome of this test is arbitrary but it guarantees that *one* node will adopt the *Sender* role and *the other* becomes *Chooser*. The SH task in every node is capable of either Sender or Chooser roles and will adopt one or the other depending on the arbitrary name-rank of the two nodes involved by any given span failure. Any arbitration rule which can be conducted independently at each node with only local information, and ensures complementary outcomes, will suffice for this aspect of the method. The *Sender* immediately begins the Selective Flooding phase by broadcasting appropriately indexed transmit signatures on some or all (details to follow) of the spare links leaving its site, wherever those links go in the network.

Tandem Node Recruitment: These signatures propagate from the Sender at carrier velocities and appear within milliseconds as new signature arrival events in the RS registers of ports on neighboring DCS machines. This raises RSDEL signature change interrupts, causing the O/S of *those nodes* to invoke their SH task. Nodes awakened by signatures, with no alarm present at their site, will enter the Tandem node state of Fig. 6.1. The main effect of Tandem node behavior at this stage of events is selective rebroadcast of incoming signatures on spare links from *that* node location. Details of selective rebroadcast are deferred except to say that the network-level effect is obviously to activate yet further DCS nodes into the Tandem state. In this way all DCS nodes within a certain range of the Sender are rapidly alerted into the Tandem state and all perform selective rebroadcast. The range of influence of the selective forward flooding signature wave is controlled by the repeat limit mechanism.

An eventual effect of this forward flooding wave is that if, within the range limit, there is *any* connectivity between *Sender* and *Chooser* in the network sub-graph comprised of spare links remaining in the network after the failure, then one or more of the forward flooding signatures arrives at the *Chooser* node. This raises a signature interrupt at the Chooser and triggers a reverse linking sequence (if no forward signatures reach the Chooser node then no method can restore the fault by diverse routing.).

6.1.2 Reverse Signature Linking

In the cases of realistic interest (networks such as in Ch. 2) one or more forward flooding signatures does reach the *Chooser* node. When the first forward flooding signature on a given index arrives at the Chooser, the Chooser initiates a distributed reverse-linking process which will trace backwards through the mesh of signatures realizing one path through an arbitrary number of co-operating Tandem nodes. (The apparent straightforwardness of these steps at this stage is deceiving, especially when proceeding in parallel for k paths at once). To initiate the reverse-linking mechanism that will result in one of the required paths, the *Chooser* applies a *complementary signature* on the transmit side of a port having a preferred receive signature. The *Chooser* node emits no signatures other than in response to qualifying forward signatures that arrive in its port registers. When the reverse-linking signature arrives at a node that is in the Tandem state, the Tandem-state SH task again acts according to the Tandem Logic block in Fig.6.1, details of which follow. The net effects of Tandem node action at this stage is however :

(a) selective deletion of certain forward flooding signatures emitted by the Tandem node site,
(b) operation of a specific DCS matrix crosspoint,
(c) retransmission of a complementary signature on the transmit direction of the circuit on which an appropriate forward signature is present,
(d) general reorganization of transmit signatures on all remaining links, including possible creation of new signatures, according to specific rules.

After transiting through one or more tandem nodes which act as above, a reverse linking signature arrives back at the *Sender* and forms a complementary signature pair with an existing Sender-originated signature. A complete bidirectional path is then known to have been established between Sender and Chooser when this occurs. This new path may be routed through a number of cooperating DCS (up to the repeat limit) which have already operated the crosspoints required along this new route. The *Sender* does not know the routing of this new path, (no single node knows more than the crossspoint(s) it has operated) but the *Sender* is assured by the method that the *other* end of the new path is indeed the Chooser node - the node to which direct span connectivity has been lost. When such a reverse-linking signature arrives at the Sender, the *Sender* operates a selected crosspoint at its site to substitute the implicitly realized restoration path into one of the working digital paths that was affected by the span failure. The end-to-end mapping aspects of the protocol ensure that *Sender* and *Chooser* both substitute this route for the *same* one of the many DS-3 entities that may be managed in a single restoration.

6.1.3 Overall Network Level Dynamics

The above description is deceptively simple because it describes the basic dynamics of a single path found through unimpeded forward flooding and a single reverse linking sequence. In reality, the process of forward flooding proceeds incrementally and simultaneously on each of several indices and reverse linking also occurs simultaneously on some indices while other index families are still expanding. Each family of signatures, comprising the network-wide set of selective rebroadcasts resulting from each original signature issued by the Sender, identifiable by their common INDEX field, competes against the others to expand. Such families then later collapse in the process of reverse linking. Through the rules of forward signature rebroadcast, the family of signatures on each index effectively seeks to expand, in competition with all other indices, into the network. Each index family is trying to place one signature of its type on every span in the network. The network-wide pattern of signatures on a given index is mesh-like because each rebroadcast from a Tandem node goes in all directions, including back towards the Sender.

When (or if) the mesh of signatures on a given index happens to expand to reach the Chooser node, the Chooser's single reverse-linking signature has the effect of dissolving all of that mesh of signatures except for a subset of selected links along which a single path back to the Chooser is consolidated through the action of Tandem nodes with reference to the set of precursor relationships on that index in existence at each node when the reverse linking process reaches it. The dissolving is effected by Tandem nodes in the reverse linking path by suspending all now-superfluous transmit signatures on the given index except for the complement signatures on the axis of the newly created path itself. Suspension of these indices causes them to disappear from the inputs of adjacent Tandem nodes, and thereby frees other links at other nodes in the network and so on completely collapsing the mesh of the index that has reverse linked. However, immediately as a link is freed by this process at a Tandem node, the free link becomes available for optimal incorporation into the meshes of remaining signature families on *unsatisfied indices* which are still seeking to expand. Tandem nodes immediately reallocate any free links optimally into the meshes of simultaneously expanding *unsatisfied* indices.

The net effect is a network-wide competition, mediated by the Tandem node, amongst all indices in the network, each of which seeks to expand its mesh to touch the Chooser. As soon as any index succeeds in doing this, its signature mesh collapses onto a single path described by the trajectory of the reverse-linking process which is guided by the precursor associations in effect at each Tandem node when reverse linking reaches it. All links of the dissolved mesh that are not in the final path are re-incorporated where possible to expand other remaining meshes. If these meshes succeed in reaching the Chooser, they too are catalyzed into a path, collapsing and making links available for other indices, and so on. A recursive dynamic picture of staggered intermittent mesh expansion followed by sudden collapse emerges from this process: At first all indices compete to expand. By chance one succeeds, and promptly collapses. This advances its old competitors, another one of which succeeds and collapses, thereby advancing the remaining group of competitors and so on. Thus there is a rapid dynamic series of interactive mesh expansions and mesh collapses, each collapse leaving behind only a thread of concatenated links that comprise one path. This asynchronous, three-dimensional unregulated process proceeds until either all index meshes have succeeded and collapsed onto a path (all required paths are found) or halts when no remaining index mesh can succeed in expanding to the Chooser (all topologically possible paths have been found). In the latter case, a Sender timeout eventually suspends any remaining unsatisfied index meshes by cancelling their originating signatures. Whenever 100% restoration is achieved, the Sender also suspends any unneeded surplus signature originations, collapsing unneeded meshes without waiting for timeout. In either of these cases, the final state of the network is that *only bidirectional complement signature pairs persist*, each pair running the length of one continuous non-branching restoration path that condensed out of that index mesh. In a given restoration event, no one index is assured of being realized into a path, but the effect is that all link-disjoint shortest routes that are topologically feasible are found by those indices which do succeed.

An index mesh may advance either by profiting from the collapse of other indices or by advancing in geographical extent (up to the maxrepeat limit) around regions of the network where all spare links are temporarily seized. Because the competition is intense enough to saturate the spectrum of possible path lengths, and because shortest path expansions enjoy first reverse linking, the overall product is a heuristic mechanism which produces k link-disjoint paths with nearly perfect k-shortest path properties. Later results quantify the frequency at which the method results in perfect k-shortest path properties, and analyze the dynamics of the few cases where it is suboptimal.

6.2 Detailed Description of Signature Processing Rules

We will now describe the signature processing rules through which Selfhealing is effected in a more detailed manner totally constrained to the isolated context of one node. We use for discussion a simple example of a span cut between two nodes P and Q in an arbitrary network in which some third node K is involved as a Tandem node. Node K is not necessarily adjacent to P or Q for any of the following cases. For simplicity, the (P-Q) span cut is assumed to have disrupted only 3 working circuits. The notation TS=(...) or RS=(...) is used to enumerate the contents of a mode 0 signature, in the implied order NID, SOURCE, TARGET, INDEX, REPEAT. For conciseness, the fields RA, MODE and checksum are omitted from the drawings and discussion. '*nul*' is used to denote a signature or signature field that is in the logically inactive or inapplicable state. 'x' denotes a 'dont care' field whose exact value is not of significance to the protocol in the current context.

6.2.1 Sender-Chooser Arbitration and Initialization

Shortly after the span cut on the facility between P and Q, alarms occur at sites P and Q, causing the O/S at both sites to invoke their SH tasks. If this was not a span failure but a single unidirectional system failure, the RA bit would have effected an equivalent bidirectional failure on the one failed system. Node P reads the last valid receive signatures on all ports with SPARE = 0 and ALARM = 1 and sees that RS.NID =Q is the common node to which connectivity is lost. (If all of the alarmed ports do not show the same NID then we have 2 or more simultaneous span cuts. Extensions to handle this will be discussed later.) The SH task at node Q is similarly activated and sees RS.NID = P on all such alarmed working ports. Nodes P and Q independently perform the arbitrary test to determine Sender and Chooser roles. Using *rep-port* as any representative port affected by the (single span) failure, each SH task performs the arbitration as follows. MyID is a local constant for the network-wide ID of the host node.

```
If (ALARM @ rep-port) and (not(SPARE @ rep-port))
        then if [ord(MyID) > ord(RS.NID @ rep-port)] state := Sender
        else state:= Chooser
```

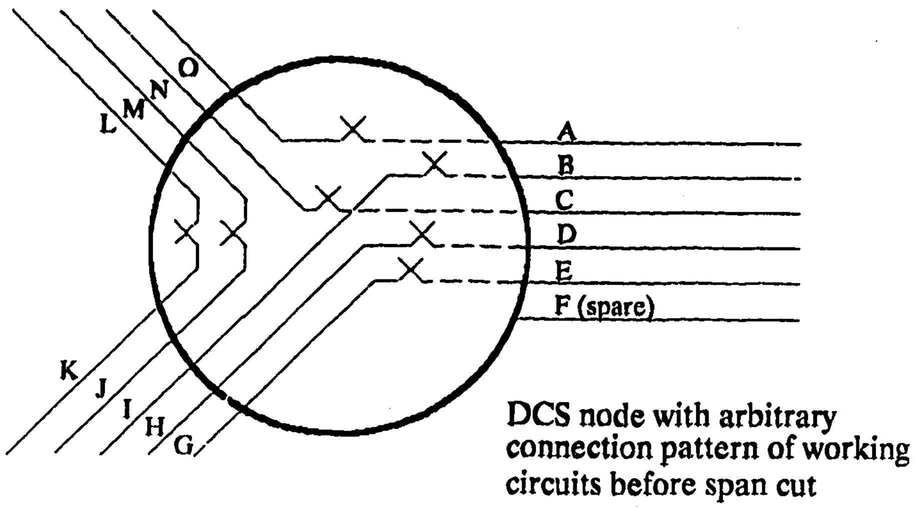
Note that a given node can be *either* a Sender or Chooser but may adopt either role for different spancuts. In the example with nodes P and Q, the above will result in Q becoming Sender and P becoming Chooser.

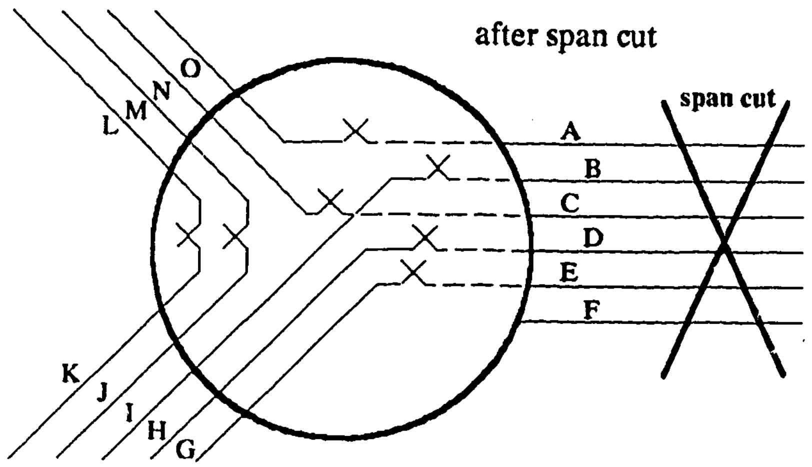
Initializations: When a node realizes it is one of the *alarm nodes* in a SH event, it performs a set of common initializations that are required whether it is to act as Sender or Chooser. First, all working ports at the node are scanned to learn how many working circuits were affected by the span fault. In this scan, temporary tables are constructed for ready working reference to the alarmed ports and to the non-alarmed but disconnected working ports to which the alarmed ports *were connected*. The basic procedure is:

```
[affected-ports] := nul ; [alarmed-ports] := nul ;
lostccts := 0
for scanport := 1 to size-of-DCS do
        if not(SPARE) @ scanport and ALARM @ scanport then
                begin
                lostccts := lostccts + 1
                [affected-ports]:= [affected-ports] + [ASSOC-PORT @ scanport]
                [alarmed-ports] := [alarmed-ports] + [scanport]
                CONNECTED @ scanport := false
                AIE @ scanport := false
                end
end {for}
restccts := 0
```

lostccts stores the number of working circuits to be restored. *restccts* will count successful restoration paths as they are found. Not shown above is an additional test to ensure that every alarmed port that is logged indeed belongs to one common span cut. The line *AIE @ scanport := false* disables further alarm interrupts from the alarmed ports. The *affected-ports* are not themselves the ports experiencing the failure. Rather, they are the working non-alarmed ports to which the failed ports were connected through the DCS matrix prior to failure. *These* are the ports to which it will be required to connect restoration paths to restore the end-to-end network path that was disrupted by the span failure. Fig.6.2 illustrates the concept of *affected-ports* and *alarmed-ports*.

[Affected_ports] and [Alarmed_ports]


O
N
M
L
A
B
C
D
E
F (spare)
K
J
I
H
G
DCS node with arbitrary connection pattern of working circuits before span cut



[Affected_ports] = [O, N, I, H, G]

[Alarmed_ports] = [A, B, C, D, E]

Fig. 6.2 **Affected-ports** and **Alarmed-ports** sets at DCS node adjacent to span failure.

Two other initialization tasks are performed by every node when first invoked. These are to scan the node to build link-to-span associations and to build temporary working lists (or sets) for each logical span that is found. These working sets organize all spare ports on each span into (non-exclusive) subsets of ports which have (a) *nul* (background only) TS, (b) an active TS, (c) an active RS. These working sets speed execution of the protocol by avoiding frequent searches of the whole node to find ports satisfying various signature conditions. The span-oriented structure of these sets is fundamental to the protocol however, as spans are a necessary logical construct in Selfhealing but the protocol must deduce the existence of spans for itself by recognizing groups of ports having common RS.NID. In the following we will explain the action of the protocol assuming it has already constructed the required span-oriented working sets.

6.2.2 Sender Selective Flooding

In our example, node Q is Sender. Immediately after the above initializations it goes on to initiate the forward flooding process. Signature flooding, and all subsequent signature transmissions are on *spare links only.* Sender flooding occurs on every span except the alarmed span. Every other span is flooded with a number of signatures which is the *lesser of* {lostccts, or number of spares available on the span}. Each forward signature initiated is also assigned a unique index and the following other attributes:

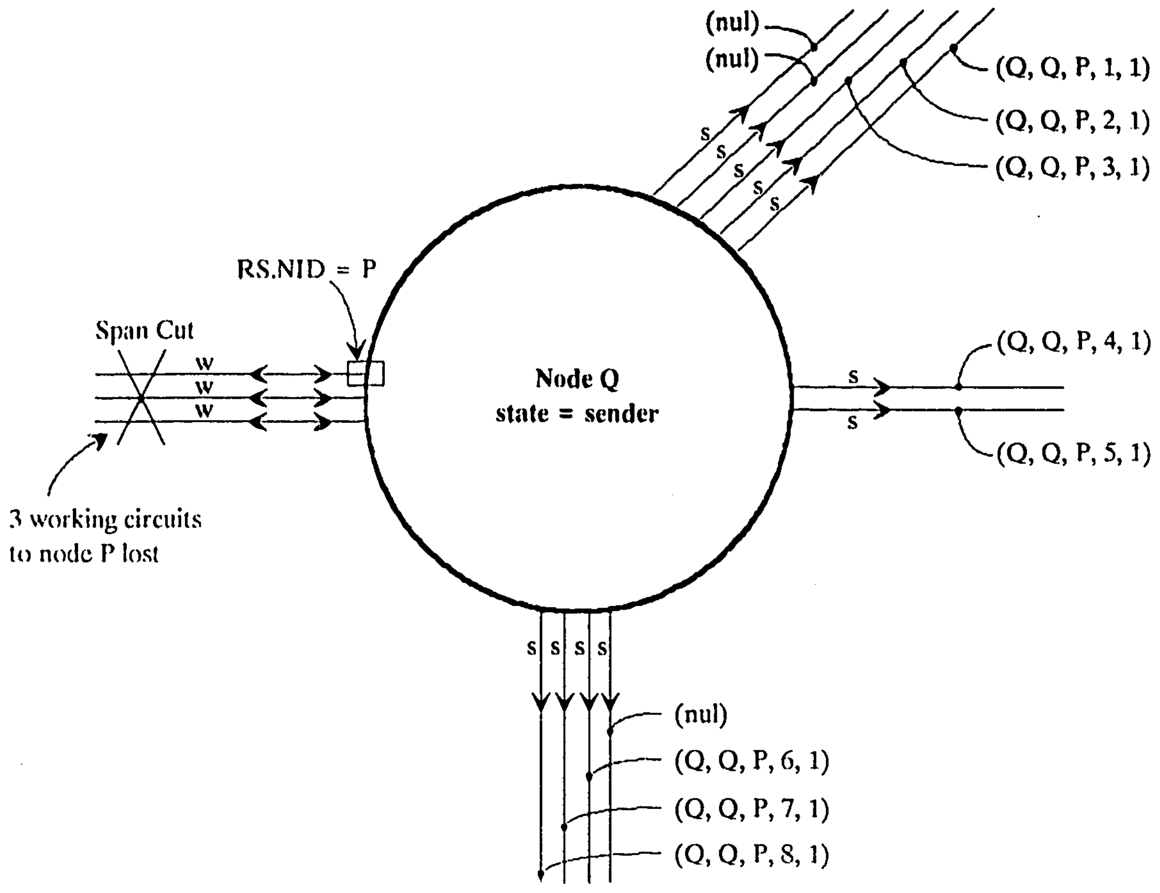
```
Sender-Flood;
begin
index-stamp := 1
for every span in [Spans] / alarm-span DO
        begin
        [Floodset] := first (min( lostccts, |span|)) ports in [span] | (TS @ port = nul and SPARE
= true)
        for every port in [Floodset] Do
                TS.index @ port = index-stamp
                TS.repeat @ port := IRV
                TS.source @ port := MyId
                TS.target @ port := alarm-span
                TS.mode @ port := 0
                index-stamp := index-stamp + 1
        end
end
```

The resulting pattern of TS origination for the example (P-Q) is shown in Fig.6.3.

6.2.3 Chooser Reverse Linking Origination

Let us temporarily skip over the actions of the Tandem nodes and deal with the simpler actions of the Chooser node when, as we know from the network-level discussion, forward signatures eventually arrive at the Chooser. Let *int-port* be the address of the port on the Chooser DCS which receives some forward signature with an index that has not yet appeared on any other signature that has arrived at the Chooser. Fig.6.4 shows the basic manner in which the Chooser responds to a valid forward signature in this case, assuming (a) *restccts* is still less than

Sender Node Selective Flooding Pattern



w	= working
s	= spare
TS	= (NID, source, target, index, repeat)

Fig. 6.3 *Sender* selective signature flooding pattern.

Chooser Node Responds to a Forward Signature

- **restccts < lostccts**
- **r = repeat ≤ maxrepeat**
- **index 3 not already responded to**

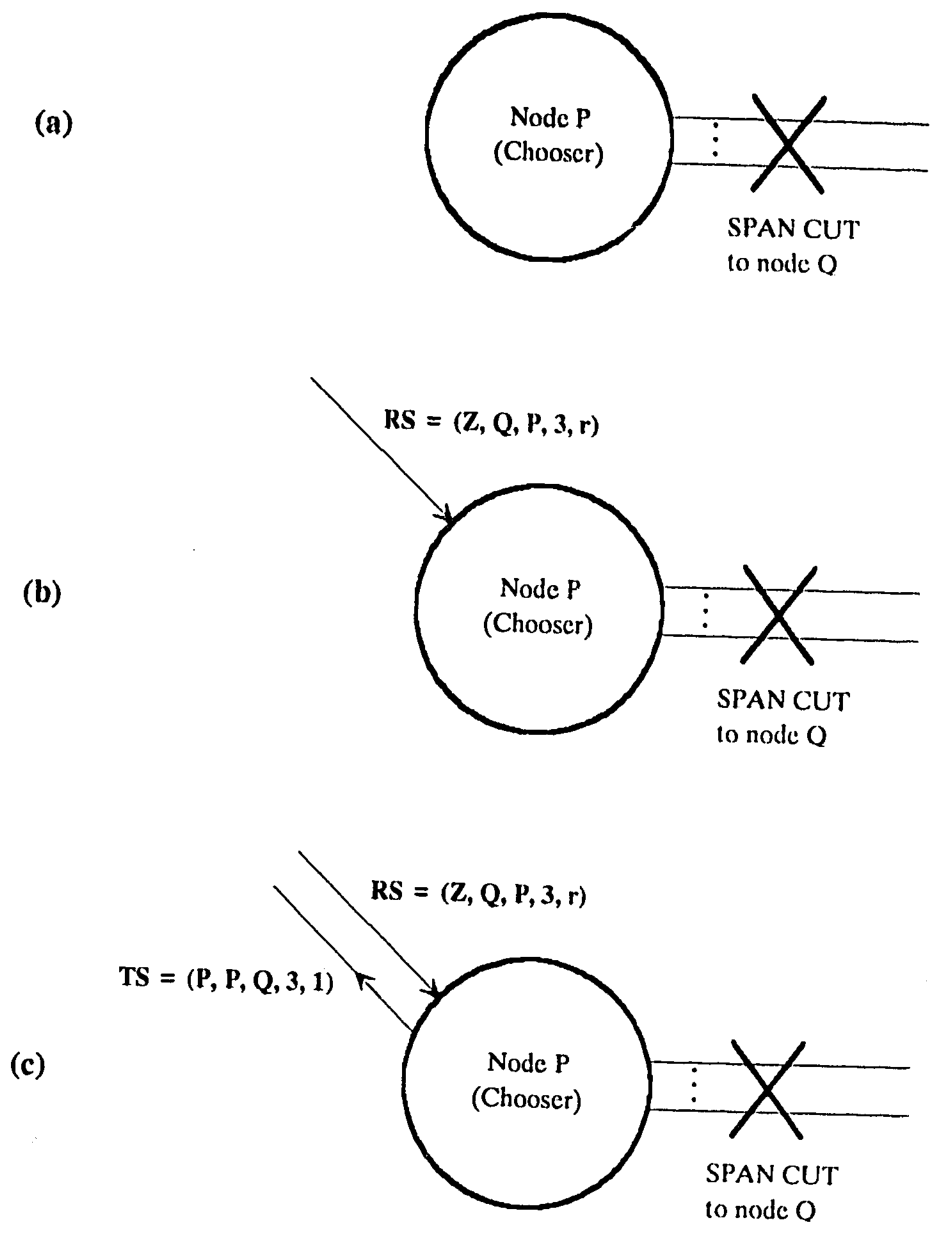


Fig. 6.4 *Chooser* response to a new forward flooding signature.

lostccts, (b) RS.SOURCE = P, RS.TARGET = Q (ie, class(RS.int-port) = current-class) and (c) RS.REPEAT <= repeatlimit. When such a signature arrives at the Chooser, it responds by originating a *complementary* TS on the transmit side of *int- port*. The Chooser creates a complement pair in the *int-port* by writing a TS which satisfies the previously defined *complement* condition and obeys the same NID and REPEAT field rules used by the Sender when it initiates a signature. The set [indices-chosen] is initially nulled in the preceding transition from *Normal* to *Chooser*.

```
New-Forward-Sig: {executed at the Chooser node}
begin
if (restccts<lostccts) then
        begin
        IF NOT ((RS.INDEX @ int-port) IN [indices-choosen])then
                begin
                [indices-chosen] := [indices-chosen] + RS.INDEX @ int-port
                TS.TARGET @ int-port := alarm-node
                TS.SOURCE @ int-port := MyId
                TS.REPEAT @ int-port := IRV
                TS.INDEX  @ int-port := RS.INDEX @ int-port
                PSR.SIE @ int-port := true
                PSR.SCIE @ int-port := false
                restccts:=restccts+1;
                end
end;
```

In the (P-Q) example, the Chooser writes the following to the TS registers in port *int-port*, as shown in Fig. 6.4:

```
TS.INDEX @ int-port = RS.INDEX @ int-port          TS.NID = Q
TS.SOURCE = P                                       TS.REPEAT = 1
TS.TARGET = Q
```

The Chooser will not subsequently respond *to any other signatures* having an index that has already been responded to. This is the purpose of the set [indices-chosen]. The lines *PSR.SIE @ int-port := true* and *PSR.SCIE @ int-port := false* have the effect of disabling any further normal RS signature interrupts from this port and enabling a subsequent interrupt for when SIGID later arrives. This is done because the Chooser expects a live-traffic signal to be applied by the Sender as soon as reverse linking is complete. Setting the SIE interrupt mask ON ensures that as soon as the live signal arrives, the SH task will be reinvoked on this port for final SIGID reception, mapping and connection to the appropriate *affected-port*.

6.2.4 Sender Recognition of Reverse Linking

Again deferring treatment of the action of the Tandem nodes, let us skip back to the Sender node and look at the basic action of the Sender when the reverse linking process, triggered by the Chooser, has its effect all the way back to the Sender. Fig.6.5(a) shows the Sender node after flooding in the example (same as Fig.6.3). Fig.6.5(b) shows a receive signature which eventually returns as a result of reverse linking. The reverse linking signature will appear in the

Sender Node Recognizes a Reverse-Linking Signature

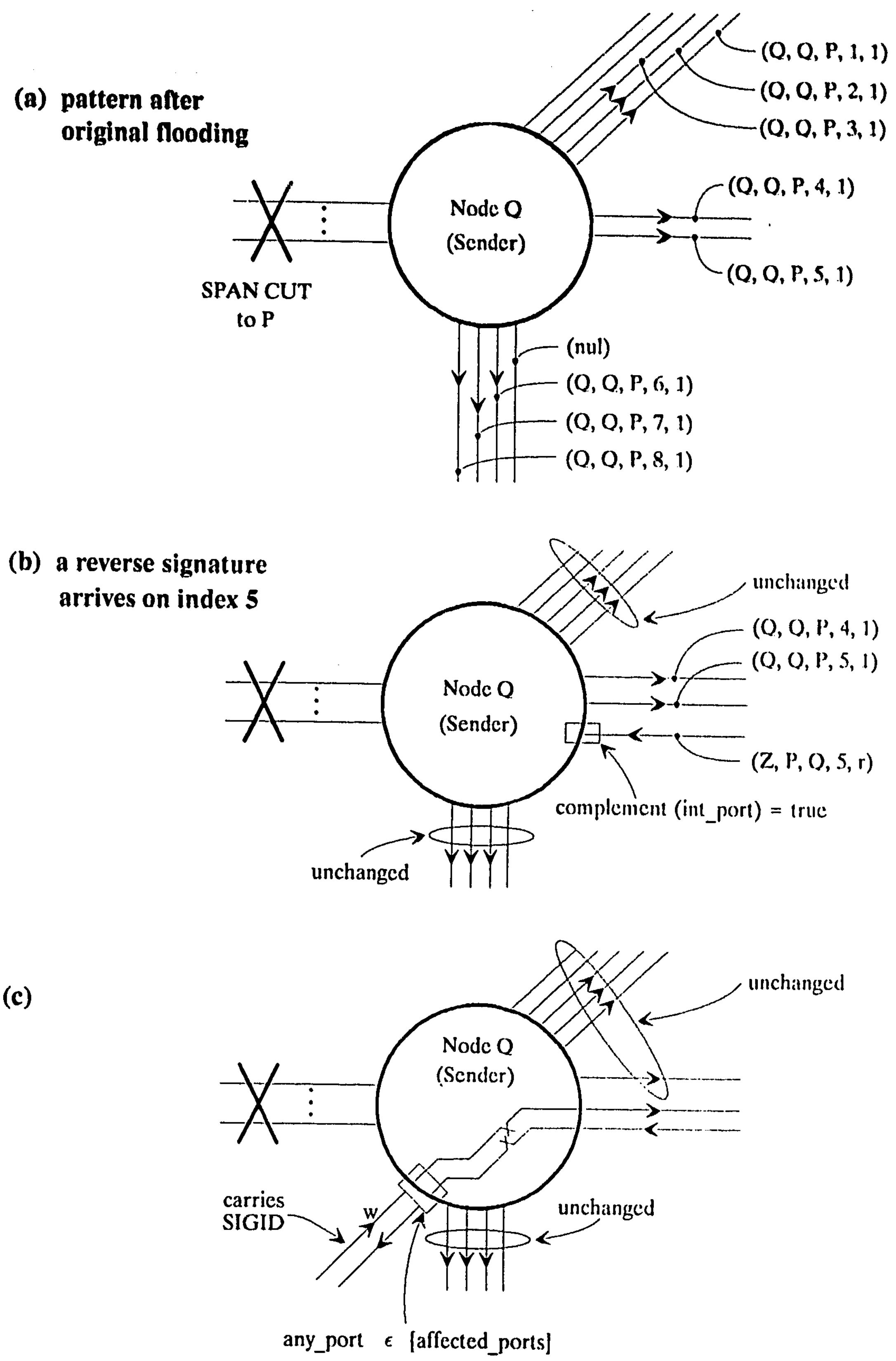


Fig. 6.5 *Sender* recognizes a reverse-linking signature.

RS register of the Sender port which has the complement SOURCE, TARGET pair and the same INDEX as the TS originally placed in that port during Sender forward flooding. We again denote the port receiving the current RS signature as *int-port* on the local DCS.

As soon as the SH task verifies that the newly arrived RS in *int-port* forms a complement signature pair with the existing TS in *int-port*, the Sender recognizes this as success in finding one individual path rerouting to the Chooser and the Sender and immediately operates a bidirectional crosspoint in its matrix to connect *int-port* to one of the *affected-ports* at its site as shown in Fig. 6.5(c). The Sender may at this point easily effect a traffic priority scheme in its order of choosing ports for restoration from the set of *affected-ports*.

In summary, the basic actions taken by the Sender node when a reverse linking signature arrives and (*restccts* < *lostccts*) is to (a) pick an affected port, (b) substitute the found path into the damaged path by operating a crosspoint to the affected port and (c) forward the SIGID of the path selected to the Chooser for its use in correct final traffic mapping (this step can be implicit in forwarding the signal itself if a network-wide SIGID plan applies), and (d) increment restccts. The Sender ignores any other signatures that arrive in its ports but do not create a *complement* signature condition in that port.

```
return-sig:
begin
if complement (int-port) then
        begin
        restccts:=restccts+1;
        luckyport := next [Affected-ports]
        Xpoint (luckyport,Int-Port);
        [Affected-ports] := [Affected-ports] - [luckyport]
        if (lostccts = restccts) then
                begin
                for every span in [Spans]/alarm-span] do
                for every port in span | (TS @ port < > nul and not complement(port)) nullout
(TS @ port)
                end
        end
end
```

The operator *next []* is used to choose an item of a set. (If a set is ordered, a corresponding *reset[ ]* operator is used when needed to reinitialize the sequence of returns from *next[ ]* on a given set. The underlying linked list implementation of these set utilities is not hard to extrapolate.) If no traffic priority scheme applies, the the next candidate for restoration is chosen from *affected-ports* arbitrarily. *Xpoint (port1,port2)* is a function (or request to the host O/S) that operates a bidirectional crosspoint in the DCS matrix between the specified pair of ports. *Nullout (port)* is a function which restores the TS register of the specified port to its non-active background state. This is used above to cancel all outstanding Sender flooding if *lostccts* = *restccts* is achieved. Not shown is a separate Sender timeout which interrupts the Sender after some predetermined maximum time for Selfhealing, after which all non-complemented TS originations from the Sender are arbitrally cancelled.

If a global network SIGID scheme does not apply, the Sender will also apply a mode 1 signature to the live signal that it substitutes into the restoration path to assist the Chooser in final mapping. This is achieved by adding:

```
TS.SIGID @ int-port := TS.SIGID @ luckyport
TS.MODE @ int-port := 1
```

where TS.SIGID @ luckyport is a *local signal identifier* that was previously assigned by the DCS to each working port at its site during normal operational times. This serves for final signal mapping at the Chooser because one of the *affected-ports* at the Chooser will have a matching identity in its PSR.SIGID register that was obtained in normal working times and frozen in the port hardware at alarm onset.

6.2.5 Final Signal Mapping at the Chooser

We will jump a final time across to the Chooser, to complete the Sender-Chooser behavior before considering the Tandem node actions. The Chooser knows as soon as it applies a reverse linking signature that a route will result and live rerouted traffic will soon appear. If only one signal path were interrupted by the span fault, the Chooser could go ahead and connect to the one *affected-port* and just wait for the Sender to forward the signal. But when multiple paths are interrupted, the Chooser does not know which signal entity will be applied by the Sender to the path on which the Chooser just initiated reverse-linking. The Chooser therefore has to wait for SIGID information to know which final connections to make.

At the Chooser a common Chooser mapping rule suffices whether network-wide SIGIDs are used, or the above local-node SIGID scheme is adopted. The following exploits the fact that just before the failure, the Assoc-port registers of the alarmed-ports would have pointed to the ports to which they were connected across the DCS matrix while in service:

```
Chooser traffic substitution
{entry is from SIGID interrupt on int-port}
matchport := port in [alarmed-ports] | PSR.SIGID @ port = PSR.SIGID @ int-port
Xpoint ( ASSOC-PORT @ matchport, int-port )
```

An example of final Chooser signal mapping is shown in Fig. 6.6 (a)-(c).

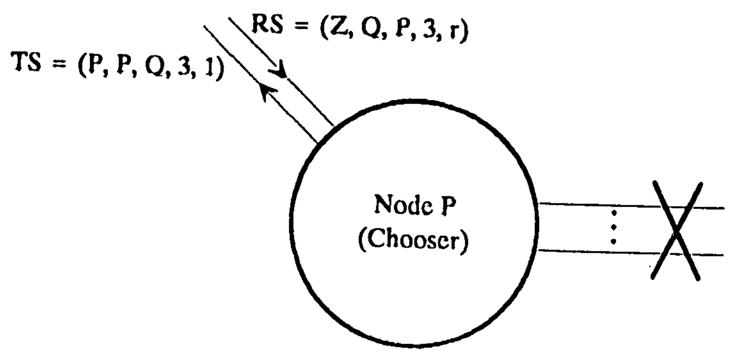
6.3 Tandem Node Signature Processing Rules

The Tandem node is where the real workload and art of the SH protocol is found. As can be appreciated from the network overview of operation in Sec. 6.1.3, all of the competing index mesh expansion and collapse dynamics are mediated by the Tandem nodes. The Tandem node rules have three effects simultaneously :

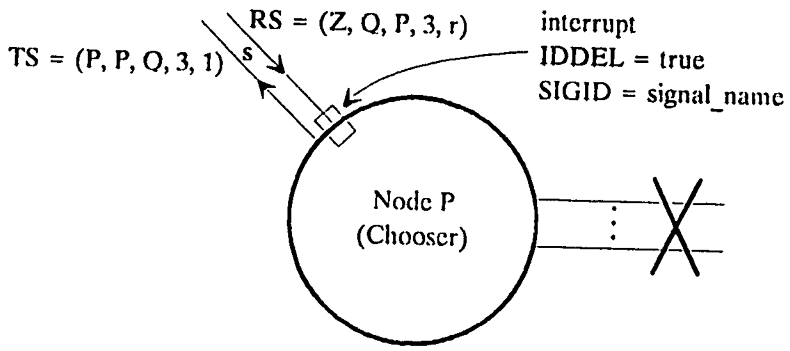
(a) application of signature rebroadcast rules for expansion of unsatisfied index meshes,

Chooser Node Receives Live Traffic Signal With Signal Identifier

(a) pattern after responding on index 3



(b)



(c)

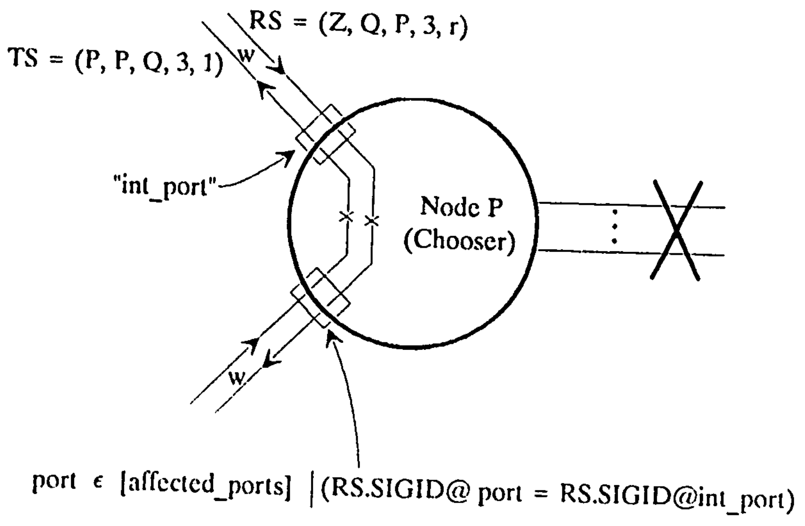


Fig. 6.6 *Chooser* receives rerouted traffic and restores one working signal with mapping directed by signal identifier.

(b) propagating reverse linking along the required axis and precipitating network-wide collapse of index meshes on which reverse linking occurs and,
(c) optimally reallocating links freed by collapse of one index into the expanding meshes of remaining indices.

6.3.1 Tandem Node Activation

A node enters the Tandem state when the SH task is invoked in response to a signature arrival interrupt (RSDEL) on a *spare link* , while there are *no alarms* at the node itself. The Tandem node first builds and initializes a set of span sets as did Sender and Chooser. A *current-class* couplet is then constructed from the RS.TARGET and RS.SOURCE of the signature at the interrupting port and this SH task instance will not thereafter react to any signature not satisfying *class*(RS @ int-port) = *current-class*. In the example, the current-class will be (P,Q) so only signatures having SOURCE=P, TARGET =Q *or vice-versa* will be processed by this instance of the SH-task until its state returns once again to Normal and is reinvoked for another span restoration. After these initialization tasks the Tandem node's first action will be to provide selective rebroadcast services to the RS that caused the interrupt. We will begin our description of a series isolated signature processing examples by explaining the *basic signature rebroadcast*. Each of the following signature processing cases is framed in terms of how a Tandem node reacts to an individual new RS arriving in one of its ports or reacts to an RS disappearance from a port. In each case we assume RS.REPEAT @ *int-port* < *repeat-limit* and the *current-class* tests are satisfied and we denote the port where the signature change interrupt occurred as *int-port*.

6.3.2 Basic Tandem Node Selective Rebroadcast

The basic signature rebroadcast pattern applies when a new RS signature arrives at a Tandem node and there are no other RS signatures at the node with the same index. This obviously applies when the first activating signature arrives and provides an opportunity to show the basic rebroadcast rules which are applied to any *repeatable* signature. A *repeatable signature* is the general term for a new RS that meets all requirements within the current context of the node for selection as a rebroadcast precursor. Other signatures with valid class, repeat and index fields may arrive and be *unrepeatable* (to follow). The basic rule for a *repeatable* signature is however to rebroadcast *one copy of the signature into each span on which that index is not already being sent, with the exception of the span on which the signature arrived*. Where this pattern results in application of a new TS from the node, the *port* where the repeatable signature is found is registered as the *precursor* for the new TS signature(s) that were created and every new TS has the REPEAT field incremented with respect to its precursor.

For a *simple* signature rebroadcast from *int-port*, there must be no other transmit signatures from this node with TS.INDEX = RS.INDEX @ int-port. The Tandem node first checks all spans for this condition, with a boolean function *already-sending (port)*, which returns true if there is any TS leaving the node with the same index as the RS at *port*. If there is no such signature already being emitted from the node, a simple broadcast pattern can follow as shown in Fig. 6.7. The basic rebroadcast pattern is INDEX oriented and seeks, where spares are available, to place one TS of the given index onto every span except the span in which the precursor lies. The following procedure effects a simple broadcast pattern and updates the ASSOC-PORT register of transmitting ports to point back to *int-port* recognizing it as the *precursor* for each of the TSs in the broadcast.

Simple broadcast (int-port)

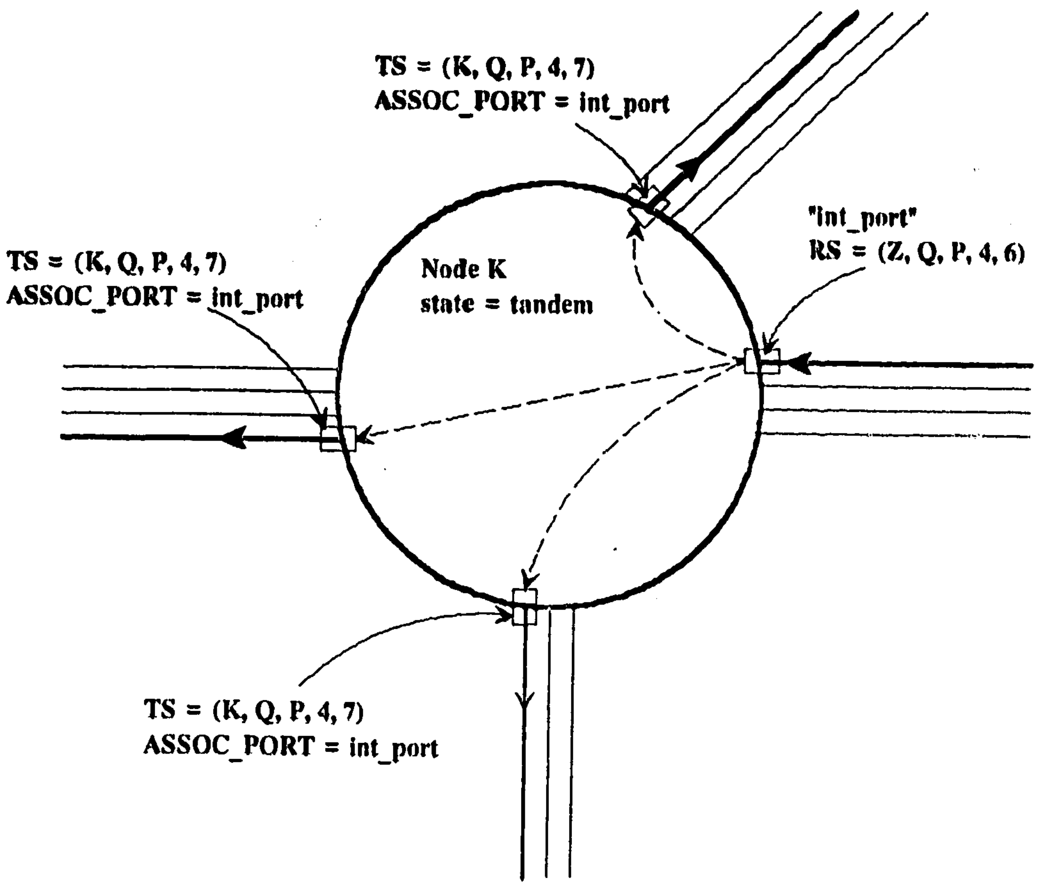
```
if Not (already-sending(RS.INDEX @ int-port) then
begin
        for every span in [spans] \ RS.NID @ int-port do
        begin
        if there exists port in span | ((TS @ port = nul) and SPARE @ port ) then
                begin
                portno := next [emp-set @ span]
                TS.INDEX @ portno := RS.INDEX @ int-port
                TS.REPEAT @ portno := RS.REPEAT @ int-port + 1
                ASSOC-PORT @ portno := int-port
                [emp-set@span] := [emp-set@span] - [portno]
                [send-set @ span] := [send-set@span] + [portno]
                end
        end
end
```

[spans] is the previously built set of spans at this node, identified by the NID of the adjacent node. [emp-set @ span], [send-set @ span] and a third set, not used above, [rec-set @ span] are the three working sets built for each span at this node upon first invocation from the *Normal* state. These respectively contain the numbers within the given span of all ports *not having an active TS*, all ports *having an active TS*, and all ports having *an active RS*.

6.3.3 Tandem Node Recognition of a Better Precursor

As the restoration event proceeds and the volume of signatures impinging on a given Tandem node increases, it will often occur that a new signature arrives in a port and it is found that TSs having the same INDEX as the new signature are already being sent from that node. In this case, a simple rebroadcast is not adequate and the issue is: 'which of the present RSs with the same INDEX, including the newly arrived RS, is the *best precursor* for transmit signatures on that INDEX ?' It can happen that after a rebroadcast justified by an RS at some port, another signature appears on some other port (or the same port), having the same INDEX but a lower REPEAT value. This makes the newcomer a *better precursor* for broadcasts on that INDEX.

Tandem Node Selective Rebroadcast Pattern



* Z = an arbitrary node
* Only spare circuits are shown
* Same pattern is attempted for every RS that qualifies as a precursor for its index
* TS, RS = (NID, source, target, index, repeat)
* int_port is the precursor for index 4 at node k

Fig. 6.7 Basic *Tandem* node selective signature rebroadcast pattern.

In this case the task is to identify the better precursor as such and to adjust and re-associate all TSs on that index with their new precursor and re-establish a new basic broadcast pattern, possibly rooted in a different span than the previous precursor.

A signature is a *better precursor* than all others on the same index at a Tandem node if (a) it is the receive signature having the lowest repeat field of all RSs on that index at the node, OR, (b) it is already part of a *complement signature pair* in the given port, (regardless of repeat value). In cases of equal lowest repeat values for two or more RSs which include the current precursor, the current precursor will continue in that role, otherwise the new precursor will be chosen at random amongst the equal minimum-valued candidates.

In the example of Fig. 6.8(a), three TS's are associated (by the dashed lines) with an RS having INDEX *i*, and REPEAT r. In Fig. 6.8(b), an otherwise equivalent signature arrives at a port in a different span and has a lower REPEAT field. In Fig. 6.8(c), the Tandem node reacts by altering the ASSOC-PORT registers of the relevant transmit signatures to reflect a new association with the better precursor at the new port. In addition, the TS.REPEAT fields are adjusted to reflect association to a precursor signature with lower REPEAT field. Note that the shifted base for the revised basic broadcast pattern means that : (a) the span in which the old precursor lay may now receive a TS on the given index, whereas before the change it did not. (b) the span in which the new precursor lies now may not have a TS on that index from the new precursor in the same span.

The same shift of precursor base would occur in principle if the better precursor was in the same span as the current precursor but this in fact does not arise because application of these same rules at the adjacent nodes means they will only send one instance of each index onto each span. No crosspoints have been operated as yet in Fig. 6.8. The precursor association between TS and RS signatures at the Tandem node is only a logical association.

Note that (b) implies that a new spare port may have become free for use in the broadcast pattern of another index after the precursor for the first index was moved. Indeed this fact manifests itself in the full implementation of the Tandem node protocol as a succession of applications of the basic Tandem logic rules, revisiting all other ports which have an active RS present and re-executing the basic tandem logic procedure until no new empty port results anywhere. We will revisit this point in closing the chapter, but in the particular example of Fig. 6.8, such reapplication of the Tandem node rules would be triggered by the suspension of the (*i*, r+1) TS in span T which was caused by the precursor shift from span M to span T for the broadcast pattern on index *i* that is shown in this figure.

Depending on the relative repeat counts of RS's on other indices present at the node when the new empty port arose in span T, the new empty port could be reincorporated into the broadcast pattern of another index before the present Tandem state invocation is complete.

Tandem Node Recognizing a Superior Precursor Signature

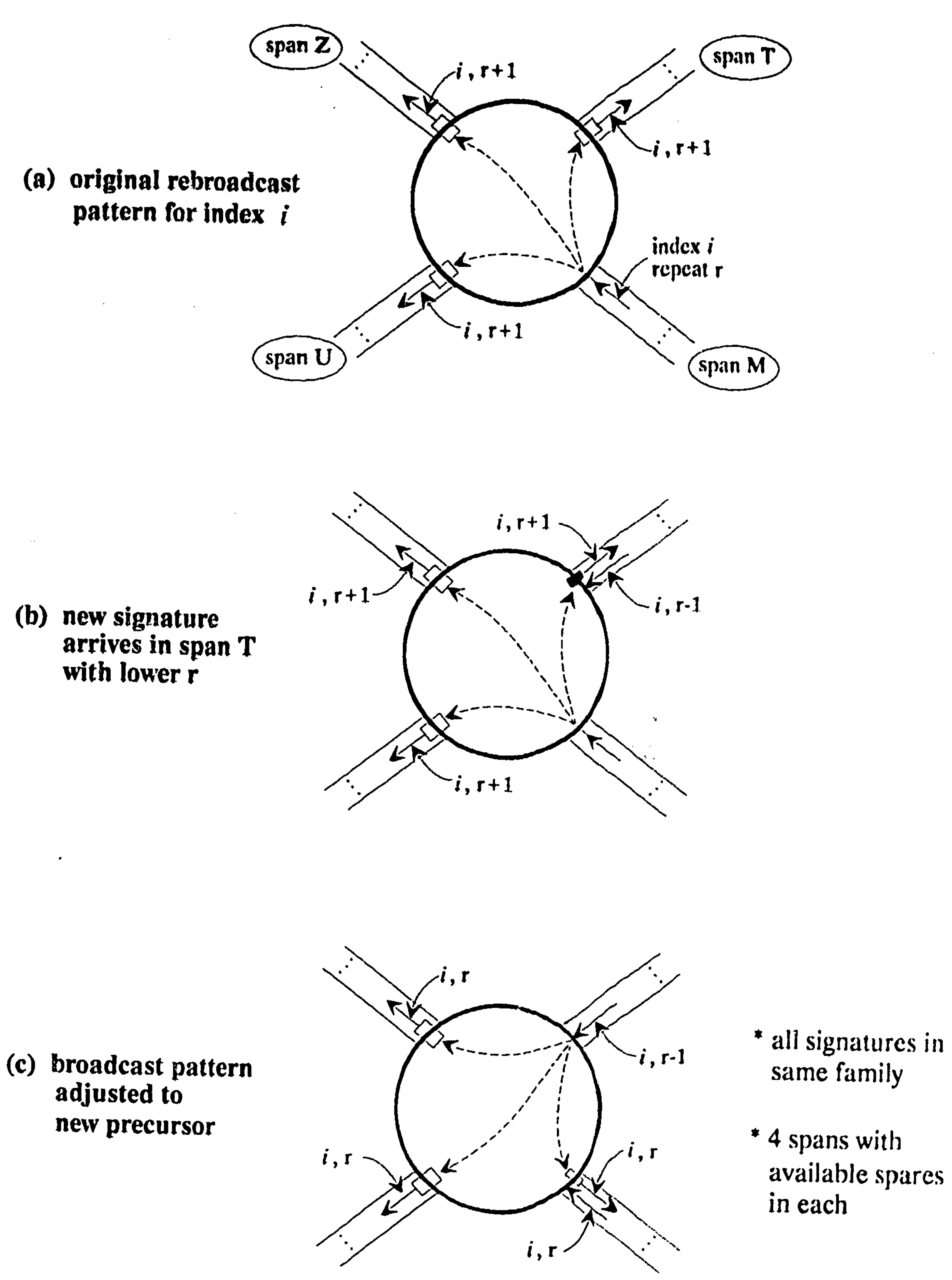

(a) original rebroadcast pattern for index *i*
span Z
span T
span U
span M
i, r+1
i, r+1
i, r+1
index *i* repeat r
(b) new signature arrives in span T with lower r
i, r+1
i, r+1
i, r-1
i, r+1
(c) broadcast pattern adjusted to new precursor
i, r
i, r-1
i, r
i, r
i, r
* all signatures in same family
* 4 spans with available spares in each

Fig. 6.8 *Tandem* node recognition and response to a superior precursor on a given index.

In fact, if the next lowest repeatcount is found to be with the old precursor for index *i* (in span M), then the free TS in span T would be used to rebroadcast the (*i*, *r*) signature (the old precursor) down the single span that remains available to that index. The protocol can accordingly support up to at most two fragments of mesh on any one index: one portion is the best precursor for TS's on that index at the node (as in Fig. 6.8(c)); the other portion is the special case of a single extension on the same index from an RS in some span other than the best precursor span, into the span of the best precursor. This can only occur if the one index involved has both the lowest and second lowest repeat values amongst all RSs at the node. This is a rather special set of circumstances however and in the majority of cases the family signatures on one index at a node will always be rooted on the best (lowest repeat) precursor for that index, in the pattern of Fig. 6.8(c).

It can also occur that without other parameters changing, the REPEAT field of an existing RS simply drops to a lower value (ie. as a result of a shift to a better precursor on that index at another node). In this case the signature change still raises an RS interrupt and thereafter the Tandem logic treats the interrupting RS as a new arrival and applies the above rules, possibly resulting in a precursor shift just as if a better precursor appeared in a port not presently in that INDEX family. An illustrative procedural description for the basic steps of recognizing a better precursor and shifting rebroadcast to that new precursor, where it is assumed that no complement signature pair already exists on the index of the new signature that arrives, is as follows: (These are the conditions corresponding to Fig. 6.8)

Better Precursor (int-port)

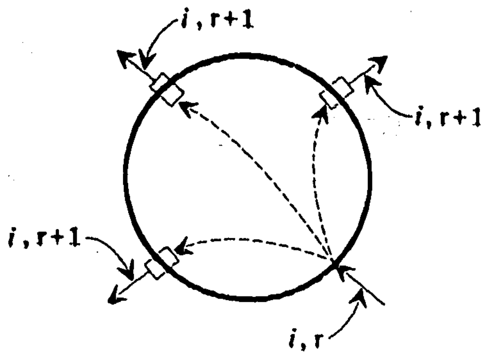
```
/* A definition of the procedure CancelTxSigs(int-port) (used here), follows in Section 6.3.5. The
procedure Simple-broadcast was defined in Section 6.3.2 */
for every span in [spans] \ RS.NID @ int-port do
begin
if there exists port in span | ((RS.INDEX @ port = RS.INDEX @ int-port) and (RS.REPEAT @ port
< RS.REPEAT @ int-port)) then begin
        CancelTxSigs(int-port)
        Simple-Broadcast(port)
end
```

6.3.4 Tandem Node Recognizing a Signature Complement

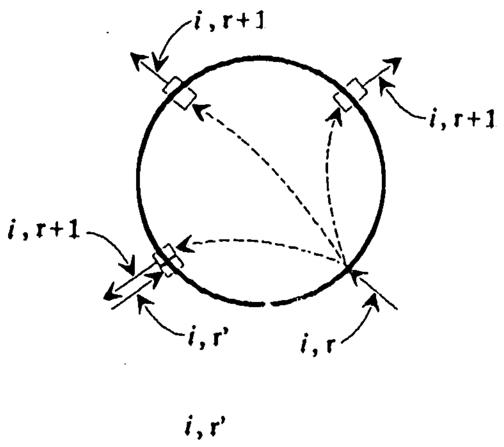
As a result of Chooser node initiation of a reverse linking signature, some Tandem node which is adjacent to the Chooser node will be the first to see and recognize a *complement signature pair*. In treating the actions of a Tandem node in this situation it will become apparent that the complement signature condition is passed on to other Tandem nodes who are *not adjacent* to the Chooser and the same behavior described here applies in such cases as well. In Fig. 6.9(a), a basic Tandem node rebroadcasting pattern is shown on a given INDEX for a node with four spans.

Tandem Node Recognizing a Signature Complement

(a) an initial broadcast pattern on index *i*



(b) reverse-linking signature arrives on index *i*



(c) reverse-linking sig operates crosspoint & follows precursor relationship. all other TS on index *i* suspended

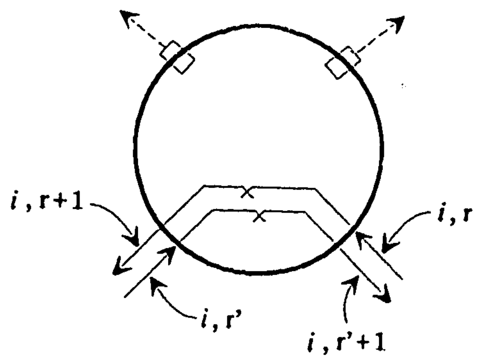


Fig. 6.9 *Tandem* node recognition and response to a signature complement condition on a given index.

The subject node will recognize a new signature as creating a complement signature pair in the port where the new signature arrives when *complement*(RS @int-port, TS @ int-port) = *true*. For illustrative purposes, this is assumed to be the case with the new signature arriving in Fig. 6.9(b). When the complement condition arises, the Tandem node takes the following actions which have the effect shown schematically in Fig. 6.9(c) in the example:

(a) A bidirectional crosspoint is operated between the port where the complement pair is found and the port where the *precursor* of the TS in the complement pair is found.

(b) All TSs from the above precursor are suspended with exception of the TS that is in the complement pair.

(c) The RS @ int-port is repeated into the TS register at the port where the precursor of int-port was found. The repeat field is incremented as usual.

Complement Signature

```
{entered with a complement pair in port int-port}
if complement(int-port) then begin
        Xpoint( int-port, ASSOC-PORT @ int-port)
        TS.INDEX @ (ASSOC-PORT @ int-port) := RS.INDEX @ int-port
        for every span in [spans]\RS.NID @ int-port do begin
                for every port in [send-set@span] | TS.INDEX @ port = RS.INDEX @ int-port do
                        begin
                        ASSOC-PORT @ port := nul
                        nullout( TS @ port)
                        [send-set@span] := [send-set@span] - [port]
                        [emp-set@span] := [emp-set@span] + [port]
                        end
        end
end
```

These actions prune off the unneeded branches of the broadcast pattern on the given index, extend the path of the complement signature in the one direction where it continues to produce a complement signature, and operates a crosspoint between the two uniquely identifiable ports involved.

6.3.5 Disappearance of a Signature at a Tandem Node

Another signature event that can occur at a Tandem node is the disappearance of receive signatures due to simultaneous application of the above processing at adjacent tandem nodes of the network. Actions at adjacent nodes can also cause a change in an existing RS from signature to signature without an observable disappearance in between. The processing of such a change is internally handled as two events: first a logical disappearance of one signature followed by the appearance of a new signature. In both of these cases the following response applies to the physical or logical disappearance of an RS signature.

If the RS that vanished was not a precursor for any TS at the node, then the signature disappearance requires no action, because no TS broadcasts were caused by it. If the disappearing signature was a precursor, then all TS's rooted to the disappearing precursor are cancelled and the ASSOC-PORT registers of those ports are nulled. In the latter case the use of the released transmit links must be re-examined from an overall nodal viewpoint before exiting the Tandem node procedure. The first part of the processing is to take down any existing broadcast pattern which may have had *int-port* as its precursor:

Cancel Tx Sigs (int-port)

```
for every span in [spans]\RS.NID @ int-port do
        for every port in [send-set@span] | ASSOC-PORT@ port = int-port do
                begin
                ASSOC-PORT @ port := nul
                nullout( TS @ port)
                [send-set@span] := [send-set@span] - [port]
                [emp-set@span] := [emp-set@span] + [port]
                end
        end {for}
end {for}
```

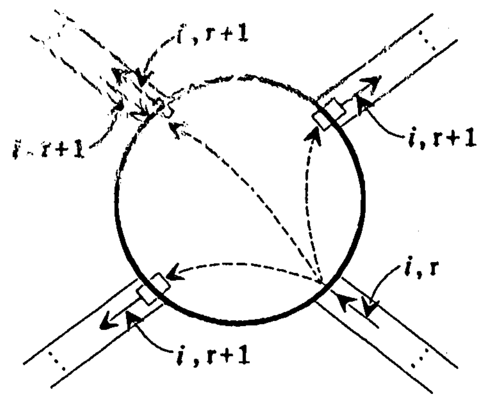
Fig. 6.10 (a) and (b) illustrates a simple case of the first stage resulting from a precursor signature disappearance or overwriting by a signature on a new index. Fig. 6.10 (c) goes on to illustrate a simple form of reallocation of the released ports to the next best precursor within the same index family. In general however, the free ports resulting from a signature disappearance will not necessarily go to a new precursor on the same index as the precursor which disappeared. If other RS indices exist that do not enjoy maximal rebroadcast but have a lower repeat count than any RS in the present index, the new free TS registers may be allocated to the broadcast pattern of one or more such other index families. The overall policy for allocation of free TS registers follows in the next section.

6.3.6 Tandem Node Reallocation of New Empty Port

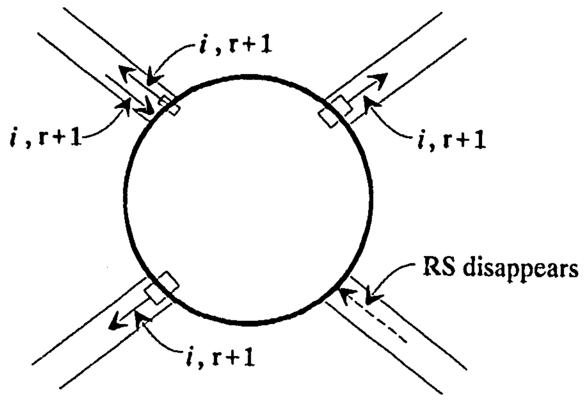
Obviously, cancelling the broadcast tree from a precursor which has disappeared creates available spare links. These may immediately be taken over by the signature broadcast trees for other indices at the node. The actions of reverse linking and precursor shift can also have the side-effect of freeing the TS registers of ports which were in use for some index before the current event at int-port. Such free TS registers are an important resource and their reallocation consumes much of the Tandem node processor time. For the protocol to work as intended, these ports must be immediately reapplied to the optimal expansion of remaining unsatisfied index meshes.

Tandem Node Recognizing Signature Disappearance

(a) initial broadcast pattern



(b) precursor for index *i* disappears



(c) previously inferior RS becomes precursor for index *i*

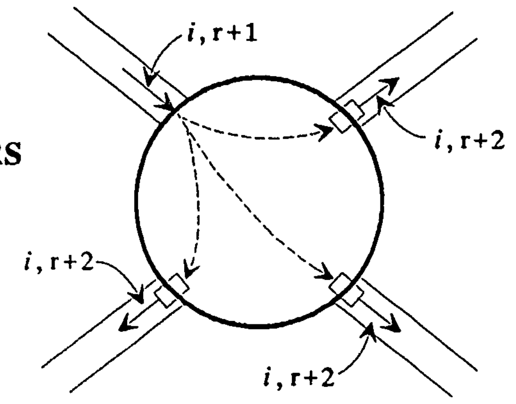


Fig. 6.10 *Tandem* node response to disappearance of a receive signature.

The basic rule for such reallocation of newly freed TS registers is to offer them for incorporation into the existing broadcast patterns or potential broadcast pattern at the node, with access to such free ports *prioritized by repeat value* and each access subject to all other Tandem node logic rules.

Basic reallocation of new-empty-port

```
{processing the interrupt at int-port resulted in a net release of one or more spare outgoing
links}
{avail-TS is a boolean function which tests if there exists any span | [emp-set@span < > nul }
begin
[dispatch] := nul
for rep-order := 1 to maxrepeat do
        for every span in [spans]; [dispatch] := [dispatch] + [rec-set @ span]
sort [dispatch] | RS.repeat @ port increases
reset [dispatch]
repeat
        psuedo-int-port := next[dispatch]
        Tandem logic (psuedo-int-port)
until next[dispatch] = nul or not (avail-TS)
end
```

The above has the effect of reallocating a newly freed TS register to signature rebroadcast for the index family which has the minimum repeat count of all indices present at the node which are not presently sending into the span where the new free port resides. An example of how three newly released TS registers would be allocated depending on the pattern of spans, indices and repeat values is shown in Fig.6.11. The procedure **Tandem logic** is the basic signature processing FSM used by the Tandem state of the protocol. Noticeably embedded in the above specification for re-allocation of a free TS register is the property that any time a free port results, the entire set of ports with active RSs at the node may have to be revisited, in order of increasing repeat field value, giving each the opportunity to reassert the Tandem node rules from its point of view as a psuedo-interrupt. This only ends when all such RS ports have been revisited or the test *avail-TS* indicates all outgoing ports are now allocated.

6.3.7 Combined Application of Tandem Node Rules

In the complete implementation of the Tandem node procedure, the Tandem node logic is applied in a two-stage method designed to ensure that all of the various by-products of applying the Tandem logic rules for one port are fully taken into account in terms of their influence on all other broadcast meshes simultaneously at the node. This is achieved in the following high-level manner with the use of a binary flags *new-empty-port* and *avail-TS* and the key procedure *Tandem-logic* which contains all the event parsing and FSM action blocks for the Tandem state:

Overall structure of tandem state:

```
{primary invocation in response to real signature event...}
newemptyport: = false
Tandem-logic(int-port, newemptyport, involved)
if newemptyport then
                {series of prioritized secondary psuedo-events}
                repeat
                psuedo-int-port := next[dispatch]
                Tandem logic (psuedo-int-port, newemptyport, involved)
                until next[dispatch] = nul or not (newemptyport)
if involved state := Tandem else state := Normal
```

The entire Tandem logic FSM is first used to react to the primary physical signature event and is also the most convenient way of reapplying all the previous rules from the viewpoint of the new pseudo-interrupt ports which will be given access to the new free port plus any other free ports that may have resulted from other secondary application of Tandem logic. Such reinvocation ensures that the original freed port will be incorporated into the broadcast pattern of the best precursor index that is not already fully rebroadcast and that upon exit all Tandem state rules are globally consistent and maximally applied.

Every processing block in the Tandem logic block is designed to set the flag *newemptyport* if it has a net effect of cancelling any TS. On every entry to the Tandem state the complete Tandem node logic is therefore first executed in context of the *primary interrupting event* and then, if needed, reiteratively applied in totality until *newemptyport* is false. Upon final exit the complete signature state of the node is then known to be globally consistent and maximally applied for each index at the node.

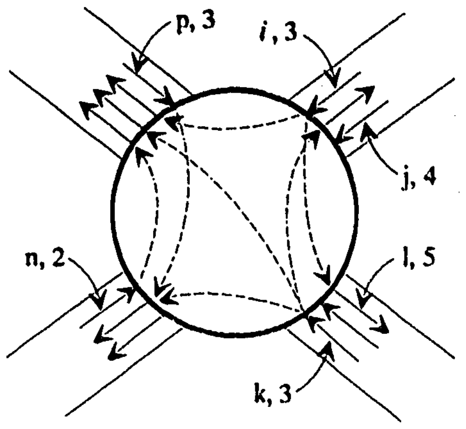
One can appreciate that the above rules, applied iteratively allow for extremely complex dynamic sequences. Each reinvocation reopens the range of possible processing events so that in principle a single RS event can catalyze drastic reconfigurations in the complete Tandem node signature state. And every such Tandem node is simultaneously interacting with direct neighbours and, indirectly thereby, with all other Tandem nodes which have the same potential for such drastic nonlinear dynamic behavior.

6.3.8 Tandem Node Recognition of Not Involved Condition

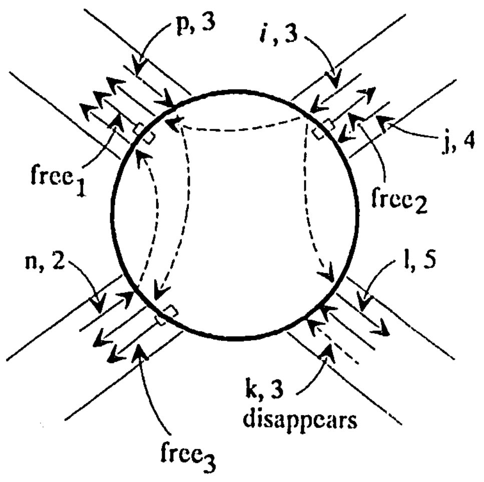
A final element of signature processing that a Tandem node must be able to do is to recognize when the situation arises that the given node is unable to participate further in the SH event. Once a Tandem node finds a complement pair in any port it knows that it is an active, crosspoint holding part of the ultimate restoration pattern. We call the set of such nodes the restoration team. In general there are always cases of nodes alerted into the Tandem state which do *not join* the restoration team.

Tandem Node Allocating New Free Transmit Link To Index Family With Lowest Repeat Precursor

(a) an initial pattern of precursor signatures and partially satisfied broadcast patterns



(b) index k collapses freeing 3 TS ports



(c) each free TS port reallocated to index family with minimum precursor repeat not already sending on that span

* free$_1$ given to index j
* free$_2$ given to index n
* free$_3$ given to index j

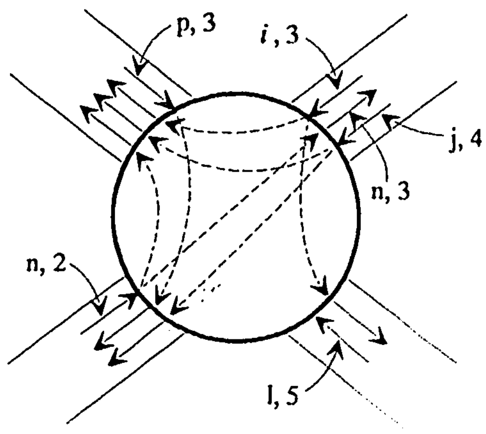


Fig. 6.11 *Tandem* node policy for allocation of new free transmit signature register amongst competing indexes present at its site.

On every invocation of the Tandem node protocol which is determined to have been caused by a signature disappearance, an internal flag called *involved* is computed. This flag is false as long as there is at least one spare port at the node with an active RS. If *involved* ever becomes true, the node does some necessary tidying up, releases its working span memories, sets *state := normal* and suspends itself, giving control back to the DCS O/S. This mechanism releases unneeded nodes from influence of the SH effect. The mechanism allows that a given node may in fact participate, drop out, and then participate again in a given SH event, and there is no objection to this.

6.4 Summary

This chapter has been devoted to the substance of how the Selfhealing protocol works. We looked first at its operative principle at the network level and then at details of the three part (Sender, Chooser, Tandem) event-driven protocol which causes the indicated network-level action purely through local signature-event processing. All operations within the node are expressed in terms of signature processing rules which are totally local to each node and involve no global network context whatsoever.

Having walked through the several independent signature-processing principles that are applied by the protocol, we can now refine the introductory FSM of Fig. 6.1. Fig. 6.12 is an exploded view of the Selfhealing FSM showing all significant processing blocks in our actual implementation. Whereas Fig. 6.1 indicated only the primary external events RS and alarm, the detailed FSM defines a wider set of higher level events which are determined with respect to context at the node when a physical RS event occurs. For further description of event parsing and processing at this level of detail, the reader is referred to Appendix B where the function and complete formal specification of each event and block in Fig. 6.12 is given. This chapter has, however, covered all of the principal concepts and methods required to appreciate the method and the experimentally obtained results which now follow.

Selfhealing Protocol Description

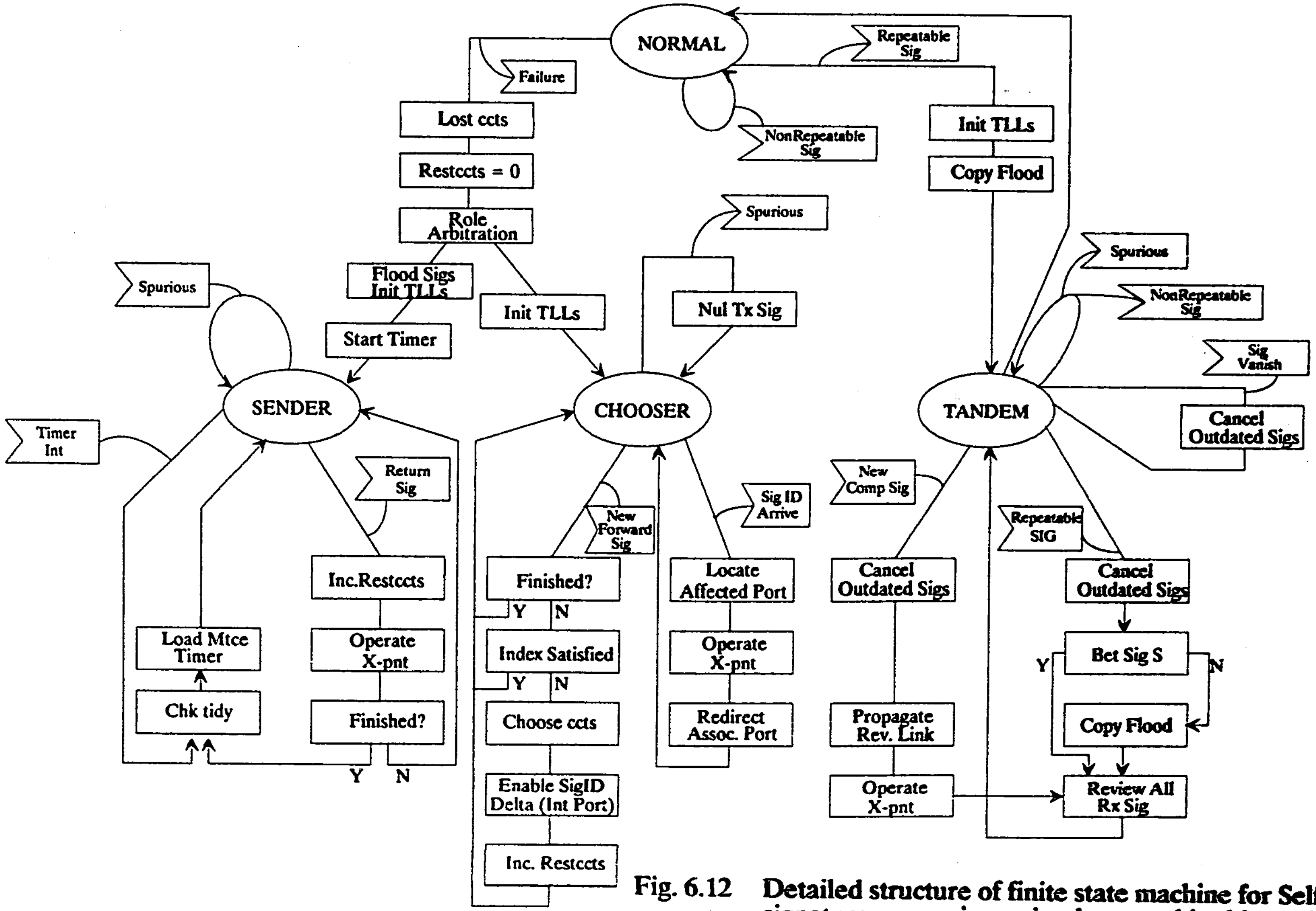


Fig. 6.12 Detailed structure of finite state machine for Selfhealing signature processing as implemented in this work.

Chapter 7
Research Methods And Implementation Techniques

In the remaining chapters we report the experimental implementation and characterization of the proposed Selfhealing method for real time network restoration. Ch. 4 identified the generic measures by which we would quantitatively assess the performance of any such restoration scheme and Ch.'s 5 and 6 gave the substance of the technique we have developed to address the problem. This chapter now describes a computer - experimental research method in which the proposed protocol was *implemented* (not simulated) for experimental characterization. All subsequent chapters report results obtained with this experimental implementation of the protocol. It will be seen how the method we now describe formally for evaluating the proposed solution, was in fact also the essential tool for synthesis of the desired protocol.

7.1 Research Method

The complexity of asynchronous transient event-driven interaction *amongst a number of nodes* in a Selfhealing network level is a system-level problem to which reductionist analysis is not helpful. Operating in a 3-dimensional sea of signatures each Selfhealing node is *indirectly influenced by all others* and *has an indirect influence on every other node*. By its nature, Selfhealing is a transient, non-linear, massively parallel, multi-dimensional process to which the usual analytical requirements of statistical equilibrium or one-dimensional contention do not apply. Consider for instance the well-known CSMA-CD protocol for which analytical results (throughput and delay) are obtainable. These results apply only under assumptions of statistical equilibrium for a protocol whose net effect can be approximated analytically by independent Poisson sources of offered load, contending on a one-dimensional medium of interaction where collisions interact in only one way (ie. destructively). By comparison, Selfhealing involves contention interactions between numerous index meshes in multi-way contention in a two dimensional medium. The Selfhealing protocol engenders many different types of *constructive* interactions between signatures and the complete state space of all node state combinations and link signature states is astronomically large and changes with time. For these reasons, an essentially experimental research method has been adopted as it is not known what analytical methods could produce the true functional verification and direct quantitative results that we have obtained and which are essential to the engineering acceptance of the method.

The main technique used in this work is computer emulation of the Selfhealing task running concurrently at every node of representative multi-node networks. In this approach the protocol itself is actually *implemented* and executed in every node of the various study networks using concurrent programming methods. The collective behavior of the system (the network plus distributed SH tasks embedded in it) is then observed. This approach not only yields highly realistic results but it was essential to advance from a conceptual principle to a working Selfhealing protocol in the first place. The equivalent large-scale experiments in real life would be prohibitively costly, time consuming, inflexible and limited in scope.

7.1.1 Research Tools and Environment

There are two completely separate and replaceable parts to the experimental system which was developed for research in this area: (a) *a protocol module*, (b) *a network emulator module*. A constant philosophy in the development of the two-part system comprised of emulator and protocol module was to make the environment that the network emulator provides to the protocol such that *the protocol cannot tell that it is not inside a real DCS machine in a real network*. The correct specification of the protocol module is an *output* of the research. By adhering to the above principle for structuring the interaction between emulator and protocol module we maximize confidence in the final protocol specification which is derived by this work.

The network emulator is a *tool* which converts one physical instance of the protocol implementation into many virtual (concurrent) instances of it and facilitates their distributed asynchronous interaction according to a defined network description. The emulator is analogous to an imaginary tool for DNA design which converts a proposed DNA structure into the resultant large-scale organism. The goal is to get the molecular design (the protocol) right, but the criteria for success are only measurable by observing the complete organism (the network). The combination of separate protocol module and emulator provides synthesis-analysis feedback loop, (Fig. 7.1), so that the experimenter can see what small changes in the low-level signature processing rules do to the large scale behavior of the network. Without mechanizing the research in this way it is doubtful that any progress would have been made. In networks of over three nodes and a half-dozen links, the signature interactions are so numerous and complex that it is prohibitively time consuming and error prone for a human to deal with them directly. In addition all "interesting" behaviors (ie. bugs) in the protocol were found by testing on far more complex networks with manual inspection only of those cases where something unusual happened and was trapped for study by automated analysis of the results.

7.1.2 The Protocol Module

Pure Code : In order to achieve the structure of a single protocol specification and an engine which executes the protocol as if it were simultaneously present in many nodes, the protocol procedure had to be written as *pure code*. This is a programming methodology in which a procedure can be used recursively or re-entrably as a shared resource simultaneously accessible by number of calling tasks or routines. The main principle is that a pure-code procedure is strictly an operator applied to data sets that belong to other procedures.

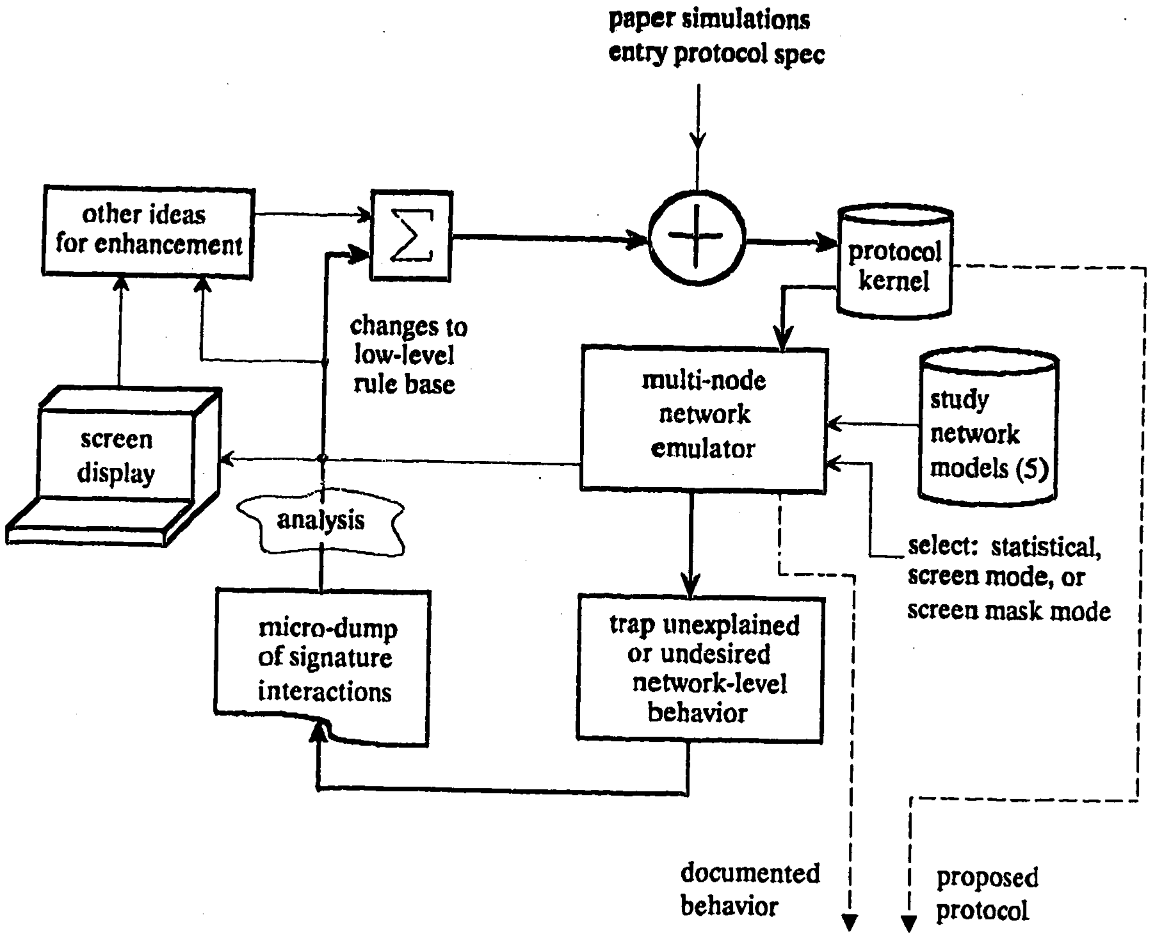

paper simulations
entry protocol spec
other ideas for enhancement
changes to low-level rule base
protocol kernel
multi-node network emulator
study network models (5)
screen display
analysis
select: statistical, screen mode, or screen mask mode
micro-dump of signature interactions
trap unexplained or undesired network-level behavior
documented behavior
proposed protocol

Figure 7.1 - Protocol Development & Research Environment

A pure-code procedure stores no variable data of its own and carries no state information of its own. It accommodates its own constants and internal state to that brought to it with the current data set or event on which it is to operate. With these methods it is possible in the network emulator for the one instance of the SH protocol to be executed first for one node, then halted and used to advance the processing for another node and so on. Recovery and continuation of the processing for the halted task is handled by restoring the respective internal state and switching to the data set that belongs to the new context. This software technique was used here to allow re-entrant re-use by the network emulator of a single selfhealing task procedure in the context of each node that exists in the network description file. It is the use of these methods that make it relatively easy to later extend Selfhealing to multiple simultaneous fault capability.

Finite State Machine Methods : Another important aspect of methodology is the use of Finite State Machine (FSM) techniques as the basic specification and encoding form for the SH protocol. The difference between an algorithm and a FSM encoding of procedural behavior is that FSM representations describe behavior in terms of actions in response to external events. An FSM has a corresponding state transition diagram (STD) whereas an algorithm is represented by a flowchart. FSM methods are often used in telecommunications software design because of their natural pacing of execution to external events with inherent task suspension when no event requires it. This is in contrast to self-paced algorithms which require wait-loops to synchronize their execution to events in the outside world. Moreover, an algorithm waiting for an external event may require knowledge of what external event is being awaited. The following nested case statements show the basic template that can be used to describe any FSM. The Selfhealing protocol uses this internal structural framework extensively.

```
begin
enter with current state of the FSM
Case state of:
        State 1: determine which event occurred
                Case event of
                event 1:    Action block 1 ;
                            assign new next state
                event 2:    Action block 2 ;
                            assign new next state
                .
                .
                .
                event N1:   Action block N ;
                            assign new next state
                end {this subcase}

        State 2: determine event
                Case event of
                event 1:    Action block 1 ;
                            assign new next state
                event 2:    Action block 2 ;
```

```
                    assign new next state
            .
            .
            .
            event N2:  Action block N ;
                       assign new next state
            end {this subcase}
        .
        .
        State N:  {etc.}
        .
    end {outer case }
```

Several benefits arise from the use of FSM methods in this work: (1) A simple structure is used repeatedly in representing all major portions of the Selfhealing protocol. The basic structure facilitates disciplined management of a protocol which is complex when taken overall. It allows very structured thinking on problems related to the protocol in localized terms such as " if the *current state* is (x), and *event* (y) occurs, then what is the *action* to take?". Each *action block* is itself a conventional algorithmic segment designed to realize the intended action and varies in practice from 2 lines to half a page of code depending on the complexity of the atomic action required. The isolation from the overall problem complexity obtained by this structuring is helpful and desirable as a discipline in the development of a software system for such massively parallel interactions as Selfhealing represents. (2) Considerable confidence is obtained about the practicality of Selfhealing in real crossconnect machines because the code is written exactly as it would be installed as an interrupt-driven or time-shared task in the operating system of a real DCS machine in a real network. Furthermore the interface between the emulator and the protocol specification can be made exactly as it would appear to the SH task in a real DCS machine. (3) When the protocol is implemented this way *the network emulator actually executes the specification of the protocol*. This elegant property means that when the emulator indicates that the desired network-level behavior is obtained, one can literally print the Selfhealing protocol module to obtain a C language formal specification of the protocol. The concept of directly *executing the specification* of a software function is a recent and advanced objective in the field of software engineering [cfi ArSi86].

7.1.3 The Network Emulator

The network emulator is itself a small, specialized, form of time-sharing operating system. We emulate |N| nodes, executing Selfhealing concurrently in every node in a manner similar to the way a multi-tasking mainframe computer makes it appear to many users that they are simultaneously executing a given application program (an editor, say) on their own virtual computers. In reality there is only one computer and only one instance of the editor program. Every time a given user task is ready to execute, the editor is executed *in the current context of that user*. The research emulator works this way but the users are nodes in the defined study network, the 'job' they are all executing is the Selfhealing protocol, and much greater attention is paid to realtime considerations than a normal operating system.

The *context* of each node consists of: (1) the identifier of the node in whose role the SH task is presently to execute (*the current node*), (2) the identity of the port on the current node which raised the interrupt (*int-port*) that invoked execution of that node, (3) the current state of the virtual SH task at the current node (the *current state*) and, (4) the array of port records that belong to the current node. The *virtual* Selfhealing task in any given node cannot tell that it is not running on its own processor interacting directly with the network in which it is embedded. Any signature applied to a link by the SH task in one node is delivered after appropriate delays as an event in the respective port at the node that is connected to the receiving end of that link, according to the network description file. The *emulator* has access to the network description file, to convert what one DCS does to the transmit links at its site into future events occurring at other nodes, but the SH task(s) have no access to any network description whatsoever. All they see of the world is a set of RS signature and status registers into which new contents appear from the outside world and a corresponding set of TS registers to which the SH task may write signature contents. All physical actions of the port card hardware such as DS-3 F-bit modulation and physical propagation delay are handled by the network emulator. The SH task's myopic view of the world is the same in the emulator as it would be in a real DCS machine of the generic architecture previously defined (Fig. 2.1).

With the combination of methods used it is true to say that we are emulating the network but implementing the protocol. Like a pilot in a flight simulator, interactions with the outside world are a simulated interface but the object under test is real in all its details, not a replica or facsimile in any way. This is exactly the case in the present work: the protocol module is an actual implementation designed to run in a real DCS. The network emulator simulates the interface to the DCS host O/S and passes events in and out through that interface to and from the other nodes of the network, as well as applying appropriate propagation and execution time delays and applying the (unseen) topological properties of the network in which the protocol is embedded. We implement the interaction between host DCS O/S and the SH task just as we propose it for a real application: the host O/S treats the SH task as an interrupt handler for a special class of real-time interrupts (signature interrupts). This relationship is functionally no different than between the O/S and a printer I/O routine or a keyboard interrupt handler. This provides a clean and realistic interface model between the emulator and the virtual SH task at each node and is directly translatable to a DCS O/S. In this approach every new signature received and every signature change (detected by the proposed port hardware) causes an interrupt which invokes the SH task. The processor is always returned to the O/S immediately when there is no signature processing load, even temporarily in the middle of a restoration event. This implementation property is preferred in telecom designs over software task designs which hold the processor. The principles of this interrupt-driven paradigm apply almost identically if a task scheduler acts as an intermediary between the port cards and the SH task, although speed probably suffers somewhat. In this variant, messages to the O/S from the port cards would replace the vectored interrupt structure. The SH task is normally suspended (asleep) but becomes high priority in scheduler as a result of any signature message from a port card.

The combined pure-code/FSM/interrupt-handler paradigm for implementation of Selfhealing greatly reduces the total complexity of the protocol design problem. While each individual signature processing event may be manageably simple, the overall *effect* of just a few successive invocations of the protocol can easily be more complex than could realistically be managed by normal single-algorithm software design methods.

7.2 Network Emulator Description

Two versions of a network emulator were implemented. The first was a prototype to verify feasibility and basic soundness of the key concepts of this work. It uses a method of synchronously stepped concurrent execution which amounts to execution of Selfhealing in a *parallel* computer environment. The second emulator is a continuous-time asynchronous concurrent emulator which effects a *distributed* computational environment for Selfhealing (as described in Ch. 3). The latter emulator is the most realistic and is used for the best predictions of exact routings and speed but it is available only on the SUN workstation. The stepped emulator has upper-bound real-time speed estimating characteristics and predicts the same number of paths as the SUN emulator but there are small differences in routing due to the additional realism and parallelism of the SUN emulator. The stepped emulator has the advantage of allowing a sequence of synchronously sampled network signature states to be inspected and, being implemented for the IBM PC, it is very portable for distribution and demonstrations. The methods for execution timing measurement, for scheduling nodes to execute, and for updating screen graphics differ between the basic and the advanced emulator but both have the overall structure in which a single *instance* of the proposed node protocol can be plugged in and studied for its effects in an entire network model. Changes in the network model or in the protocol under test are completely independent.

The main reason for its developing the truly asynchronous emulator was to capture the significant speed benefit of asynchronous parallel execution that is expected for Selfhealing in reality. The stepped emulator could not accurately reflect this true asynchronism. Also this additional aspect of realism was desirable for a rigorous verification of Selfhealing. The stepped-concurrency emulator for the PC was described in [Grov87b]. We will now focus on the truly asynchronous concurrent emulator for the SUN workstation. The basic emulation structure is as follows:

```
basic network emulator structure
        begin
1       initialize all node states to normal
2       read description of nodes and links in network description file
3       create links between nodes as described in network file
4       initialize all links to have nonactive background signatures
5       accept user input about span that is to fail
6       set alarm indications in DCS ports attached to any of the failed links in a receive direction
        and start the virtual time clock
        Repeat
7       Select (presently asleep) node i, (according to event scheduler)
8       Gather set of all links attached to node i
9       Recover last state of node i from emulator records
10      give the processor over to the Selfhealing protocol
11              {protocol code executes with local link states and node state as provided to
                it on this entry}
12      schedule all future events at other nodes resulting from present invocation
13      observe and record self-determined next-state for node i
14      output any cross-point operate decisions made by node i
15      Update stored image of port registers at node i
16              {optionally, update screen graphic display}
17      Determine which node to wake up next (event scheduler)
Until (no further events scheduled or time limit reached)
18      Output summary of paths created through network and timing measurements etc.
end
```

The SUN Selfhealing research environment actually consists of three programs developed as research tools.[1] Program **ntwkbuild** is an environment for graphical input and editing of network models. It generates the network files and system command files required for subsequent execution of Selfhealing in that network. Program **Selfheal** is the actual concurrent network emulation program which executes the specified protocol version in the specified network model. **shgraph** is a post-processor for near real time graphic display to permit review of the network dynamics as Selfhealing occurred. Event-by-event forward and backward time stepping is provided by **shgraph** for careful inspection and visualization of the network-level dynamics to obtain both detailed and intuitive understandings of operation.

7.2.1 Virtual Time

The truly time concurrent simulation is built upon a few key utilities provided in LANSF, a program package originated by Gburzynski and Rudnicki [LANSF87]. LANSF was developed for precision discrete-event simulation of local area networks and was adapted in this work to provide the basic virtual-time scheduling utilities for concurrent asynchronous execution of Selfhealing with realistic link propagation and other delays including the execution time of the protocol itself. In LANSF, link propagation times and other delays can be calculated and incorporated in the scheduling of future events which will occur at other nodes. The LANSF-based emulator effects an arrow of virtual time with 1 microsecond resolution. An event originating at time *now* and place *here* can be scheduled to have its effect as an event occurring *somewhere else in the network* at a time (*now* + *delay*) where *delay* can be computed or assigned by the causing process. The emulator manages the advance of virtual time in a manner which allows for the protocols at different nodes to execute, effectively, at overlapping moments in virtual time. All events are scheduled on the time line in 1 microsecond resolution bins and nodes are only invoked at those points in virtual time when an event occurs for them.

The basic principles of the virtual-time event-driven simulation method are explained with the aid of Fig. 7.2. The virtual time clock is a discrete clock with a granularity of 1 microsecond. The *ready queue* contains all of the simulation events scheduled to execute at the present time of the virtual clock and all such events must be completed before the virtual clock is incremented. If there are several ready queue events, one is randomly selected and the process (node) for whom the event is scheduled is awakened and given the data indicating the event. For example, at time t1, node 23 may apply a signature to link 178. This schedules node 26 to be awakened at time t2 (in the future) with an indication of a new signature in the port connected to link 178.

When active, a node executes the SH protocol according to its last state (stored by the emulator) and the port records visible to it. This may result in the creation of new signatures or modification of currently transmitted signatures. Eventually the node protocol suspends itself. Any future events which have been caused by the execution of this node are then put in the queue of events scheduled for that future bin in virtual time. Another event from the ready queue is then selected and the corresponding node runs in response to the event scheduled for it at that instant on the virtual time line. When all events in the ready queue have been dispatched, the virtual clock is incremented tick by tick until another bin on the time line is reached where the timer queue indicates one or more events are scheduled. These events are placed in the ready queue and similarly dispatched to invoke the nodes for which they are destined. Any future events which result are in turn scheduled for execution at their respective places in the virtual future by entering them in the timer queue. The virtual clock is then advanced again and so on until "the end of (virtual) time" or until no event remains scheduled for the future.

Whenever a node suspends execution it may insert into the Event Queue a list of special event types which it wishes to have trigger its re-awakening. These insertions are called "wait requests".

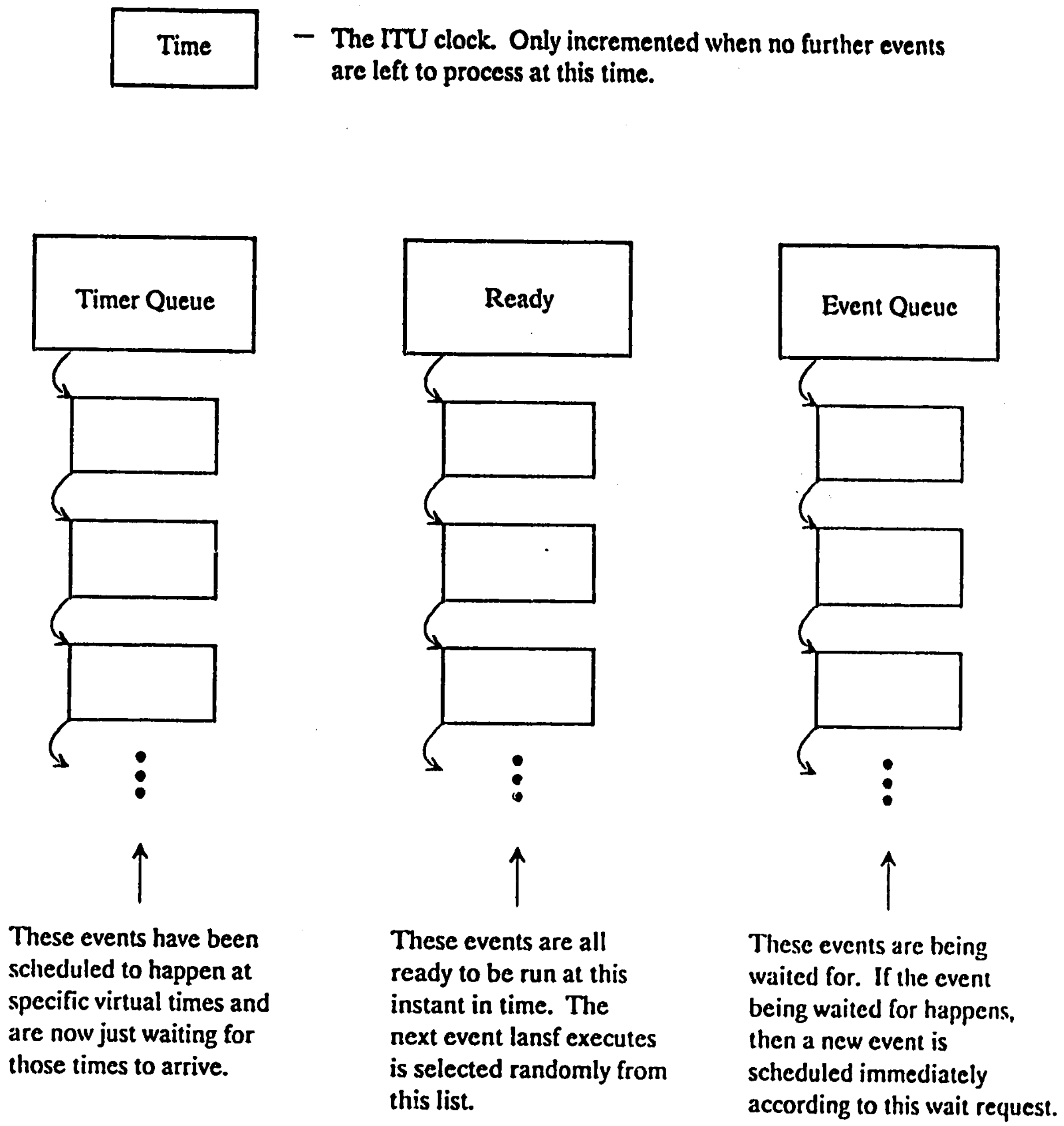

Time
— The ITU clock. Only incremented when no further events are left to process at this time.
Timer Queue
Ready
Event Queue
These events have been scheduled to happen at specific virtual times and are now just waiting for those times to arrive.
These events are all ready to be run at this instant in time. The next event lansf executes is selected randomly from this list.
These events are being waited for. If the event being waited for happens, then a new event is scheduled immediately according to this wait request.

Fig. 7.2 Principles of virtual time event scheduling method for concurrent execution of numerous Selfhealing task instances.

At different times and contexts the SH protocol can then incorporate the various interrupt inhibits which are involved in the protocol and implement internal interrupts within a node for purposes such as the Sender time-out interrupt.

7.2.2 Processor Speed Calibration

The actual processing time of a node is included in the factors which determine the time of scheduling future events. Because LANSF completes process execution within one virtual time tick, special measures are required to obtain and add the self-execution time of the protocol to the network-emulation. In the PC-emulator, a physical realtime clock was used to measure self-execution times but it was found technically infeasible to obtain a realtime clock for the SUN 3/60 that had the required time resolution.[1]

Therefore, another approach was taken in which an estimate of self-execution time is computed in the course of execution based on previously calibrated execution times of respective source code statements. This was soon realized to be a preferred approach because it allows one also to vary the effective processor speed for research purposes; something that is not easily done in reality. The effects of processor execution time are included in event scheduling by incorporating self-execution times in the delays given to future events that result from the current node invocation. Every time the SH task executes it runs internal totalizing counters which form a weighted sum of estimated unit delays (microseconds) accumulated in terms of the number and type of source code statements executed in that invocation. For instance, a *Repeat-While* loop may count actual iterations executed and multiply by a precomputed weight based on the line count of each source code statement in the source code loop, for which we have previously measured *actual* execution times. In this process, the internal running total can be sampled at any point where protocol initiates an event and then used in calculating the ultimate delay into the future at which the initiated event is scheduled for its recipient. For instance, suppose a TS register is newly written when the running self-timing estimate is C1 and sometime later the node suspends itself at complete self-timing estimate of Ct. Then, when control reverts to the emulator, the following computation will determine the future time at which the TS will become an RS at some other node. The *time* of the future event will be:

$$t_{fu} = t_{present} + C1 + d/v^{*} + nbits/S \qquad (7.1)$$

where:

$t_{present}$	=	present instant of virtual time (stopped while task executes)
d	=	length of link
v^{*}	=	propagation velocity on link
nbits	=	number of bits in a signature
S	=	rate of signature bit transport

The corresponding *place* of this future event will be the node and port number which the emulator determines from the network file is connected to the link to which the TS was applied. In this manner several signature events can be emitted during one invocation of the protocol without all such events appearing in an apparent burst at the end of the current execution phase. (In actuality, the sequencing of signature initiations and the alarm event onsets makes use of local timer events to suspend a node to effect its own execution time by delaying itself to a future time where it catches up with its own estimated execution time, or with a desired amount of deliberate delay. The effect achieved is however equivalent to the simpler explanation offered above.)

To perform the self-estimation of execution times a software calibration exercise was undertaken in which 43 categories of C language instruction types were identified and calibrated for the SUN 3/60 processor by running timing loop programs in which the subject source statement was executed thousands of times and total time measured at 20 msec resolution, by the SUN O/S. When executing Selfhealing the line counters for each statement category are multiplied by their calibration times to estimate the actual execution time on each SH node invocation. The execution time of the time-estimating code is itself omitted from the estimation. Appendix C contains a list of the source code instruction categories and their execution times calibrated by this method.

With the use of this method, any arbitrary processor speed can be effected through a scalar P which scales the self-estimated times used in the event scheduling calculations. $P = 1.0$ represents the calibrated speed of a real SUN 3/60. A faster processor is modelled by $P > 1.0$. This feature is used in Ch. 12 to explore sensitivity of performance to processing power with respect to other delays inherent in the SH process, most notably, signature transfer time. This principle can also be used to test effects due to possibly mixed processor speeds in the same network.

7.3 Summary

This chapter has explained the methods developed for this research on Selfhealing networks. The overall approach comprises an essentially experimental implementation of the intended protocol running concurrently in a simulated network environment. We believe this gives a degree of confidence in the results which is higher than would be obtained either by analytical means or simulation studies not based on concurrent execution principles. True parallel distributed asynchronous execution of the protocol, with accurately modelled processor execution times and signature propagation delays is effected by the network emulator. The performance results and research findings that now follow were obtained through this method of investigation and the successful final version of the protocol was derived by repeated utilization of these same tools and research environment.

CHAPTER 8 RESULTS IN SMALLNET STUDY NETWORK

In this chapter and the next, the performance of Selfhealing is reported quantitatively in terms of path-finding efficiency and speed in four study networks which have been employed. Processor speed (P), repeat limit (RL), and signature transfer rate (S) are fixed at values most likely envisaged for real applications in these first results. Chapters 10 - 12 are then devoted to parametric experimental studies to explore characteristic dependencies on each of these parameters.

Smallnet is an arbitrary 10-node study network model which is amenable to complete graphical representation of every link in the network. For these first results, we use this property of Smallnet to present a topological picture of the each restoration plan obtained for every spancut. We then compare each Selfhealing solution to *MetaDijkstra's* ideal topologies and we use the few cases where they differ as discussion vehicles to develop insights into Selfhealing dynamics. Comprehensive statistical results for three larger and more realistic network models - *US*, *Metro*, and *Bellcore* - then follow in Ch. 9, without the continued luxury of full graphical representation of each restoration topology.

8.1 Topological Path Performance in Smallnet

All possible spancut experience in *Smallnet* were performed under the following conditions:

Maximum path finding stress	signature transfer rate : 8.0 kb/s
Repeat Limit (RL): 6	signature length : 64 bits
all span distances: 1,000 km	processor speed (normalized to SUN 3/60): 1.0
carrier velocity : 5 microseconds/ km	alarm collection holdoff : 100 msec

The restoration plans derived by Selfhealing are shown in the 22 inset panels of Fig. 8.1. Each panel corresponds to one span cut and the failed span is marked by the "X". Where a bracketed number in the lower right corner is present (Spancuts 9, 11, 14, 17) the total path length (TPL) is greater than the corresponding *MetaDijkstra* solution by the number of links indicated. All other restoration patterns are identical, or equivalent in performance measures, to the solutions from *MetaDijkstra*. For comparison where there is a difference, the ideal solutions for span cuts 9, 11, 14 and 17 follow in similar form in Fig. 8.2.

8.1.1 Path Number Efficiency

A PNE of 100% was obtained for every span cut. This means that in each case the *number* of paths found by Selfhealing was equal to the maximum number of paths that are theoretically achievable by an ideal k-shortest paths algorithm applied between the two alarm nodes. The 100% PNE result implies that the highest possible degree of traffic restoration would be obtained because the network restorability is limited only by the topology of the network, not by the mechanism used to solve the restoration routing problem.

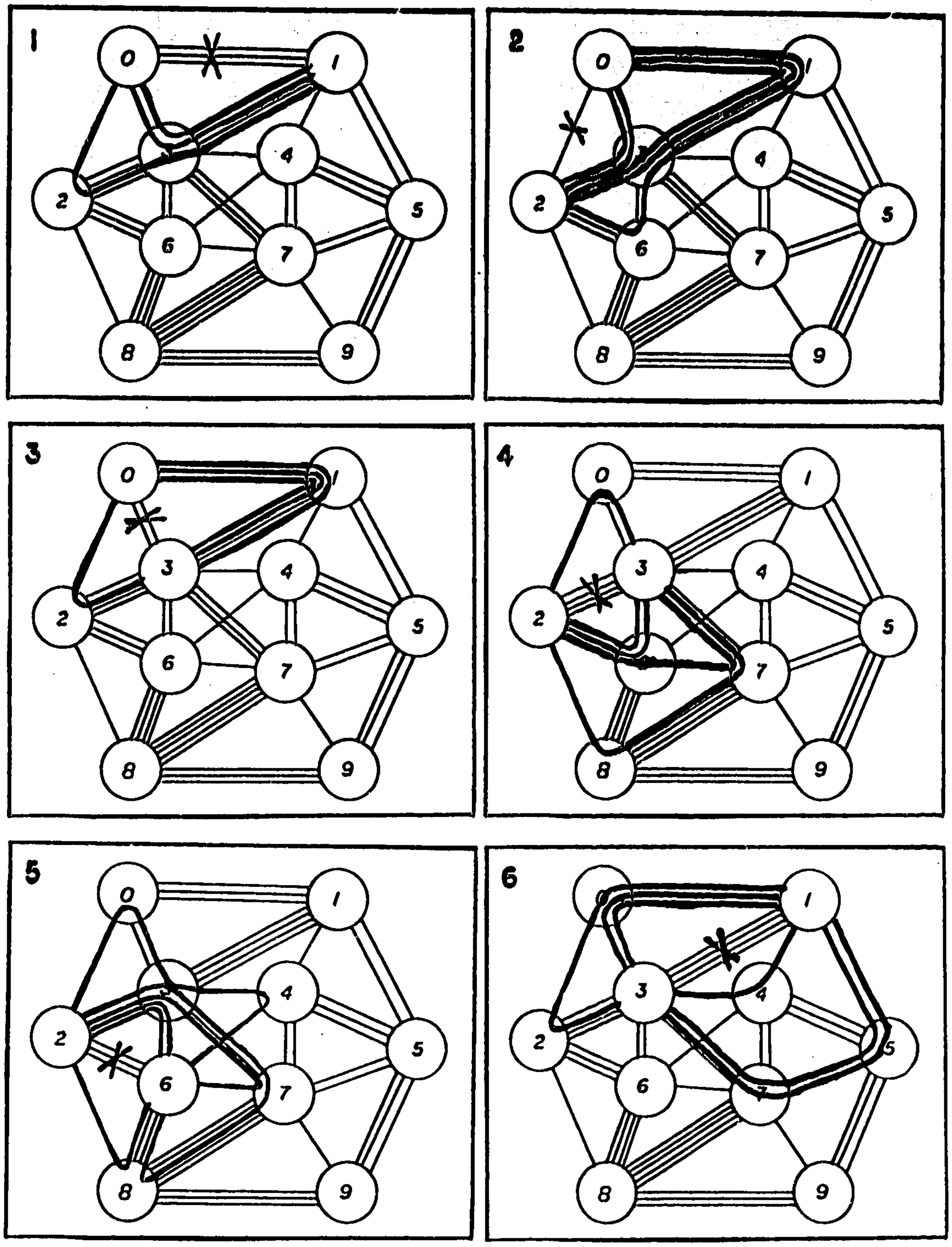


Fig. 8.1 Restoration topologies derived by Selfhealing in Smallnet with RL=6, S=8.0, P=1.0 (p.1 of 4)

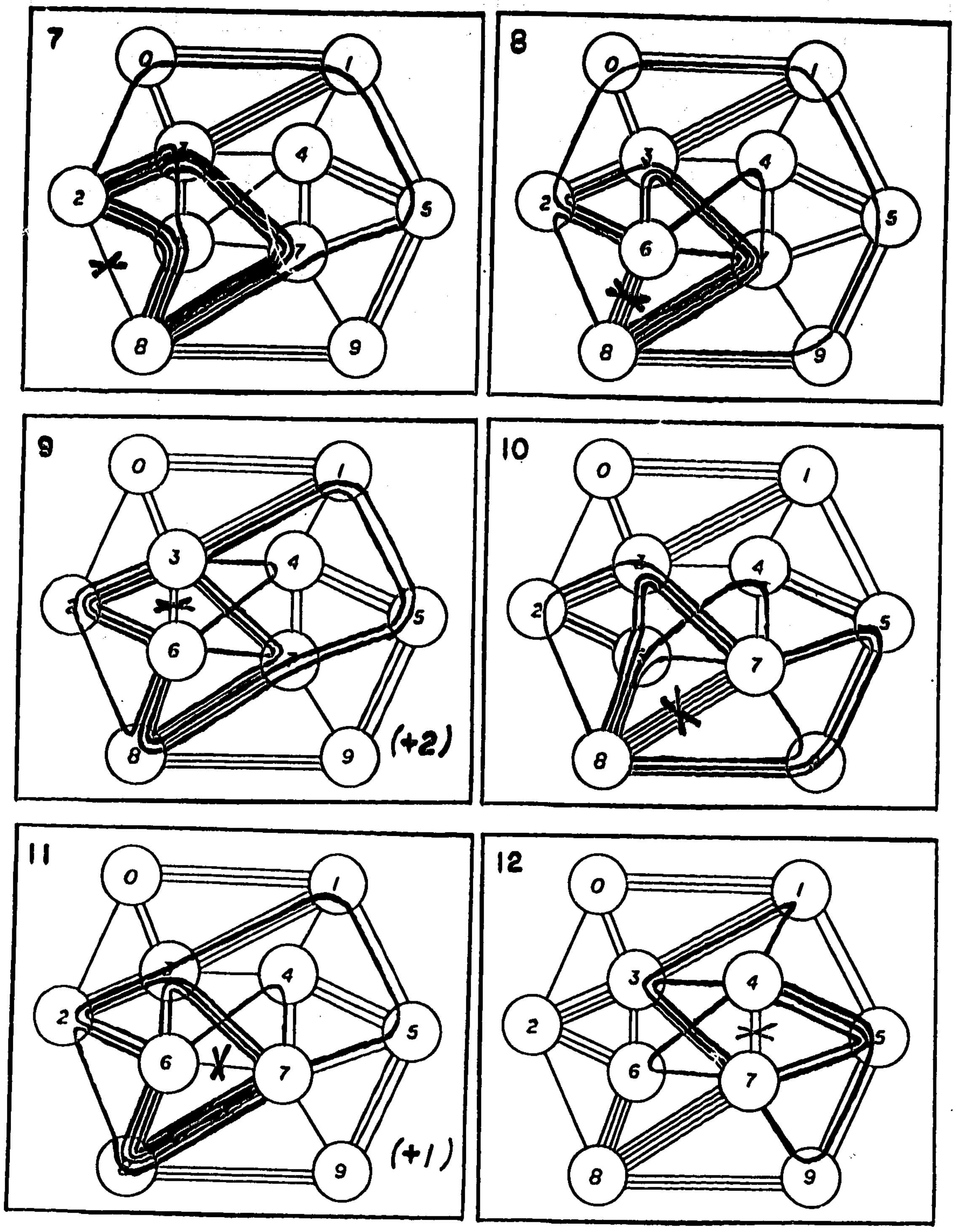


Fig. 8.1 Restoration topologies derived by Selfhealing in Smallnet with RL=6, S=8.0, P=1.0 (p.2 of 4)

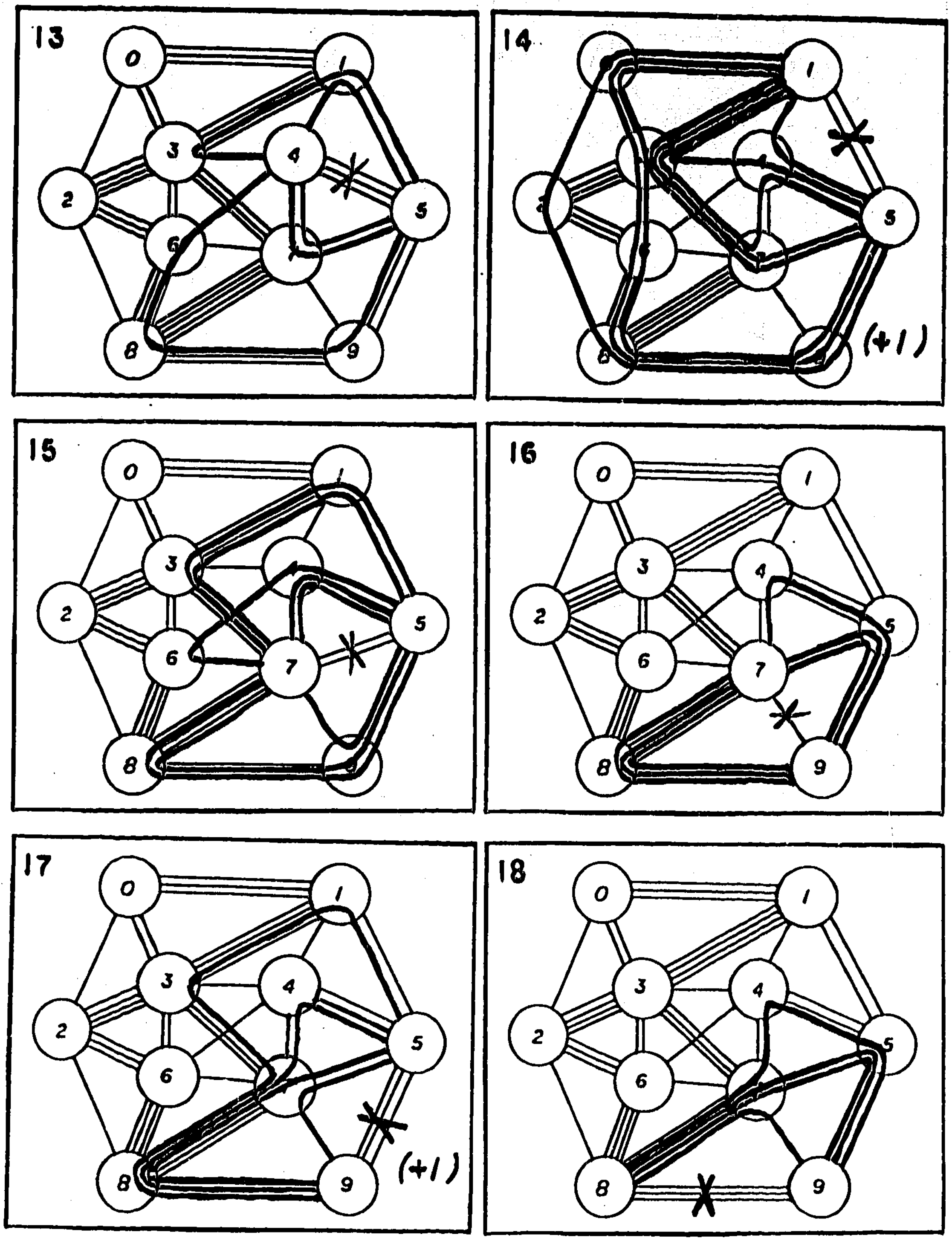


Fig. 8.1 Restoration topologies derived by Selfhealing in Smallnet with RL=6, S=8.0, P=1.0 (p.3 of 4)

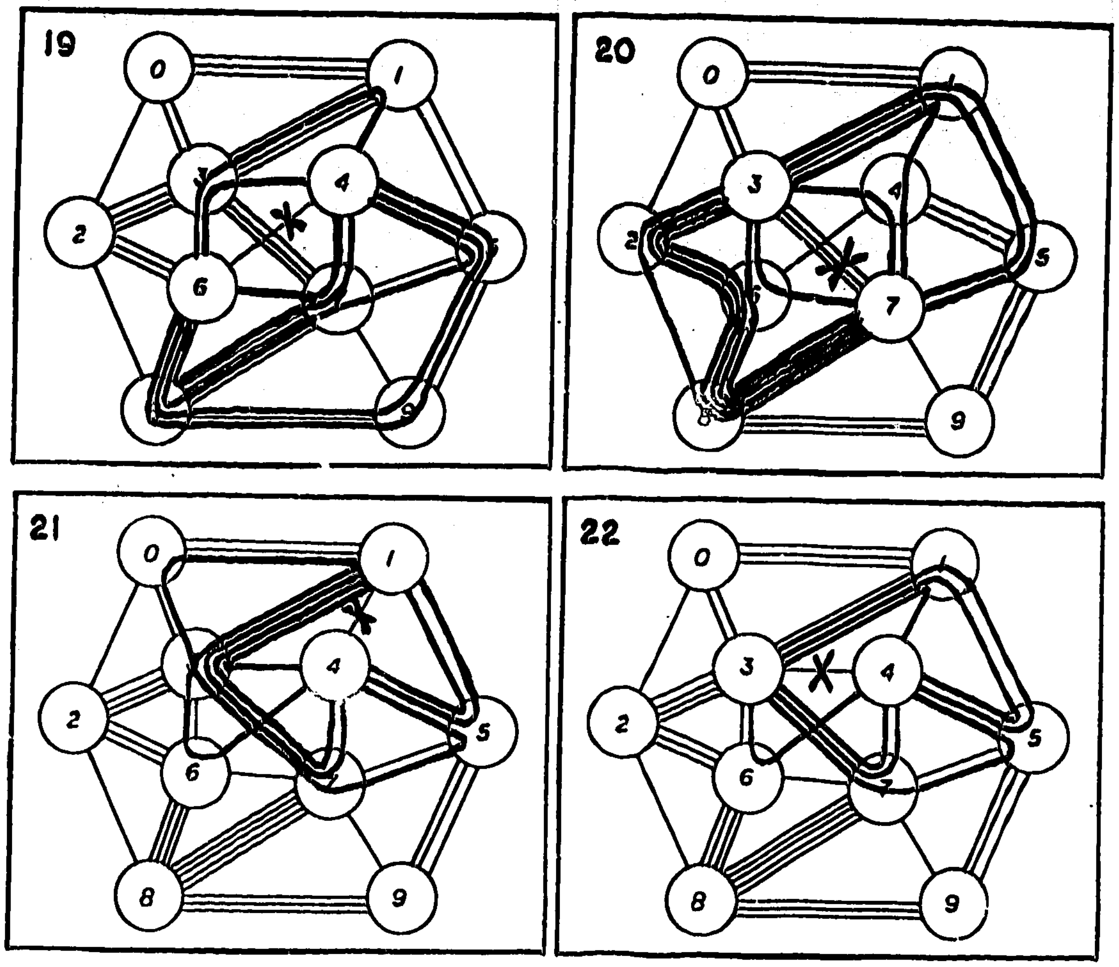


Fig. 8.1 Restoration topologies derived by Selfhealing in Smallnet with RL=6, S=8.0, P=1.0 (p.4 of 4)

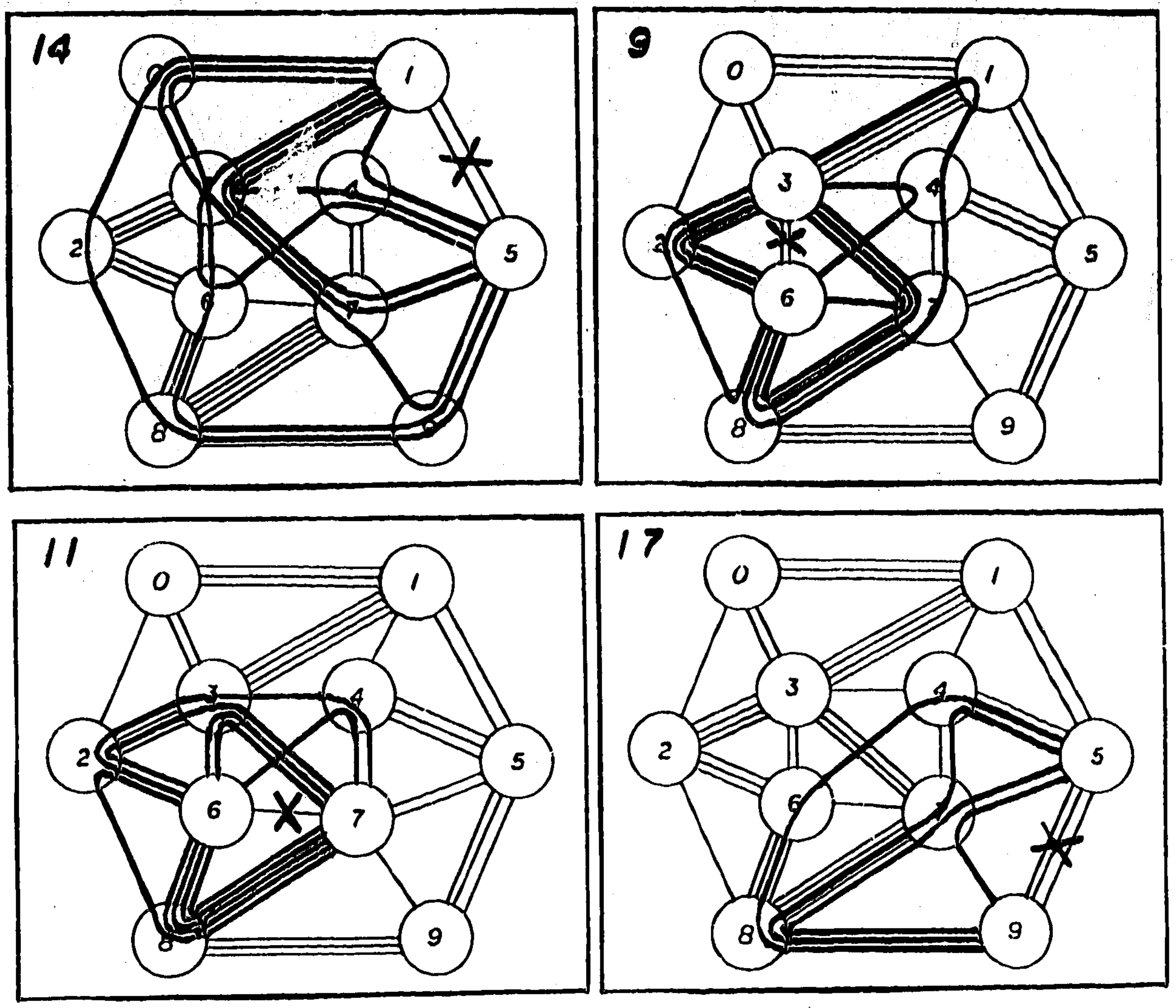


Fig. 8.2 Ideal restoration patterns from for Smallnet spancuts (1-5), (3-6), (6-7) and (5-9)

A total of 143 individual paths were created in response to the 22 different span cuts for an average path-establishment load of 6.5 paths per cut. In all cases but one, the limiting factor in determining path number was the number of spare circuits available at one of the alarm nodes. This "end-node bottleneck effect" is by far the most frequent mechanism limiting the topologically possible number of paths in the networks studied. This was not, however, the case for span cut 3-7, (panel 20. Fig. 8.1). In this case the network topology is such that as spare circuits between the two failure nodes are synthesized, the graph of remaining spares becomes disconnected across cutset [(2,8) (2,6) (3,6) (3,4) (4,1) (1,5)] before the available spares at either of the alarm nodes are saturated.

8.1.2 Path Length Efficiency and Shortest Path Probability

Fig. 8.3 is a histogram showing the statistical frequencies of individual path lengths obtained by Selfhealing in comparison to the frequencies of path length from the ideal solutions. The sample set is the lengths (in links) of each of the 143 paths resulting from all spancuts in Smallnet. Selfhealing used 401 individual links to create the required 143 paths for an average restoration path length of 2.8 spans. On the same set of spancuts, the *MetaDijkstra* solutions totaled 143 paths using 396 links. The network average PLE is therefore 396/401 = 98.75 %.

Four of the 22 restoration plans constructed by Selfhealing are imperfect in terms of length and four individual paths are longer than the corresponding paths in the solutions from *Metadijkstra*. The network average SPP is therefore (143- 4)/143 = 97.2 %. Three of the individually overlength paths are one span longer than ideal and the fourth is two spans longer. In spancut (3-6), the ideal solution has a length 3 path that is realized as a length 5 path in the actual solution. In 18 out of 22 span cuts the PLE is 100 %. These are the spans (1-8,10,12,13,15,16, 18-22). In the remaining four restoration patterns the PLE values are:

24/26 = 92.3 % for span cut 9	31/32 = 96.9 % for span cut 14
13/14 = 92.9 % for span cut 17	24/25 = 96.0 % for span cut 11

Inspection of cases where excess path length occurred shows that it was in each case one of the last and longest paths that was imperfect. Therefore in operational cases of less than maximum path finding stress, some or all of the strictly overlength path routings would not arise. Only spancut 3-6 has a path that is overlength and does not correspond to the longest length path in the ideal plan. If the single longest path in each case is excepted from the SPP estimate it rises to 99.3%.

8.2 Dynamics of Overlength Paths

Let us now consider the four cases in the results from Smallnet where overlength paths occurred. It is of interest first to consider whether or not the occurrence of excess links is simply related to the complexity of the restoration plan. To this end, Fig. 8.4 is a scatterplot of the 22 cuts

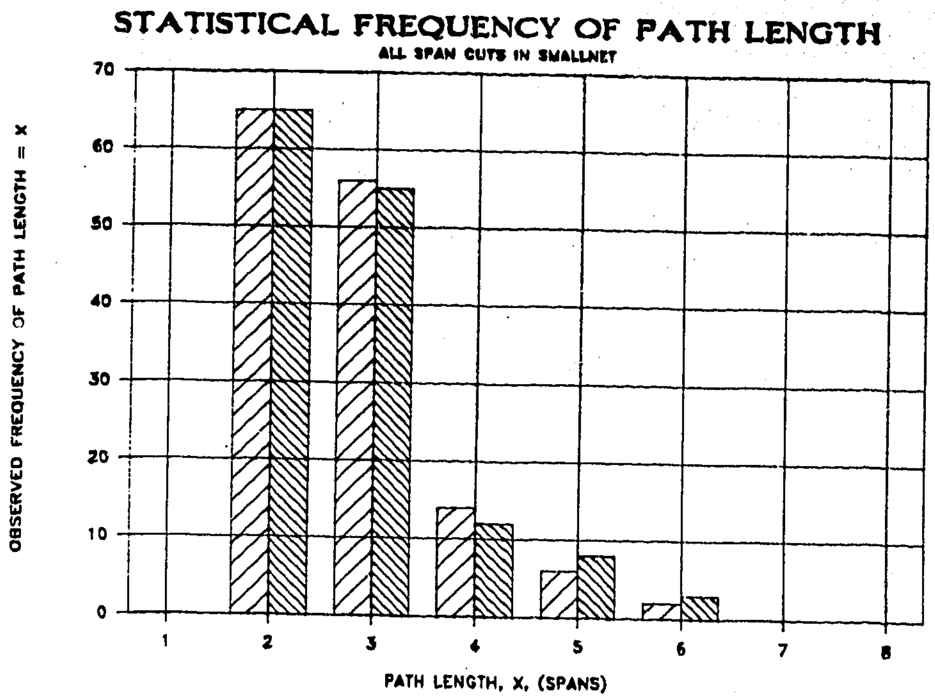


Fig. 8.3 Distribution of ideal and experimental path lengths for all spancuts in Smallnet

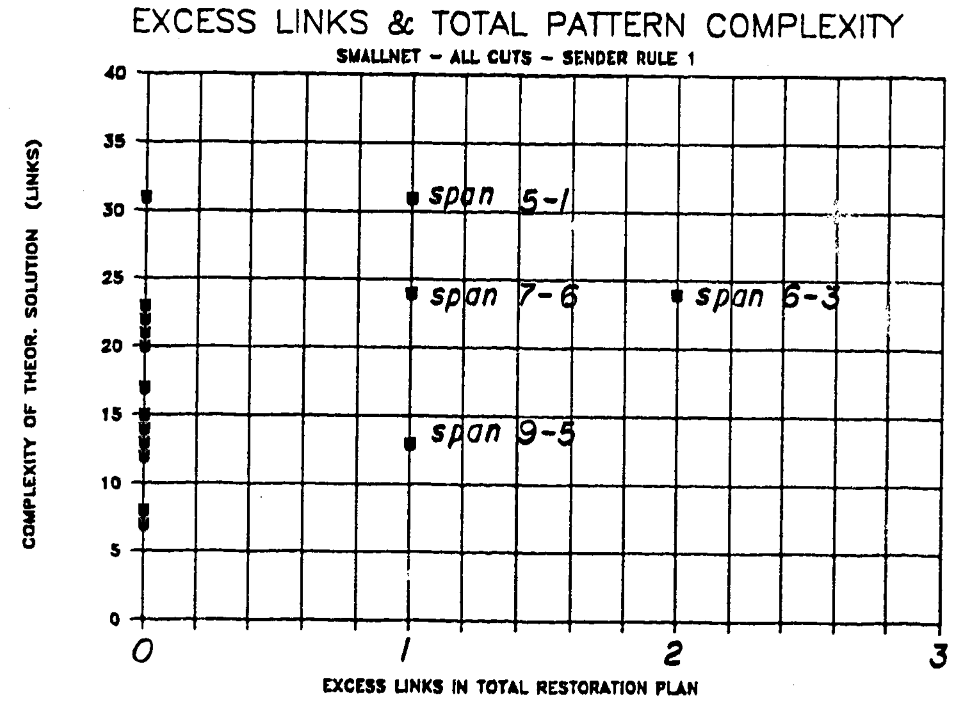


Fig. 8.4 Experimental Smallnet spancuts having overlength paths plotted in terms of pattern complexity and number of excess links

in Smallnet in terms of the number of excess links in the Selfhealing restoration plan and the total number of links in the theoretical solution. No simple correlation is obvious between plan complexity (in total links) and the number of excess links. Other plots (not shown) of the excess links against total number of competing indexes, total number of paths found, and against the path length (of the corresponding ideal path) also show no strong diagnostic correlation for excess path length in Smallnet or the other networks studied.

This leads us to suspect that cases of excess links are attributable to purely topological effects which may arise in either simple or complex restoration patterns. Two such effects have been identified: One is an intrinsic effect that can arise in any k-shortest link disjoint paths algorithm when path length is measured in logical hops. We call it the *early path choice* effect. In this case a choice amongst two equal-length alternatives for some *early* (short) path can have a differing influence on the longer (later) paths of the k-shortest set. The other is a dynamic effect intrinsic to Selfhealing called *temporary index blocking* (TIB). TIB is a situation where links in one part of the network are temporarily held in the mesh of one index but then released, due to success of the index via some other expansion trajectory. TIB only has the effect of increasing path length if, during the time of temporarily holding these links, the expansion of some other mesh is caused to reach the Chooser via a longer than ideal sequence of precursor associations and these longer path associations persist long enough to be frozen in by reverse linking.

8.2.1 Overlength Paths in Smallnet

We will now investigate each of the four PLE-imperfect restoration solutions occurring in Smallnet when RL=6, first explaining the dynamics to which we attribute each overlength path and then offering a general discussion of the two effects mentioned above. The insights gained lead to a prediction, which is verified, that 3 of the 4 Smallnet cases of overlength paths can be corrected to 100% PLE by appropriate reduction of the repeat limit. The four overlength path instances are:

Span cut 5-9: The Selfhealing solution for this case is shown in Fig. 8.1, panel 17 and the ideal solution is in Fig. 8.2, panel 17. In this case four paths were found and three of the Selfhealing paths are identical to paths in the ideal solution. The fourth path of the experimental solution, (5-1-3-7-8-9) is of length 5. It should ideally have been either of two other paths, (5-4-6-8-9) or (5-4-7-8-9) of length four. The creation of this path provides a illustration of *temporary index blocking* (TIB) effects.

In spancut 9-5, the critical index blocking occurred on span (4-7) as follows: In cut (9-5), node 9 acts as the Sender and, in this case, floods all the spares leaving its site. Let us say that it floods with index numbers 1 through 4 in a clockwise pattern starting on span (9-8). The first index to arrive at node 7 will be index 4 on span (9-7). Node 7 rebroadcasts index 4 once onto each of the spans (7-5), (7-4), (7-3), (7-6) and (7-8). This seizes one of the two spares on span (4-7) for index 4 but it can be easily seen in this case that index 4 destined to succeed quickly over route (9-7-5). The next index to get onto span (4-7) is index 2, arriving via route (9-8-7). The latter index is repeatered on to node 5 (the Chooser) and will result in one of the ideal restoration paths (9-8-7-5). At this stage another index, 1, cannot get on to span (4-7), because it is full with indices 2 and 4. Soon after index 4 is applied to span (4-7) however it reaches the Chooser via span (7-5). The Chooser responds with a reverse-linking signature and when this arrives at node 7, index 4 is removed from span (4-7) and the freed link becomes available for inclusion in the broadcast pattern of another index at that node on the basis of lowest repeat count of all indexes not currently sending on span (4-7).

Even if the newly freed link on (4-7) is now given to index 1, it is too late to avoid creation of the overlength path because the other index 4 signature is already in a race over the path (7-4-5) with index 1 as it now advances from node 3 over the route (3-1-5). The shorter path may win the race, but it is now up to elements of chance, distance, and facility velocity. In this particular case the longer path (7-3-1-5) succeeded in arriving at the Chooser first and thereby commits the collective mechanism to reverse linking synthesis of a path over the longer precursor chain for that route.

Spancut 6-7 : (Experimental in Fig. 8.1, panel 11. Ideal in Fig. 8.2, panel 11) In this cut, an ideal path (7-4-3-2-6) was replaced by the longer path (7-5-1-3-2-6). This is also attributable to TIB effects as follows: the index that creates path (6-4-7) temporarily blocks span (3-4), prohibiting creation of a path (7-4-3-2-6) long enough in time for the index that creates the eventual experimental path to reach the Chooser via the longer route (7-5-1-3-2-6). In this case the index that succeeded on span (6-4) was also the index selected by node 4 for repeatering to the single spare on span (3-4). When node 4 later sees reverse-linking on the index that created path (6-4-7), it immediately applies the other index received on span (7-4) to span (3-4) but now it will take three span times for the new index from node 4 to reach the Chooser (node 6) via the remaining spans 4-3-2-6. Before this happens signatures on the same index have traversed the 5 spans on route (7-5-1-3-2-6) arriving at the Chooser and triggering reverse-linking, creating the overlength path in the solution.

Spancut 3-6: (Experimental in Fig. 8.1, panel 9. Ideal in Fig. 8.2, panel 9) This spancut presents another variation on the index blocking effect. One of the ideal paths (6-8-7-3) was realized as actual path (6-8-7-5-1-3) for an excess link count of +2. The reason this occurred was that one of the spares on span (3-7) was flooded with an index which was applied initially to span 6-8 by the Sender. This index mesh then split in two directions at node 8, expanding right and left (as we view the network). One route arrived at the Chooser via spans 8-2-3. The other route also reached the Chooser via route 8-7-3. The index then succeeded in the direction of node 2 via (6-8-2-3) rather than in the direction of node 7. When this occurred the other potential route was now obsolete for that index and although in the process of collapsing, it will continue to hold links until reverse-linking on that index can propagate to effect mesh collapse fully on the unneeded links . When reverse linking on the affected index reached node 8, then the link in span (8-7), followed by the link on span (7-3) on that index, were released and reapplied to a new index. By this time however, the longer path (6-8-7-5-1-3) had reached Chooser and triggered reverse linking.

The reason this results in +2 excess links is that topologically the next shortest route in this network, after spans (6-7), (7-3), and (3-4) are used (or index-blocked), is the length 5 path that was found. The other path on the route (6-8-7-5-1-3) is equivalent in length to the (6-8-7-4-1-3) path used in the ideal solution for spancut (3-6). *MetaDijkstra* also could equally have chosen either of these length 5 paths when all spans have unit lengths. It depends simply on the internal sequence of the latter algorithm when searching the network presented to it as to which is selected. Both are equivalent when path lengths are measured in logical span lengths. The latter point arises again in the next case of interest.

Spancut 1-5 : (Experimental in Fig. 8.1, panel 14. Ideal in Fig. 8.2, panel 14) In this cut the TPL is +1 due to the overlength path (1-0-3-6-8-9-5) which could have been routed as (1-0-3-7-9-5) if the prior path (1-3-7-4-5) had instead been routed as (1-3-6-4-5), the latter two alternatives being of equal length. The outcome in this case is not attributable to index blocking or any Selfhealing effect but is rather an instance of the *early path choice* effect.

In cut (5-1) there is no difference between the Selfhealing and MetaDijkstra solutions for the first four paths, ranked by length. There are however, two equal choices presented by the network topology for the 5th path which, in either case, will be of length 4. The choices are (1-3-7-4-5) or (1-3-6-4-5). What this has to do with the length 6 path (1-0-3-6-8-9-5) which is overlength is that *depending on which early path is chosen*, different possibilities remain for that later path. By chance order of internal execution, *Metadijkstra* chose (1-3-6-4-5) for its fifth-shortest route whereas Selfhealing realized the route (1-3-7-4-5). These paths are equally correct choices for the fifth path in the restoration plan but it turns out that the *Metadijkstra* choice is in this instance globally beneficial in terms of the downstream effect on the longer path which is at issue here. By studying the two panels involved, it can be seen that if route (1-3-6-4-5) is chosen instead of (1-3-7-4-5) for the fourth path, a link on span (3-7) is left free for creation of the *later* length 5 path (1-0-3-7-9-5) as part of the overall solution. If however, the fourth path is chosen as (1-3-7-4-5), the last spare link on span (3-7) is used thereby topologically requiring the sixth path in the total solution to adopt the length 6 routing (1-0-3-6-8-9-5).

We will now discuss the TIB and early path choice effects which these four exception cases have shown us in more depth.

8.2.2 Discussion of Temporary Index Blocking

TIB is a normal and frequent phenomenon within the dynamics of Selfhealing. It is an inherent result of the way Tandem nodes mediate competition amongst multiple indexes which are simultaneously expanding their meshes and then collapse eventually, if successful. TIB is intrinsically related to the speed and parallelism of Selfhealing because it is an aspect of the mechanism that permits the greatest promptness in allocating resources at every node to synthesize simultaneously all the disjoint paths that are possible without needing to find each path serially in time. Any given index may expect to be frequently and repeatedly blocked temporarily as the shuffling and jockeying of competing meshes is at the core of Selfhealing's speed and fluid parallelism. An index is said to be blocked on a certain span when the span is a legitimate candidate for that index but all spares on that span are presently held by other indexes.

TIB can infrequently cause an overlength path if the blocking on a desired span endures long enough to permit an end-run effect by indexes advancing around the blocked span by a longer path. The general effect is as follows: Part of the spare capacity of a span, which is in the route of a path found in the ideal solution, is initially incorporated in the mesh of an index that is destined to succeed over another route which does not ultimately include the subject span. Reverse linking on the successful index later frees at least one link of the key span for use by other index(es), but in the meantime mesh expansion of the temporarily blocked index over the key span has been halted, but it continues over other longer routes or fronts ultimately up to the length permitted by RL. If the index that is blocked on the key route expands enough via other routes to reach the Chooser before the same index reaches the Chooser on the shorter path after it unblocks, then the Chooser responds to the first arrival of that index and the longer route may be realized, rather than the theoretically possible shorter route. The only remaining requirement for the TIB event to cause realization of an overlength path is that blocking on the desired span(s) also lasts long enough after the Chooser response for sufficient reverse linking to occur to freeze-in the overlength path. Otherwise it is still possible for the shortest path to be realized even after Chooser response because the precursor structure within the mesh is still dynamic at each node, up until reverse linking is complete at each node.

Based on this explanation of TIB-induced excess path lengths, we can predict that the effect is dependant on RL. This is because TIB effects could apparently only result in an overlength path if *end-runs* of the required lengths are permitted to support mesh expansion around the blocked regions. Therefore to the extent that RL can be reduced without impinging on the actual number of paths that are created (100% PNE should never be sacrificed in practise), it should be possible to ensure that solutions are found which have even better PLE statistics.

The principle is that by reducing RL, we should be able to make overlength routes unviable in some cases, especially since we have seen that the overlength paths also tend to be the longest paths. By defeating an end-run to the Chooser we effectively force the temporarily blocked index front to *wait* as required for eventual mesh collapse on the links required for realization of the strictly preferred shorter paths. One may suspect that this forced waiting could be at some cost in total route-finding time. This is one of the questions considered in Ch. 10. Presently however, a simple experiment with Smallnet can test the predicted effect of reducing RL in cases where TIB caused excess-length paths. To test this hypothesis, the first three of the above cases were repeated with RL reduced from 6 to 4 for spancuts (5-9) and (3-6) and RL = 5 for spancut (6-7). The results are that in each case the solutions do become identical to the ideal restoration plans shown in Fig. 8.2. This confirms the principle that PLE is influenced by RL and where TIB is responsible for overlength path(s), a reduction in RL can, in at least some cases, cause a return to 100% PLE at 100% PNE. Ch.10 further explores the effects of RL.

It may be thought that TIB path effects could also be mitigated by adopting a Chooser node pause after each index is received, to see if a lower repeat count arrives on the same index before committing to reverse linking. The latter would however, further delay the helpful readjustment of index meshes that results from reverse linking and mesh collapse. It is also not a complete cure to TIB effects. In restoration however, it is far more important to accept new routes as fast as possible and so this measure has not been pursued. Another possibility that may occur to the reader is to allow the Chooser to switch its reverse linking response to a new port if, after an original reverse linking response, a forward signature on the same index arrives thereafter with a lower repeat count. We have investigated this possibility and it has the fundamental danger that the process of reverse-linking, set in motion by the Chooser response when it *first* responds on an index, can in fact dissolve the remaining chain of precursor associations that is behind the *second* shorter path possibility that is postulated in this case. This implies the loss of an otherwise guaranteed path. In addition the Tandem node protocol would have to be modified to recognize changes in reverse-linking direction and be able to initiate switches of reverse-linking propagation direction similar to the postulated Chooser ability. It is not clear how to incorporate such changes without other fundamental side-effects primarily stemming from the loss of built-up system state information implicit in the mesh of indexes that is lost when reverse linking occurs.

One can infer from these dynamics however that TIB can never influence path length in a one-index Selfhealing event. Therefore if there were some application where time is less critical than obtaining 100% PLE, a series of *time consecutive* single-index path finding is always possible. In this application, the Sender would conduct a one-index flood and wait for reverse linking. Then it would move on to another single-index flood and so on. Working this way, Selfhealing would be literally emulating MetaDijkstra but in a distributed fashion.[1]

It is important to note as a final point that TIB has never been observed to reduce PNE from 100% . Nor can it do so because, by definition, if there is no end-run path the given index will have no alternate avenue for flow and will inherently wait for collapse on prior meshes to expand onto the only spans available to that index. If the blocked index is permanently blocked then this is equivalent to a statement that all paths supported by the topology of the network have been found, within the allowed RL. Deep within the Selfhealing method is the fact that the sequence of dynamics which occurs are in fact statements about the topology of the network so that questions such as above tend to be self-referential. That is to say that a question such as "What would the outcome have been for path so and so if the sequence of dynamics had been different within that network?" is a self-contradictory statement because the sequence of dynamics is ultimately determined entirely by the network itself. One cannot change the intrinsic dynamics except by changing the network.

8.2.3 Discussion of Early Path Choice Effect

Spancut (5-1) brings to our attention a particular property of any k-shortest paths algorithm when applied to graphs where logical or integer link lengths apply. In such circumstances the k-shortest paths obtained by finding first the shortest path, then the second shortest path not using any links of the first, and so on can have more than one outcome. This is because without using real-valued link costs, two exactly equal choices can exist for the nth longest disjoint path. Depending on the arbitrary choice made in such circumstances, there can be differing consequences for subsequent paths, with possibly different TPL for the complete restoration plan.

In a graph where edge lengths are real-valued there is never any choice between paths for the nth shortest route: real-valued equality between span lengths is infinitely improbable and the multiple-outcome effect is not seen. In one sense therefore, this is an artifact of our interest in logical span lengths in the present work. Nonetheless we want to continue to use logical link lengths in Metadijkstra because it is our yard-stick against which Selfhealing is measured and Selfhealing (at least to date) inherently uses *logical* path lengths. The effect we encounter is not an issue in the literature of usual k-shortest (non-disjoint) paths because early use of one link does not prohibit reuse of that link by other paths. Only when link-disjointness is introduced and combined with logical path length measures is the combination of factors present which permit more than one unique solution.

One could argue on this basis that the Selfhealing result in spancut (5-1) is equivalent to MetaDijkstra's result because, over a large number of trials, the globally fortuitous choices amongst equivalent early path alternatives will average out to be equal between any two k-shortest link disjoint paths algorithm. Nonetheless, to have a systematic basis for comparison, we propose to bias the outcome against Selfhealing on this score. That is, in cases of multiple solutions, we will consistently take the *best* solution found by MetaDijkstra as the reference against which

Selfhealing is measured, as in spancut (5-1). This will bias the PLE and SPP measures achieved by Selfhealing slightly towards the pessimistic side and not affect PNE at all.

8.3 Speed of Selfhealing in Smallnet

The above sections focused on the topological completeness and efficiency of the Selfhealing restoration patterns. We now add the dimension of time to the dynamics of Selfhealing. Let us first consider the speed results in terms of histograms and averages obtained over all spancuts in Smallnet. Two sets of statistics are of interest: (a) the times required for establishment of individual paths (143 data points), and (b) the times required for restoration event completion (as defined in Ch. 4) (22 data points).

Fig. 8.5 shows the statistical frequency of the time for individual restoration paths. The mean path completion time (t_{pav}) is 196 msec and it has a std dev of 49 msec. 95% of all paths are completed in less than 300 msec. Fig. 8.6. shows the statistical frequency of the time required to complete the entire restoration event. The mean time to complete restoration (t_R) is 269 msec, with std dev 58.7 msec and 95 % of all observed restoration events are complete by 375 msec (t_{95}). These results imply that with the parameters RL =6, P=1.0, S= 8 kb/s, no spancut in the basic Smallnet model would result in loss of connections. The longest any transport entity suffered interruption was about 450 msec, implying a large remaining margin (at least 1.5 seconds) of time before the onset of call-dropping. The average time to establish the first path (t_{p1}) is 160 msec with a very small variance of only ~ 5 msec.

8.3.1 Restoration Trajectories

We can gain more insight into Selfhealing as a dynamic process by preserving, to an extent, the association between time and topology which is obscured by the histograms and averages above. As a tool for presenting and analyzing the dynamics of a Selfhealing event, we introduce the *restoration trajectory (RT) plot*. An RT plot (or just "trajectory plot") represents a complete restoration event resulting from a given spancut in the space of elapsed time versus path number. The ordinate is time since the onset of failure and the abcsissa is the running number of paths completed. Every path that is part of a restoration event is represented by a point at the time at which the path was completed and the consecutive number of completed paths that it represents. The RT plots for all 22 individual spancuts in Smallnet are shown side-by-side Fig. 8.7. Each square marks the time of completion an additional path, measured as the time at which reverse linking arrives at the Sender node. Each line segment represents a separate restoration event numbered 1 to 22 corresponding to all the spancuts in Smallnet in the same order that their topologies are presented in Fig. 8.1.

Fig. 8.7 shows that in all cases, a minimum of approximately 155 msec is required before the first path is completed, even on the simplest 2-span triangular restoration paths found at the start

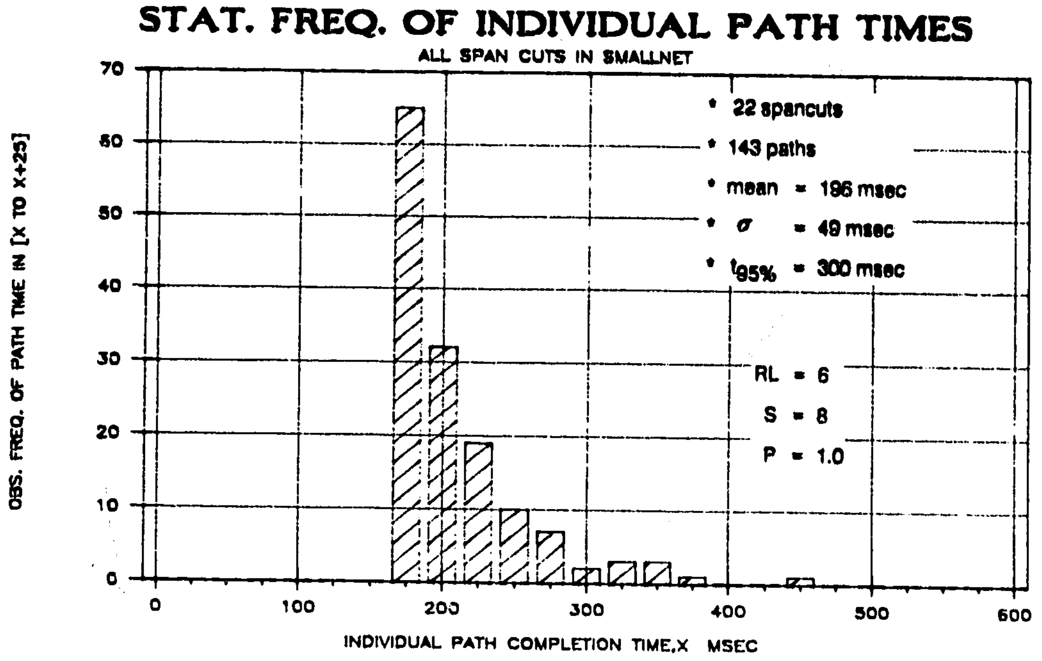


Fig. 8.5 Statistical frequencies of individual path completion times for all spancuts in Smallnet.

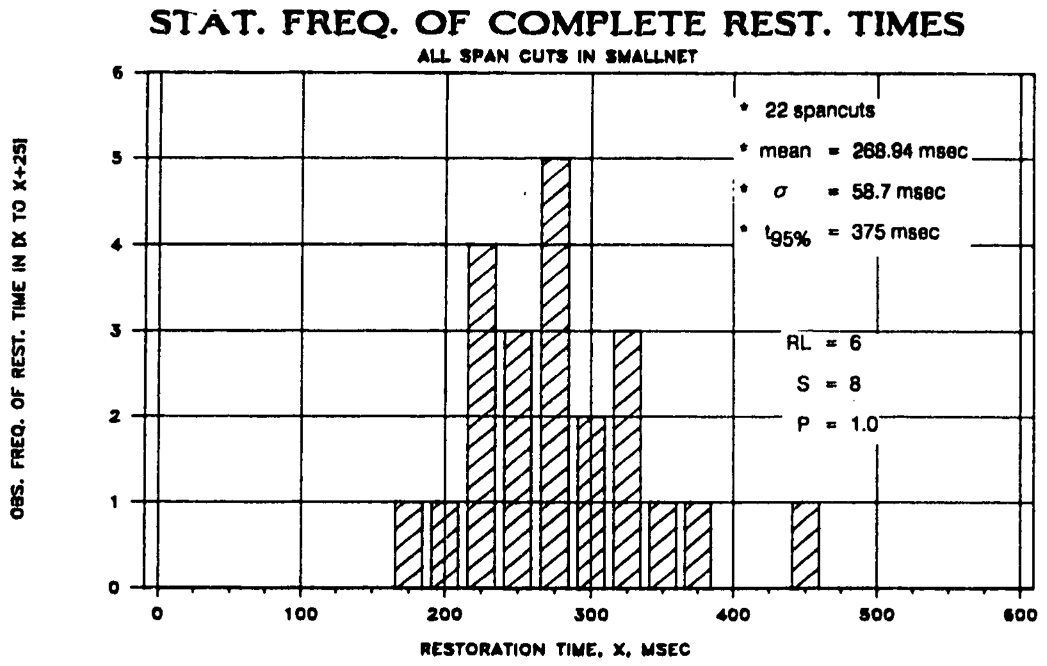


Fig. 8.6 Statistical frequencies of times required for complete restoration in Smallnet.

of most restoration events. The maximum time to complete any restoration was 450 msec in spancut 5-1 (panel 14 of Fig. 8.1). Several different characteristic slopes and time intervals between completions of individual paths are noted in varying segments of the RT plots. We will later see the circumstances to which they correspond such as accumulating success on one route or via several diverse routes. The most significant differences between individual trajectories are attributable to the different number of paths to be found, the lengths of the paths to be realized, and the intrinsic topological route complexity of the plan that Selfhealing has to synthesize.

When all restoration trajectories for a given network are overlaid we obtain one plot called a *restoration space (RS) plot* as in Fig. 8.8. This form of performance representation is thought to be as characteristic of a given combination of network and restoration method as the 'eye' diagram is characteristic of a transmission system. The RS plot is formed by overlaying all the individual RTs obtained from cutting all spans in a network. Such a plot shows the performance envelope of the given network and restoration technique in terms of dimensions of prime concern; coverage (how many paths can be restored ?) and time (how long does it take?). Characteristics common to all restoration events in the network and the outliers that define the envelope of performance are both readily identifiable in the RS plot.

A variation on the RS plot for operational contexts is to apply a normalization on the horizontal axis converting it to *percent working circuits restored* (ie. N / W(u,v)) rather than number of paths completed (N). This may be of greater utility for planning and operations with real networks. The actual number of paths has more utility for making intrinsic speed/pathfinding comparisons between alternative methods under maximum path finding stress. The comparison of candidate network topologies and/or restoration methods with the RS plot makes it immediately apparent in what ways a network may be either topologically or speed limited, both concerns of the restoration planner. Further diagnostic insights which can be directly read from the RS and RT plots follow in Ch.'s 10 - 12.

8.3.2 Analysis of Sample Restoration Trajectories

Three particular restoration trajectories have been selected for discussion of Selfhealing dynamics with the aid of the RT plot. These are: (a) a case with relatively high path parallelism involving relatively few distinct routes (cut 2-8, Fig. 8.1, panel 7), (b) a case with relatively few paths realized over relatively many diverse routings (cut 7-8 ,Fig. 8.1, panel 10), (c) a case containing a large number of relatively long paths (cut 5-1, Fig. 8.1, panel 14). These three trajectories are shown together in Fig. 8.9 with the individual path routings indicated so that the time when individual paths are completed can be related to the topologies of the individual paths shown in Fig. 8.1.

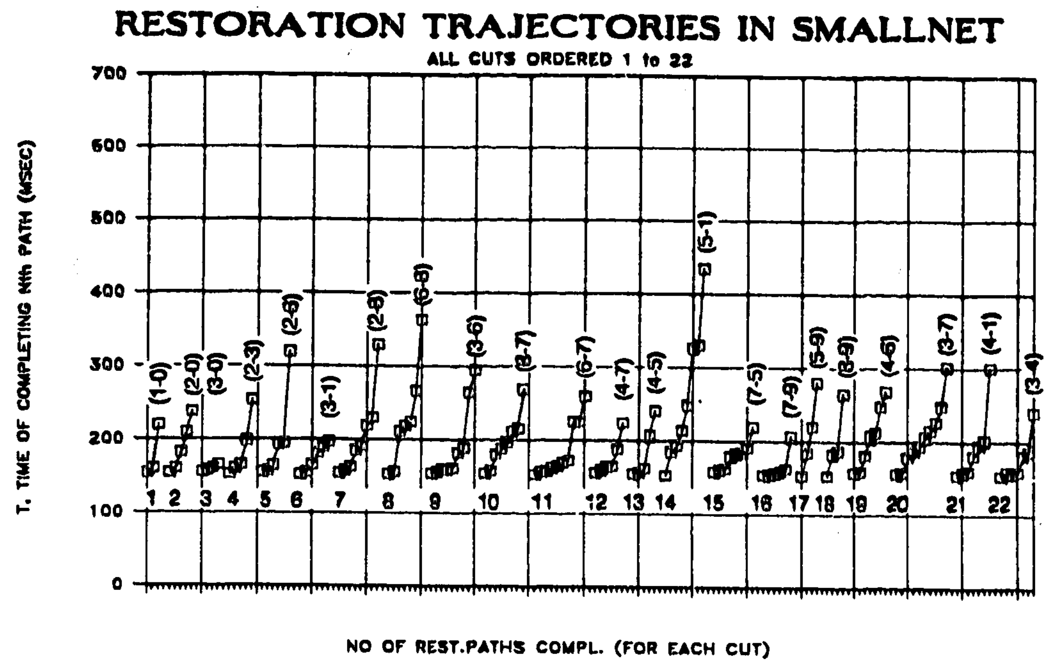


Fig. 8.7 Side by side display of all restoration trajectories in Smallnet (no. of paths axis repeats, starting at 1, at beginning of each individual trajectory).

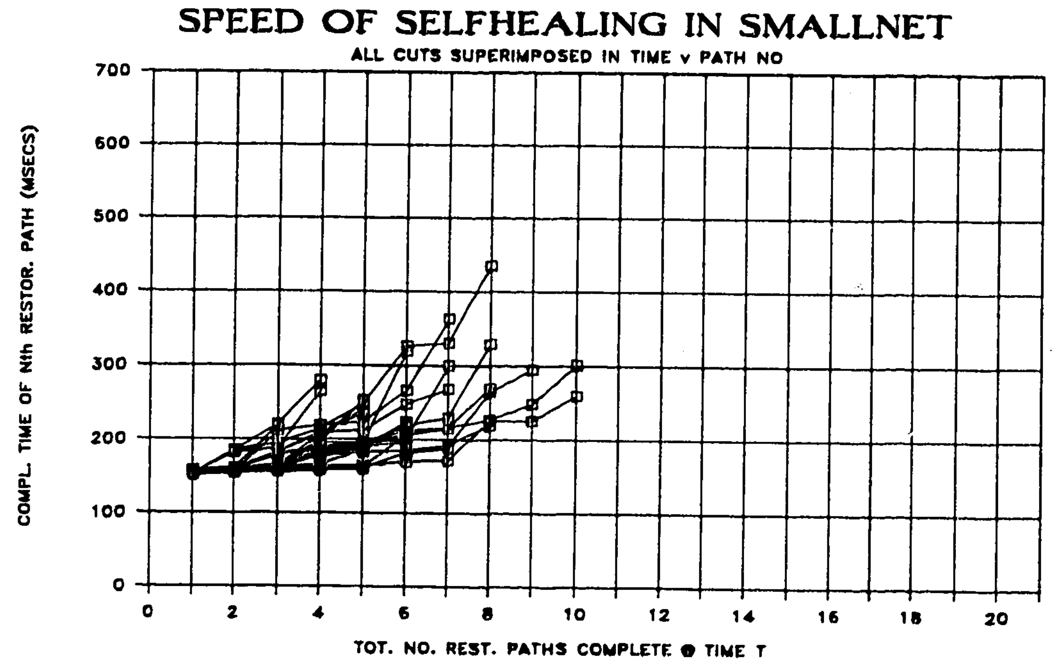


Fig. 8.8 Restoration space diagram for Selfhealing in Smallnet with RL=6, S=8.0, P=1.0 (overlay plot of all individual restoration trajectories).

The minimum time to establish the first path in each trajectory is relatively simple to explain. A fixed time of 100 msec is invested by the Sender/Chooser protocols to ensure complete alarm pickup before continuing. This holdoff realistically allows staggered alarm onsets to be collected before Sender flooding begins. The protocol can accept 'late arriving' alarms but it is more efficient if all alarmed circuits are known before beginning Sender flooding. Another contributor to the pedestal of time before even one route is found is the initialization activities required at Sender, Chooser and in at least one Tandem node before active signature processing begins. To complete the first path of any restoration, at least one Tandem node must be invoked, scan its node, build working lists and execute the Tandem logic protocol for the first time. In addition to processor times, assuming the first path found comes via only one Tandem site, a minimum of 4 one-way signature propagation times are required. The present Smallnet results assume 1,000 km spans, 8 kb/s signature transfer rate and a 64 bit signature field for a physical one-way span propagation time of 13 msec. Therefore the minimum number of signature exchanges for path establishment accounts for a further pedestal of about 52 msec. The remaining time to realize a first path consists of 3 to 5 msec of initialization and signature processing delay in the Sender and Tandem node for a total minimum time to complete the first path of around 155 msec. This breakdown gives us an early indication that in the above conditions (d=1,000 km, S=8 kb/s, P=1.0), processor speed contributes relatively little to the overall restoration times.

In the trajectory of spancut (8-2), we see that after the initial minimum time pedestal, all three paths on the simple route (2-6-8) come in rapidly. The incremental delay in accumulating these paths over the common route is 4.4 msec/path. About 23 msec later, the paths to be found on the route (2-3-7-8) come in. The first two have an incremental delay between them of about 7 msec. The increase with respect to the interarrival time of 4.4 msec on the route (2-6-8) is due to more intensive tandem-node processing events compared to the simple forward broadcast and linking returns performed by node 6 to realize the (2-6-8) paths. It is also likely that the span (7-3) experienced some TIB due to the competition of 5 original indexes from node 8, for the three spares on span (7-3). Indeed the extra delay in completion of the third path on route (2-3-6-8) of about 26 msec is close to the two-span extra delay that would result from index blocking of (7-3) until reverse linking was complete on one of the (2-6-8) paths.

The seventh path to come in is (2-3-6-8) which follows quite quickly, implying it was also just waiting for collapse of the residual meshes on the earlier-to-succeed indexes. The eighth path found is the long one (0-2-1-5-7-8). The time of this path and its peripheral routing imply that it encountered no index blocking or other interactions with other indexes but simply took a long time by virtue of total transit time for forward flooding and reverse linking propagation (about 130 msec here). The index that lead to this path would also have encountered a slight time penalty because it is the pioneer index along its front and has to wake up each Tandem node. Its overall path time is therefore relatively simply accountable by the addition of 10 signature transfer times and 4 Tandem node initialization times, although the latter are hardly significant.

It is instructive to compare this trajectory to that of spancut 6-7 (refer to Fig. 8.1, panel 11 and Fig. 8.5). Spancut (2-8) has 3-deep parallelism on two nested routes and we saw that the paths on (2-6-8) came in tightly grouped in time but the other set of three paths, on route (2-3-7-8), were dispersed in arrival by TIB. By comparison, spancut (6-7) presents three different two-span routes having, respectively, 1, 2, and 4 paths on each route. From the trajectory of spancut (6-7), we see an initial run of rapid path accumulation in which the 6 paths on 3 different length 2 routes are realized more quickly than 6 paths over two routes of length 2 and 3. The contrast shows that there is almost no difference in speed between path parallelism on one route and path parallelism on diverse routes of equal length. The time penalties come from route length more than number of different routes. The long flat runs of path accumulation such as we see in the trajectory of spancut (6-7) also imply that as the network size increases (while preserving the topology), we should generally see less than a linear increase in restoration time.

The restoration plan derived in response to spancut (7-8) is shown in panel 10 of Fig. 8.1 and its RT plot is in the centre of Fig.8.9. Cut (7-8) was selected for discussion because it is relatively rich in route diversity for the number of paths created: Selfhealing had to enlist and co-ordinate 6 Tandem nodes to establish 6 unique routes and 8 paths. One of the Tandem nodes (6) is simultaneously cooperating in 4 paths on 3 different routes. Notwithstanding these facts, the (7-8) restoration event is the fastest overall of the three trajectories selected in Fig.8.9, even though each restoration has 8 paths in total. This illustrates that in the present Smallnet conditions the time penalty for establishing numerous diverse *routes* is not as important as the length of the routes which are required, which are greater in the other two trajectories. Spancut (7-8) also provides a characteristic sample segment of trajectory line advancing under a relatively high route-finding load with numerous instances of TIB, mesh collapse and mesh expansion. This segment is provided by the first seven paths found. They come in intermixed and spaced with delays characteristic of link release from one reverse-linked index into the mesh of another index. The slope of this segment, as taken from the first to the seventh path completion, averages 10 msec per path.

The third trajectory in Fig. 8.9 is from spancut (5-1) includes three of the longest paths found in any Smallnet cut and has the highest total time of all restoration events in Smallnet. The topology of the corresponding solution is shown in Fig. 8.1, panel 14. The trajectory begins with a straightforward realization of the path (1-4-5) and two parallel paths over route (1-3-7-5) after experiencing only normal propagation delays. Path (1-3-4-5) completes next, after waiting for mesh collapse on the indexes of paths (1-3-7-5). Path (1-3-7-4-5) follows with a greater delay because it appears to have been index blocked on span (4-7) and only unblocked by the reverse linking that realized path (1-3-7-5) after which three span traversals still remained for its index to reach the Chooser. The next two paths to come in are dominated purely by their two-way propagation delays. These two long paths would have progressed from node 5 independently of all other signatures until briefly waiting for index collapse opportunities in the links going out of nodes 3 and 0. The final path found appears to have been blocked at span (3-6) by the index that created (1-3-7-4-5) and then by the index that created one of the two penultimate paths, each of which themselves had to await prior index collapses before their own meshes collapsed and finally let the second path on route (1-0-3-6-8-9-5) be realized. This trajectory of this spancut shows that, with present parameters, signature transfer delays tend to be significant when long paths are involved in the restoration event but we also note that times are significantly influenced by the sequence of blocking dynamics that occurs.

8.3.3 Characteristic RT plots of Various Restoration Schemes

Having looked in detail at some actual restoration trajectories from Selfhealing and having proposed that in general proposals for new restoration methods be compared in terms of restoration trajectories, we can now use the RT plot to show how the characteristics of Selfhealing differ fundamentally from two other classes of system for advanced network restoration. Fig. 8.10 shows three different generic trajectory characteristics. These are: (a) restoration via time-sequential single-path-at-a-time methods such as in [YaHa88], (b) Selfhealing and, (c) restoration via centralized download of preplans such as in [Flue87].

The Selfhealing trajectory is an arbitrary sample from the double-sized Smallnet network in Ch. 11. Because no performance data has been published on the other schemes, the trajectories shown for them are representative ones predicted by the author based on the fundamental principles of operation of those methods as they have been described in [YaHa88] and [Fluery87]. The point here is not to claim quantitative accuracy for those other methods but to convey a qualitative understanding of how these schemes differ fundamentally in kind. Nonetheless, to provide the most fair and meaningful comparison possible, the predicted trajectories for the other schemes are quantitatively constructed using the same assumptions of processor speed, signalling speeds, link distances and incremental crosspoint operate times that are used in the Selfhealing experiments.

Fig. 8.10(a) shows the predicted form of trajectory for the class of a distributed single-path-at-a-time scheme such as Bellcore's FITNESS proposal [YaHa88]. The path completions are relatively widely spaced in time and the interval between path completions increases with path number. The first path time is on the order of Selfhealing's the first path time. The inter-path time remains relatively constant while the method exploits all of the minimum length (2-span) paths which can be found. These are paths that Selfhealing tends to find very rapidly as a burst of

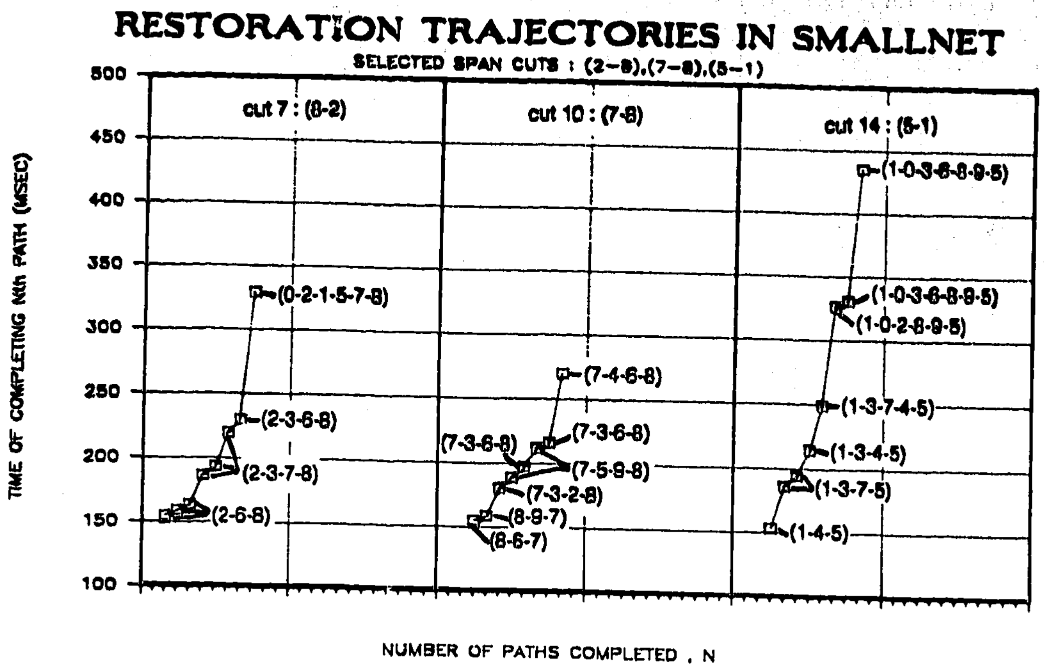


Fig. 8.9 Detailed dynamic structure of three restoration trajectories selected for discussion from Smallnet.

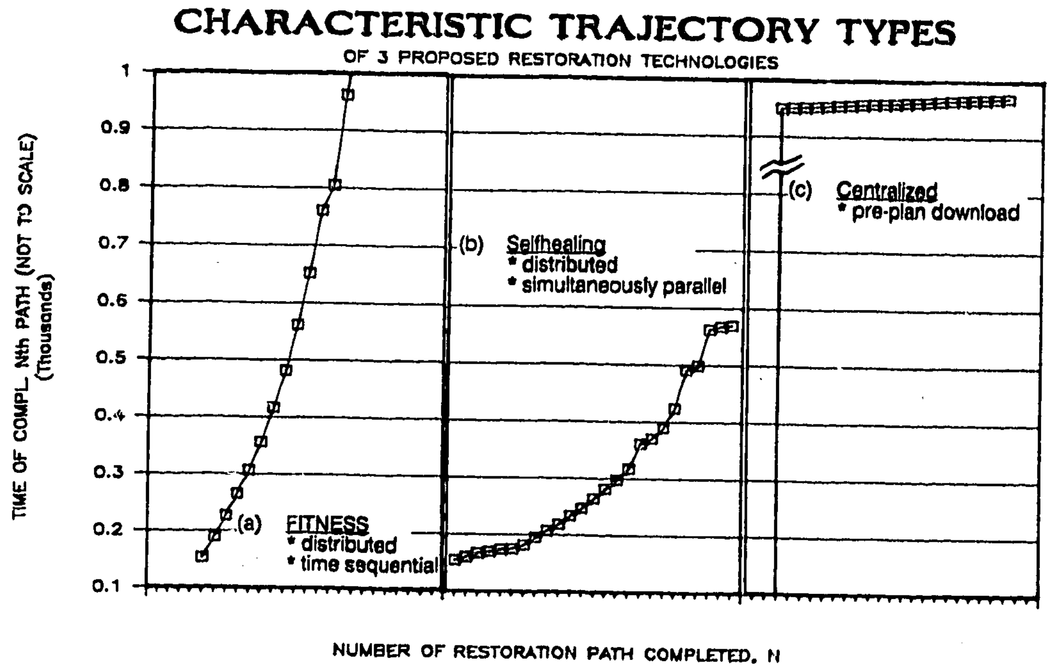


Fig. 8.10 Relative trajectory characteristics of three currently proposed restoration technologies : (a) Bellcore FITNESS, (b) Selfhealing, (c) Centralized download.

parallel path completions in little more time than it takes to find the first path. In FITNESS-like schemes, the inter-path completion time then grows as longer and longer individual paths are found, effectively raising the power of the function that describes total restoration time each time a longer routing is necessary to gain more paths. Selfhealing times also rise as longer *routes* are exploited but generally each time a longer route is established, all of the paths on it are realized in another relatively rapid burst and all paths on several different routes can be exploited simultaneously. More importantly in Selfhealing the time investment to realize a long route begins from fault onset, not from the time of completing the previous path in sequence.

Fig. 8.10 (c) shows the characteristic trajectory predicted for the class of restoration schemes which use pre-plan download. Their characteristic is expected to be a relatively long time before even one path is realized, followed by a rapid series of individual path completions as the last node is downloaded and the plan is executed. The final path completion phase can be very rapid but it is feared that in practise the entire burst of path completions is likely to occur *well after* the call-dropping threshold has passed. In addition, there may be no restoration trajectory at all if the centralized database or control centre is not available or if the spancut to be restored also took out telemetry to one of the end nodes. In comparison the typical Selfhealing trajectory is generally more complex and variable with a diversity simultaneous individual route-finding dynamics. Various regions of the Selfhealing trajectory have local resemblances to the other two alternatives but in general the Selfhealing trajectory reflects greater parallelism than the time-sequential distributed schemes and reflects greater speed than centralized control schemes.

8.3.4 A Trace Experiment to View Network-level Selfhealing Action

As a means to help convey and consolidate the concepts by which Selfhealing works at the network level, a specially instrumented spancut was performed in Smallnet. In this particular experiment a log file of every signature occurring on every link through time was kept through the dynamic phase of the Selfhealing event. Special filter programs were then used produce a picture of the signature state of every link at each node in the network as time advances. Other filters were applied to abstract the following data from the network as a whole as a function of time: (a) total volume of signatures in the network: This is the total number of individual links in the network having a non-null signature at the sampling time. (b) the total number of unique surviving indexes in the network: This is the total number of different indexes found amongst the volume of signatures extant in the network at the sampling time. (c) the furthest range of any signature from the Sender: This is the maximum repeatcount found on any signature anywhere in the network at each sample time. (d) total number of links in the mesh of each index: This is the total number of links found included within the mesh constructed by each index issued by the Sender at each sample time.

Smallnet spancut (2-3) was selected for presentation in this manner. To control the total number of competing indexes, the cut was performed with only four working circuits on the span. A priori inspection of span (2-3) shows that node 3 will act as Sender and the total number of indexes created under these circumstances will be 12, making it manageable to represent graphically the extent of the mesh of each individual index. The spancut was run with the same parameter values (S, P, RL) as the Smallnet results above.

Results: The restoration plan that obtained is identical to that shown in Fig. 8.1, panel 4, except that the length 3 path (3-7-8-2) is not needed with W(2,3) = 4 and is not created. Fig. 8.11 shows the total number of links in the mesh of the indexes 1 through 12 as a function of time after the Sender node begins to execute. Fig. 8.12 shows the total number of all signatures in the network versus time. Fig. 8.13 shows the total number of surviving indexes in the network as a function of time. Fig. 8.14 shows the furthest reach of any signature from the Sender node against time. For technical reasons, the plotted time in the above figures is the actual time since failure onset less 90 msec of the 100 msec alarm holdoff time. The effective onset of the event is therefore at 10 msec in these data. In Fig.s 8.11 and 8.12, we can see at time t= 20 that Sender flooding has placed one instance of each index in signatures on the 12 spare links leaving its site. At t=60, nodes 0, 4, 1, 6, 7 have all been invoked and executed the Tandem logic rules. Some indices are able to expand up to another five links in total at this stage.

8.4 Summary

The all span cuts study of Smallnet under maximum path finding stress has shown a Path Number Efficiency of 100% in all cases. Every spancut in the network was restored to the fullest extent topologically possible in an average of 269 msec and an observed maximum of 451 msec; well before the call dropping threshold. Path lengths are equivalent to those that would be obtained by an algorithm that selected the k-shortest link-disjoint paths, except where TIB effects extended these lengths when a uniform RL of 6 was employed. With the uniform RL of 6, Selfhealing exhibited a PLE of 98.75% for all cuts in Smallnet. If the RL is selectively reduced, 100% PLE can be obtained except for spancut (5-1) where the early path choice effect is involved.

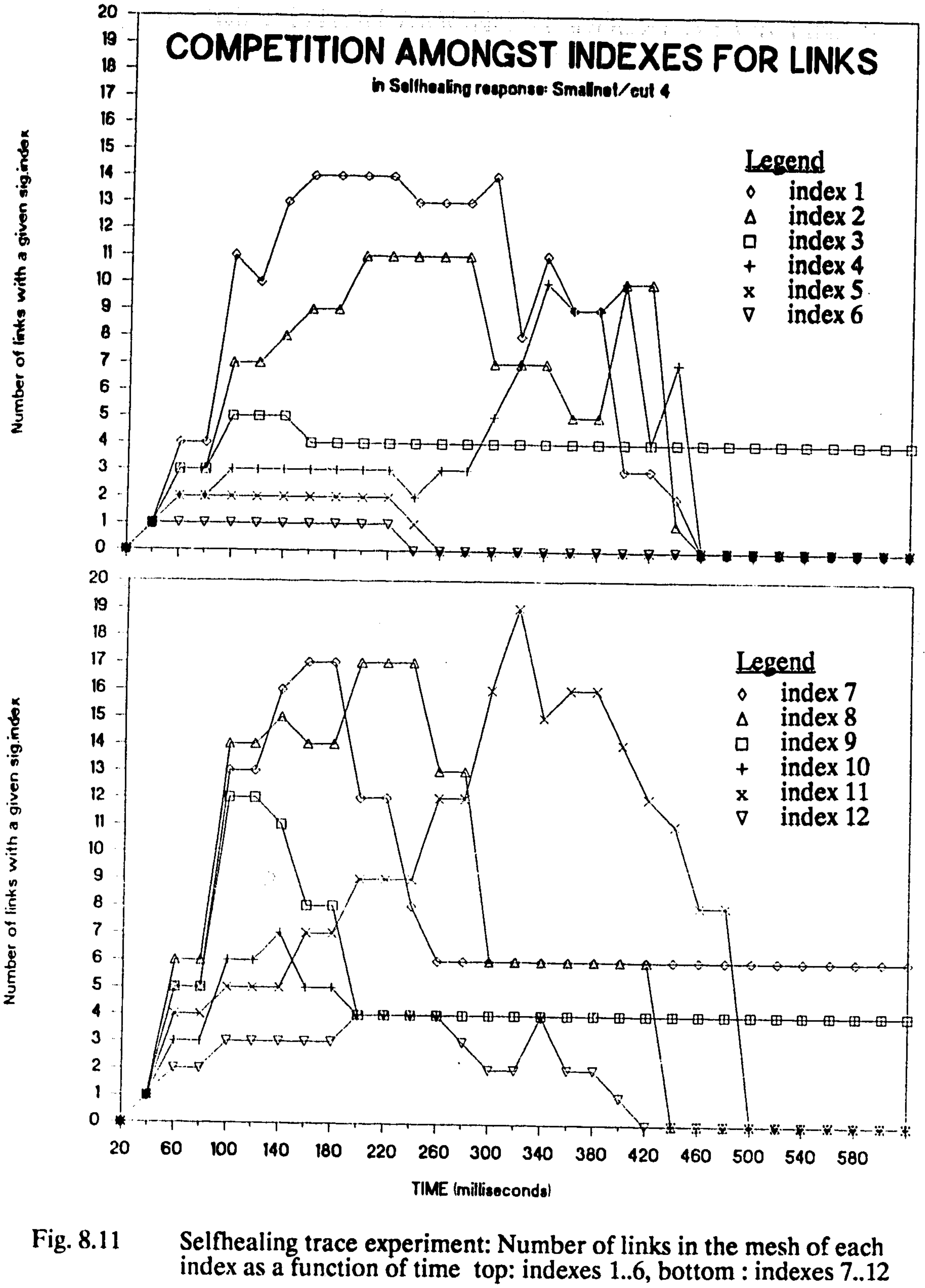


Fig. 8.11 Selfhealing trace experiment: Number of links in the mesh of each index as a function of time top: indexes 1..6, bottom : indexes 7..12

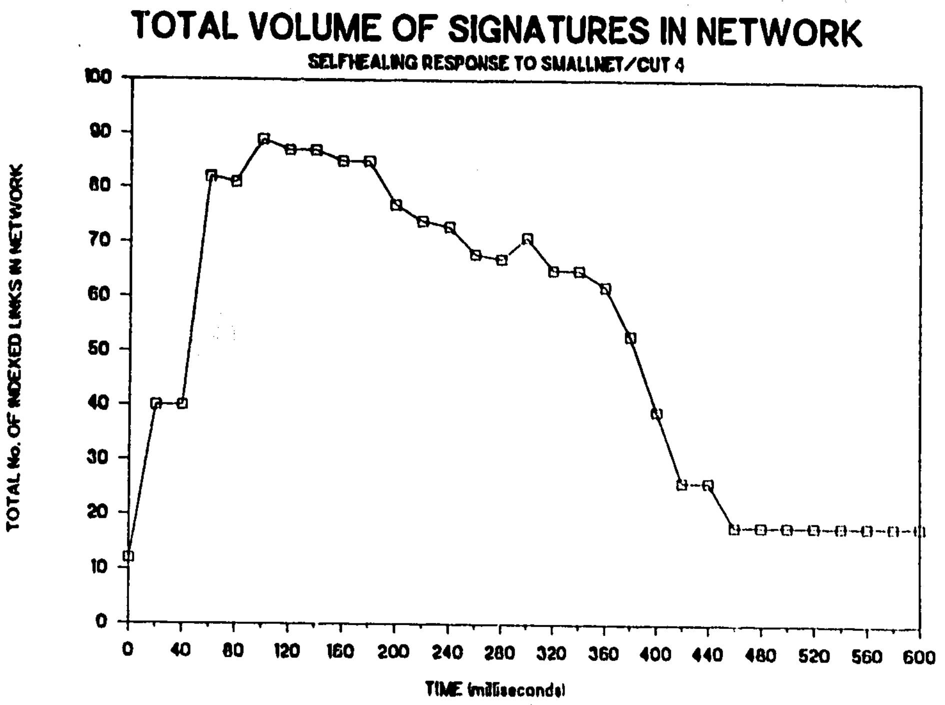


Fig. 8.12 Selfhealing trace experiment: Total number of signatures in network as a function of time

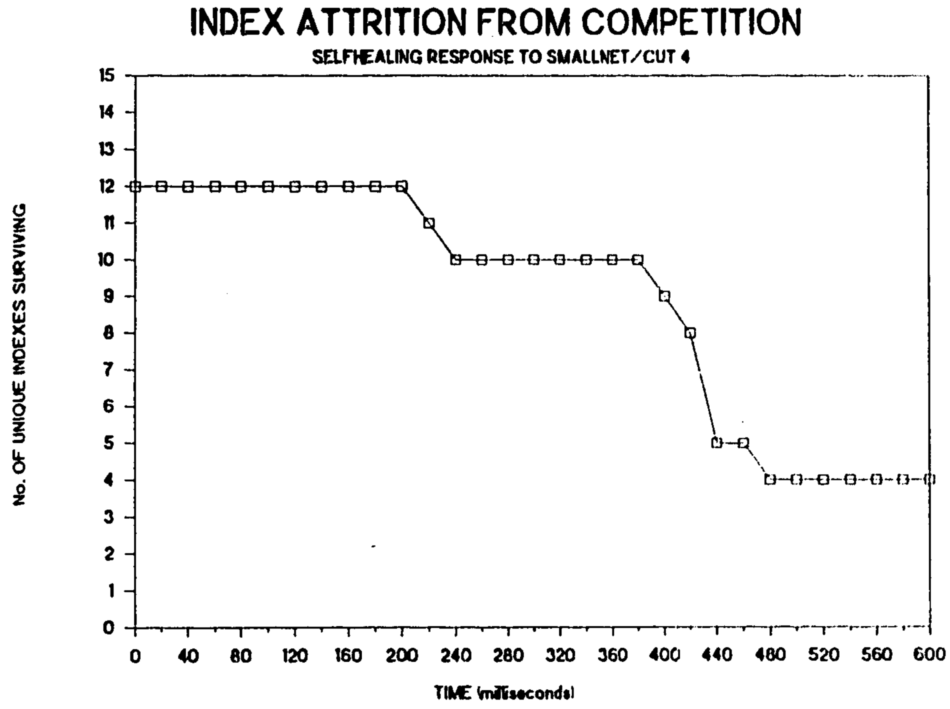


Fig. 8.13 Selfhealing trace experiment: Number of unique surviving indexes as a function of time

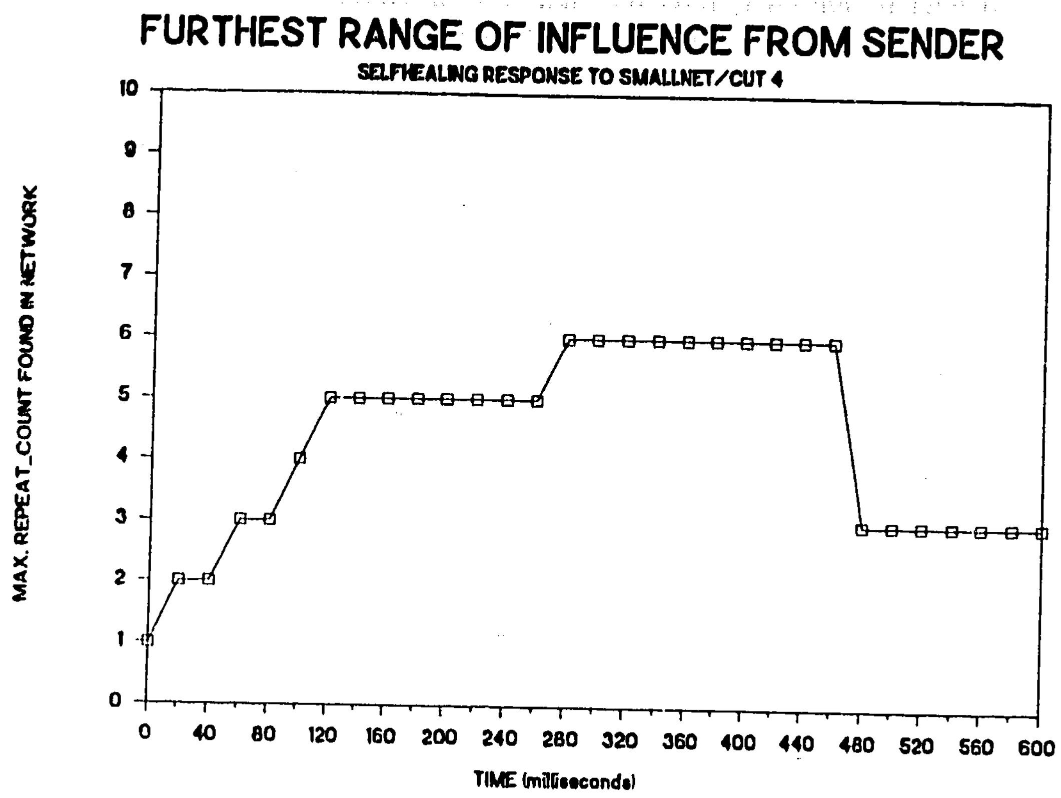


Fig. 8.14 Selfhealing trace experiment: Farthest extent of influence as a function of time

CHAPTER 9

RESULTS IN BELLCORE, METRO AND US STUDY NETWORKS

The previous Smallnet network served primarily as a study network that was manageable to work with when developing the protocol. The relative ease of conducting all-span-cut experiments, plotting and diagnosing the results of network behavior in Smallnet facilitated the numerous developmental experiments required to reach the present protocol specification. However, because the verification and characterization of the protocol is experimental in nature, it is important to empirically verify the correctness, sanity, and performance of the protocol in as many other representative network models as could be obtained. We now focus on three of these additional study networks, referred to as *Bellcore*, *Metro* and *US*.

9.1 Properties of the Study Networks Employed

Ch. 2 described some of the real networks for which Selfhealing is intended. The *Bellcore*, *Metro* and *US* network models of this chapter were designed or obtained to reflect the relevant characteristics of these real networks. Two of these networks, (US and Metro) are essentially the same networks used in the 1987 publication on this work [Grov87b]. Some differences apply however in the US and Metro network models used in the present work. The third (Bellcore) is based on additional data which only became available in November 1988 as a result of Bellcore's entry into this area of study.

Both of the former networks are now used in the maximum path finding condition wherein the number of working circuits is effectively infinite. The 1987 published results which use these networks assumed an arbitrary quasi-random distribution of working circuit quantities on each span. Since then it has been considered more meaningful and defensible from a research viewpoint to eliminate unnecessary assumptions about working circuit quantities and report the maximum path finding abilities of the method.[1] The Bellcore network however provides us with a data set based in the real world for which operational metrics can be assessed.

Since publishing the Metro and US models in 1987, feedback from industry members and statistical comparison to details of Telecom Canada's network has confirmed the suitability of these networks as realistic test cases. However, because of improvements to the protocol since last reporting [Grov87b], and because of the shift from the stepped-concurrent PC emulator to the truly asynchronous emulator, the US and Metro network results now presented are *not* necessarily identical to the previous published results. The node identification convention has also been changed from letters to numbers, starting at zero, to accommodate certain technical aspects of software reused in implementing the advanced (SUN workstation) emulator. The basic topological parameters of each of these networks are given in Table 9.1

Table 9.1 Properties of Study Networks

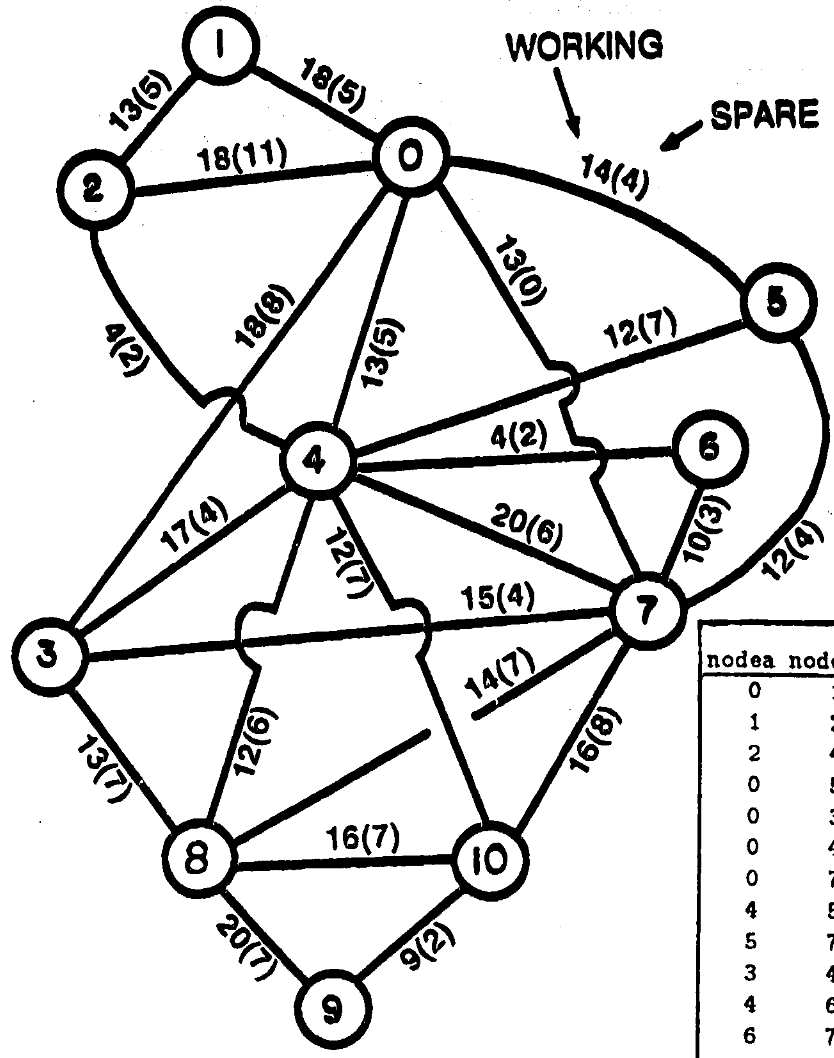
	no. nodes	no. spans	average span connectivity
Bellcore	11	23	4.18
US	28	47	3.36
Metro	25	55	4.4
Smallnet	10	21	4.2

9.1.1 Bellcore Network Model

Fig. 9.1 shows the Bellcore network model. Each node is identified by a node number and the quantities of working and spare circuits on each span are indicated by the ordered pair of numbers on each span. Span distances are to scale in the drawing and are as listed (in kilometers) in the inset. The data for this network became available as a result of Bellcore's interest in the present work and their own subsequent work on the FITNESS restoration proposal. This is the study network which they used as a study vehicle in [YaHa87]. They indicate that the network is based on a real LATA of one of Bellcore's US member companies but do not disclose exactly which locale is modelled. This network is one of the few examples available of a real network with specified ratios of working to spare DS-3 circuit quantities. Perhaps surprisingly, it was found in the early development of the SH protocol that correct operation in conditions of *less-than-maximum* path finding stress could be a problem. Therefore realistic test conditions in which not all of the possible paths are required is a welcome addition to the portfolio of test cases. The Bellcore network is also of value to give at least one example of traffic restorability with working/spare ratios and network configurations that were *not* devised by the present investigator. We also speculate that if new investigators continue to enter this field, Bellcore's published study network could become a common benchmark on which to compare results.

9.1.2 US Continental Study Network

Fig. 9.2 shows the US transcontinental study network. This network is inspired by the topology of real fiber networks as charted by Kessler [Kess86,88] and reproduced in Plate 3 (Ch.2). Two spans have been added to the US network model compared to the model shown in [Grov87] to reflect increased route density in the eastern seaboard area as apparent in [Kess88] with respect to [Kess86] which had been previously used. (These are spans (15-23 and 21-23).) In the US network each span has a uniform random number between 1 to 8 spare DS-3 circuits assigned to each span. Working circuit quantities are assumed to be infinite so that maximum path finding stress applies.



nodea	nodeb	distance
0	1	20
1	2	20
2	4	40
0	5	50
0	3	75
0	4	35
0	7	60
4	5	55
5	7	40
3	4	35
4	6	45
6	7	15
0	2	30
4	7	40
3	7	70
8	4	45
8	7	60
3	8	25
8	10	30
8	9	18
9	10	20
7	10	30
4	10	50

Fig. 9.1 Bellcore study network model (*Bellcore net*)

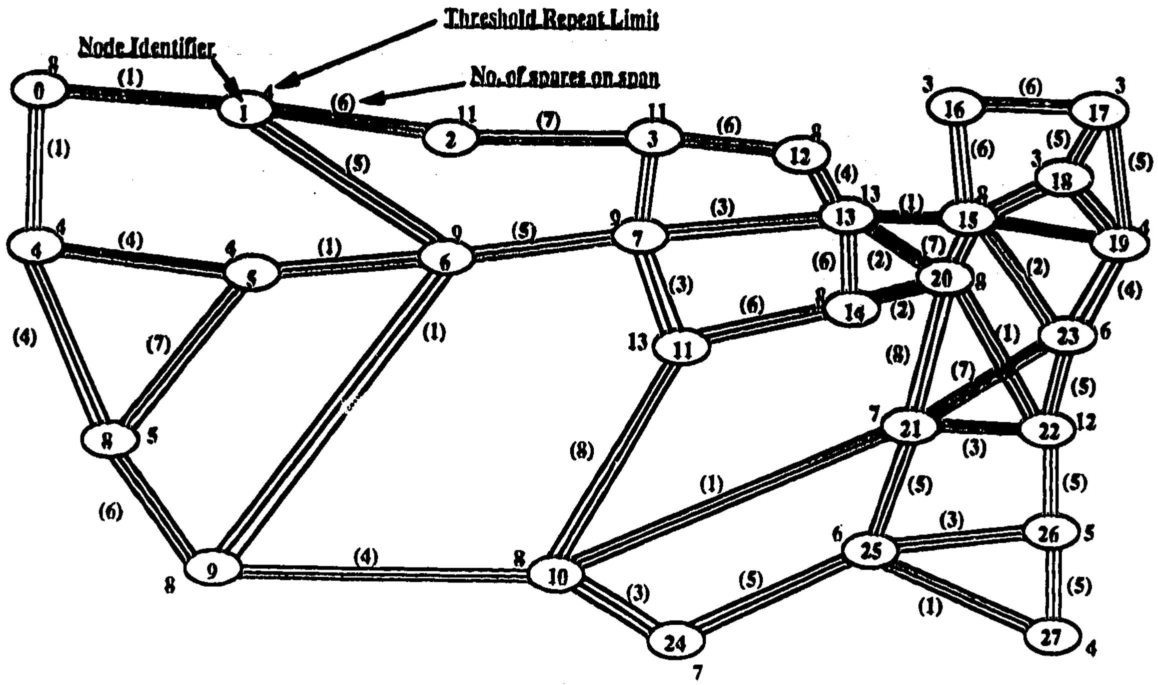


Fig. 9.2 Continental US study network model (*US net*)

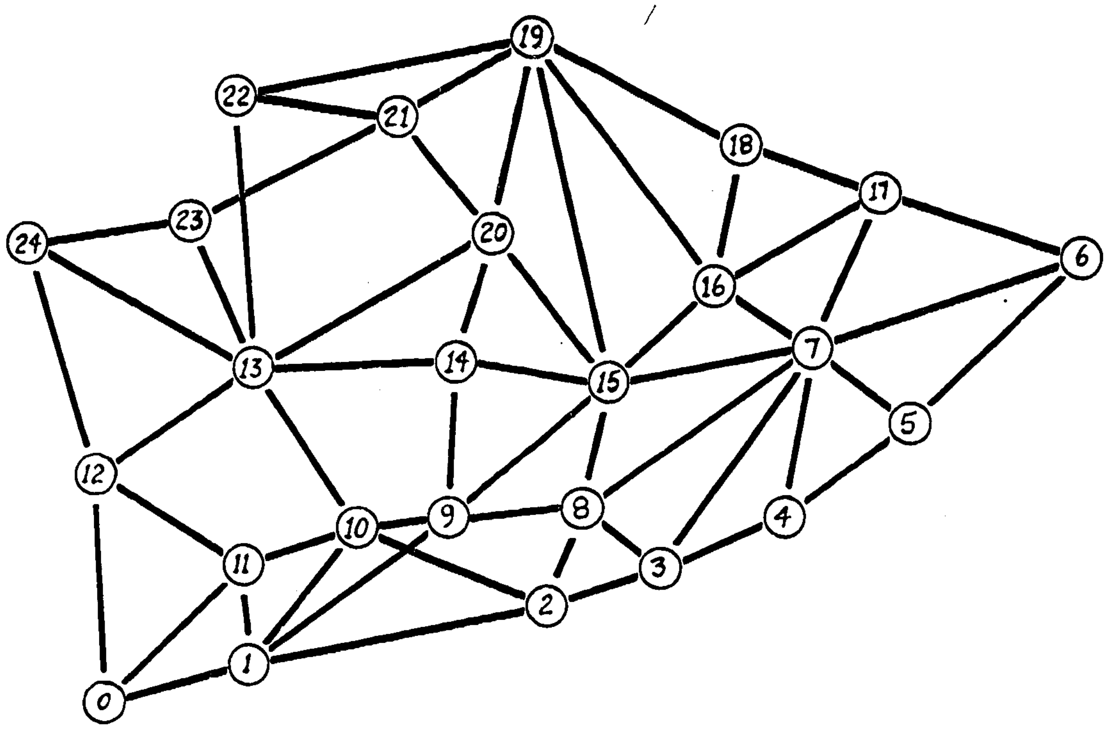


Fig. 9.3 Metropolitan study network model (*Metro*)

The US network model (Fig. 9.2) was used with d =1,000 km on every span, S=8 kb/s and P=1.0. This is the most geographically and logically extensive network model. Paths of up to 13 spans are theoretically possible for restorations in this network although such long reroutings would probably not be permitted in practise. However, because routing lengths range from 13 to as low as 3 for topologically-limited performance in this network, we decided to apply Selfhealing in the US network model using a *node-specific RL plan*, rather than the usual constant RL value. The repeat limit values that will be effected by each node when it acts as Sender is noted by the unbracketed number beside each node in Fig. 9.2.

9.1.3 Metropolitan Study Network Model

Fig. 9.3 shows the metropolitan study network. It is based on an approximate fiber network topology for Toronto in the mid 1990's as fitted together from public domain [AbRi86],[Bour86] and private sources. This network has a special property that is of realistic interest to planners as a possible strategy for providing minimum redundancy. The strategy is to use only those circuits which are presently equipped as spares for span protection switching as the spares for a DCS-based Selfhealing network. In this scheme no new spare facilities would have to be provisioned until future growth warranted. To convert the existing network plan to Selfhealing, the presently dedicated span protection switching spares would be terminated directly on the DCS equipment and the protection switching bays eliminated. This has the benefits of improved overall protection against single circuit electronic failures and complete span failures due to cable cuts since every spare is now accessible in principle for rerouting traffic from any other span.

From a research viewpoint this type of network (one spare per span) is also a special case of interest to characterize because it demands maximum route finding performance for each path that is found (in generalized networks every route may yield several paths). It also represents the possible planning policy of always having a constant number of spares on every span. For instance as the Metro net shown in Fig. 9.3 grows, there might come a time where *exactly two* spares are provided on every span. It is the aspect of a constant number of spares per span, whether one or more, that is of interest to Selfhealing because in these conditions every span is *matched* in size to every other span that it might be concatenated with in restoration. This means that index blocking dynamics may be expected to differ somewhat because the spares on a span will generally be entirely seized or entirely free when any given index arrives. Because network connectivity influences restoration, the US and metro network models were selected to give the extremes of average span degree shown in Table 9.1 and these extremes slightly exceed the range of average connectivities seen in the real networks considered in Ch. 2.

9.2 Results - General

In the following results a signature length of 64 bits is used with signature transfer rate, S= 8.0 kb/s and processor speed P = 1.0. Repeat limit and span distances vary with each network. A carrier velocity of 5 microseconds/ km is assumed. An alarm collection holdoff of 100 msec applies and the Sender role is always adopted by the end node with *max[ord(namea), ord(nameb)]*.

The complete results of the response to each span cut in these networks are presented in Tables 9.2, 9.3, and 9.4. Table 9.2, for the Bellcore network, explicitly lists the ideal (MetaDijkstra) path set as well as the path set obtained with Selfhealing and shows the corresponding PNE and PLE values for each cut. Because relatively few restoration plans actually differ from the ideal, this lengthy form is not repeated in Tables 9.3 and 9.4 for the Metro and US networks. The latter tables list the Selfhealing restoration plan only, and wherever the Selfhealing restoration is *not identical* to the ideal solution in the latter two tables, the details are given in the text.

9.3 Bellcore Study Network Results

The finite working circuit quantities of the Bellcore net do not in general force topologically-limited path finding. Table 9.2 was obtained using RL =8. Results show this is twice as high as it needs to be for this network because no ideal path is greater than length 4 for any span cut. Table 9.2 contains the following entries from left to right in columns (a) to (e): (a) identity of the span that is cut, (b) the number of working circuits on the span, (c) a listing of each distinct routing found in the MetaDijkstra restoration plan, (d) the number of individual paths on each route in the ideal plan, (e) the ratio of working circuits lost to paths restored. The next two columns (f,g) detail the Selfhealing restoration plan and the percent span restoration achieved, in the same manner as columns (c,d,e). PNE and PLE columns are as previously defined. Operational PNE applies in this case. The last column details the time of the first restoration path found (t_{p1}) and the time of the last restoration path found (t_R).

9.3.1 Path Performance

Table 9.2 shows that all spancuts in the *Bellcore* network are either 100% restored or restored to the fullest extent topologically possible, producing an (operational) PNE value of 100%. Fig. 9.4 shows the distribution over all cuts of the individual path lengths and compares them to the ideal path length distribution found from MetaDijkstra seeking the given number of working circuit restoration paths. The overall PLE 97.55%. Of 237 individual paths created, 20 were longer than the corresponding paths in the ideal solution for a SPP of 91.6 %. 19 of these paths were in excess of the MetaDijkstra path length by one span, and the 20th was in excess of the ideal by two spans. The redundancy of the Bellcore network model (121 spare / 313 working circuits) is 38.66 % and the average restorability of traffic in the network is 75.77 %. 12 out of the 23 spans are 100%

Table 9.2 Results of Selfhealing in Bellcore Study Network (RL = 8, S = 8 kb/s, P = 1.0)

SPAN CUT	Paths Lost	Theoretical Ideal			Selfhealing					
		Route	No Paths	% Rest.	Route	No Paths	% Rest.	PNE	PLE	Time [1..N] (msec)
1-0	18	0-2-1	5	27.78	0-2-1	5	27.28	1.0	1.0	137..167
2-1	13	1-0-2	5	38.46	1-0-2	5	38.46	1.0	1.0	138..218
4-2	4	2-0-4	4	100.0	2-0-4	4	100.0	1.0	1.0	138..177
5-0	14	0-4-5	5		0-4-5	5				
		0-2-4-5	2		0-3-4-5	1				
		0-3-7-5	4	78.57	0-2-4-5	1				
					0-3-7-5	4	78.57	1.0	1.0	138..306
3-0	18	0-4-3	4		0-4-3	4				
		0-4-7-3	1		0-4-7-3	1				
		0-5-7-3	3		0-5-7-3	3				
		0-2-4-8-3	2		0-5-4-8-3	1				
		0-5-4-8-3	1	61.11	0-2-4-8-3	1				
					0-2-4-10-8-3	1	61.11	1.0	32/33	141..462
4-0	13	0-2-4	2		0-2-4	2				
		0-3-4	4		0-3-4	4				
		0-5-4	4		0-5-4	4				
		0-3-7-4	3	100.0	0-3-7-4	3	100.0	1.0	1.0	135..254
7-0	13	0-3-7	4		0-3-7	4				
		0-4-7	5		0-4-7	4				
		0-5-7	4	100.0	0-5-7	4				
					0-3-8-7	1	100.0	1.0	26/27	139..360
5-4	12	4-0-5	4		4-0-5	4				
		4-7-5	4	66.67	4-7-5	4	66.67	1.0	1.0	135..191
7-5	12	5-4-7	6		4-5-7	6				
		5-0-3-7	4		5-0-3-7	3				
		5-4-6-7	1	91.67	5-4-6-7	1				
					5-0-4-8-7	1	91.67	1.0	36/37	133..365

SPAN CUT	Paths Lost	Theoretical Ideal Route	Theoretical Ideal No Paths	Theoretical Ideal % Rest.	Selfhealing Route	Selfhealing No Paths	Selfhealing % Rest.	Selfhealing PNE	Selfhealing PLE	Time [1..N] (msec)
4-3	17	3-0-4	5		3-0-4	5				
		3-7-4	4		3-0-2-4	2				
		3-8-4	6		3-8-4	3				
		3-0-2-4	2	100.0	3-8-7-4	3				
					3-0-5-4	1				
					3-7-5-4	3	100.0	1.0	36/43	139..294
6-4	4	4-7-6	3	75.0	4-7-6	3	75.0	1.0	1.0	135..146
7-6	10	6-4-7	2	20.0	4-6-5-7	2	20.0	1.0	4/6	133..189
2-0	18	0-1-2	5		0-1-2	5				
		0-4-2	2	38.89	0-4-2	2	38.89	1.0	1.0	118..144
7-4	20	4-3-7	4		4-5-7	4				
		4-5-7	4		4-6-7	2				
		4-6-7	2		4-3-7	3				
		4-8-7	6		4-8-7	6				
		4-10-7	4	100.0	4-10-7	5	100.0	1.0	1.0	139..259
7-3	15	3-4-7	4		3-0-5-7	4				
		3-8-7	7		3-4-7	1				
		3-0-4-7	2		3-0-4-7	4				
		3-0-5-7	2	100.0	3-8-7	6	100.0	1.0	34/38	158..352
8-4	12	4-3-8	4		4-7-8	6				
		4-7-8	6		4-3-8	4				
		4-10-8	2	100.0	4-10-8	2	100.0	1.0	1.0	141..163
8-7	14	7-3-8	4		7-4-8	6				
		7-4-8	6		7-3-8	4				
		7-10-8	4	100.0	7-10-8	4	100.0	1.0	1.0	136..250
8-3	13	3-4-8	4		3-4-8	4				
		3-7-8	4		3-7-8	4				
		3-0-4-8	2		3-0-4-8	2				
		3-0-4-7-8	3	100.0	3-0-4-7-8	1				
					3-0-5-7-8	2	100.0	1.0	1.0	138..361

SPAN CUT	Paths Last	Theoretical Best			Selfhealing						
		Route	No Paths	% Rest.	Route	No Paths	% Rest.	PNE	PLE	NNU	Time [1..N] (msec)
10-8	16	8-4-10	6		8-9-10	2					
		8-7-10	7		8-7-10	7					
		8-9-10	2		8-4-10	6					
		8-3-4-10	1	100.0	8-3-4-10	1	100.0	1.0	1.0	1.0	138..288
9-8	20	8-10-9	2	10.0	8-10-9	2	10.0	1.0	1.0	1.0	137..141
10-9	9	9-8-10	7	77.78	9-8-10	7	77.78	1.0	1.0	1.0	135..240
10-7	16	7-4-10	6		7-8-10	7					
		7-8-10	7		7-4-10	3					
		7-3-4-10	1		7-5-4-10	2					
		7-3-8-9-10	2	100.0	7-3-4-10	1					
					7-3-8-4-10	1					
					7-4-8-9-10	2	100.0	1.0	37/41	7/6	136..465
10-4	12	4-7-10	6		4-8-10	6					
		4-8-10	6	100.0	4-7-10	6	100.0	1.0	1.0	1.0	135..259
		Theoretical Network Restorability	75.7%		Selfhealing Restoration	75.7%		Overall PLE	= 97.55	Overall NNU	= 1.08

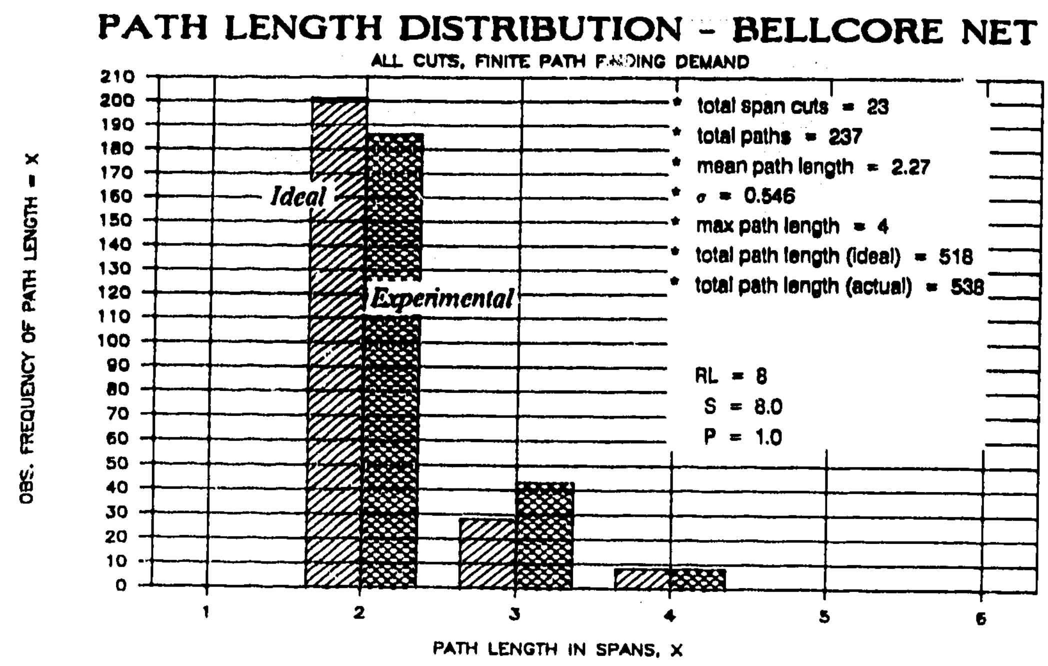


Fig. 9.4 Distribution of ideal and experimental path lengths in Bellcore net (RL=8, S=8.0, P=1.0).

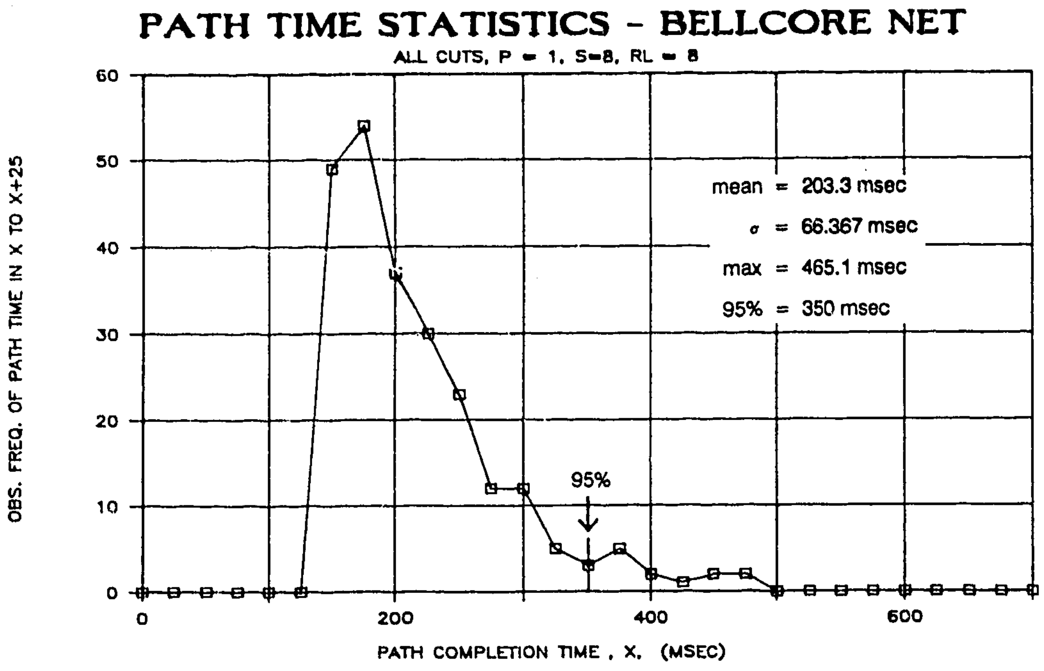


Fig. 9.5 Distribution of individual path completion times in Bellcore net (RL=8, S=8.0, P=1.0).

restorable but some spans are as little as 10% restorable suggesting that the placement of the spare capacity in this network has probably not been optimized but is simply the outcome of normal growth and provisioning activities within the network from which Bellcore abstracted this model.

Let us now consider the cases in which Selfhealing derived a restoration plan with paths which were in excess of the ideal paths for a k-shortest paths algorithm. In all there are 7 cases: (3-0), (7-0), (7-5), (4-3), (7-6), (7-3) and (10-7). To assist in interpreting these cases a second set of results was obtained with RL reduced to 4 as a means to illuminate which excess path lengths are due to TIB effects when RL = 8 , a relatively high value for this network. RL = 4 was chosen because it is the lowest value that can be uniformly applied in this network without reducing PNE from 100%. This may seem contradictory because spancut (3-0) resulted in a length 5 path in Table 9.2. However, this path is overlength by virtue of TIB in the presence of a high RL. It reduces to length 4 when RL=4 so that 100% PNE is indeed realized.

When the RL was reduced to 4, the ideal solution for spancuts (3-0) and (7-5) were obtained. In (4-3) one of the length 3 overlength paths was reduced to length 2 but 6 other paths which ideally have length 2 in that solution remained at length 3. Span (10-7) also showed a reduction of one length 4 path into an ideal length 2 path, but two paths of length 3, which should ideally be of length 2, remained. All other solutions remained topologically unchanged when the RL was reduced from 8 to 4. In summary, reducing the RL to 4 from 8 perfected 2 of the longest solutions and improved another 2. With RL= 4, 5 non-ideal solutions remain and they include 14 individual paths which are each overlength by one span. The remaining overlength paths are all paths of length 3 that should ideally be of length 2. All but three of these overlength paths are attributed to remaining TIB effects at the reduced RL of 4. The relatively high number of overlength paths and the TIB effects persisting even for length 3 paths is partly due to the short span distances in this network, as we shall see in the next section.

9.3.2 Path Spreading Effect

Three of the cases where overlength paths occur, (10-7), (7-3) and (4-3) are not attributable to TIB as seen in Smallnet but to a new phenomenon we call *path spreading.* It results in Bellcore because signature transfer times are now much smaller, with respect to nodal processing times, than they were in Smallnet. Span distances of only 10's of km apply in Bellcore as opposed to 1,000 km. We will use these three cases to introduce and explain this phenomenon.

Consider cut (10-7) with reference to Fig. 9.1. The ideal solution and the actual solutions with RL =4 compare as follows:

Route	**Ideal # paths**	**Actual # paths**
(7-8-10)	7	7
(7-4-10)	6	4
(7-3-4-10)	1	1
(7-5-4-10)	0	2
(7-3-8-9-10)	2	1
(7-4-8-9-10)	0	1

The significant difference is that two of the six paths which the ideal plan places on the route (7-4-10) turn up in the actual plan routed over the route (7-5-4-10). All other routing differences are equivalent from a PLE viewpoint. In this case the overlength routing of these paths came about as follows: Node 10 acts as the Sender for spancut (10-7). Because of its position adjacent to node 10, node 4, which is also the largest node in this network, bears the load of managing Selfhealing processing for 39 spares on 8 distinct spans very shortly after onset of the (10-7) spancut. By comparison node 5 has a total signature processing load of 15 ports on three distinct spans. With the very short span distances in Bellcore net, it becomes possible for a signature from node 10 to be rebroadcast by node 4 , transit node 5 and arrive at the Chooser (node 7) before the last index subjected to the rebroadcast burden of node 4 was completely processed. Since the signature transfer time over a link of 50 km is only 8.3 msec (S = 8 kb/s) and processing times of 3 to 5 msec per span are observed, it is possible, depending on the detailed construction of working span lists at nodes 4 and 5, for a signature from node 10 to be immediately repeatered onto span (4-5) and also be repeatered onto span (5-7) while node 4 is still processing the same index with a local span list that happens to have (4-7) at its end.

The general effect here is that when processing times become on the order of signature transfer times, the chance order of processing indices in the protocol can be factor in the competition amongst indices and specifically, can serve to advance a given mesh faster over a long path than it advances over the geographically shorter path, even though the shorter path *is not blocked* as in normal TIB. In other words, in regimes of low d/P ratio, path solutions may tend to leak or spread out from the ideal set of shortest paths. Imagine the introduction of a single index into a network and allowing it to expand without interaction from other indices. With limit of long spans and fast processors (high d/P) it is clear that the network offers a uniform impedance to the advance of the index front in all directions. If we reverse the extremes to make nodal delays significant and link delays very small (low d/P) then the network becomes 'lumpy'. Large nodes will tend to be slower than small nodes and index meshes will tend to flow around them. Chance dependencies on the internal order of processing events at a node will begin to manifest themselves externally in the network. Of course such effects would never be seen if one ignored processor execution times in the dynamics as is done in many analytical treatments of distributed system (eg. [MoBh86]).

To test the hypothesis of path spreading due to a low d/P ratio in the above results, two further all-spancuts experiments were performed on Bellcore net with the observation that the general effect of interest is really dependent on the STT/(1/P) = STT*P product where STT = d/v + nbits/fsig is the complete transfer time: (1) The STT*P product was increased by changing S to 1 kb/s, with P = 1.0, (2) The STT*P term was increased by changing P to 10.0 with S = 8 kb/s. Both cases use RL = 4. When the outcome of these two experiments are added to the results of Table 9.3 (RL = 8) a complete summary shows a systematic behavior in terms of PLE that is consistent with our understanding of TIB and path spreading effects: (PNE = 100% in all cases).

Case	RL	S	P	Number of cuts with PLE < 100%
1	8	8	1	7
2	4	8	1	4
3	4	1	1	0 (ideal)
4	4	8	10	0 (ideal)

In case 1 (which is Table 9.3) we now know that a combination of TIB-induced effects and pure path spreading effects contributed to cause 7 restoration plans having excess path length. Reducing RL to a more appropriate value for this network (RL=4, case 2) eliminated most TIB effects on path length but left the pure path spreading effects and some shorter-path TIB effects that were aggravated by the low STT*P product. (A high STT*P product indicates the combination of long signature transfer times and fast nodal processors; the conditions under which best topological (PLE) efficiency is obtained.) By then increasing the ratio of signature transfer time to processor speed (in case 3 by reducing S, in case 4 by increasing P) 100% ideal path length efficiency was obtained. We summarize the topological performance of Selfhealing in the Bellcore study network by observing that, as in Smallnet, 100% PNE was achieved in all cases. In no case was PLE below 95% and it was improved to 100% by applying the understanding of TIB and path spreading effects which has been gained.

9.3.3 Speed of Restoration in Bellcore Network Model

The distribution of individual path completion times in Bellcore net is shown in Fig. 9.5. The average path completion time (t_{pav}) is 203 msec (std dev = 66.4 msec) and 95% of all paths are complete in under 350 msec. Fig. 9.6 shows the RS plot (overlaid restoration trajectories from all span cuts) in Bellcore net. This plot identifies spancut (3-0) as requiring the greatest amount of time for the number of paths realized (462 msec for 11 paths). Another trajectory that emerges from the pack is (10-7) in which 16 paths are created in 465 msec. At the other extreme is spancut (7-4) in which 20 paths are synthesized in only 259 msec. Some insights come from these trajectories. When the solution topologies for (3-0) and (10-7) are compared to all others topologies for the cuts in Bellcore, they are *not* the solutions with the highest number of diverse routes *or* with the most paths, as might be expected. They do however involve the *longest* paths

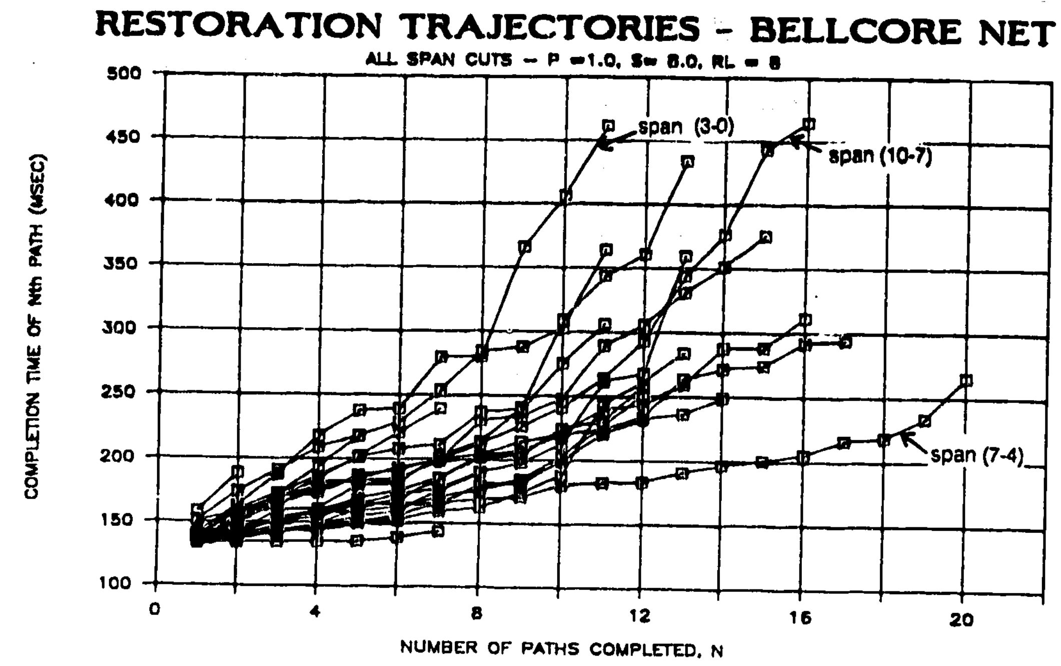


Fig. 9.6 Restoration space diagram for all span cuts in Bellcore net (RL=8, S=8.0, P=1.0).

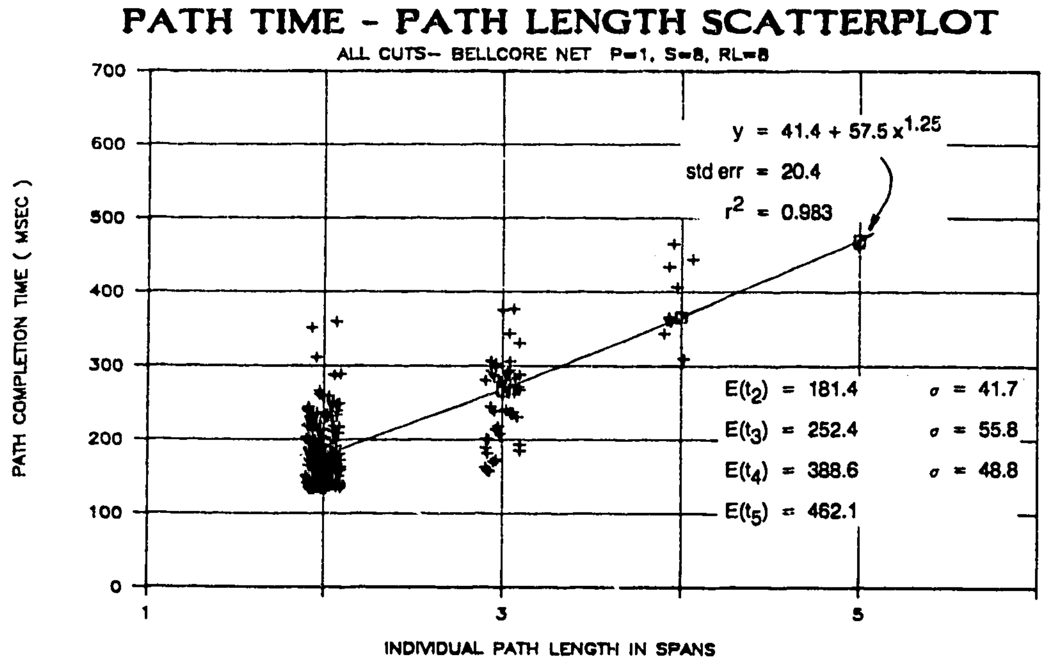


Fig. 9.7 Path time versus path length scatterplot for Bellcore net (RL=8 S=8.0, P=1.0 - random x offsets applied for presentation).

that are found in any spancut but this does not adequately account for their total time. Spancut (3-0) has one length 5 route and also includes two length 4 routes. Spancut (10-7) has two length 4 routes. If no other interactions were involved, we would expect the length 5 path in (3-0) to require on the order of 5 two-way signature exchange times for its creation and this predicts a total time of only 100 + 10*64/8000 = 180 msec (span distances being insignificant in Bellcore), not 462 msec. In addition, the longest route in (10-7) is of length 4 and yet takes slightly *more* total time than does (3-0) which has a length 5 path.

What accounts for these two spancuts requiring the most restoration time is this: the restoration plans for spancuts (3-0) and (10-7) both have *the fullest inventory of increasing lengths of routes*. Like the spectrum of shells of an atomic model, these two solutions exhibit the fullest occupancy of each possible *route length* in the paths that are created. For instance in Table 9.2, (3-0) has 1 length 2 route, 2 different length 3 routes, 2 different length 4 routes, and one length 5 route. The significance of this *cascade of route lengths* is that it implies a *telescoping series of TIB dynamics* at intermediate stages of mesh expansion for most indices. The topologically dictated cascade of route lengths in the solution implies that until the index mesh(es) that will result in the length 2 paths collapse in reverse linking, the indices that will result in length 3 paths tend to be blocked. And the indices destined to create length 4 paths are blocked until the length 3 paths are reverse linking, and so on. In the limit, path synthesis over the longest of routes in these conditions may have attend $O(k^2)$ prior TIB episodes. A check of the spancuts with next highest restoration times ((7-5), (8-3)) shows that they too follow the pattern of a relatively full cascade of route lengths. (see Table 9.3.)

It is the full *cascade* of route lengths that implies a particularly punctuated series of mesh expansions. In these circumstances, wave after wave advance further toward the Chooser only *after* all preceding waves ebb away. In reality the view is complicated by the fact that mesh advances and TIB events are non-uniformly distributed in space: the wavefronts in the above analogy are spatially quite irregular. Amoeboid may be a more accurate image for the mesh on each index than waves. In the opposite extreme of a topology that permits an equivalent number of paths over one route, or a set of equal-length routes, all mesh fronts arrive nearly together at the Chooser and collapse more or less simultaneously without interacting with each other in TIB dynamics. By definition when the network topology supports the latter type of solution, TIB contentions between indices are not involved in the path synthesis at all. (There can always be other indices that *are* blocked even in these circumstances but our interest is always in the dynamics of the subset of indices that do eventually result in paths.)

Spancut (7-4) of Bellcore net offers an example of the opposite extreme to a cascade of routes. In (7-4), 20 paths are created over 5 different routes but because all of these routes are of length 2, it is a very fast restoration. The (7-4) trajectory is nearly linear and requires only 259 msec for an average slope of about 5 msec/path completion. This is indicative of essentially processor-limited operation in these study conditions: the protocol is going flat out accumulating path completions without TIB-induced delay dynamics. In general topologies, in between the extremes of a single-route solution and a full cascade of routes, Selfhealing proceeds in a complex combination of partly simultaneous parallel mesh expansions and partly interacting dynamic series of TIB interdependencies.

Fig. 9.7 is a scatterplot of the time taken to complete each path versus the length of a path based on the all-span cuts data for the Bellcore net.[2] Notwithstanding the significant scatter, the data show that the average time to create a path is essentially linear in path length for the Bellcore network. This is consistent with the view that the time to create paths depends individually much more on the properties of the *set of routes* required than either the length of the path or the number of paths required, but that over a large set of paths these individual dynamic variations should be uncorrelated and leave a constant average incremental time per span. If we consider that a minimum requirement for path creation is a two-way signature exchange over the total length of a path, this implies a minimum regression slope of 16 msec / incremental span of path length. The actual slope of about 58 msec / incremental span is much greater than this implying that the creation of most paths involves significant signature processing dynamics (an average investment of ~ 3 STT times for index dynamics per incremental span) and that path creation by straightforward signature exchange without index blocking is rare. A curve joining the lowest of all scatter points at lengths 2 and 3 on the plot is however consistent with the theoretical minimum slope of 16 msec / incremental span corresponding to path construction by pure signature exchange. The apparent reduction of scatter on the longer paths is an artifact of there being fewer total data points at the longer path lengths. All principles indicate that at longer path lengths the path time variance should increase.

9.4 US Study Network Results

Table 9.4 shows the Selfhealing response to each span cut in the US study network. The 47 span cuts in this network resulted in 316 individual restoration paths. When each of these solutions is compared to the output of MetaDijkstra on the same network, the following is found:

(a) PNE = 100% for all span cuts

(b) Network average PLE = 99.45 % (1268/1275)

(c) SPP = 97.78 % (309/316)

(d) Mean path time = 247 msec , Std. Dev = 110 msec

(e) Mean time to complete restoration = 348 msec, Std. dev = 154 msec.

Table 9.3 Results of Selfhealing in US Network Model (RL varies, S = 8 kb/s, P = 1.0)

Span Cut (A-B)	Route	Length	Number of Paths	$Time_1..Time_N$
1-0	0-4-5-6-1	4	1	206.8
12-3	3-7-13-12	3	2	180.9..191.9
	3-2-1-6-7-13-12	6	1	261.9
	3-2-1-6-7-11-14-13-12	8	1	367.6
4-0	0-1-6-5-4	4	1	206.9
5-4	4-8-5	2	4	154.7..166.3
	4-0-1-6-5	4	1	207.1
6-5	5-8-9-6	3	1	182.2
	5-4-0-1-6	4	1	207.1
	5-8-9-10-11-7-6	6	3	262.4..268.8
	5-8-9-10-11-14-13-7-6	8	1	320
6-1	1-0-4-5-6	4	1	206.3
	1-2-3-7-6	4	2	209.4..217.2
	1-2-3-12-13-7-6	6	3	263.5..369.3
	1-2-3-12-13-20-21-10-9-6	9	1	370.0
8-4	4-5-8	2	4	153.8..160.5
	4-0-1-6-9-8	5	1	266.5
8-5	5-4-8	2	4	153.1..159.6
	5-6-9-8	3	1	181.8
9-8	8-5-6-9	3	1	180.7
	8-4-0-1-6-7-11-10-9	8	1	325.9
9-6	6-5-8-9	3	1	180.0
	6-7-11-10-9	4	3	208.0..221.6
	6-1-0-4-8-9	5	1	287.2
	6-7-13-14-11-10-9	6	1	330.4
7-6	6-1-2-3-7	4	2	208.5..218.1
	6-9-10-11-7	4	1	
	6-1-2-3-12-13-7	6	3	278.0..306.8
	6-5-8-9-10-11-7	6	1	291.9
7-3	3-12-13-7	3	3	185.2..220.7
	3-2-1-6-7	4	5	206.9..251.6
	3-12-13-14-11-7	5	1	287.5
	3-2-1-0-4-8-9-10-11-7	9	1	540.7
3-2	2-1-6-7-3	4	2	207.9..225.0
	2-1-6-7-13-12-3	6	3	280.9..322.0
	2-1-0-4-8-9-10-11-14-13-12-3	11	1	734.1
2-1	1-6-7-3-2	4	2	207.1..218.4
	1-6-7-13-12-3-2	6	3	270.6..365.4
	1-0-4-8-9-10-11-14-13-12-3-2	11	1	516.3
10-9	9-6-7-11-10	4	1	207.7
	9-8-5-6-7-11-10	6	1	288.4
	9-8-4-0-1-6-7-11-10	8	1	420.7

Span Cut (A-B)	Route	Length	Number of Paths	$Time_1..Time_N$
11-10	10-9-6-7-11	4	1	208.4
	10-21-20-14-11	4	1	210.9
	10-24-25-21-20-14-11	6	2	285.5..289.0
	10-24-25-21-20-13-7-11	7	1	348.4..447.6
	10-24-25-21-20-13-14-11	7	1	632.8
11-7	7-13-14-11	3	3	183.2..211.2
	7-6-9-10-11	4	1	207.2
	7-6-5-8-9-10-11	6	1	271.2
	7-6-1-0-4-8-9-10-11	8	1	411.3
	7-3-12-13-20-21-25-24-10-11	9	1	685.1
	7-3-12-13-20-21-10-11	7	1	663.6
	7-6-1-2-3-12-13-14-20-21-25-24-10-11	13	2	856.3..878.1
13-12	12-3-7-13	3	2	181.7..189.7
	12-3-2-1-6-7-13	6	1	263.9
	12-3-2-1-6-7-11-14-13	8	3	315.5..347.4
13-7	7-3-12-13	3	2	180.2..187.6
	7-11-14-13	3	3	183.4..194.1
	7-6-1-2-3-12-13-	6	2	261.9..266.2
	7-6-9-10-21-20-13	6	1	320.7
14-13	13-20-14	2	2	155.6..160.9
	13-7-11-14	3	3	181.3..192.3
14-11	11-7-13-14	3	3	180.4..219.6
	11-10-21-20-14	4	1	218.7
	11-10-24-25-21-20-14	6	1	301.7
	11-20-24-25-21-20-13-14	7	2	418.6..617.6
	11-10-9-6-7-3-12-13-14	8	1	682.7
20-13	13-14-20	2	2	154.9..158.2
	13-15-20	2	1	162.4
	13-14-11-10-21-20	5	1	245.0
	13-14-11-10-24-25-21-20	7	3	290.3..406.9
20-14	14-13-20	2	2	153.9..158.8
	14-13-15-20	3	1	189.3
	14-11-10-21-20	4	1	210.2
	14-11-10-24-25-21-20	6	2	302.5..331.6
	14-13-7-11-10-24-25-21-20	8	1	403.9
24-10	10-21-25-24	3	1	180.9
	10-11-14-20-21-25-24	6	2	262.0..287.4
	10-11-14-13-20-21-25-24	7	2	369.7..395.2
25-24	24-10-21-25	3	1	183.2
	24-10-11-14-20-21-25	6	2	263.6..316.2
27-25	25-26-27	2	3	153.4
26-25	25-27-26	2	1	153.0
	25-21-22-26	3	3	18[illegible].9..199.8
	25-21-23-22-26	4	2	221.8..230.9
25-21	21-10-24-25	3	1	180.2
	21-22-26-25	3	3	181.9..193.3
	21-23-22-26-27-25	5	1	234.2
	21-20-14-11-10-24-25	6	2	265.8..287.6

Span Cut (A-B)	Route	Length	Number of Paths	$Time_1..Time_N$
21-10	10-24-25-21	3	3	180.0..196.9
	10-11-14-20-21	4	2	211.4..225.4
	10-11-14-13-20-21	5	1	318.4
	10-11-7-13-20-21	5	1	339.1
	10-11-7-13-15-20-21	6	1	362.8
21-20	20-22-21	2	1	160.8
	20-15-23-21	3	2	191.1..203.7
	20-14-11-10-21	4	1	209.8
	20-15-19-23-21	4	4	224.5..300.3
	20-14-11-10-24-25-21	6	1	288.9
	20-13-7-11-10-24-25-21	7	2	393.6..409.2
22-20	20-21-22	2	3	155.6..177.8
	20-21-23-22	3	5	205.8..271.6
	20-15-23-21-25-26-22	6	1	370.6
	20-14-11-10-24-25-26-22	7	1	383.7
	20-14-11-10-24-25-27-26-22	8	1	614.1
22-21	21-20-22	2	1	154.1
	21-23-22	2	5	158.2..194.9
	21-25-26-22	3	3	189.8..196.3
	21-25-27-22	4	1	275.9
26-22	22-21-25-26	3	3	181.5..196.2
	22-23-21-25-27-26	5	1	233.3
23-22	22-21-23	2	3	160.3..188.2
	22-20-15-23	3	1	185.7
	22-26-25-21-23	4	3	245.1..279.0
	22-26-27-25-21-23	5	1	356.9
23-19	19-15-23	2	2	158.5..162.2
	19-15-20-22-23	4	6	207.2..293.1
	19-18-15-20-21-23	5	1	309.2
	19-18-15-13-20-21-23	6	1	394.3
19-15	15-23-19	2	2	154.0..159.5
	15-18-19	2	5	160.9..187.9
	15-16-17-19	3	5	185.6..211.4
	15-20-21-23-19	4	2	209.8..215.9
20-15	15-13-20	2	1	154.0
	15-23-21-20	3	2	184.1..197.6
	15-19-23-21-20	4	3	215.4..284.5
	15-19-23-22-20	4	1	250.6
15-13	13-20-15	2	2	158.8..171.4
	13-14-20-15	3	2	212.5..229.4
	13-17-11-10-24-25-21-20-15	8	2	368.9..379.1
	13-14-11-10-24-25-21-23-15	8	1	429.0
	13-14-11-10-21-20-15	6	1	463.7
16-15	15-18-17-16	3	1	180.8
	15-19-17-16	3	5	186.6..243.2
17-16	16-15-19-17	3	6	182.4..209.5
19-17	17-18-19	2	1	161.0
	17-16-15-19	3	6	183.1..245.9

Span Cut (A-B)	Route	Length	Number of Paths	Time$_1$..Time$_N$
19-18	18-15-19	2	5	156.4..181.2
	18-17-19	2	1	159.6
18-17	17-19-18	2	5	153.8..168.3
	17-16-15-18	3	5	190.4..203.6
18-15	15-19-18	2	5	153.7..186.6
	15-19-17-18	3	1	208.6
23-15	15-19-23	2	4	156.2..175.1
	15-20-21-23	3	7	180.9..292.3
	15-13-20-21-23	4	1	309.3
23-21	21-22-23	2	3	153.8..167.6
	21-20-15-23	3	2	185.4..197.0
	21-20-19-15-23	4	4	229.0..270.6
	21-25-26-22-23	4	2	235.9..253.7
Totals:				
47 spans			320 paths	

Fig. 9.8 shows the statistics of individual path lengths taken over all span cuts compared to the ideal path length distribution for all span cuts in this network. Fig. 9.9 shows the corresponding distribution of individual path completion times. 95% of all paths are completed within 450 msec. Fig. 9.10 gives the RS plot for Selfhealing in the US network and Fig. 9.11 plots path time versus path length for each of the 320 paths that were synthesized. Because the US network provides the longest routings of any network studied to date, its data were of particular interest for a regression study to see how path completion time varies with path length. The best regression fit to the data in Fig.9.11 was found to occur with a polynomial of order 1.3, closely repeating the equivalent result for the Bellcore network but beginning to show a slightly greater than linear rise with long paths in a network comprised of long spans. In the 47 span cuts performed on the US network only two of the Selfhealing solutions had PLE values less than 100%. These were spans (22-20) and (23-21) for which the Selfhealing restoration plans and the ideal plans have been studied to find the cause. The generally far fewer PLE-imperfect results than in Bellcore is attributed to the much longer spans giving a higher STT*P product.

For reference in the following discussion of spancuts (22-20) and (23-21) the reader can reconstruct these two spancuts from Table 9.3. Inspection shows that in spancut (22-20) the difference between the experimental and ideal plans reduces to two paths of length 4 in the MetaDijkstra solution which are realized by length 7 paths in the Selfhealing solution for a total of 6 excess spans. The two plans otherwise have the same number of length 2, 3, 7 and 8 span paths. Study shows that this outcome is attributable to the early path choice effect. The source of the

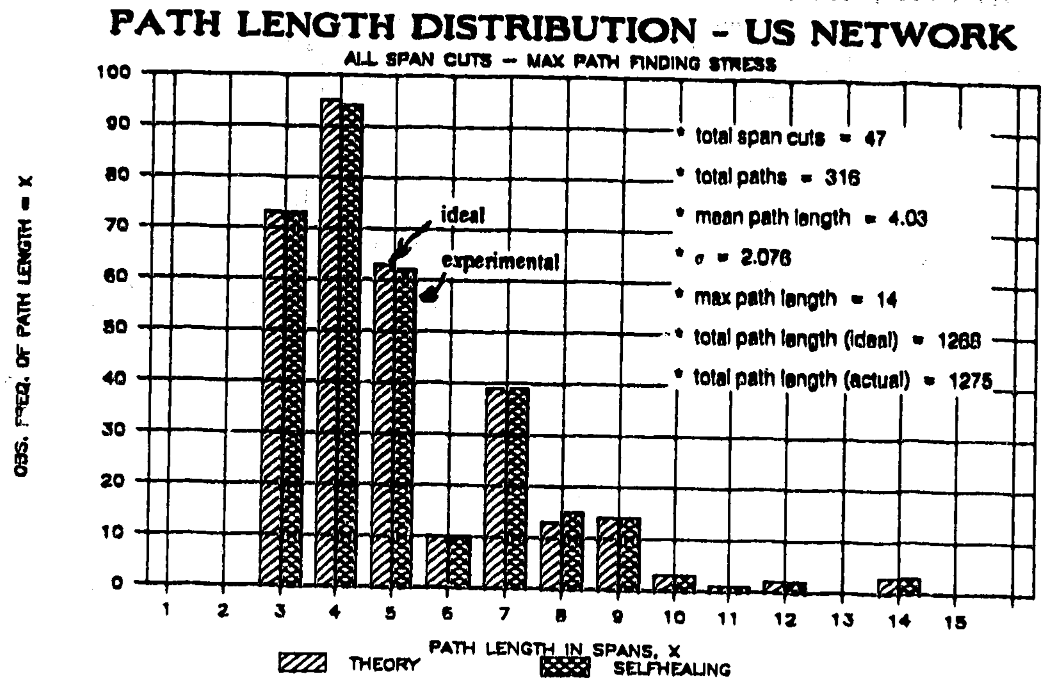


Fig. 9.8 Distribution of ideal and experimental restoration path lengths in US network model.

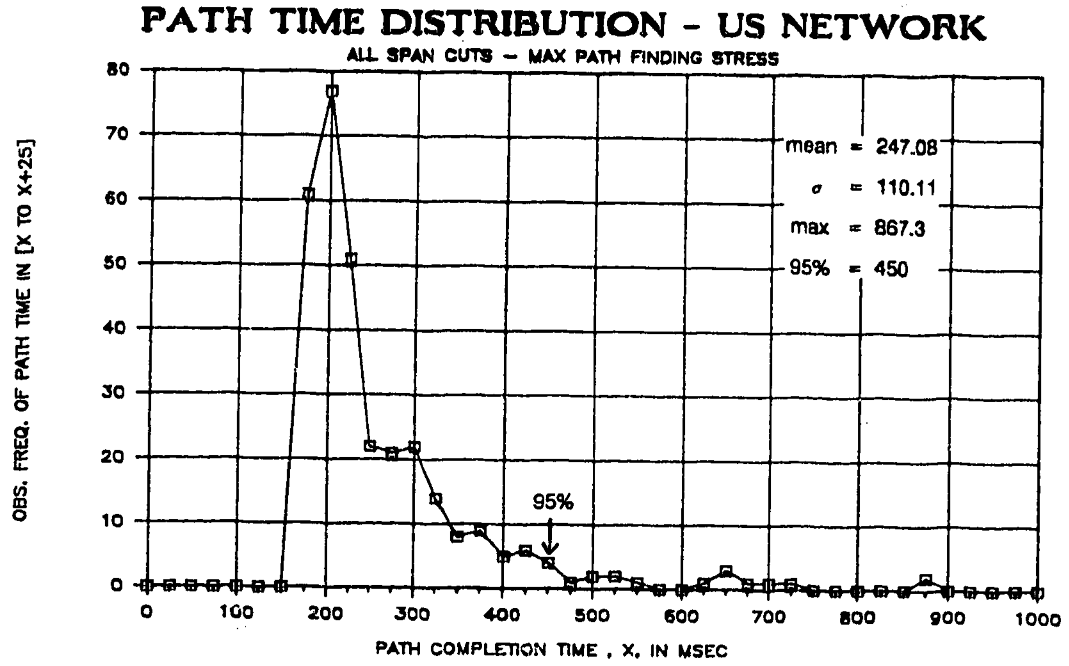


Fig. 9.9 Distribution of Selfhealing path times in US study network.

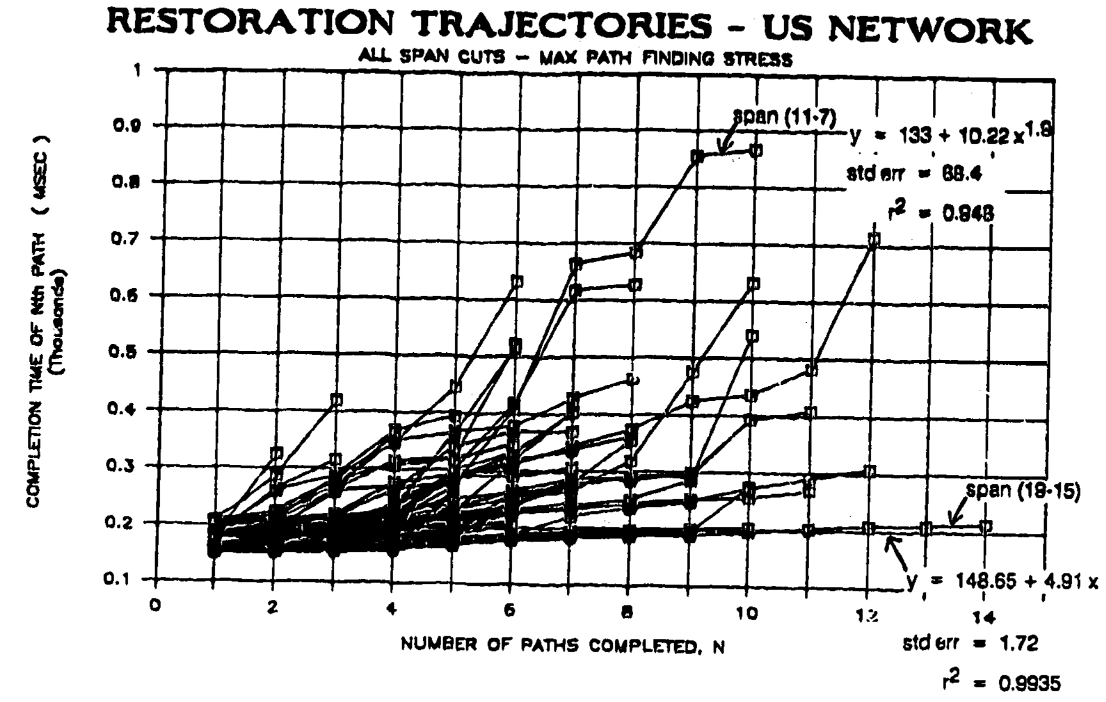


Fig. 9.10 Restoration space diagram for all span cuts in US network model.

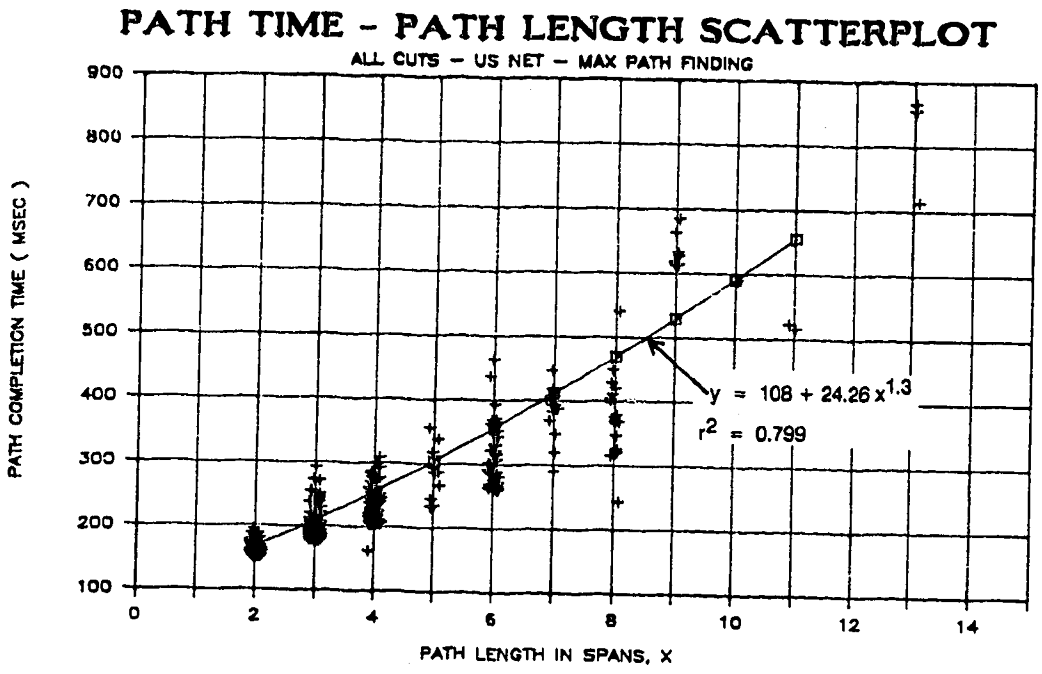


Fig. 9.11 Path time versus path length scatterplot for all span cuts in US study network.

effect in spancut (22-20) is traceable back to a difference in the choice for two of the five 5 length-3 paths which were possible. Both Selfhealing and MetaDijkstra realize three paths over (20-21-22) and three paths over (20-21-23-22) (refer to Fig. 9.2). At this point there are only two more length-3 paths possible in the network and since their lengths are identical, an arbitrary choice must be effected between them. MetaDijkstra happened to realize two paths on route (20-15-23-22) and Selfhealing effected the equally valid choice of two more paths on route (20-21-23-22). By chance Selfhealing's choice had the side effect of prohibiting subsequent paths of length 4 over route (20-21-25-26-22) because span (20-21) is then full. Thus Selfhealing's early choice forced it to subsequently realize the route segment (20-14-10-11-24-25-, to get around span (20-21) and thereafter complete the paths over the same last three spans, -25-26-22).

In spancut (21-23), 10 out of the 11 paths found in the ideal and in the Selfhealing solution are identical. The difference of +1 comes from a path (21-25-26-22-23) of length 4 which corresponds in the ideal solution to a length 3 path (21-20-22-23). This outcome is attributed to legitimate TIB effects on span (20-22). Node 23 acted as Sender in this case and placed five indices on span (23-22). Because span (20-22) has only one spare it is easily blocked by one of the five indices arriving at node 22 on (23-22) which happens to be the first repeatered into span (20-22). But this index is destined to succeed over another route before it can succeed over a route involving (20-22). The result was that because of the delay attending index collapse after reverse-linking for one of the 3 (23-22-23) paths, both of the two remaining indices at node 22, which could (and preferably would have) succeeded over span (20-22), were already advanced far enough that they reached the Chooser over longer than ideal routes by the time (20-22) was finally released by reverse linking. One of these other routes was the length 4 route (22-26-25-21). The path (23-20-21) was no longer possible after that because all spares in the span (22-23) were used. Had there been 6 spares here instead of 5, (23-20-21) would eventually have been realized because when all other indices are collapsed after reverse linking, the remaining outstanding index would have had no further blocking contention on (22-20).

Selfhealing's performance in this network is, in summary, also very good. The most important objective of 100% PNE has again been obtained. In addition only two restoration events out of the 47 spancuts in this network exhibited less-than-ideal k-shortest paths length properties. Some of the restoration plans autonomously devised by Selfhealing in this network are particularly impressive. Consider for instance the restoration routing plan derived for spancut (11-7) in Table 9.3: *10 paths were created over 7 distinct routings requiring the coordinated efforts of* <u>*16 different Tandem nodes*</u>. *One of the routes is* <u>*13 spans long*</u> *and another is 9 spans long. The entire restoration plan was* <u>*computed and put into effect in 878 msec*</u> *and it is topologically equivalent to the best that an ideal centralized k-shortest link disjoint paths algorithm could produce.*

9.5 Metropolitan Study Network Results

The metropolitan study network of Fig. 9.3 was used with every span distance set to d= 40 km. There are a total of 55 possible span cuts. The Selfhealing reaction to each of these is listed in Table 9.4. The 55 spancuts resulted in a total of 163 paths. A summary of the results is as follows:

(a) PNE = 100% for all span cuts

(b) Network average PLE = 99.8 % (506/507)

(c) SPP = 98.18 % (54/55)

(d) Mean path time = 163 msec , Std. Dev = 35 msec

(e) Mean time to complete restoration = 193 msec, Std. dev = 37 msec.

Fig. 9.12 shows the distribution of path lengths created by Selfhealing and by MetaDijkstra in the metro network model. The distributions differ by only one span in total, a length 7 path created by Selfhealing that ideally could have been of length 6. Fig. 9.13 gives the distribution of path completion times. 95% of paths are completed in under 226 msec. Fig. 9.14 shows the RS plot for all spancuts in the metro network. Fig. 9.15 is a scatterplot showing the path-time versus path length relationship for Selfhealing in this network.

All but one restoration event resulted in 100% PLE. The case where MetaDijkstra and Selfhealing differed was spancut (9-1), shown in Fig. 9.16. The discrepancy is attributable to fortuitous early path choices by MetaDijkstra as we have seen twice before. The Selfhealing solution has a total path length of 17 spans, comprised of single paths of length 2, 3, 5 and 7 respectively. The ideal solution in this case has a total length of 16 spans contributed by paths of length 2, 3, 5 and 6. The extra span is traceable to the choice between two length 5 paths. Selfhealing chose (1-11-12-13-14-9) whereas MetaDijkstra chose (1-0-12-13-14-9). Both paths are by themselves equally valid length 5 alternatives but the path realized by Selfhealing in this case had the side-effect of using up span 1-11 and forcing the next longest route to use (1-0-11-10- as an alternative to the shorter (1-11-10- route segment. After this segment, the remainder of the length 7 Selfhealing path is identical to the length 6 MetaDijkstra path.

9.6 Comparative Discussion

Comparison of the all-spans trajectory (RS) plots for these networks (Fig.s 9.6, 9.10, 9.14) shows a range of dynamic characteristics that varies considerably with individual span cut from linear path accumulations at a processor-limited rate of about 5 msec/ path in route-simple restoration plans to TIB-limited quadratically increasing restoration times in solutions requiring a cascaded series of route lengths. In the US network spancut (19-15) gives a lower bound trajectory in which 14 paths are accumulated over 4 routes in an incremental time of approximately 70 msec. At the other extreme, (11-7) in the US network required approximately 880 msec to accumulate 10 paths. Polynomial regression shows that individual trajectory of this spancut rises

Table 9.4 Results of Selfhealing in Metropolitan Network Model (RL = 8, S = 8 kb/s, P = 1.0)

Span	Route	Length (spans)	Number of Paths	Time
1-0	0-11-1	2	1	134.9
	0-12-11-10-1	4	1	177.3
11-1	1-0-11	2	1	134.7
	1-10-11	2	1	136.7
	1-9-10-13-12-11	5	1	188.1
7-3	3-8-7	2	1	137.5
	3-4-7	2	1	139.0
	3-2-8-15-7	4	1	179.8
7-6	6-5-7	2	1	143.8
	6-17-7	2	1	145.4
9-8	8-15-9	2	1	138.3
	8-2-10-9	3	1	151.9
	8-7-15-14-9	4	1	184.8
	8-3-2-1-9	4	1	192.9
13-10	10-11-12-13	3	1	155.5
	10-9-14-13	3	1	158.2
	10-1-0-12-24-13	5	1	214.6
	10-2-8-15-20-13	5	1	226.5
22-13	13-20-19-22	3	1	151.8
	13-23-21-22	3	1	152.2
19-15	15-20-19	2	1	136.5
	15-16-19	2	1	140.4
	15-7-16-18-19	4	1	183.6
	15-14-13-22-19	4	1	187.5
	15-9-14-20-21-19	5	1	239.8
19-18	18-16-19	2	1	140.2
	18-17-16-15-19	4	1	182.9
23-21	21-22-13-23	3	1	151.6
	21-20-13-20-24-23	4	1	203.6
11-0	0-1-11	2	1	134.0
	0-12-11	2	1	135.7
8-2	2-3-8	2	1	136.1
	2-10-9-8	3	1	151.3
	2-1-9-15-8	4	1	194.2
3-2	2-8-3	2	1	135.7
	2-10-9-8-7-3	5	1	195.1
	2-10-9-15-7-4-3	6	1	
8-3	3-2-8	2	1	135.1
	3-7-8	2	1	138.3
	3-4-7-15-8	4	1	190.2
17-6	6-7-17	2	1	135.8
	6-5-7-16-17	4	1	187.3

Span	Route	Length (spans)	Number of Paths	Time
15-8	8-7-15	2	1	138.7
	8-9-15	2	1	141.2
	8-3-7-16-15	4	1	181.3
	8-2-1-9-14-15	5	1	219.5
12-11	11-0-12	2	1	134.8
	11-10-13-12	3	1	152.8
	11-1-9-14-13-24-12	6	1	
23-13	13-24-23	2	1	135.1
	13-22-23-21	3	1	151.1
20-15	15-19-20	2	1	135.1
	15-14-20	2	1	136.1
	15-9-10-13-20	4	1	178.6
	15-16-19-21-20	4	1	186.4
20-19	19-15-20	2	1	134.2
	19-21-20	2	1	137.3
	19-22-13-20	3	1	174.1
	19-16-15-14-20	4	1	203.3
24-23	23-13-24	2	1	134.8
	23-21-20-13-12-24	5	1	221.2
12-0	0-11-12	2	1	134.1
	0-1-10-13-12	4	1	184.6
5-4	4-7-5	2	1	135.5
	4-3-7-6-5	4	1	178.4
8-7	7-3-8	2	1	136.1
	7-15-8	2	1	137.4
	7-4-3-2-8	4	1	178.6
	7-16-15-9-8	4	1	189.0
10-9	9-1-10	2	1	137.2
	9-14-13-10	2	1	151.5
	9-8-2-10	3	1	152.9
	9-15-20-13-12-11-10	6	1	236.3
13-12	12-24-13	2	1	134.0
	13-12-24	2	1	135.4
17-16	16-7-17	2	1	136.1
	16-18-17	2	1	136.5
	16-15-7-6-17	4	1	197.2
21-19	19-20-21	2	1	135.7
	19-22-21	2	1	136.5
	19-15-20-13-23-21	5	1	195.5
2-1	1-10-2	2	1	136.6
	1-9-8-2	3	1	151.3
	1-11-10-9-15-8-3-2	7	1	267.7
10-2	2-1-10	2	1	137.1
	2-8-9-10	3	1	152.2
	2-3-7-15-20-13-10	6	1	229.4

Span	Route	Length (spans)	Number of Paths	Time
7-4	4-3-7	2	1	134.3
	4-5-7	2	1	143.8
15-7	7-8-15	2	1	135.9
	7-16-15	2	1	142.8
	7-17-16-19-15	4	1	185.3
	7-3-8-9-15	4	1	186.6
	7-6-17-18-19-20-15	6	1	261.7
	7-4-3-2-1-9-14-15	7	1	282.5
14-9	9-15-14	2	1	135.1
	9-10-13-14	3	1	152.7
	9-8-15-20-14	4	1	177.7
24-12	12-13-24	2	1	134.9
	12-11-10-13-23-24	5	1	205.6
15-14	14-20-15	2	1	137.1
	14-9-15	2	1	141.4
	14-13-22-19-15	4	1	192.6
18-16	16-19-18	2	1	134.5
	16-17-18	2	1	135.3
22-19	19-21-22	2	1	135.2
	19-20-13-22	3		151.4
9-1	1-10-9	2	1	135.2
	1-2-8-9	3	1	151.0
	1-11-12-13-14-9	5	1	203.7
	1-0-11-10-13-20-15-9	7	1	283.4
4-3	3-4-7	2	1	134.7
	3-8-7-5-4	4	1	187.1
6-5	5-7-6	2	1	134.4
	5-4-7-17-6	4	1	177.2
16-7	7-17-16	2	1	134.1
	7-15-16	2	1	138.5
	7-6-17-18-16	4	1	176.3
	7-8-15-19-16	4	1	193.0
15-9	9-8-15	2	1	135.6
	9-14-15	2	1	139.8
	9-10-13-20-15	4	1	186.3
	9-1-2-8-7-15	5	1	194.8
14-13	13-20-14	2	1	136.7
	13-10-9-14	3	1	151.2
	13-22-19-15-14	4	1	177.6
20-14	14-15-20	2	1	134.4
	14-13-20	2	1	138.5
	14-9-15-19-20	4	1	190.0
19-16	16-15-19	2	1	134.5
	16-18-19	2	1	135.3
	16-7-15-20-19	4	1	195.5
	16-7-17-8-9-10-13-22-19	8	1	316.0

Span	Route	Length (spans)	Number of Paths	Time
21-20	20-19-21	2	1	135.0
	20-13-23-21	3	1	151.4
	20-15-19-22-21	4	1	177.4
10-1	1-9-10	2	1	135.2
	1-2-10	2	1	136.3
	1-11-10	2	1	138.1
	1-0-12-13-10	4	1	180.7
7-5	5-6-7	2	1	135.8
	5-4-7	2	1	139.0
17-7	7-6-17	2	1	134.0
	7-16-17	2	1	135.0
	7-15-19-18-17	4	1	186.1
11-10	10-1-11	2	1	134.2
	10-13-12-11	3	1	152.8
	10-9-1-0-11	4	1	176.7
20-13	13-14-20	2	1	136.2
	13-22-19-20	3	1	152.2
	13-23-21-20	3	1	154.1
	13-10-9-15-20	4	1	169.0
16-15	15-7-16	2	1	136.6
	15-19-16	2	1	137.3
	15-20-19-18-16	4	1	179.5
	15-8-7-17-16	4	1	189.9
18-17	17-16-18	2	1	134.6
	17-7-15-19-18	4	1	178.6
22-21	21-19-22	2	1	135.1
	21-23-13-22	3	1	151.0
Totals:				
55 spans		2.96 spans /hop	163 paths	

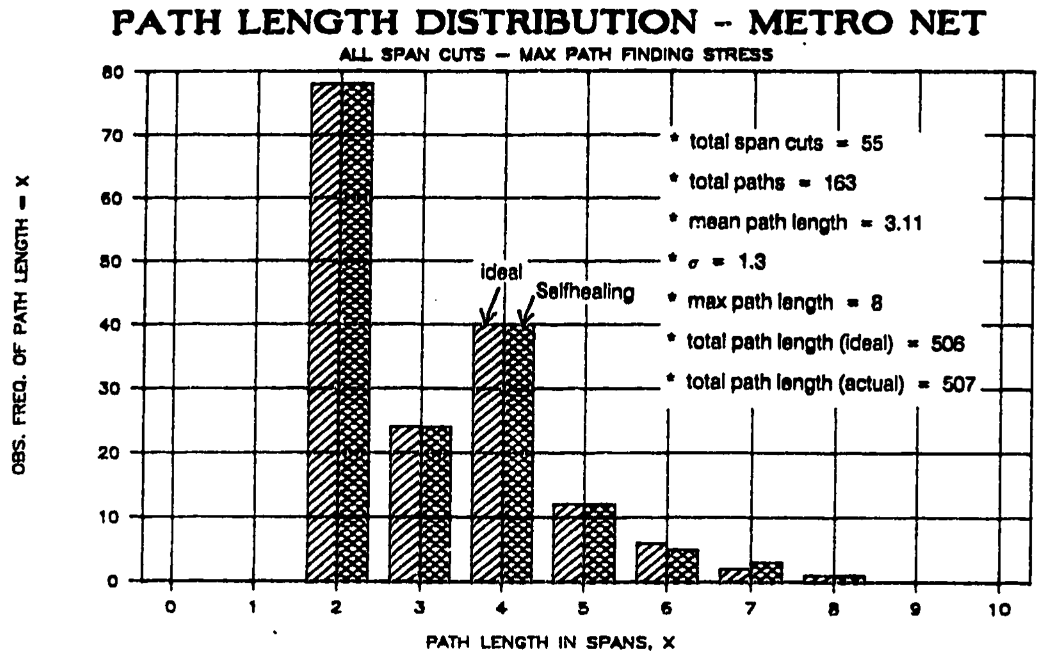


Fig. 9.12 Distribution of ideal and experimental path lengths in Metro network model.

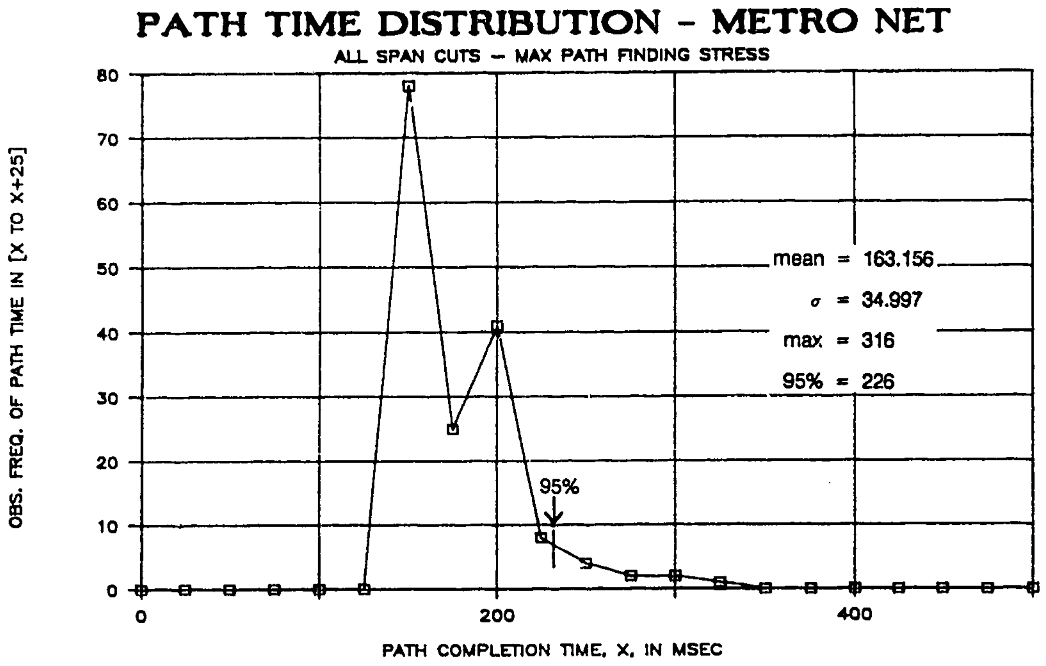


Fig. 9.13 Distribution of individual path times in Metro study network.

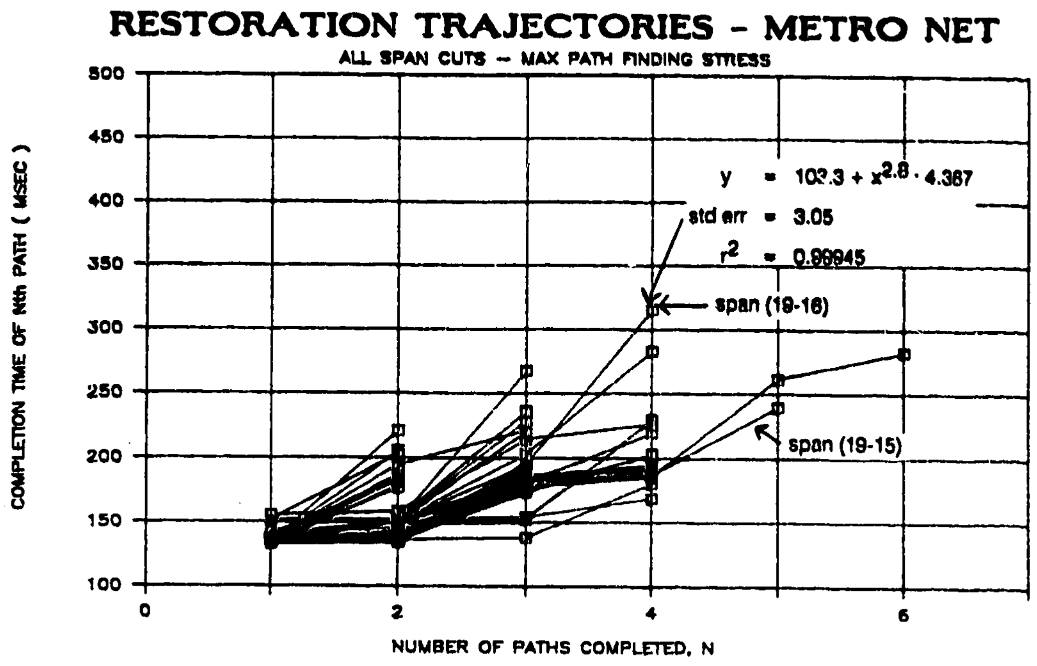


Fig. 9.14 Restoration space diagram for all span cuts in Metro network model.

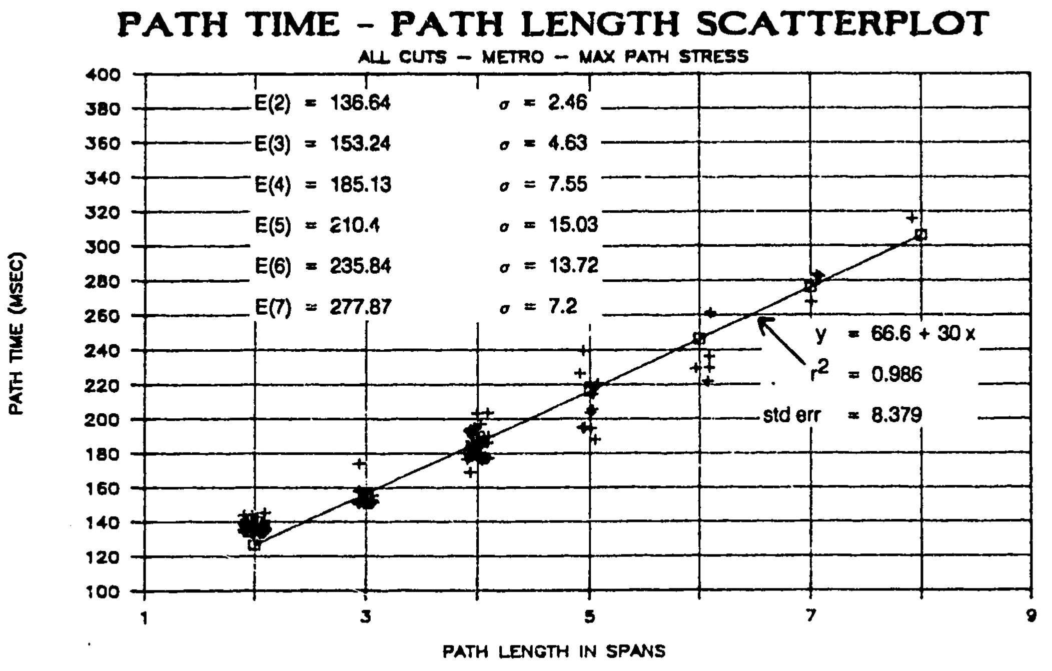
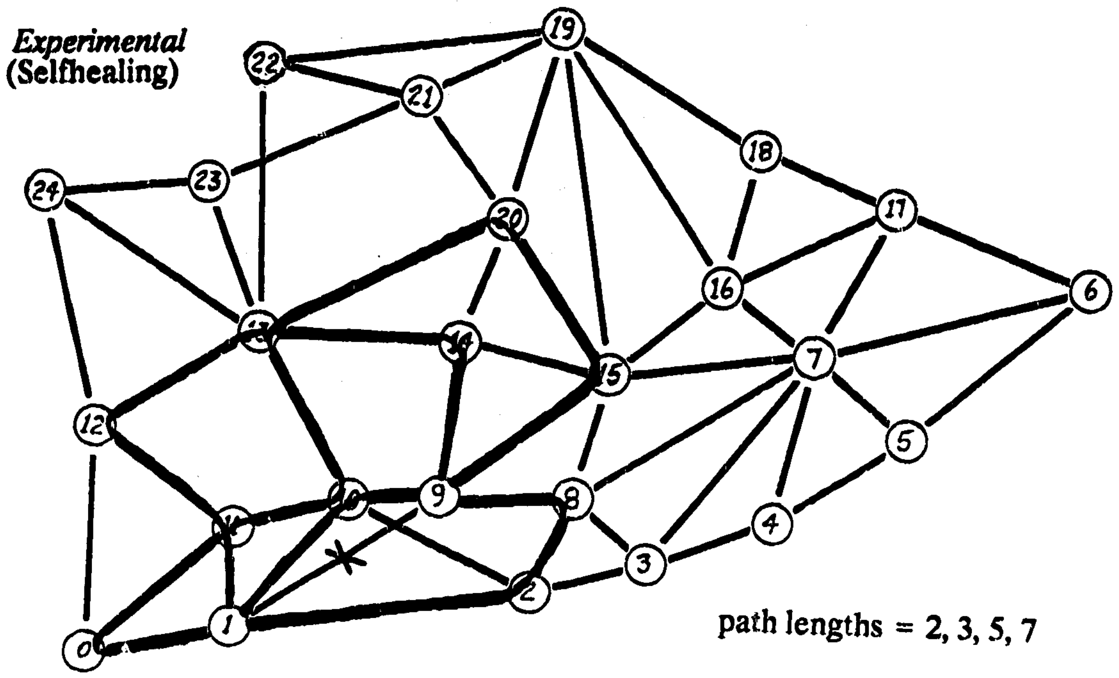


Fig. 9.15 Path time versus path length scatterplot for all span cuts in Metro study network.



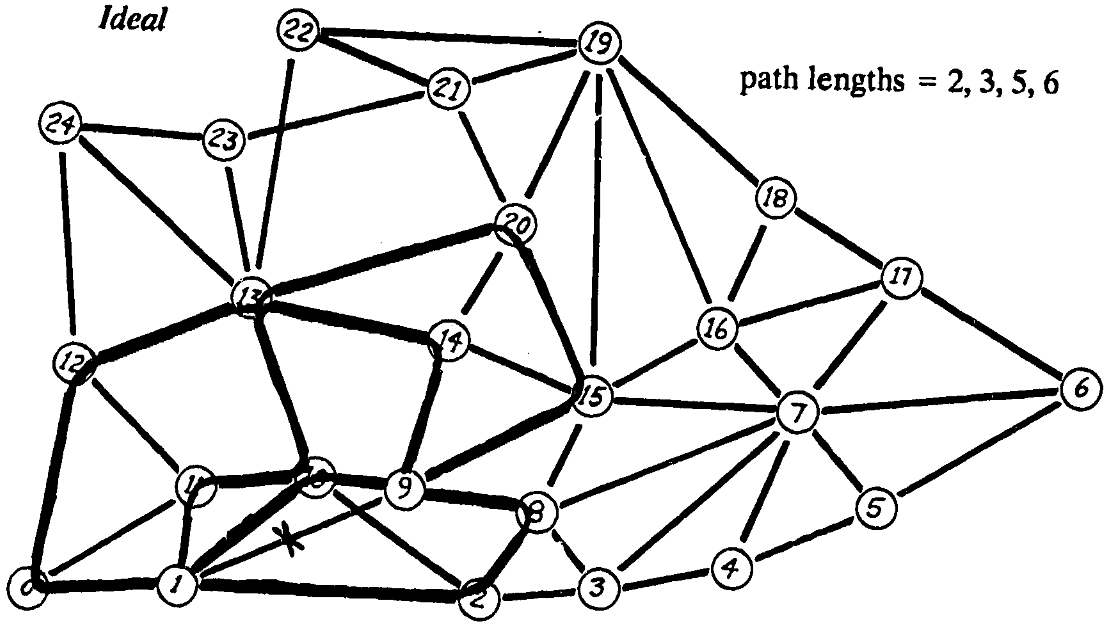


Fig. 9.16 Ideal and experimental solution topologies for spancut (9-1) in Metro study network.

approximately as the square of the path number ($x^{1.9}$). Similar inspection in the Bellcore network shows spans (7-4) and (3-0) which delimit the extremes of path finding difficulty. Regression shows that the trajectory for spancut (3-0) in Bellcore net rises approximately as the 2.2th power of the path number. In metro net, the trajectory of span (19-16) (Fig. 9.14) appears to rise at the highest rate but with the number of data points available, the trajectory of this spancut can be fitted with the same standard error to polynomials with any order from 1.8 to 2.8. The path time versus path length scatterplots (Fig.s 9.7, 9.11, 9.15) show that individual path completion time is very nearly linear with path length. The highest rate of rise seen is $O(x^{1.3})$ in the US network where paths up to length 13 are found.

9.7 Summary of Results

The experimental studies on three model networks constitute a thorough body of test results giving empirical verification of the Selfhealing technique under a range of network characteristics. The data of this chapter comprise a total of 125 individual span restoration experiments in which a total of 717 individual path routings were created. The most important result is that *in every case Selfhealing achieved 100% PNE.* In the case of networks tested in maximum path finding stress this means that Selfhealing restored a number of paths which was equal to highest topologically possible number of paths. In the case of Bellcore net (which had finite working circuit quantities) Selfhealing found either the topological limit or the finite number of paths required.

The second most important result is that every restoration event was completed well within the 2 second objective. The longest observed restoration time observed was for span (11-7) in the US study network and this took a total of 878 msec. Fig. 9.17 plots the *combined distribution of complete restoration times* for all 125 span cuts of the three study networks. In 95% of all span cuts the restoration event is complete in under 529 msec.

The third most important outcome is that in over 95% of all cases the path routings obtained are as efficient length-wise as can be computed by a centralized k-shortest paths algorithm which enjoys global network knowledge. The observed overall shortest path probability (with RL =4 for Bellcore net) was 96.9 % (695 out of 717 paths). This includes 4 paths which have excess length attributable to the early path choice effect and these outcomes are unavoidable for any k-shortest paths algorithm that does not permute every combination of possible path sets.

The path length distributions in Fig.s 9.4, 9.8 and 9.12 show that in each of these networks, Selfhealing provides a close approximation to the path lengths produced by MetaDijkstra, even before the optimizing considerations of reduced RL, S and P were introduced in the developments of this chapter. We now go on to explore such parametric effects, beginning in Ch. 10 with the repeat limit.

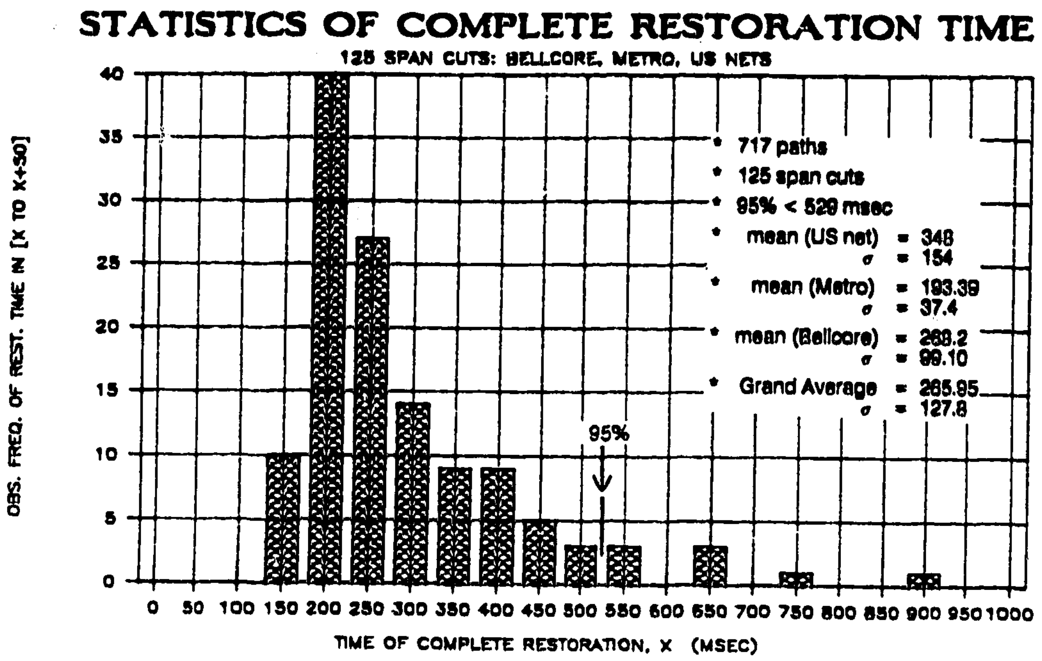


Fig. 9.17 Combined distribution of times to complete restoration in a total of 125 spancuts from Bellcore, US and metro study networks.

CHAPTER 10 EFFECTS OF REPEAT LIMIT

Our goal in this chapter is to understand how the Repeat Limit (RL) parameter affects both the topological properties and the speed of Selfhealing. We will begin with theoretical discussion of the influence of RL and an inherent tradeoff that implies existence of an optimum RL value. Next we outline a series of Selfhealing experiments designed to test and quantify the dependence on RL. Finally we present the results of these experiments and interpret them in light of our theoretical expectations and in terms of impact on engineering applications of the method.

10.1 Theory of Repeat Limit

Any signature arriving at a Tandem node is ignored by the node if (RS.repeat @ int-port >= RL). This is the basic mechanism which limits the geographical extent of the influence of a Selfhealing event. From this viewpoint, an arbitrarily high RL might seem advisable so that any cut in a network would have the highest chance of full restoration, even if the paths found are longer than theoretically required. In practice the detailed network configuration changes continually on a daily or hourly time scale as a result of ongoing maintenance and growth, so a high RL ensures the widest scope for path finding in the presence of uncertainty about the actual network dimensioning. In addition a high RL facilitates the use of a single value uniformly at all nodes in the network.

10.1.1 Remnant Signatures

On the other hand, a less obvious role of RL is in determining not just the spatial range of signatures but also the *temporal lifetime* of freely propagating *remnant signatures*. Recall that when a successful index undergoes reverse linking, collapse of the previously built mesh of signatures on that index is initiated by Tandem nodes on the reverse linking route by suspension of all transmit signatures on that index from ports which are not in the reverse linking path (Fig. 6.9). Before reverse linking occurs, all signatures in the network on any given index are ultimately related by a chain of precursor relationships back to the Sender. As soon as reverse linking slashes through the fabric of the mesh this is no longer true for the *remnant portions* of the given index mesh. Any signature in the network that is no longer related by a cause-effect chain back to the Sender, we call a *remnant signature*.

The time for network-wide elimination of *remnant signatures* resulting from reverse linking on a given index is proportional to the RL for the following reason: Remnant signatures are not identifiable as such by Tandem nodes and are therefore still subjected to normal Tandem node rebroadcast rules as if these free remnant signatures were ordinary forward flooding signatures until the number of rebroadcasts reaches the RL limit. The result is a fleeting propagating existence for the remnant portions of a collapsed signature mesh. They can exist only so long as they manage to enjoy rebroadcast by Tandem nodes with free links into which they can be repeatered. Behind the front of the remnant mesh the precursors for this index are gone and this is rolling that mesh up from behind. But as long as the leading indices of the remnant mesh portion can keep propagating via rebroadcast, expansion onto new links can keep pace with the rate at which Tandem nodes learn of the disappearance of the precursors for this index and suspend TS's on the index. This can only go on until the number of rebroadcasts in each portion of the remnant mesh reaches the RL. Then further rebroadcast by Tandem nodes is denied and the remnant mesh is finally caught from behind and eliminated as the disappearance of its precursor casuality chain catches up with it.

During the lifetime of a remnant mesh fragment however it will generally have propagated back into the primary region of path creation in the network. Tandem node rebroadcast is essentially omnidirectional so remnant signatures will reinvade the very territory of their origin if they can, and in principle, compete with other indices that are still in the crucial forward expansion phase. There is no memory or knowledge in the system to reliably allow a Tandem node to distinguish an arriving remnant signature from an ordinary forward signature. An unsatisfied index may therefore have to compete with remnant signatures as well as other legitimate unsatisfied indices. The latter considerations counteract those above and tend to encourage use of the lowest possible RL that provides adequate path-finding reach.

10.1.2 Threshold Repeat Limit

These considerations lead to the notion of a Threshold Repeat Limit (TRL) for any given span cut. We define the TRL for a spancut (u,v) as the lowest RL value for which 100% PNE can be topologically obtained in the given network. TRL is meaningfully defined for either individual spancuts or an entire network. Because we rank PNE above all other criteria in importance, followed next by speed, TRL is also the optimum RL by our criteria because it is the fastest condition on RL which still gives 100% PNE. The concept of TRL allows identification of three regimes of Selfhealing dependence on RL. The bounds of these regions may be slightly different for each span in a network, but the overall behavior of Selfhealing in a network can be described by three regions of RL which correspond physically to the following effects:

(1) *range of reach-limited path finding:* In this region restoration times are low but not all of the attainable coverage is found because RL is not high enough to permit creation of the longest paths that are topologically possible. We can identify this region by virtue of PNE < 100%.

(2) *range of threshold repeat limits* : This is the regime of repeat limit values at which individual span cuts of a network first reach a PNE of 100%. For a network as a whole we can define this as the range between the lowest and highest TRL values for individual span cuts of the network. A constant RL network plan will tend to be more appropriate for a network with a narrow *band of* TRL values. A wide spread of TRL values will tend to encourage a node-specific RL plan.

(3) *range of excessive index lifetime:* This is the range in which RL values are uniformly higher than necessary for 100% PNE in a given network. In this range, no further increase in PNE is obtained by increasing RL but PLE and speed may suffer due to TIB effects in the presence of a high RL. Two observable phenomena are expected in this region: (a) shifts in the routing topologies with changing RL values and, (b) increasing restoration times due to *index congestion*.

A *topology shift* is defined as a change in the exact pattern of path routings generated by Selfhealing when one of the parameters of Selfhealing is varied, in this case RL. Such shifts may either conserve or alter TPL and PLE. *Index congestion* is our way of referring generally to conditions where significant numbers of remnant signatures are extant and either burdening Selfhealing nodes with a spurious processing load or competing outright with unsatisfied indices creating spurious index blocking dynamics.

10.1.3 Index Congestion

There are two hypotheses for how index congestion can affect the speed of Selfhealing. One mechanism is that index congestion can slow the average progress of unsatisfied index meshes by introducing a large number of unnecessary instances of TIB. In most cases however we expect that an unsatisfied index mesh should tend to be comprised of signatures with a lower repeat count than a remnant signature created by reverse linking on another index. Even when the mesh of an index that results in a length 2 path collapses, the repeat count of the remnant signatures from that mesh will rapidly exceed the repeat count of signatures in the remaining unsatisfied indices against which they might compete. This means that in the competition for free outgoing links at the Tandem node, the unsatisfied indices will most often have preference over remnants. This will not always be guaranteed however, because remnants from a very short path may, for some time after reverse linking, have a repeat count lower than signatures on an unsatisfied index that is destined to realize a particularly long path. Certainly therefore there is the possibility for remnant signatures to extend path length by being the instruments of the same TIB induced path extension effects we have already seen.

Another hypothesis is that an unnecessarily high RL will increase restoration times by generating an increased spurious processing load on all nodes of the network. The present protocol implementation maintains a dynamically constructed set of linked lists indicating which ports have active receive signatures, which ports are free for new signatures and which ports have currently active transmit signatures as a means to implement the set operations of the protocol. Frequent searches on these lists are linearly dependent on the length of the lists and in addition each signature that arrives or disappears at a node causes the protocol accordingly to update the lists by moving ports from one list into another. The nuisance effect of remnant signatures which appear and disappear as they race around within the range of influence can then be well imagined.

Even if they are recognized as having too high a repeat count to take over any existing broadcasts at a Tandem node, they will nevertheless trigger list processing, *better precursor* test searches (which are computationally expensive) and will often enjoy rebroadcast into such spare links as may be available at the node.

We know from implementation of the protocol that arrival and departure of a remnant signature at a Tandem node will most often require execution of linear-time segments of code. Assuming that the total number of signature events caused at any one node by remnant signatures rises as the square of signature lifetime (ie,RL) then the overall effect on network performance is expected to be a quadratic rise in restoration time with a linear increase in RL.[1]

10.2 Experiments

To study the effects of RL on restoration times and on solution topologies, all span cut experiments were performed in Smallnet with RL varying from 2 to 20 with P= 1, S= 8 kb/s, and d= 1,000 km in all cases. An all span cuts experiment with the Bellcore network was also performed with RL= 4 for comparison to the previous results obtained with RL= 8 in Ch.9. The solutions to each span cut in Smallnet were examined at each RL value to find any cases of topology shifts. Frequent changes in total path length and solution topology were seen as expected for RL values below the threshold where all possible paths are realized (RL < TRL). In this region path number and total path length naturally increase with RL as longer paths are allowed and accumulated. Our particular interest is in any topology shift that occurs *above* the TRL for each spancut, as these are strictly not expected and may be detrimental to PLE.

Of the 22 spancuts in Smallnet (Fig. 8.1), topology shifting was observed above the RL value at which PNE first reached 100%, in four cases. Three of these, (7-6), (5-1) and (9-5), had topology shifts in which the total path length (TPL) increased, decreasing PLE. In the fourth (6-8) the topology shift was self compensating in terms of total path length and so PLE remained at 100%. Although the speed of restoration was affected by further increases in RL, the topologies for all other spancuts remained unchanged as RL was increased from the TRL for span (6-8) all the way up to RL = 20.

Five of the span cuts in which no topology shifting occurred were selected for analysis of the effect of RL on the speed of restoration in the majority of cases where topology shifting *does not* occur. Fig. 10.1 shows how the restoration time of each of these sample spancuts individually vary with RL. (Spancut (7-6) is omitted from the group of five although it also appears in Fig. 10.1) Solutions with topology shifting were excluded so that effects which are intrinsically due to *index congestion* are not obscured by changes in restoration time which arise in conjunction with a topology shift.

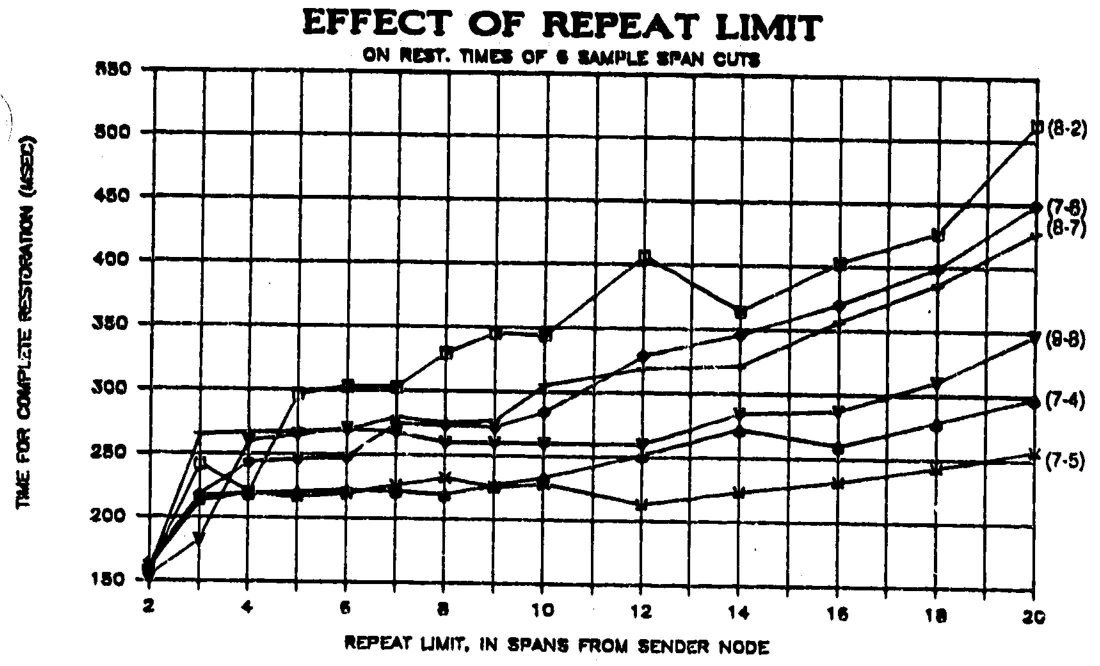


Fig. 10.1 Restoration times of six sample spancuts from Smallnet as a function of repeat limit (RL).

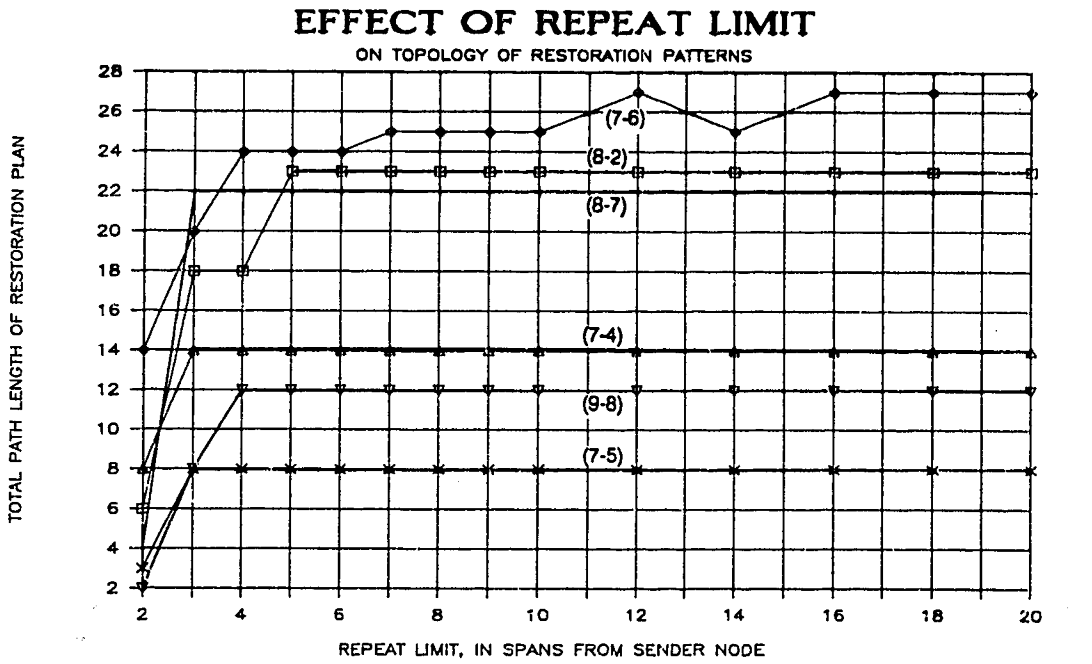


Fig. 10.2 Total path length (TPL) of restoration plans as a function of RL for six sample spancuts in Smallnet.

For the six spancuts in Fig. 10.1, a plot of the total path length of the solution against RL was obtained (Fig. 10.2). A corresponding series of RT plots was made with RL values spanning the range of interest to assist in understanding the most interesting of the events seen as RL varied widely. These are shown as Fig.s 10.3 through 10.6 for spancuts (7-6), (3-6), (8-6) and (5-1) respectively. An RL-dependent pair of trajectory plots for Smallnet span (5-9) was also obtained (part of Fig. 10.7).

Trajectories for spancuts (3-0) and (7-5) were also obtained in the Bellcore network because these were the two spans whose solutions shifted from a non-ideal PLE value to 100% PLE when RL was reduced from 8 to 4 in Ch. 9. These are shown in Fig. 10.7. Finally the all spans trajectory plot for Bellcore at RL =4 and 8 was obtained (Fig. 10.8) and path time statistics were abstracted from it for these two RL values (Fig. 10.9).

The above notions about remnant signature lifetime and propagation was investigated as an aside using an experiment in which a single index was launched into Smallnet from node 0 in response to a spancut (3-0). It was arranged for this one index to experience reverse linking over a two-span path and we then tracked behavior of the single remnant index mesh that remained, with different RL values. With RL = 4, we found there were no remnant signatures surviving after 5 iterations. When RL = 9 the same path is completed in the fourth iteration but 69 additional link-iterations are consumed and there are in total 20 extra node invocations caused by the remnants of the reverse-linked index, before it finally disappears. By choosing one node and counting the total number of signature events (defined as arriving or disappearing signatures) at that node we found that doubling RL approximately squared the total number of remnant signature events at selected nodes. It is of interest to mention that the back and forth reverberation of the uprooted index remnant mesh was in fact visibly apparent at high RL values.[2]

10.3 Analysis of Results

10.3.1 Effects of RL on Path Topology

The main effect of RL on path topology is that RL above the threshold value can result in excess path lengths due to TIB effects. In Ch. 8 we noted that reducing RL could improve PLE by forcing an index to wait out TIB events, rather than producing an end-run around to the Chooser. We then asked whether the forced waiting could mean that total restoration time *increased*, even though shorter paths resulted. We now have the data to investigate this question in the form of the RL-dependent RT plots of Fig.s 10.3 - 10.7. We will now revisit this question and also analyze the dynamics of RL-dependent topology shifts with the aid of the RL-dependent trajectory plots which have been obtained.

Span (7-6): Fig. 10.3 shows the trajectories of the Selfhealing response to spancut (7-6), parametric in RL. The corresponding TPL at each RL value is shown in Fig. 10.2. The ideal

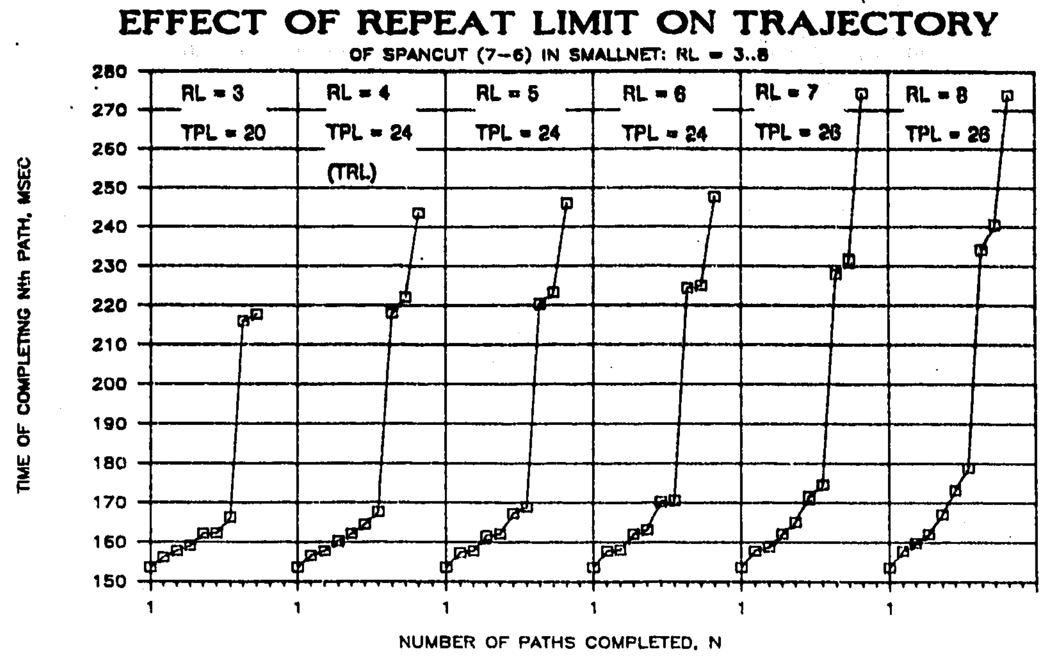


Fig. 10.3 Restoration trajectory plots for spancut (7-6) with RL varying from 3 to 8.

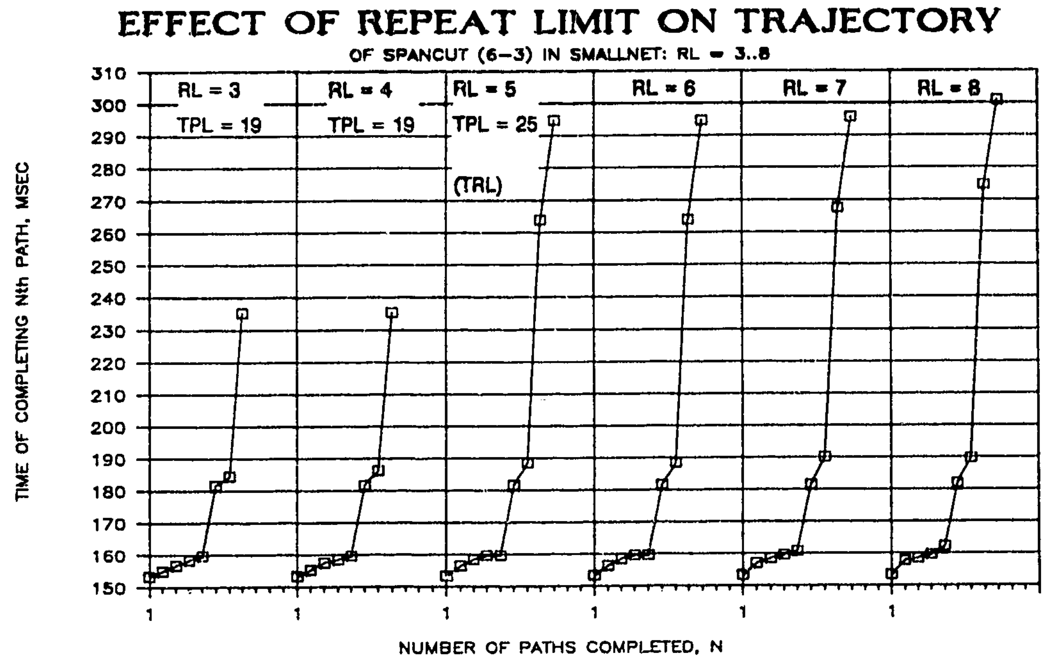


Fig. 10.4 Restoration trajectory plots for spancut (3-6) with RL varying from 3 to 8.

solution (PNE = 1.0, PLE = 1.0) is first reached when RL = 4 with 7 paths at length 2, two at length 3, and one at length 4 (denoted 7@2, 2@3, 1@4) for a TPL = 24. This topology is stable through RL= 4,5,6 but at RL =7 the length 4 path (7-4-3-2-6) shifts to the longer route (7-5-1-3-2-6). This path is then stable at all higher RLs tested but a different path shifts at RL = 12 (and at RL > 16) from (7-8-2-6) to (7-5-1-3-2-6) adding a further +2 to the TPL. The actual shifts in topology are illustrated in Fig. 10.10.

The RT plot (Fig. 10.3) shows the first jump in total time between RL=3 and 4. This understandably corresponds to the transition to 100% PNE as the last path in the ideal restoration pattern is realized. Another significant jump in total restoration time occurs in conjunction with the length 4 path at RL = 6 shifting to a length 5 path at RL = 7. This path shift is attributed to a combination of ordinary TIB effects aggravated by slowing at nodes 3 and 4 due to processing remnant signatures from seven prior indices.

In this case, creation of a longer path as a result the end-run of an index mesh around a blocked or slow area did *not* result in a lower total restoration time. This is because the index that wins the *forward* race to the Chooser via the 'long' path does so by a very small margin but the ensuing time for *reverse linking* is determined by the long route. The logically longer route will therefore require more reverse-linking time. We see in general that an arbitrarily small time difference in the forward rate of mesh expansion can therefore be quantized into a difference in overall path completion time of one or more span propagation times. For instance, indices x and y may arrive at the Chooser separated by as little as one microsecond but if index y has established a precursor chain that is one span longer than index x's, and reverse linking occurs along the longer path, then this necessarily requires one extra span time in reverse linking. Therefore the notion of saving time over a longer route is sound only in the context of *forward* mesh propagation time. Forward expansion may indeed succeed over a longer path before it can over a shorter path but we now see that this is unlikely to reduce total path time because path time is measured as the time of reverse-linking completion. In most cases the extra STT of even one additional reverse-linking stage will be greater than the differential forward times of the competing indices.

If we return to Fig. 10.3 with this insight we can in fact check that the jump in total restoration time when RL goes from 6 to 7 is about 28 msec which is very close to exactly two STT times (13 msec each). The change in path length by (+1) accounts for 13 msec of additional reverse linking time. The remaining difference of 15 msec implies (a) that the longer route was placed in contention with the shorter route by a TIB event that did not occur in the dynamics for RL=6, and (b) the longer route apparently won the *forward* race between indices by a small amount, indicative of differential processor times (2 or 3 msec only).

Span (3-6): The series of trajectory plots for this spancut are shown in Fig. 10.4. The TRL of this span is 5 and we see the jump to a PNE=100% restoration pattern as RL goes from 4 to 5. At

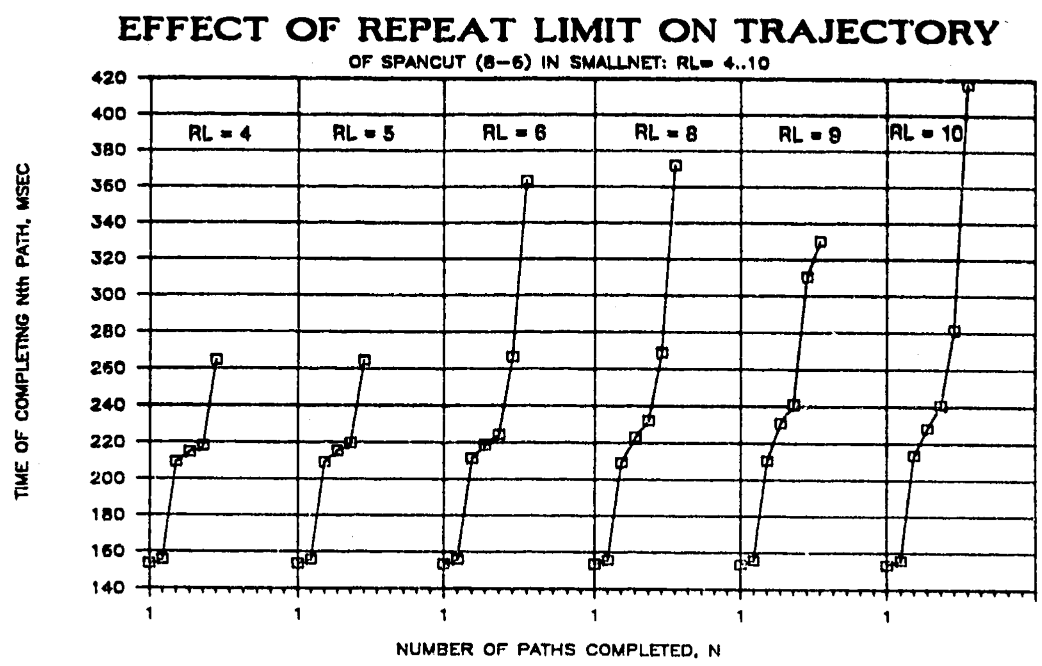


Fig. 10.5 Restoration trajectory plots for spancut (8-6) with RL varying from 4 to 10.

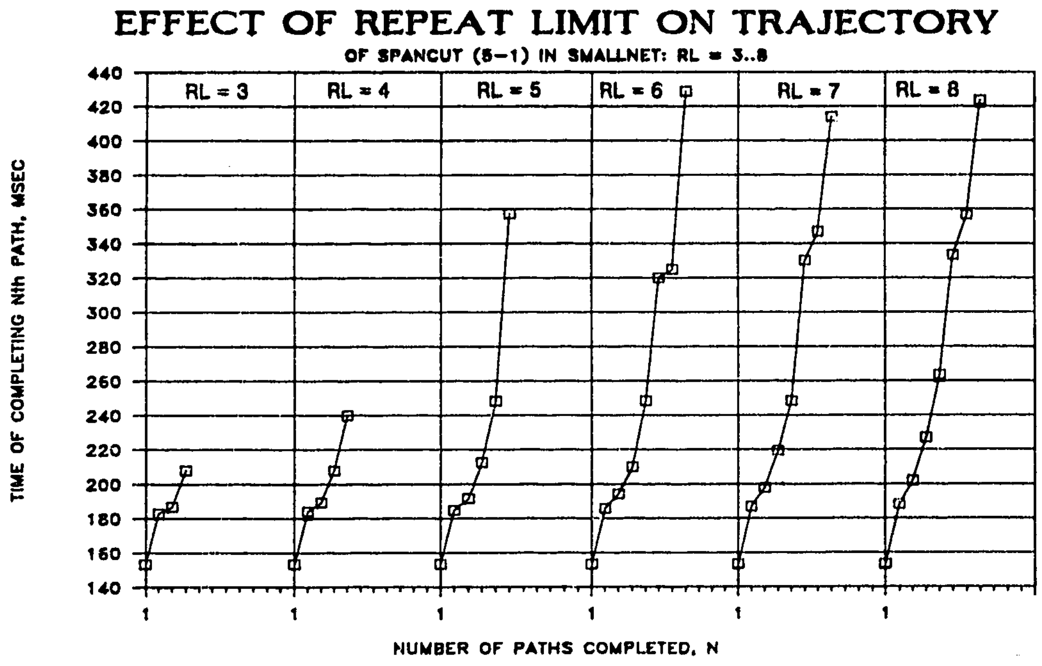


Fig. 10.6 Restoration trajectory plots for spancut (5-1) with RL varying from 3 to 8.

the threshold RL one additional path is achieved which is length 5. At the same time a path of the length 3 jumps to a length 5 routing for an overall pattern that is (+2) in TPL (Fig. 8.1, panel 9). The path that shifted jumped from (3-7-8-6) to the route (3-1-5-7-8-6). The jump to a longer route is attributed to the TIB interactions already explained for this spancut in Ch.8. For all subsequent values of RL up to 20 the topology remains stable. Fig. 10.4 shows that an increase of approximately 29 msec was associated with the jump in topology. This time is characteristic of exactly two extra span traversals as would be required to reverse link on the longer path. This implies that in the forward direction the index for this path must have reached the Chooser over the longer path only briefly before it would have succeeded on the shorter path.

Because the jump to an excess path length occurred just as RL reached the theoretical TRL for this cut, we are shown that for some spancuts in general there may be no single RL value for which PLE = 1.0 (and PNE = 1.0) is guaranteed. An idea suggested by observing this spancut is to have the Sender originate all signatures with an initially very low effective RL (it does this by manipulating the IRV) and then gradually increase the effective RL with time for those indices that remain uncomplemented at the Sender. (see Ch. 13 - Further Research)

Span (5-1): This spancut is of particular interest because it provides an instance of total time *reduction* accompanying a topology shift towards a *higher* TPL. Fig. 10.6 shows the RL dependant trajectory series for (5-1). 100% PNE is reached at RL =6 with a total time of 429 msec. The TPL of the solution at RL = 6 is in excess of the ideal by one link due to an early path choice effect as discussed in Ch 8. At RL values below the TRL, no shifting is observed amongst paths, new paths are simply accumulated as the range of influence increases. However when RL reaches 7, one of the length 5 paths of the solution jumps to a length 6 path (illustrated in Fig. 10.11) and TPL increases by 2 spans but total restoration time <u>decreases</u> by 15 msec. The 15 msec is again clearly identifiable as one STT plus some processing time. All paths in the inner group of shorter paths are unchanged and the RL=7 solution is thereafter stable at least until RL=20.

The trajectories of Fig. 10.6 help us understand the dynamics that lead to this outcome. In the RL =6 trajectory, the sixth and seventh paths, which are length 5 and 6 respectively, are found at about 320 msec and the eighth path is found at 429 msec. The difference corresponds to 8 characteristic span times in total. Investigation shows that the relevant index was blocked on span (6-8) by part of the mesh on the index that created the sixth path (length 5) and by other indices rebroadcast from node 7. Four span times were required for reverse linking to clear the blocking index from (6-8) and allow the index of the eighth path to advance its mesh and reach the Chooser through four further spans. The topology shift when RL is increased to 7 is attributable to slightly different processing times, probably caused by remnant signatures, allowing the altered outcome of competition at some node. The changed outcome results in the creation of two intertwined length six paths in lieu of one length five path and one length 6 path. There is a time penalty on one path but a time reduction to the other. However, because the path that happened to benefit in its time was also the *longest* (and latest) path, a net benefit is reflected in the *total restoration time*.

The topological shift in these two paths can be directly related to the change in trajectory between RL=6 and 7. The amount of time *added* to creation of the sixth path is approximately one STT and the amount of time *reduced* from the eighth path also this amount. Because the sixth path happened to reach the Chooser in the manner it did when RL=7, the index of the eighth path was either able to expand one span further towards the Sender before encountering TIB or was unblocked one span time earlier in the reverse linking process and therefore benefited by roughly one STT quantum in total path time. But how was the total time *reduced* by an exchange of equal amounts of time between paths? This occurs simply because all this happens *in parallel*. A reduction in time to the *last* path will show up as a reduction in *total* restoration time. The equal amount of time that was added to the faster path is visible but is not added to the *total* time because of parallelism. This is consistent with the general view that when an index reaches the Chooser via a longer path in the *forward* direction: the overall path time *of that index* will still suffer, although the time for complete restoration could be reduced if this is the last path in the solution.

Span (5-9): This is the third of the spans for which reduction of RL to the TRL value was predicted in Ch. 8 to give PLE = 1.0 and this does occur. When RL =4 the ideal solution shown in Fig. 8.1 (panel 17) is obtained. The trajectory for this spancut is shown in Fig. 10.7 at RL=4 and RL=6. The difference between RL =4 and RL=6 solutions is one path which is either routed (5-4-6-8-9) for PLE = 1.0 or is routed (5-1-3-7-8-9). Fig. 10.11 shows again that when the longer path is realized, overall path time does suffer. The mechanism of the overlength path in this case is TIB on span (4-7) in the presence of an RL which permits the longer path to be realized. The difference in times for completion of the fourth path is 40 msec (approx. 3 STT) and is consistent with the explanation given in Ch.8. One STT is attributable to the reverse-linking time of the extra span. The other two STT are the extra time lost due to blocking by other indices, first on span (4-7) and then on span (4-5), during which time forward expansion of the relevant index reached the Chooser by the longer route.

Span (8-6): The only remaining spancut where topology shifting was observed in Smallnet was (8-6). In this case two paths interacted somewhat as in (5-1), one gaining path length and time, the other losing path length and path time with a net reduction in *total* restoration time. The shift of interest is shown in the trajectory plots of Fig. 10.5 and illustrated topologically in Fig. 10.12 and occurs only at RL =9. At all other RL values from 6 to 14 the solution on the left of Fig. 10.12 is obtained. The explanation is essentially the same as that for the topology shift in (5-1): a chance perturbation in the order of events has resulted in a chance for the last path to complete by switching to a routing that allows it to avoid one TIB event. Another path had to accommodate this and it picked up an extra TIB waiting time but the exchange is beneficial overall because in

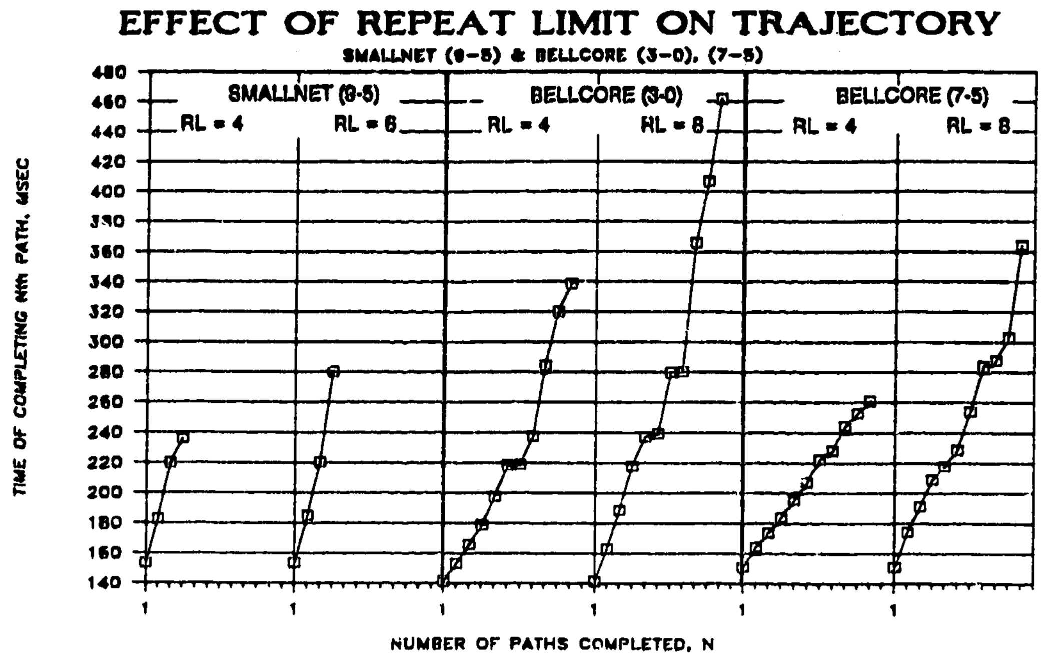


Fig. 10.7 Restoration trajectories of Smallnet spancut (9-5) and Bellcore spancuts (3-0) and (7-5) at two values of RL across which a topology shift occurs.

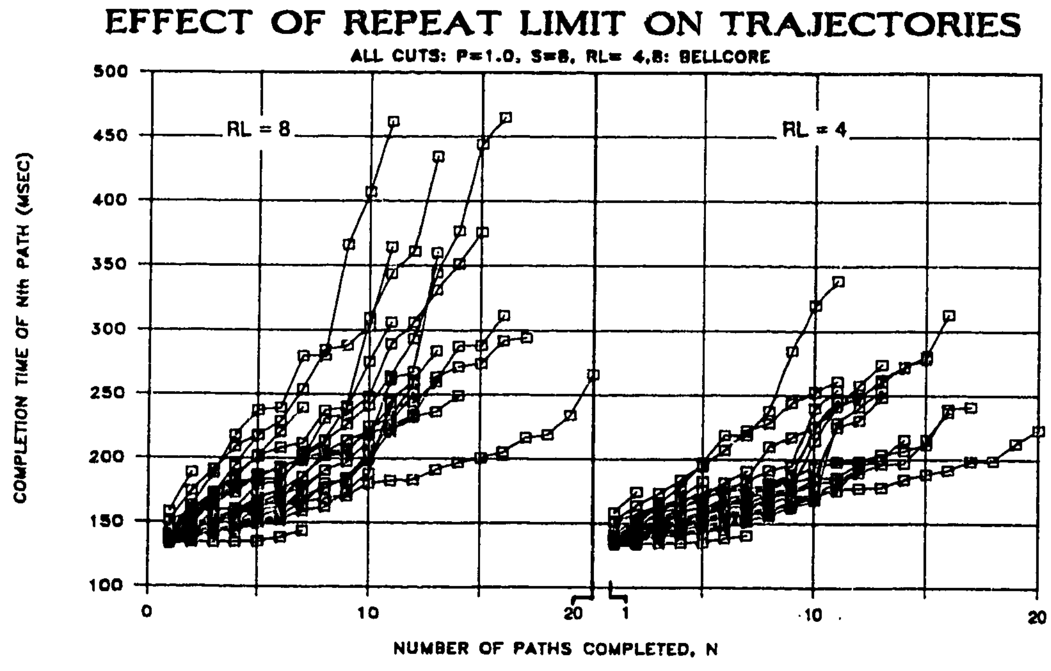


Fig. 10.8 Comparative all spans restoration space diagrams for Bellcore net with RL = 8 and RL = 4.

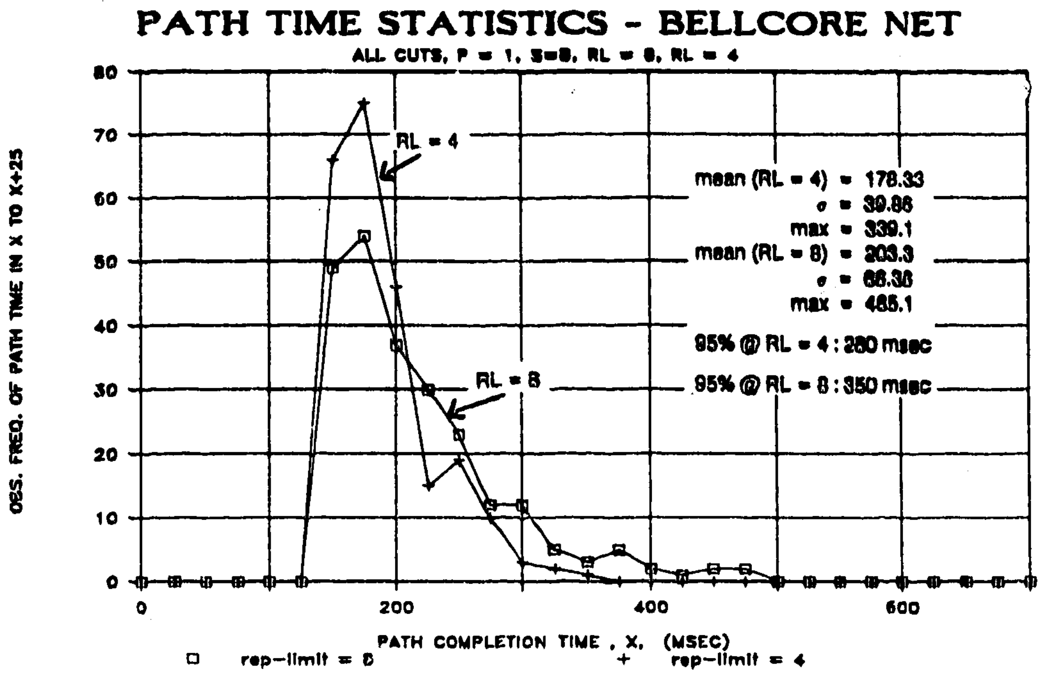


Fig. 10.9 Comparative distributions of path completion times for all span cuts in Bellcore net with RL = 8 and RL = 4.

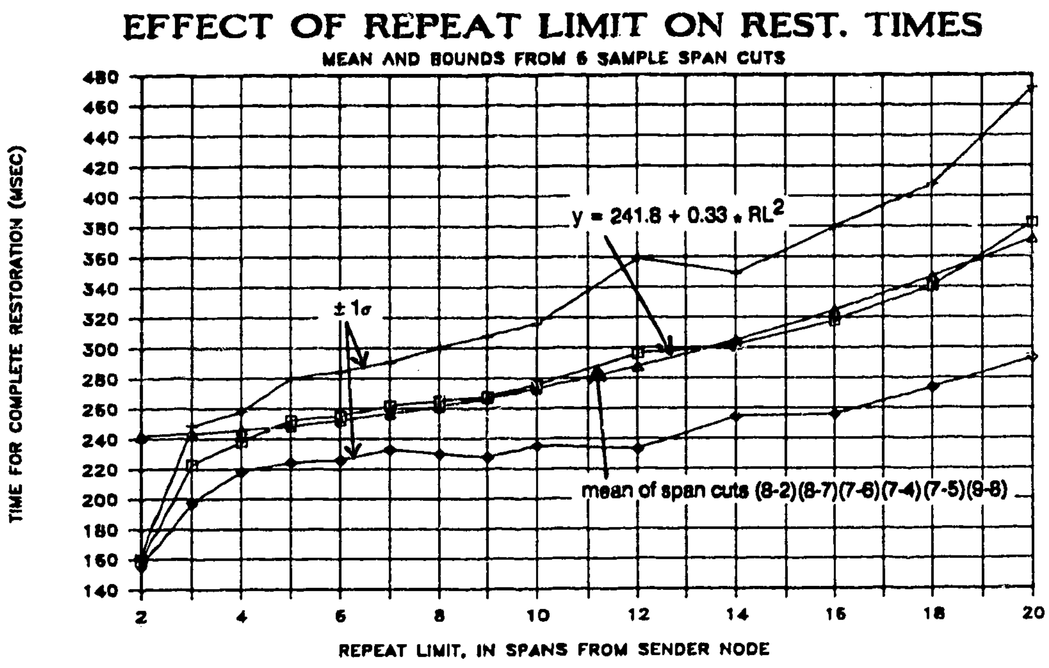


Fig. 10.13 Regression curve and std. dev. for restoration time versus RL from six sample spancuts in Smallnet.

reduced the time of the latest path. Unlike the interchange between the two paths above however, this topology shift is without cost in TPL because the path length changes are counteracting. In a time-sequential system the overall time of restoration for (8-6) (and (5-1)) would in both cases be constant because the amount by which one path increases is essentially equal to the reduction in time seen in the other path. But because of the parallelism here the overall time is reduced.

Bellcore spans (3-0) and (7-5): Topology shifts were also observed in two spans (3-0) and (7-5) when the RL was changed from 8 to 4 in the Bellcore network. The RL= 4 and 8 trajectories for these two spancuts are shown in Fig. 10.7. At RL = 4 both solutions have PLE = 100%. The topology shifts that occur are explained by the combination of normal TIB mechanisms and path spreading effects due to short span distances in Bellcore net as already explained in Ch. 9. Comparison of the RL =4 and 8 trajectories is of interest however because nodal processing delays are proportionately more important in Bellcore than in Smallnet. This leads us to expect a more pronounced slowing due to index congestion in Bellcore than in Smallnet because index congestion primarily acts to slow processors, rather than to introduce new TIB events. It does seem to be the case in the sample Bellcore trajectories of Fig. 10.7 that the overall slowing of path completion is more apparent in the end, and even at the start of the Bellcore trajectories, and than any of the equivalent changes in Smallnet trajectories occurring between RL=4 and RL=8.

10.3.2 Effects of RL on Speed of Restoration

Let us now focus on the general effect of increasing RL on the speed of Selfhealing purely due to index congestion effects without the complications of any topology shifts. In particular we want to develop engineering knowledge as to how far we can increase RL as a means to ensure 100% PNE in the face of uncertainty without incurring a significant speed penalty? To investigate this, a regression curve was fitted to the mean restoration times of the five sample cuts in Fig. 10.1 in which topology shifting did not occur. The curve and one std dev bounds of the data which was averaged to obtain it are shown in Fig. 10.13. Beyond the range of reach-limited path finding (RL > 5), the mean time to complete restoration is well approximated by $y = 242 + 0.329\,x^2$, where y = mean time to complete restoration in msecs and x = RL. This curve describes the data to within 1 msec of standard error with $r^2 = 0.9998$. Addition of a linear term did not increase the goodness of fit implying it is a relatively pure square-law rise.

Although the coefficient of the squared term appears relatively small with these data (changing RL from 4 to 16 only increases the average restoration time by 79 msec), it would rise if processor speed were decreased and it will also vary with network characteristics. A more general interpretation of the result may be obtained if we normalize the regression curve to the

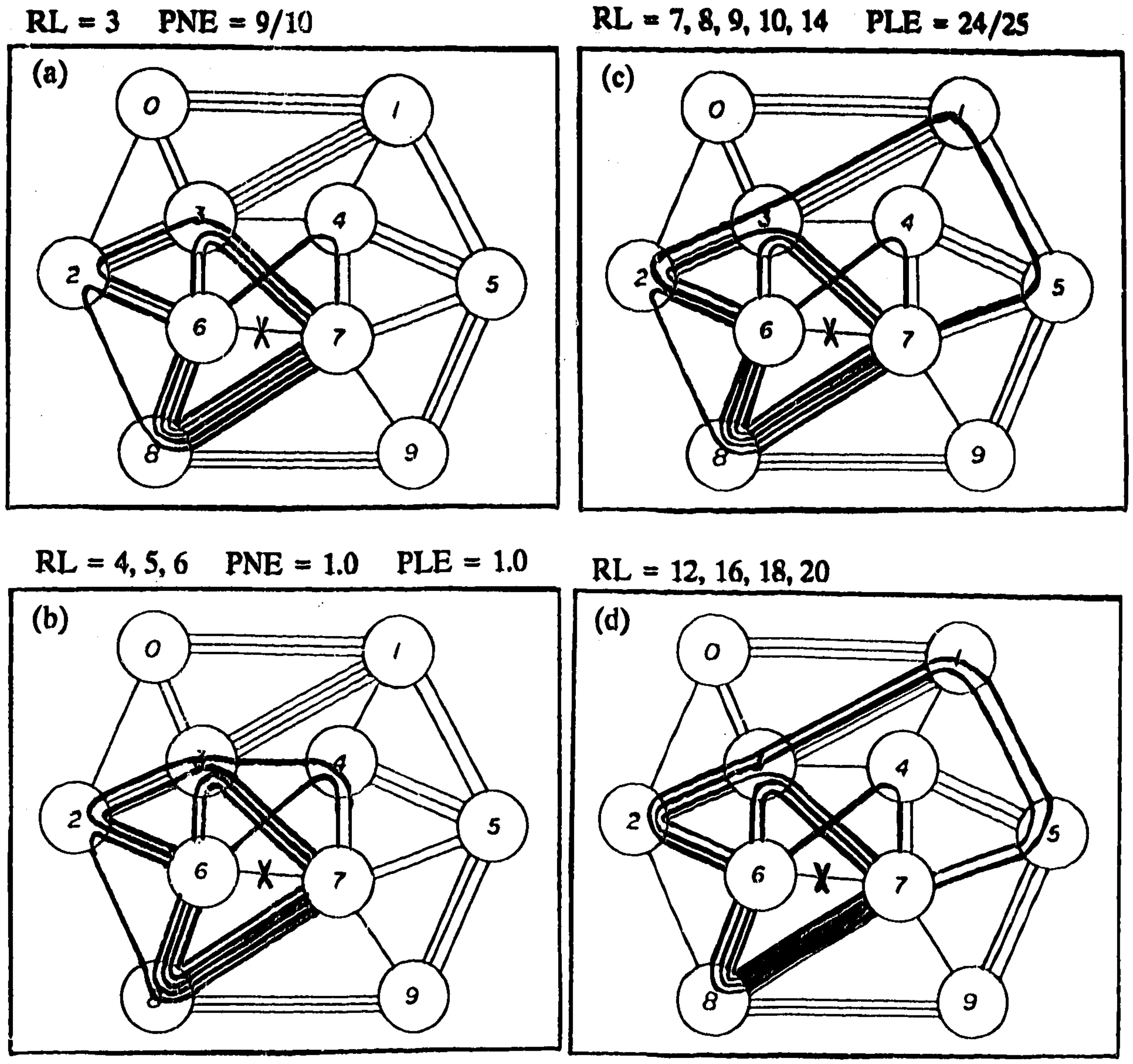

RL = 3 PNE = 9/10
(a)
RL = 7, 8, 9, 10, 14 PLE = 24/25
(c)
RL = 4, 5, 6 PNE = 1.0 PLE = 1.0
(b)
RL = 12, 16, 18, 20
(d)
0
1
2
3
4
5
6
7
8
9

Fig. 10.10 Topology shift in Smallnet spancut (7-6) occurring between RL = 6 and RL = 7.

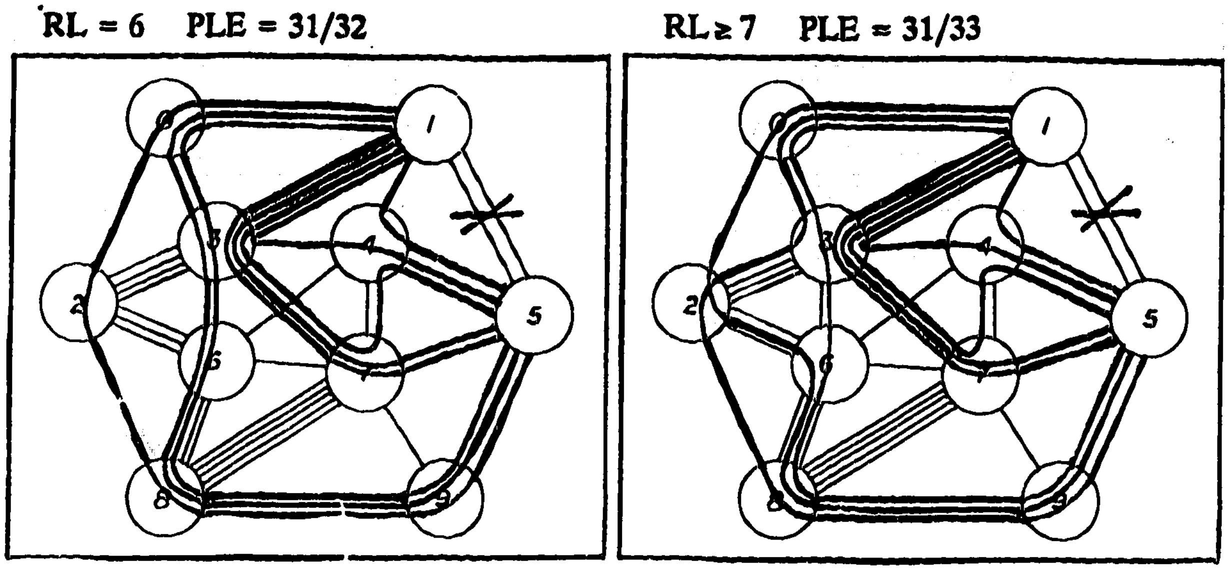


Fig. 10.11 Topology shift in Smallnet spancut (5-1) occurring between RL = 6 and RL = 7.

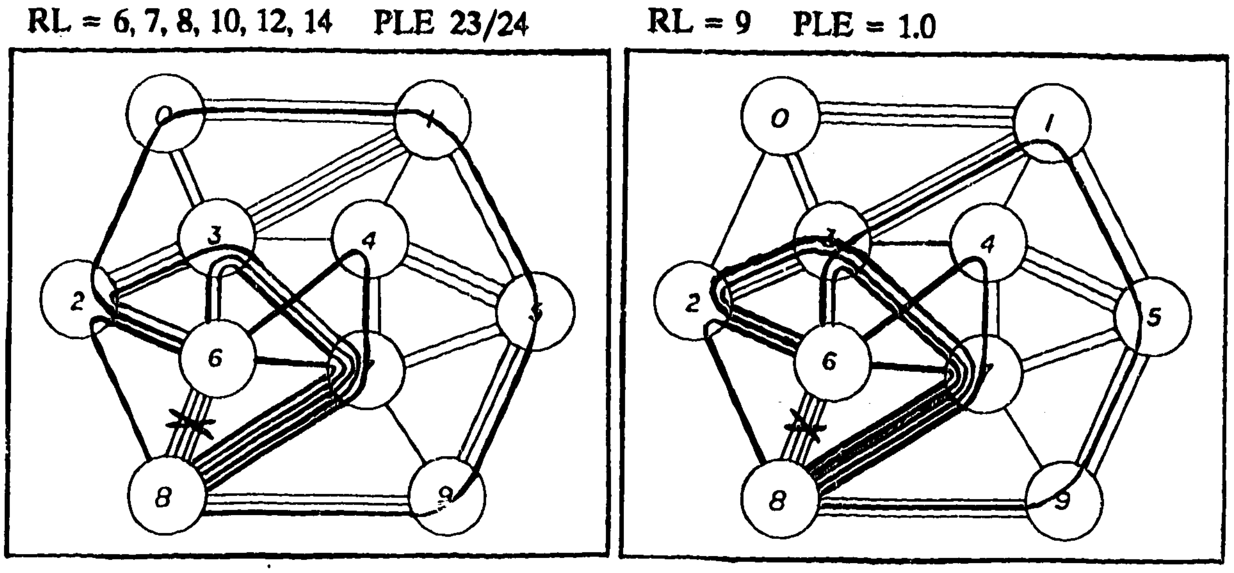


Fig. 10.12 Topology shift in Smallnet spancut (6-8) occurring at RL = 9.

approximate intercept of 242 msec. We then obtain a normalized predictor of the increase in the *incremental* time for restoration after the first path time, as a function of RL:

$$t/t_0 = 1 + 1.4 \cdot 10^{-3} \cdot (RL - TRL)^2 \qquad ;RL > TRL \qquad (10.1)$$

t/t_0 is the ratio of increase in the incremental restoration time as RL is increased beyond the TRL. The measurable but not strong increase in restoration time implied by the above says that there is relatively little penalty (5%) in *average* speed of restoration for using an RL as high as 12 in a network where most threshold repeat limits are at or around 6. Nonetheless, the coefficient of dependency of RL may vary with protocol implementation, processor speed, network size, span distances (STT*P product), and topology so that the above process of characterization should be repeated by real-world planners to derive equivalent planning relationships for any combination of real network, DCS machine, and protocol implementation. In addition the change in *maximum* restoration time with RL can in general be more extreme than the change for the average restoration times. In the particular case of Smallnet, the maximum restoration time occurs for spancut (5-1). With a doubling in RL from 6 (the TRL for this spancut) to 12, total restoration time for (5-1) increases from 430 msec to 512 msec (19%), about three times the *mean* increase predicted by (10.1).

Bellcore All Spancuts: In addition to the span-specific phenomena investigated above in Bellcore, an all spancuts experiment was performed with RL = 4 so that in conjunction with the RL=8 data on Bellcore used in Ch. 9, we could present a statistically thorough test of the effect of changing RL on speed of Selfhealing in a network other than Smallnet. Fig. 10.8 shows the RL=8 and RL=4 RS plots for Bellcore and Fig.10.9 shows the corresponding distribution of path completion times. The average path completion time increased from 178 msec at RL =4 to 203 msec at RL=8. This 14% increase is stronger than would be predicted from the observed dependancy of restoration time on RL in Smallnet and would imply a coefficient value of 3.5×10^{-2} in (10.1) for use in Bellcore, much higher than in Smallnet. The very short span lengths in Bellcore lead us to expect this more significant percentage change due to index congestion in Bellcore because processor delays are a much more significant component of the total time in Bellcore. As also expected, the change in *maximum* and *95% path completion times* due to changing RL is more dramatic than the change in averages. In Bellcore, the 95 percentile path time rises from 260 msec at RL=4 to 350 msec at RL=8, about a 35% increase.

10.4 Summary

This chapter has given us insights into how Selfhealing's topological and speed performance measures depend on Repeat Limit. RL must first of all be high enough to permit 100% PNE. The level at which 100% PNE first occurs is called the Threshold Repeat Limit. We deduced the existence of *remnant signature* effects and identified *index congestion* as the primary side-effect of a high RL value on the speed of Selfhealing. RL-dependent shifts in path routing are the main topological effects of a high RL value. Such *topology shifts* are relatively rare and do not always imply a reduction in PLE or an increase in restoration time. In no case was a drop from PNE = 100% observed at any RL value above the initial TRL of a given spancut. We deduced and empirically verified a relatively pure square-law dependence of restoration time on RL due to index congestion. The form of this dependency tells us to have regard for this effect in practice but the quantitative results imply that sensitivity to RL is relatively moderate; only 19% to 35% increase in maximum restoration time with a doubling in RL with respect to TRL. This relative insensitivity is desirable in real transport networks where continual growth and maintenance create uncertainty as to the true TRL value of the network. Our results suggest that it should be possible to compute a TRL *estimate* based on application of MetaDijstra to a *nominal* network model and then adding 2 or 3 spans to this value to ensure 100% PNE in the real (slightly uncertain) network without incurring an unacceptable speed penalty. Some advanced strategies for the repeat limit mechanism will be discussed in Ch.13.

Chapter 11 Effect of Network Size

In this chapter we pursue the question: *How does the time for Selfhealing restoration rise as the network grows in size?* Knowledge of this type is important to assess reasonably how Selfhealing would perform as the networks in which it is deployed, grow with time. For example; will restoration times seen in networks like Smallnet rise linearly or exponentially with network size? A finding of exponential growth would in practise mean an abrupt limit to the size of network in which Selfhealing could beat the CDT and it would mean that the fastest possible processors should be used in the design of DCS machines. It would also create much pressure to aggressively optimize the protocol implementation for speed. On the other hand, the actual finding that Selfhealing varies between linear and quadratic growth depending on spancut tells us as engineers that the method can grow gracefully and serve reasonably far into the future of the networks in which it is applied. These characteristics also imply that Selfhealing is well suited to benefit from future increases in processor speed by the time a Selfhealing implementation of today has exhausted. With higher than $O(n^2)$ functional dependency, the increase in feasible problem size bought by gains in processor speed is dramatically lessened.[1]

In the language of algorithm design, this is the problem of *time complexity* of an algorithm. Formal analysis of algorithms for their time complexity is described in texts such as [AHU74]. The formal statement that an algorithm has a time complexity $O(f(n))$ means that there are some constants C and n_0 such that for an input of size $n > n_0$, the running time of the algorithm is less than $C^*f(n)$. Such analysis tells us only the theoretical upper bound on the functional order that an algorithm can exhibit. As [AHU74] and others explain, this is most often vastly pessimistic compared to the performance realized in practice. The upper bound would be realized only with an absolute worst case input data set. In practise for large problems, a worst case data set can be so unlikely that it would have to be deliberately designed. A recent example of extremes in this discrepancy was reported by investigators who are using numerical simulated-annealing methods on the well-known travelling salesman problem (TSP) [Natu88]. This classic problem formally has exponential time complexity (it is NP complete). Yet these investigators report heuristic solutions to a 100 node TSP that, with their methods, run many thousands of times faster than the theoretical bound and often produce exact solutions.

For our present work we must adapt the classical questions of time complexity for algorithms to a context relevant for Selfhealing. In particular: (a) what is meant by the *size* of the *input* to Selfhealing?, (b) what is the *theoretical* time complexity of the Selfhealing protocol as a standalone piece of code ?, (c) does the theoretical time complexity of the protocol alone reflect the observed behavior of interacting Selfhealing nodes in a real restoration event? This chapter is devoted to these questions. The first two are relatively quickly dealt with and most of our effort goes into experimental characterization of the *effective* time complexity of Selfhealing using successively scaled versions of Smallnet as a primary study vehicle.

1[illegible].1 Definition of Network Size

If we take the view that multiple instances of the Selfhealing protocol, executing concurrently at each node of a network, collectively constitute a distributed computer on which a distributed heuristic algorithm for k-shortest link disjoint paths is executed, then we can view the *input* to the algorithm as being comprised of (a) the network topology graph (nodes and spans), (b) the identity of the failed span, (c) the number of working circuits on the failed span and (d) the number of spare circuits on every other span. The *output* from the algorithm is a specification, available locally in each node, of which crosspoints (if any) are to operate to synthesize the collective restoration plan. The computation time of interest is the time from failure onset until the last crosspoint operate decision that is determined anywhere in the network. The *size* of the input can be expressed in many dimensions; number of nodes, number of spans, number of lost circuits, total number of spares, but there are two notions of problem size which seem most meaningful and amenable to characterization. The first of these leads to the question : *(Q.1) How does the time for full restoration of any given span vary as the size of every span in the network increases uniformly in size, without topological alterations to the spans of the network?* The second dependency on "size" is expressed by the question: *(Q.2) How does the time for restoration of a given span increase with the size of <u>that</u> span, in a network where all other spans are fixed in size and topology?*

Over suitable periods of time, Q.1 approximates the way a real network grows. In reality not all spans will grow uniformly in step with each other and extensions to the topology may also occur for research purposes, a meaningful and systematic way to proceed seems to be to characterize the increase in restoration times as the size of *every span in the network is multiplicatively scaled by some factor F*. Results obtained under these circumstances will broadly address the notion of network growth, not in geographical extent or connectivity, but in capacity on existing spans. To obtain experimental results we will define F as the *network scale factor* and work with a selected *base network*. Each of the networks generated by scaling the selected base network will be given a new name : *basenet*x*F*. For example, Smallnetx2 will be a network with spans topologically identical to Smallnet but having twice as many circuits on every span.

Q.2 corresponds to the real world situation where the number of working circuits *on a given span* is increased without corresponding immediate growth in the remainder of the network. This may be a by-product of time delays in normal provisioning or it may be the result of unpredicted changes in network traffic patterns. This latter type of dependency on size is in fact what we have already been seeing in the trajectory plots which we have been using. We will make further use of this tool by examining trajectory plots of some very large span cuts (up to 50 lost circuits).

To address Q.1 or Q.2 we feel it is essential to use an experimental approach. We know from the protocol implementation that it can exhibit two extremes of computational complexity but the issue in practise is how its actual event-driven execution path is statistically amortized over these various code segments. Most code segments in it are linear in terms of number of ports processed but one procedure has the capability for $O(n^2)$ behavior in node size; and node size obviously scales as |[spans]|*F.

Both Q.1 and Q.2 involve dynamic interactions in a population of concurrent asynchronous tasks interacting in a massively parallel and indirect manner. No node executes the protocol in a predictable sequence because the execution path through the protocol is not determined within the protocol (unlike an algorithm) but is determined by hundreds of external events that drive the node through some actual sequence of sub-actions. These cause new events which influence the execution trajectory of other nodes and so on. The exact nature and order of these events and the exact state of each node when they occur are very important in determining which protocol paths are executed and what size of internal subsearch or other sub-problem is executed in that program path. In addition no statistical equilibrium is reached which would allow one to approximate the average distribution of sub-calls within the protocol. The data needed for engineering applications is really equivalent to knowing *the relative frequency* with which Selfhealing is required to execute linear code as opposed to $O(n^2)$ code in an actual concurrent event-driven restoration event. Having taken an experimental approach our results show that a complete range of execution characteristics are possible from almost purely linear growth to almost purely quadratic behavior, varying greatly between individual spancuts within one network.

11.2 Theoretical Time Complexity

In formal analysis the complexity of an algorithm is the order of the portion of the algorithm that has the highest individual time complexity. On this basis it is fairly easy to identify the theoretical time complexity of the Selfhealing protocol specification itself as $O(n^2)$. This derives from the behavior of the Tandem node. Square-law dependence on network size arises because it is possible in the limit for a signature change on one port at a Tandem node to require reexecution of the Tandem logic procedure, itself $O(n)$, once for every other spare port at the node. This leads to $O(n(n-1)) = O(n^2)$ theoretical time complexity for the Tandem node protocol.[2]

A separate, network-based viewpoint, leads to the same theoretical result. This is the realization that the worst-case complexity of path finding dynamics would arise for a spancut occurring in a network where the topology of the network dictates that the required restoration plan has (a) only one path per route and (b) a *pure cascade* of distinct routes of all lengths up to the length of the longest path. In such worst-case deterministic circumstances the n^{th} path to be created will require a path length of (n+1) spans. The dynamics of the absolute worst case rate of index mesh progression occurs under these circumstances because the index mesh of the n^{th} path is blocked in the first span after Sender flooding and remains blocked until the previous path has completely reverse linked. It has n spans of further progress to make but has to wait at each one for a prior sequence of lesser reverse-linking events. Using idealized signature transfer intervals and symbols to label each conceptual epoch of the dynamics for the given index as either one of forward flooding (F), temporary index blocking (T) or reverse linking (R), the resulting sequence of mesh expansions and reverse linking series in the pure cascade topology looks like:

path	route length	event sequence
1	2	F F R R
2	3	F T T T F F R R R
3	4	F T T T T T T T T F F F R R R R
4	5	F T T T T T T T T T T T T T T T F F F F R R R R R

The total number of epochs in the event series leading to completion of the n^{th} path follows the sequence $t_n = 2(n+1) + t_{n-1}$ which is a $\sum_{1}^{n} i$ form that generates the values 4, 9, 16, 25, 36 ... Therefore when approached from the viewpoint of worst-case network dynamics we also deduce that Selfhealing at the network level also has a theoretical time complexity of $O(n^2)$.

Whereas the above represents an extreme series of interlocking dependent mesh expansions, another set of topological circumstances gives the opposite extreme that Selfhealing can theoretically exhibit. This is in a network which provides all the topologically possible paths over *one or more routes of equal logical length.* In this case the equivalent table to the above has every entry comprised of an equal length sequence structured as {F...FR...R} where length of each part (F..F), (R..R) is the number of spans in the path length. These are circumstances of pure unimpeded forward flooding and reverse linking in perfect parallelism and they imply $O(1)$ theoretical dependency on network size and $O(n)$ linear dependency on path length. $O(1)$ dependency on network size means that there is *no increase* in restoration time with network size ! - any number of paths can be restored in the same time. This in fact is very nearly observed in some of the trajectories to follow. In practise there always remains a slight $O(n)$ dependency even in such an idealized topology because processor speeds are finite and cause a slight delay between path completions. One of the following experiments raises processor speed to P = 100 to indeed verify the principle that under certain conditions in Selfhealing there is literally *no dependency* of total time on the number of paths that are realized.

11.3 Experimental Determination of Time Complexity

The remainder of this chapter reports all span cut experiments on Smallnet under maximum path-finding stress with d = 1000, S= 8 kb/s, RL=6 and a 64 bit signature at F=2 and F=3. Because of technical limits on the network emulator, networks larger than Smallnetx3 could not be run in their entirety for all spancuts. A second set of span-specific experiments was therefore used to model selected individual spancuts in networks with scale factors effectively beyond F=3 in order to obtain several sets of data with enough points for regression studies of the rate of growth of restoration time with network size up to F=9.

11.3.1 Effects of Doubling Smallnet

A first tool to examine the effect of network growth is to look at the effect of the first *doubling in* network size on some sample trajectory plots. Fig. 11.1 shows five pairs of restoration trajectories consisting of the response to a particular spancut in Smallnet and the same spancut in Smallnetx2. First we note that in each Smallnetx2 trajectory, the number of paths created for each cut is twice that in Smallnet, as it should be. In addition the complete relative distribution of path lengths was confirmed to be unchanged from the distribution in Fig. 8.3 indicating that no overlength paths or topology shifts resulted simply from uniformly doubling network size. The implied preservation of each solution topology implied by constancy in PLE was also explicitly verified by graphical inspection of each spancut in Smallnetx2.

The sample trajectories show that doubling network size *did not double* the total restoration time in general. The actual increase in time varies for every particular spancut, even when two cuts involve the same number of paths. The following effects can generally be seen however: In each sample pair, the trajectories at F=1 and F=2 are almost identical over an initial phase of rapid path completions. This phase in each trajectory corresponds to the parallel realization of paths on simple two or three-span routes; paths that are notionally short, easy to identify, and established through fully parallel dynamics. In operational cases where less than maximum route finding stress is required this phase will sometimes be all that is required implying almost no penalty for doubling network size if adequate restoration is possible over relatively short routes. In cases where the first few paths found are reserved for priority traffic, this implies that the time for priority path restoration is almost assured to be independent of network size. In all cases inspected, the time of *first* path found is indistinguishable in Smallnet and Smallnetx2 and most often no significant departure between the trajectories is observed for at least the first 4 or 5 path restorations.

The next noticeable phase in comparing the trajectories of Fig. 11.1 is a steeper-slope regime where path completions are not so rapid and the x1 and x2 trajectories start to diverge. This corresponds to the exploitation of longer routes, whose propagation times are more significant, and which depend on collapse of the index meshes of prior paths for their realization. Longer

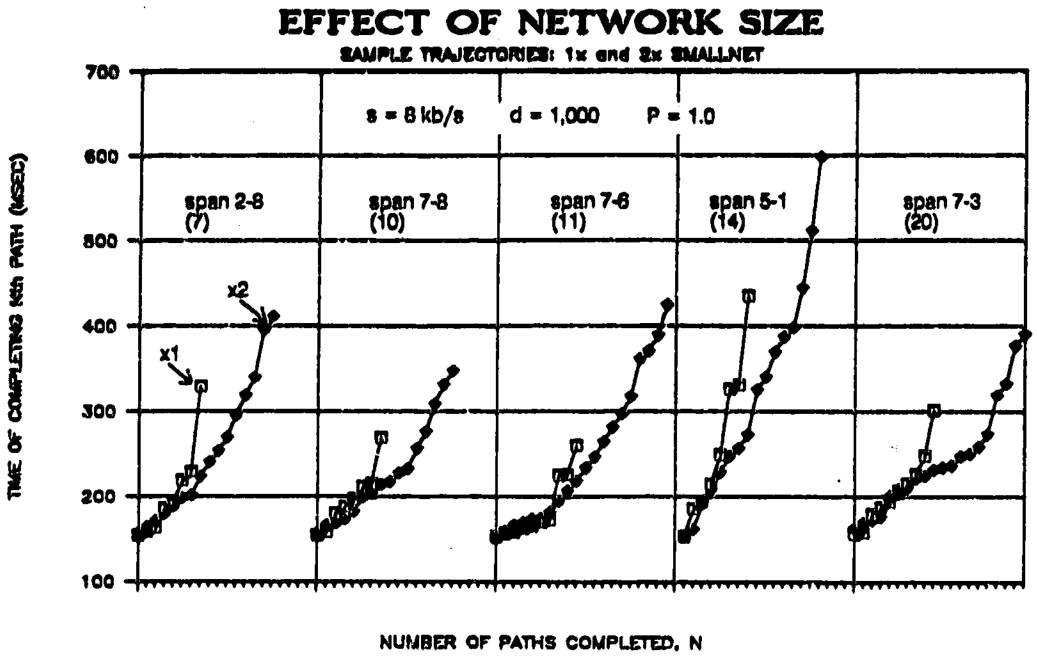


Fig. 11.1 Effect of doubling network size on five sample restoration trajectories in Smallnet.

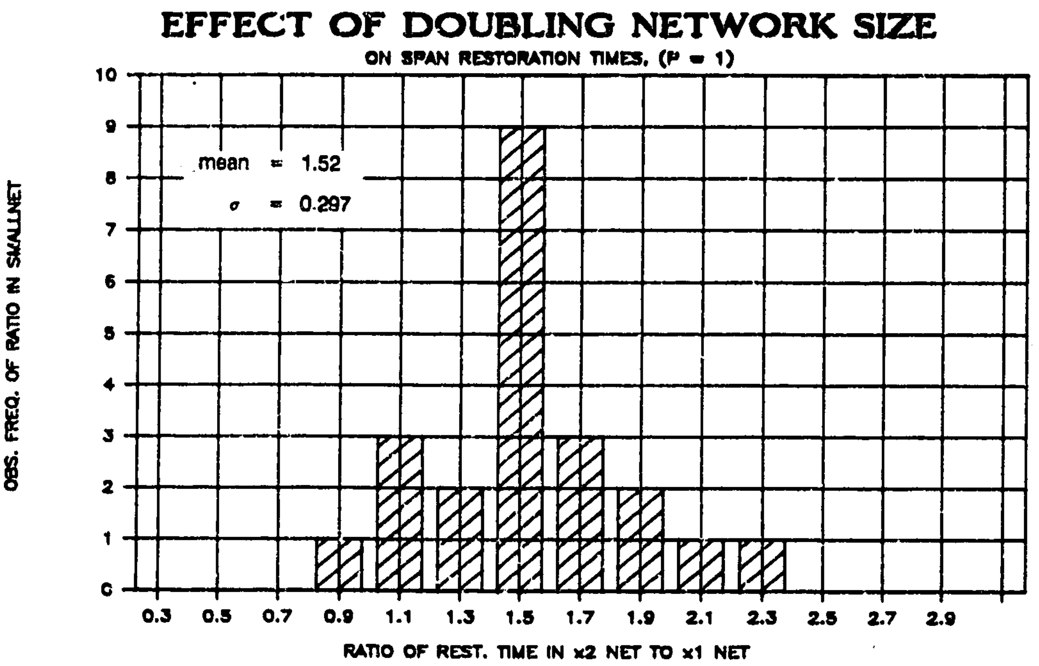


Fig. 11.2 Distribution of restoration time increase factors when size of Smallnet is doubled.

paths and TIB dynamics involve more Tandem node processing and so the Smallnetx2 path times tend to disperse and depart from the x1 trajectories and to be more smoothed by processing time effects.

The sample trajectories of Fig. 11.1 also show that the x2 trajectories are in each case *well ahead of half finished* by the time the corresponding x1 trajectory is complete. For instance, by the time all 8 paths for cut (2-8) are found in the x1 network, Selfhealing in the x2 network has found 13 out of the 16 paths that are to be found. Similarly 13 out of 16 are complete in the equivalent time in spancut (7-8), 13/20 in (7-6), 14/16 in (5-1) and 16/20 in (7-3). This is a desirable effect because it means that even while the time for *total* restoration increases with network size, *the proportion of failed paths which experience restoration by a given time is higher in a large network than in a small network of the same topology.* This beneficial property results from the *automatic exploitation by Selfhealing of whatever parallelism is possible* in the path synthesis process. The opposite effect will be manifested in a centralized restoration scheme or time-consecutive single-path-at-a-time schemes.

In four of the five sample trajectories, the time for *total restoration* in the x2 network is also *less than twice* that in the base network but in spancut (7-6) it is slightly more than doubled. When all cuts in Smallnet are considered, the distribution of the ratios by which *total* restoration time increased due to doubling network size, is shown in Fig. 11.2. The *average* increase due to doubling network size is 52%. In total, 20 out of the 22 spancuts (91%) experienced *less* than a proportional increase in restoration time. (No absolute reductions in time actually occurred; the one point apparently at 0.9 in Fig. 11.2 represents the histogram bin spanning 0.9 to 1.1. The value was 1.05.) When all span cuts are considered from the viewpoint of the proportion of span restoration occurring in the x2 network within the restoration time of the x1 network, Fig. 11.3 is obtained. It shows that on average restoration is 78.1% complete in the x2 network by the time restoration is complete in the x1 network.

The effect of doubling network size on the distribution of individual path times, as opposed to complete restoration times, is shown in Fig. 11.4. The lower curve is the observed frequency of path time in the x1 network, the upper curve is the observed frequency of path time in the doubled network and the middle curve is the x1 data normalized to the same peak as the x2 data for comparison of their shapes. The mean path finding time in the x1 network is 196 msec with a std dev of 49 msec. When the network size is doubled, the mean path time becomes 240 msec with std dev about 100 msec. This is an *increase of only about 22% in the average path time in response to a 100% increase in network size.*

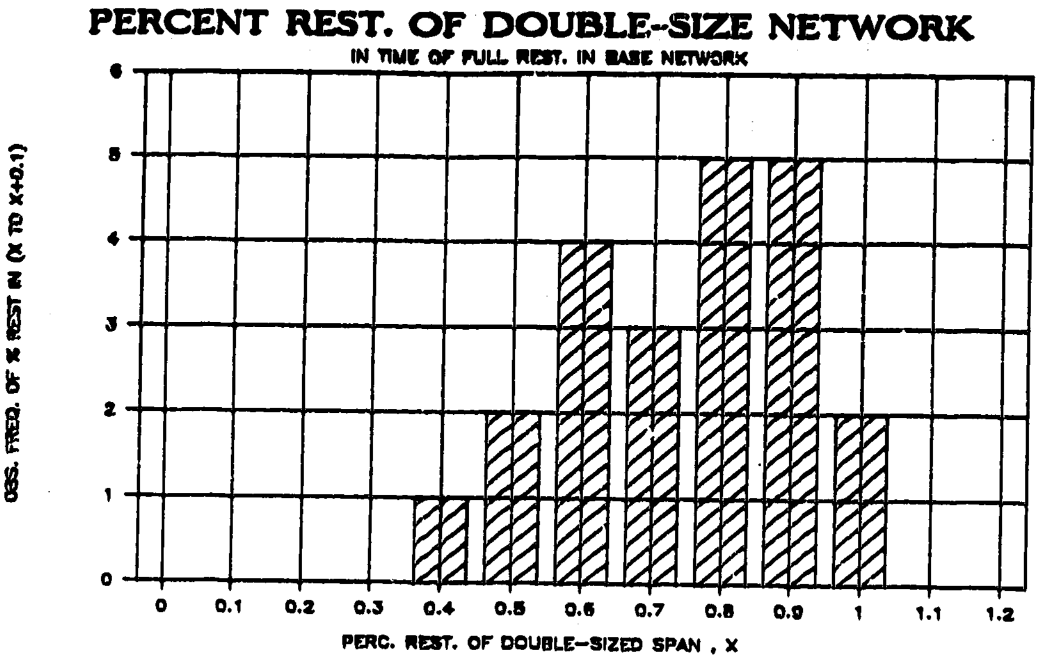


Fig. 11.3 Distribution of the percentage of restoration achieved in Smallnetx2 by the time of complete restoration for the equivalent spancut in Smallnet.

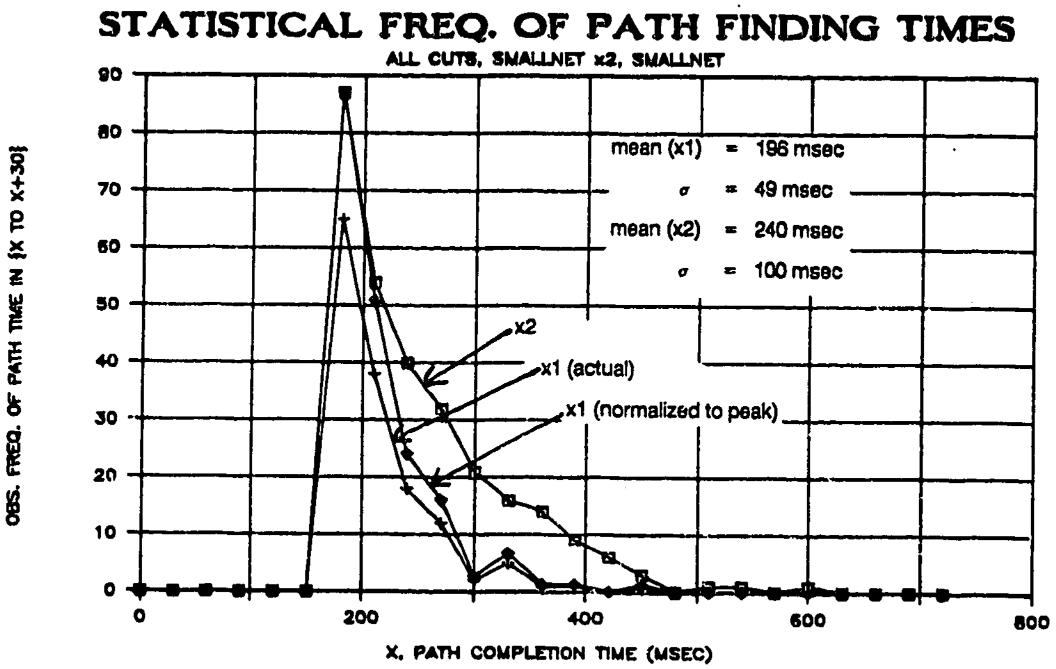


Fig. 11.4 Comparative distributions of path finding times in Smallnet and Smallnetx2.

11.3.2 Concept of a Size-Invariant Attractor Trajectory

In the trajectories of spancuts (7-8) and (7-3) of Fig. 11.1 it seems particularly noticeable that during about the first half of the Smallnetx2 trajectories *very nearly twice* as many paths are found in a given amount of time, as are found in the Smallnet trajectory. This could be explained by a tendency for the trajectory to depend more on the basic temporal sequence of signature dynamics required to establish various *routes* than on the number of paths directly. In this reasoning the basic sequence of F, T, and R dynamics that occurs would be largely identical under uniform network growth as these are determined more by the *span* topologies and the relative sizes of *spans*, than the number of paths in parallel on each route. This leads us to hypothesize that the underlying structure describing the trajectory of any given spancut is, to a first approximation, *independent of network size*, with the tendency to accumulate an arbitrary number of path completions at each stage in the basic underlying dynamic sequence.

We postulate that the shape of any restoration trajectory is determined, to a first order, solely by network topology and relative span dimensions, regardless of absolute number of paths. The principle is most easily seen if one thinks of Selfhealing dynamics (flooding, parallel mesh growth, collapse, further growth of other index meshes and so on), with infinitely fast nodal processors: In such a case, the sequence of dynamics that occur from start to complete realization of all possible (or required) disjoint paths would be in principle repeated exactly for any uniform scalar change in the network. The only difference would be the absolute numbers of signatures on each span. But with infinite processor speeds, there would be no time penalty for this.

In a real trial we expect this simplifying principle to be obscured, perhaps even hidden, by smearing and smoothing due to increased processor delays in larger networks and by instances where the order of signature processing events are not exactly repeated in the scaled network simply because the actual protocol implementation may happen to structure its lists differently in the two trial events. A way to test this principle is by an experiment where processor speed (P) is made essentially infinite. Then wherever the change in P does not result in a topology shift in the solution of a scaled network or cause any other major artifact, we would expect to see some form of underlying *attractor trajectory* (or a skeleton trajectory) which is essentially determined by route determination dynamics and is otherwise invariant with network size.

To test the hypothesis of an so-called attractor trajectory that dictates the underlying structure of the restoration trajectory for a given spancut under any degree of uniform network growth, the five sample spancuts of Fig. 11.1 were repeated with P =100. This has the effect of making processor speed virtually infinite with respect to the time for signature events to propagate across a span in Smallnet. Fig. 11.6 shows the corresponding x1 and x2 trajectory pairs with P= 100 and strikingly demonstrates the validity of the principle proposed. The trajectories are nearly identical in terms of their intrinsic shape. The x2 trajectories of course extend horizontally twice as far due simply to the fact that axis is path number and twice as many paths are found in the 2x trajectories. The most significant aspect is that almost without deviation every path completion plateau in the x1 trajectory is matched in the x2 network by a corresponding plateau of exactly *twice as many* path completions occurring at the very same time. The differences that do appear in (2-8) and (7-8) are due to random differences in TIB effects because the outcome of certain index competitions in one experiment differs with respect to the other. Nonetheless even when these differences occur, the four path completions that are slightly staggered out as two groups of two in the x2 trajectories are still easily recognized as corresponding to the adjacent plateau of two path completions in the x1 trajectory.

When the equivalent experiment is performed for all spancuts in Smallnet we get further evidence for this principle. Fig. 11.7 shows the distribution of the ratio of x2 to x1 restoration times for all span cuts in Smallnet when the network is doubled in size with P = 100. This figure corresponds exactly to Fig. 11.3 except that P is changed from 1 to 100. Fig. 11.7 implies that the effect seen directly in the five sample trajectories essentially also occurs in all spancuts. Aside from some minor discrepancies due to topology shifts there is *essentially no increase in restoration time* when Smallnet is doubled with P = 100. Figures 11.6 and 11.7 together confirm the principle that the underlying structure of a restoration trajectory is fundamentally unchanged by the number of path completions in the trajectory. In practise, exact processing times do depend on absolute size of the DCS node and on certain elements of chance in the protocol such as when a transmit link becomes free and two equal repeat count indices compete for the link. These effects generally flesh out or soften the actual appearance of the trajectory, dispersing path times and rounding out the idealized zero-time path completion plateaus of the underlying attractor trajectory. Nonetheless this is a helpful as a conceptual principle which tells us that Selfhealing speed theoretically has no dependence on the number of paths realized as long as the set of routings taken by these paths is unchanged. This implies both that uniformity in network growth is indeed desirable to maximize the predictability of Selfhealing times as the network grows and that increases in processor speed can in principle make Selfhealing times almost invariant under arbitrary amounts of uniform growth.

11.3.3 The Separate Influences of Network Topology and Network Size

We can go on to apply the above insights to the resolution of an apparent paradox regarding the speed of Selfhealing. The results so far show that the time of total restoration grew in a less-than-proportional manner when network size was doubled in Smallnet. Now consider Fig. 11.5 which is the all cuts trajectory plot for Smallnetx2. Fitted to the edges of the region occupied by these trajectories in path-number versus path-time space are two curves defining bounds within which all trajectories are approximately contained. Now we have an apparent paradox: The

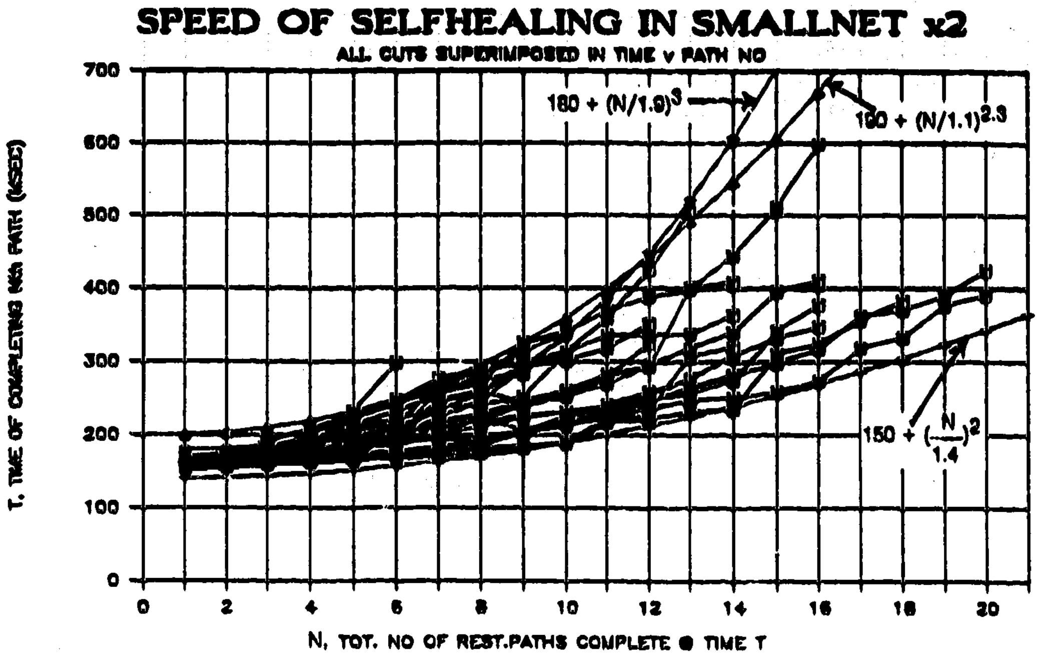


Fig. 11.5 All span cuts restoration space diagram for Smallnetx2.

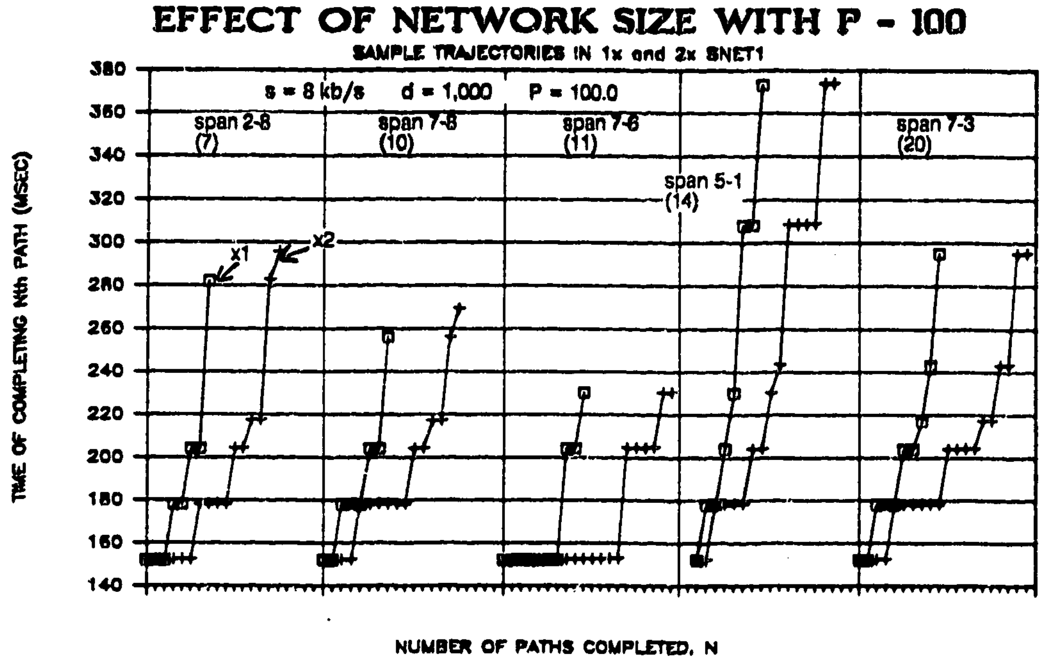


Fig. 11.6 Five comparative restoration trajectories from Smallnet and Smallnet x2 with P = 100 to test topology template hypothesis.

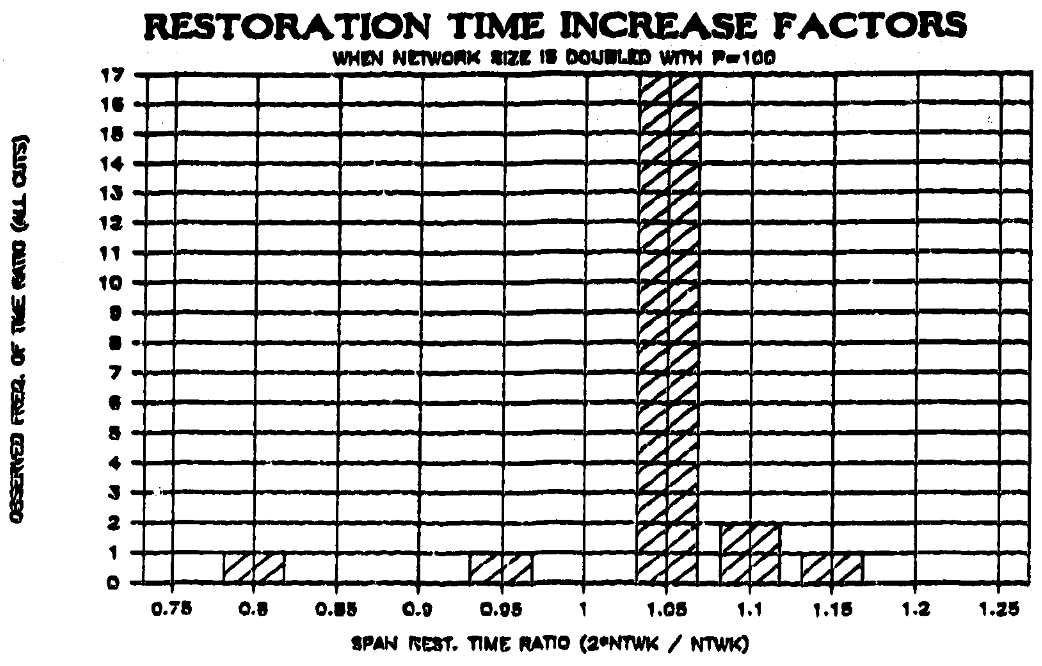


Fig. 11.7 Effect of doubling network size with P= 100: distribution of restoration time increase factors.

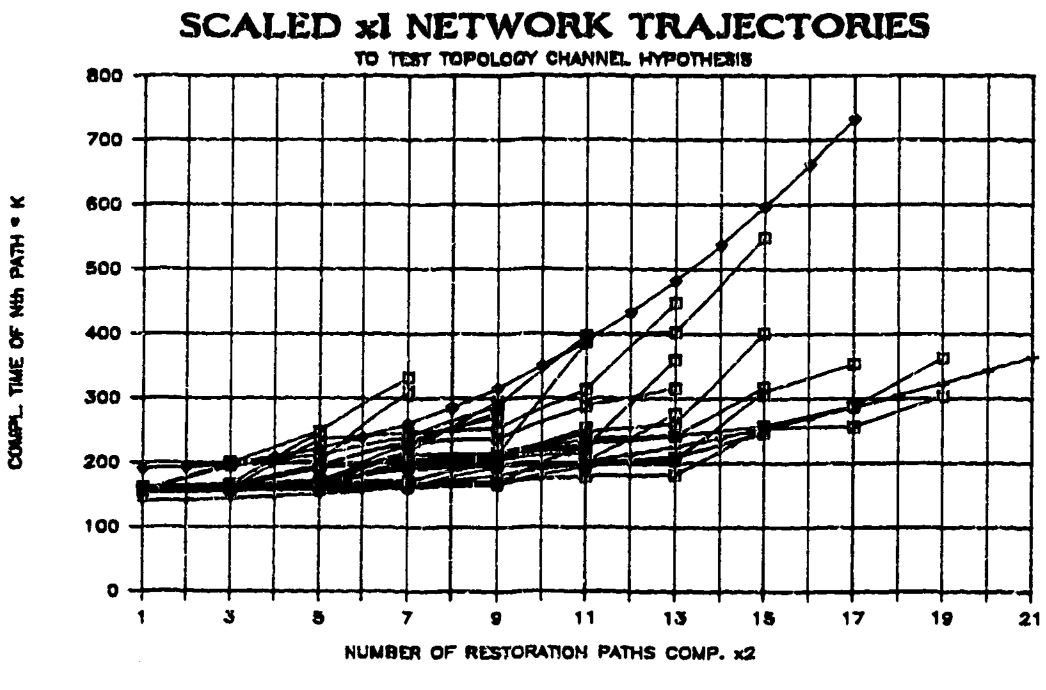


Fig. 11.8 Fitting Smallnet restoration trajectories into topology bounds obtained from Smallnetx2 to test topology template hypothesis.

bounds in Fig. 11.5 show that the time of completing the nth path grows with path number at a rate apparently quite close to $O(n^2)$. How can this be consistent with the previous findings that the time of total restoration grew less than proportionally when Smallnet was doubled?

The answer lies in the separate effects of basic network topology and network size as implied by the principle of a size-invariant attractor trajectory. The principle explanation is that for any given spancut two independent factors are at work : (a) the topology of the network (spans and their relative sizes) determines the structure of the attractor trajectory and the number and position of each plateau in the trajectory at which an arbitrary number of path completions may occur. (b) network size (absolute no of circuits on each span) determines the total processing load on the nodes of the network, hence their signature processing speed and the actual achieved rate of path completions during the plateau opportunities provided by the attractor trajectory. This explains the paradox because it is completely possible for the attractor trajectory to rise as $O(n^2)$ while a uniform doubling of the network produces no increase in overall restoration times. The point is that the idealized attractor trajectory is like a template determined purely by the required dynamics of Selfhealing which are encoded into the network *relative* span sizes and topology. Its intrinsic shape may be steep but if processors are fast enough they can accumulate all of the extra paths of a scaled up network without deviating from the intrinsic attractor trajectory for that spancut. Of course when P is noticeably finite, these conceptually separate effects are not so cleanly decoupled. The principles still predict however that in a case where P is finite, the observed growth rate of restoration times (final point of a trajectory) can be quite independent of the functional shape of the trajectory itself.

This finding implies that if the restoration trajectories for all span cuts in a network are overlaid, as in Fig. 11.5 they will sweep out some area of path-number versus path-time space that is primarily a characteristic of the basic network span topology and the relative span dimensions. We call this region a *topology template* because it is determined solely by topological factors, and in the case of 'fast' processors, it bounds all restoration trajectories of the network *regardless of network size.* The *shape* of this topology channel can rise faster than total restoration times actually grow with network size.

The topology template for a network reflects the extreme ranges of topological ease and difficulty of the series of Selfhealing dynamics that are involved in the process of successive route finding for cuts in that network. A trajectory running near the upper bound may require a relatively tortuous series of single path routes with many waits for prior mesh collapses. On the other hand, a trajectory running near the lower edge may be one in which all required paths are found immediately over large two-span routes. The important concept is that in the limit, an $O(n^2)$ rise of the topology template indicates only the intrinsic complexity of finding the nth path within the restoration plan with respect to a prior path in the same plan, not the extent to which actual total time of complete restoration rises with increasing total size of the restoration event in terms of number of paths.

With *finite processor* speeds, an actual growth of total restoration times is observed and this can be thought of as *a scaling in the topology template* (in the vertical axis of the RS port) due to finite processor speeds. This can be tested with the available data by determining the topology channel from the x1 network, multiplying the template shape in the abscissa direction by two (to normalize the abscissa) and multiplying the ordinate values by the *average* increase in total restoration time observed above for Smallnetx2, which is much less than $O(n^2)$. This creates a renormalized trajectory template that should describe the family of actual trajectories in the x2 network if the notion that processor speed effects from network size are separable from effects due to inherent encoding of the dynamic sequence requirements in the network topology. To apply this test on the available data, the equivalent scaling operations were actually performed in the reverse order to take advantage of the more voluminous data in the x2 network on which to fit the topology template. (ie. The topology template was obtained from bounds on the x2 network and then the x1 trajectory data was scaled (2.0 in x, 1.5 in y) into it to see if the x2 topology channels results in equivalent bounds describing the x1 data.) The result is shown in Fig. 11.8. Given the relatively small numbers of circuits in the x1 network, the match seen in Fig. 11.8 is taken as supporting the hypothesis that uniform network growth with finite processor speeds can be treated as an expansion of the topology template and that the overall family of trajectories at any size can be described by normalized application of the basic topology template shape.

11.3.4 Regression Studies with Higher Scale Factors

Fig. 11.9 shows how the total restoration time averaged over all spancuts in Smallnet the varies with F = 1, 2, and 3, with +/- 1 std. dev bounds. The path-time versus path-length scatterplots for the corresponding all span cut data sets are shown in Fig. 11.10 (a) to (c). Fig. 11.11 shows the mean path times for each path length in the three networks and linear regression lines fitted to describe the mean path time as a function of path length at each network size. Fig. 11.12 shows how the average time to create a path of a given length varies in each network size and shows an $O(F^{2.1})$ reference curve. The 3 data points representing mean values of restoration time are consistent with a square-law growth of the average restoration time but within the statistical bounds there is considerable latitude for individual spans to grow more or less steep than the regression curve through the mean values. More than 3 data points are needed to obtain a high quality empirical estimation of the rate of growth in restoration time.

To obtain such data nine out of the 22 spancuts in Smallnet were chosen for individual study of their restoration times as network size increased using the *span-specific method of network scaling.*[3] The spancuts were selected by virtue of their individual trajectories as seen at x1 and x2

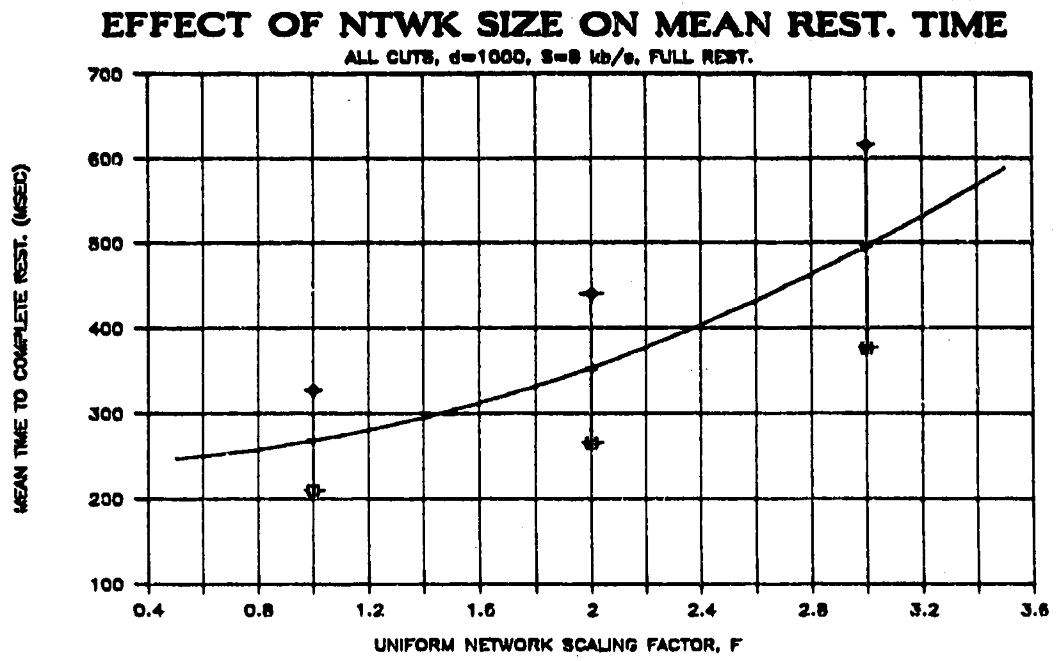


Fig. 11.9 Mean time of complete span restoration in Smallnet as a function of network scale factor, F.

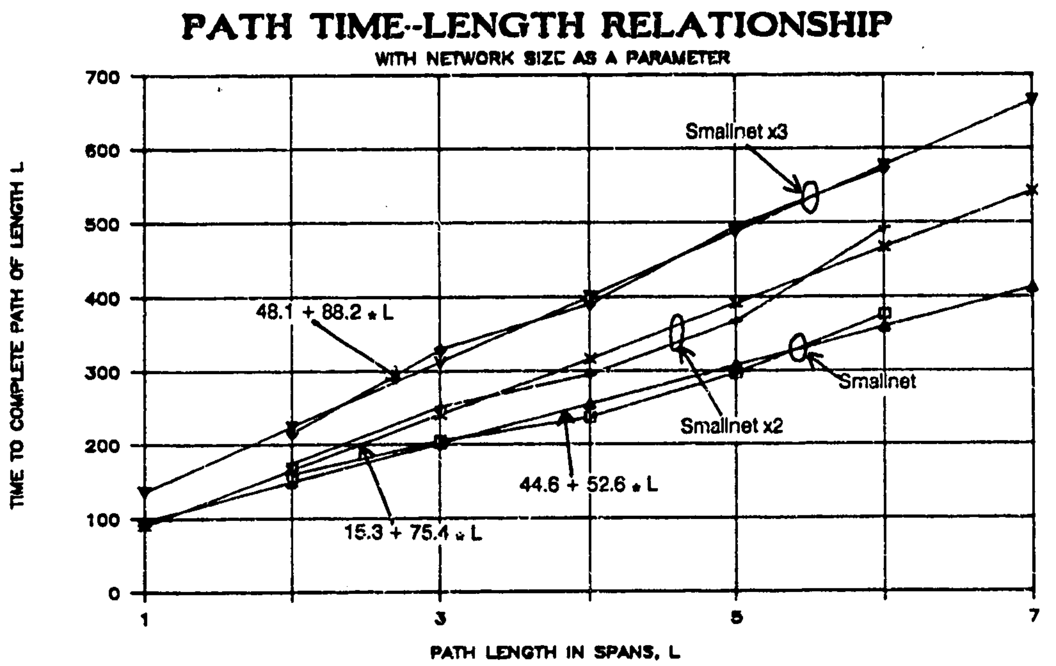


Fig. 11.11 Linear regression fits to mean path time as a function of path length with network size as a parameter.

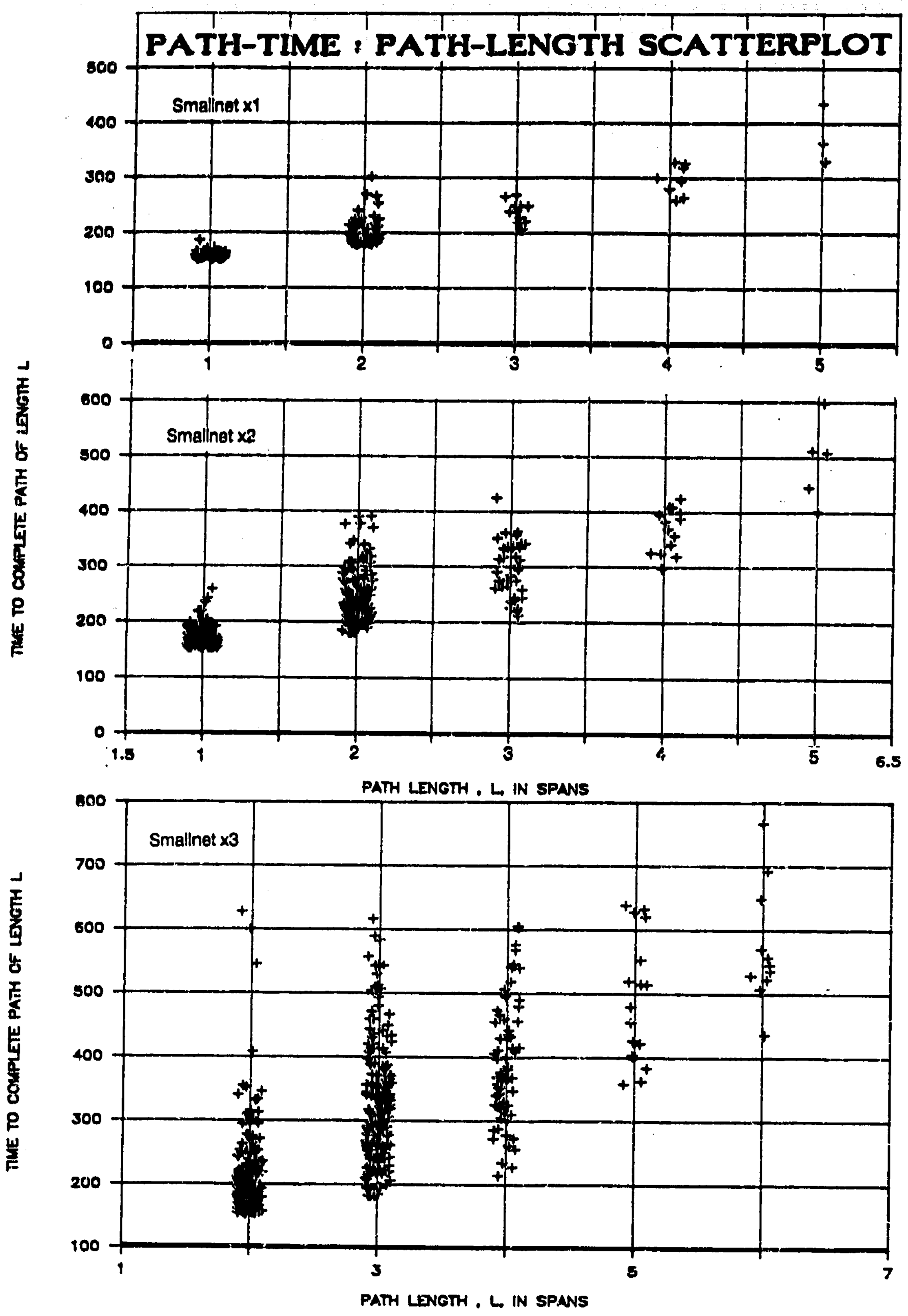


Fig. 11.10 Path time versus path length scatterplots for all spancuts in scaled versions of Smallnet (a) F = 1, (b) F = 2, (c) F = 3 .

network sizes and include the most steeply rising trajectory, the spancut with the most paths, the richest path diversity, and the most parallelism. Restoration time data for each selected spancut was taken from the x1, x2, x3 all-span cuts series and augmented with span-specific runs in networks scaled up to the largest size for which that spancut could be run by the emulator.

Generalized polynomial regressions were then performed to maximize the goodness of fit to the available data points, both in coefficients and functional degree. Because the *order* of the best-fitting polynomial is our primary interest this was first estimated by a best linear regression fit to the *logarithm* of the data as a means to obtain an initial estimate for the order of the polynomial that would then be fitted directly to the data. The initial order of the polynomial was then iteratively adjusted to find the polynomial order (within +/- 0.1) at which the regression package was able to produce the maximum possible correlation. The results of best regression fitting to the form $t = c + b{*}F^a$ are shown in Table 11.1.

Table 11.1 shows that every spancut really has its own individual characteristic ranging from much less than linear growth where route parallelism is high (7-6) to essentially square-law growth when the paths are longest and TIB events are dense (5-1). The spancuts in Table 11.1 generally exhibited the highest growth with network size of the 22 spancuts in Smallnet. All other spancuts generally demonstrated a *less than proportional* response.

Table 11.1 Large Network Regressions for Selected Span Cuts in Smallnet

Span Cut	Regression Results Constant	Coefficient	Power	r^2	Basis data at 1x, 2x, 3x plus;
(9-5)	104	61.9	1.53	0.981	6x, 9x
(6-2)	85	92.4	1.72	0.995	8x
(3-0)	123	19.1	1.8	0.994	6x, 8x
(3-2)	36.8	92.1	1.59	0.989	8x
(3-1)	59.1	109	1.17	0.991	6x
(7-6)	4.9	259	0.784	0.994	4x
(8-7)	206	44	1.8	0.999	6x
(5-1)	363	47.3	2.1	0.997	6x
(4-5)	138	2.71	1.73	0.991	4x, 5x, 6x, 7x, 8x, 9x

11.4 Summary

From the results of this chapter we conclude that for engineering purposes the upper bound of $O(F^2)$ is a sound and conservative assumption for worst case planning purposes for the increase of restoration time with uniform network growth factor. Theoretical considerations predict and experimental assessments confirm that the actual rate of increase can vary from almost no dependency on uniform network growth if P is high. Typically, most restoration times can be expected to grow at about $O(F^{3/2})$ or less with P = 1. If processor speeds less than a SUN 3/60 are used, an increase in average growth rate will ensue up to the limit of $O(F^2)$. If faster processors are used, the restoration times of a greater number of spans will tend towards *no dependence* on uniform growth in network size. When the growth of a single span is considered without corresponding uniform increases in the network, the time complexity is determined by the route-establishment dynamics dictated for Selfhealing by the topology of the network. Changes in network size which trigger changes in solutions towards a greater number of distinct routes will tend to increase restoration times up to a maximum of $O(r^2)$ where r is the number of distinct routes required in the restoration plan. In a topology where all required paths are found over a set of equi-length routes, the time complexity is in the limit of fast processors, O(1) with number of paths and O(L) in length of paths.

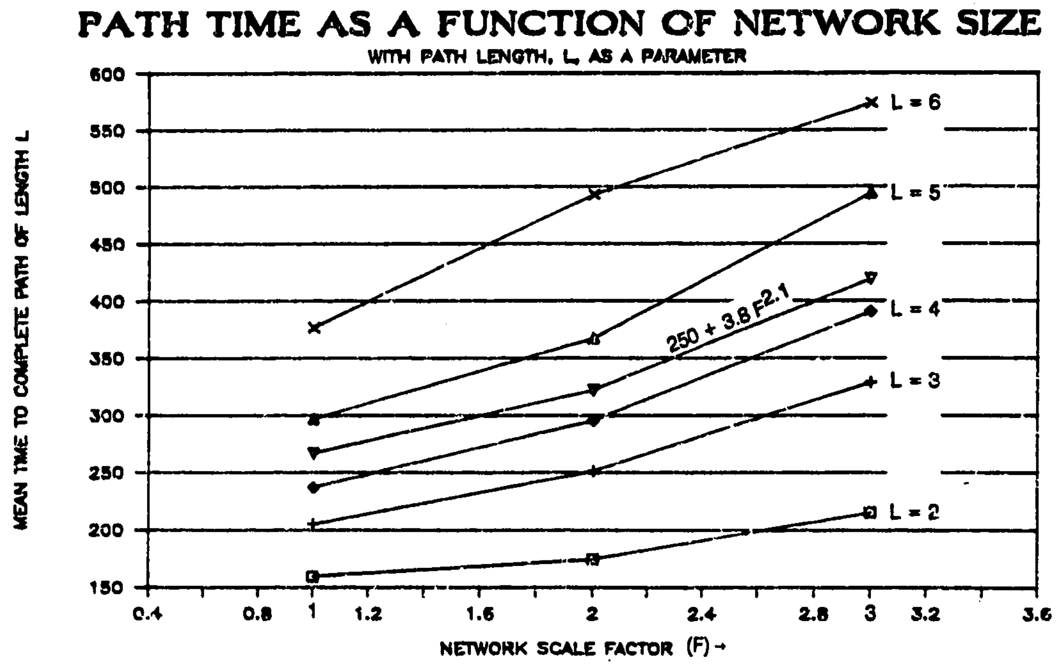


Fig. 11.12 Path completion time versus network size with path length as a parameter, based on scaled versions of Smallnet.

Chapter 12

Effects of Processor Speed and Signature transfer time

In this chapter we consider processor speed (P) and signature transfer time (STT) as engineering parameters which influence the performance of Selfhealing. It is obvious that each of these directly influences the real time speed of Selfhealing but in Ch. 9 we saw evidence that the *ratio* of signature transfer times to nodal processing delays influenced *path topologies* as well, via path spreading. We are here interested in the direct dependence of restoration times on variations in STT and P as well as determining a guideline for S/P ratio (or strictly STT*P product) which marks the onset of path spreading effects. Physical considerations based on the principles of Selfhealing imply that path times will be directly proportional to STT if the node processors are arbitrarily fast. Conversely, if STT times approach zero because of relatively short spans and signalling rates, we expect path times to vary inversely with P. In a generalized application where both processor and STT delays are significant, one needs to understand the relative balance of these effects so as to apply extra efforts for greater processing power or faster in-band signalling channels as appropriate to most influence the overall performance. We first look parametrically at the separate effects of P and STT with the other parameter fixed as follows: STT = 13 msec when P is varied, P = 1.0 when STT is varied. We later interpret the findings in terms of STT and P parameter combinations and identify approximate regions where STT is limiting, where processor speed is limiting, and where path spreading effects begin to manifest themselves.

This chapter makes use of the following symbols, some of which were introduced in Chapter 7:

P = processor speed normalized to Sun 3/60

S = signature transfer rate (kb/s)

d = link distance (km)

nbits = number of bits in a signature (including all alignment and checkbit overheads)

v^* = link propagation velocity (km/sec)

and the signature transfer time (STT) is defined as:

$$STT = d/v^* + nbits/S \qquad (12.1)$$

12.1 Effects of Varying Processor Speed

To study the effect of processor speed, the Selfhealing emulator was run with P = 10, 5, 1, 0.5 and 0.1 times the actual processor speed in Smallnet. The various values of P were effected by multiplying the self-measured (calibrated) execution time of the protocol by the factor P before scheduling external events generated by each node to appear in the virtual time concurrent emulation method described in Ch. 7. P = 1.0 represents a SUN 3/60 running compiled 'C' language code under the SUN/UNIX operating system. The SUN 3/60 uses a 68020 32 bit processor clocked at 20 MHz and is quite representative of the processing power typical built into processor clocked at 20 MHz and is quite representative of the processing power typical built into a modern DCS design. In experiments where P was varied RL = 6, d = 1,000 km, and S = 8 kb/s is maintained and a signature length of 64 bits applies. These conditions give a constant signature transfer time (STT) on all spans of Smallnet equal to 13 msec (1,000 km * 5 microsec/km + 64 bits / 8 kb/s).

The effect of processor speed on the restoration trajectories of two sample spancuts [(2-8) and (6-7)] is shown in Fig. 12.1. This series of trajectories for (2-8) show that a ten-fold *increase* in processor speed with respect to P = 1.0 reduces the time of total restoration by 44.8 msec or 24.3 % if we neglect the 100 msec constant time for alarm collection. The corresponding reduction of total time for (6-7) is 21.6 %. We can see from the series of trajectories in Fig. 12.1 that at P = 10 restoration time is essentially asymptotic and will be reduced very little by any further increases in P. Any further increases in processor speed yield no benefit because overall restoration time is strongly essentially determined by signature transfer times in this region.

Even in the absence of the series of trajectories in Fig.12.1 which in which the asymptotic behaviour is visibly apparent, the P=10 trajectory is by itself indicative of this fact by virtue of the extreme flatness on each plateau of path accumulation in the P=10 trajectory. As discussed in Ch. 11 this is evidence that we have essentially exposed the ideal attractor trajectory that underlies the dynamic structure of all trajectories for that spancut. We think that in general, whenever the pronounced characteristic geometric sharpness is seen, such as here for P= 10, one can infer that processor delays are insignificant in the dynamics of that restoration event.

Fig. 12.2 shows the distribution of path completion times for all span cuts with processor speed as a parameter. The general indication from the selected spancuts (2-8) and (6-7) in Fig. 12.1 is corroborated in the path time statistics of Fig. 12.2 in the sense that a tenfold increase in processor speed has a very minor effect on the overall distribution of path times.

A tenfold *decrease* in P is very significant however, increasing restoration times 4 to 5 fold on average. When P is *reduced*, from 1.0 to 0.5 and 0.1, the overall speed of restoration shows a 25% increase when P is first halved from 1.0 to 0.5 in Fig. 12.2 . When P is reduced further to 0.1 from 0.5, a 2.7 times increase occurs for (2-8) and a 4.1 times increase occurs for (6-7) indicating a clear transition from STT-dominated operation to processor-delay dominated operation in the region between P=0.5 and P =0.1.

This behavior is understandable as a result of the parallelism in Selfhealing. With parallelism, the net system time tends to the largest *individual* delay contributors, not the *sum* of all delays involved. As long as solution topology does not shift, the overall dependency of restoration time on processor speed is remarkably well represented by a functional form which arises from the simple notion of two delay elements in parallel, one of which is constant (STT limited operation) and the other having an approximate $1/x^a$ characteristics (inverse dependence on processor

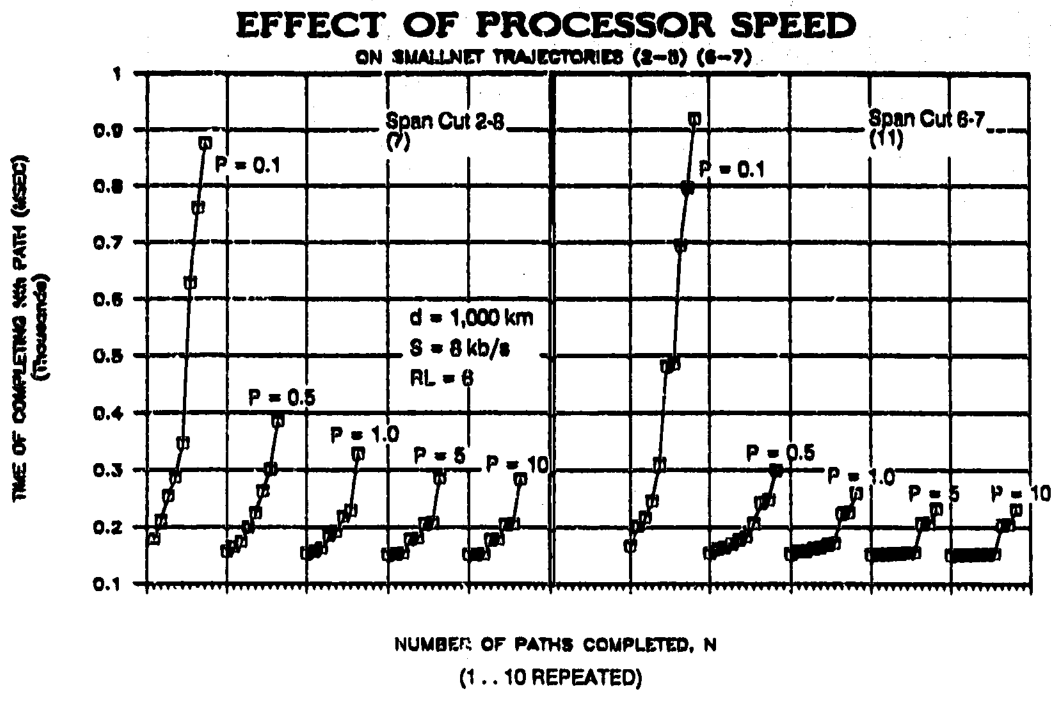


Fig. 12.1 Effect of varying processor speed from P=0.1 to P=10.0 on two sample restoration trajectories.

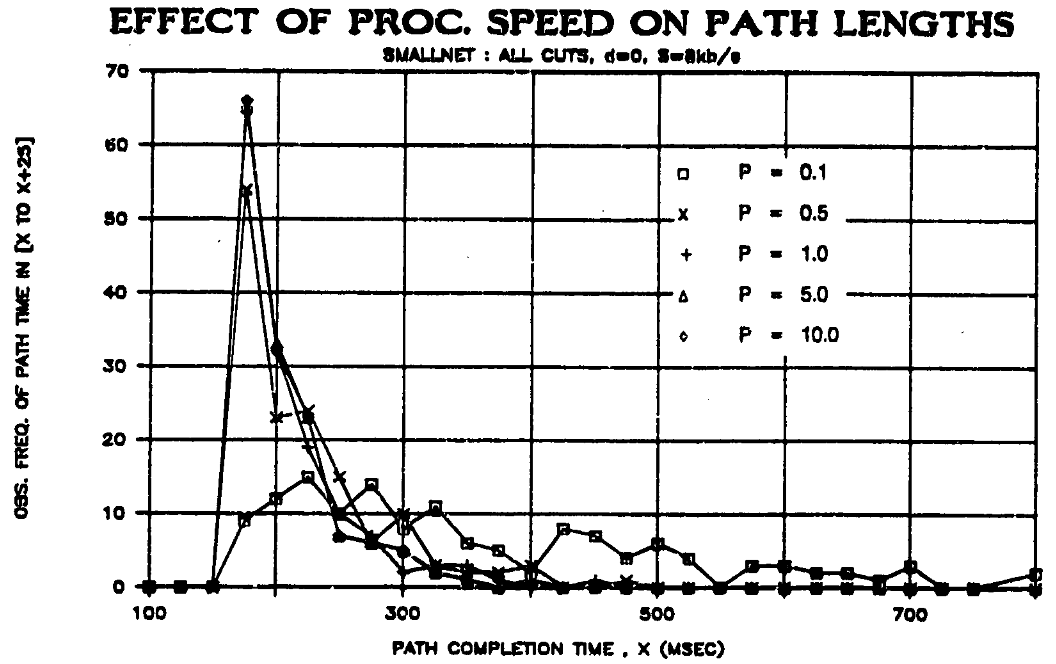


Fig. 12.2 Distribution of path completion times in Smallnet with processor speed as a parameter.

speed). For instance, every final restoration time shown for (2-8), plus a data point for P =0.01 which is not shown, is predicted within 1% by the relation $t_R(P) = 281 + 48.4 / P^{1.06}$. The data for span cut (7-6) is similarly described within 1% by $t_R(P) = 230 + 30.3 / P^{1.36}$.

The data in Fig. 12.1 also tell us that a Selfhealing design based on the particular combination of a SUN 3/60-equivalent DCS processor, 8 kb/s (F-bit) signature modulation, and a 64 bit signature format constitutes a reasonably well-centered system design for long-haul network applications. By a 'centered' design we mean that performance in this regime is not extremely sensitive to processor speed nor is it independent of it. Under the conditions of Fig. 12.1 it is apparent that some improvement would be obtainable through higher processing speed but great efforts to provide more processing power in the DCS design would probably not be justified.

The effect of a ten-fold increase or decrease in processor speed on the all-spans trajectory plot for Smallnet is shown in Fig.s 12.3 and 12.4. In Smallnet with P=1.0, the longest restoration time occurs for (5-1) and is 435 msec. With P = 10, (5-1) remains the longest restoration time but it is reduced to 375 msec. With P = 0.1, spancut (5-1) again has the highest restoration time, 969 msec. The trajectories in Fig. 12.4 (P = 10) have a regular, discrete time or skeletal sort of appearance compared to P=1 or 0.1. This reflects the tendency with arbitrarily high P for all path times to become quantized to some nearly whole number of STT times because they depend essentially only on the dynamic sequences of F, T and R events that occurs, each requiring an almost equal amount of time when P is high and span lengths are constant.

12.2 Effects of Signature Transfer Time

Signature transfer time (STT) is the total time from when a signature is written to a TS register at one DCS port to when that signature is subsequently received in full, error checked in hardware, and raises an interrupt in the distant receiving port. To systematically study the effects of variation in STT we set d = 0 and nbits= 64 in the emulator so that STT can be conveniently parameterized solely by varying S. In this case STT = 64/S seconds. Of course any actual combination of distance, velocity, signature length, signature transfer rates and hardware delays that yields the same STT represents an equivalent case. This way of parameterizing STT is reflective of the reality that S is in practice the only variable contributing to STT that the engineer will have under design control. All others factors are given by the network. Five values of S have been considered: 80, 64, 13, 8, and 1 kb/s. S= 80 kb/s and S=1 kb/s values were chosen as arbitrary extremes of the high and low ranges of interest. S = 8 kb/s is the rate obtainable in an existing asynchronous DS-3 based transport network if *gated* F-bit modulation is used as described in [Grov87a]. S = 13 kb/s is the rate obtainable using *continuous* F-bit modulation at an F-bit spacing of 20 as also described in [Grov87a]. Gated F-bit modulation is a more conservative approach which ensures strict bounds on reframe time, whereas continuous F-bit modulation will not significantly affect

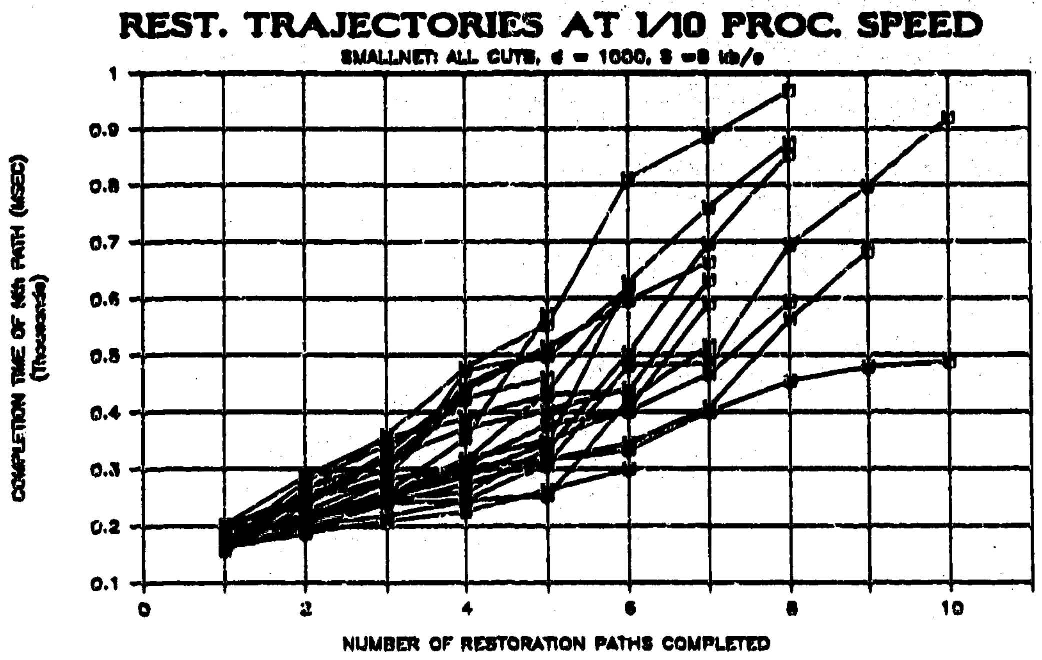


Fig. 12.3 All spancuts restoration space diagram for Smallnet with 1/10th nominal processor speed (P=0.1).

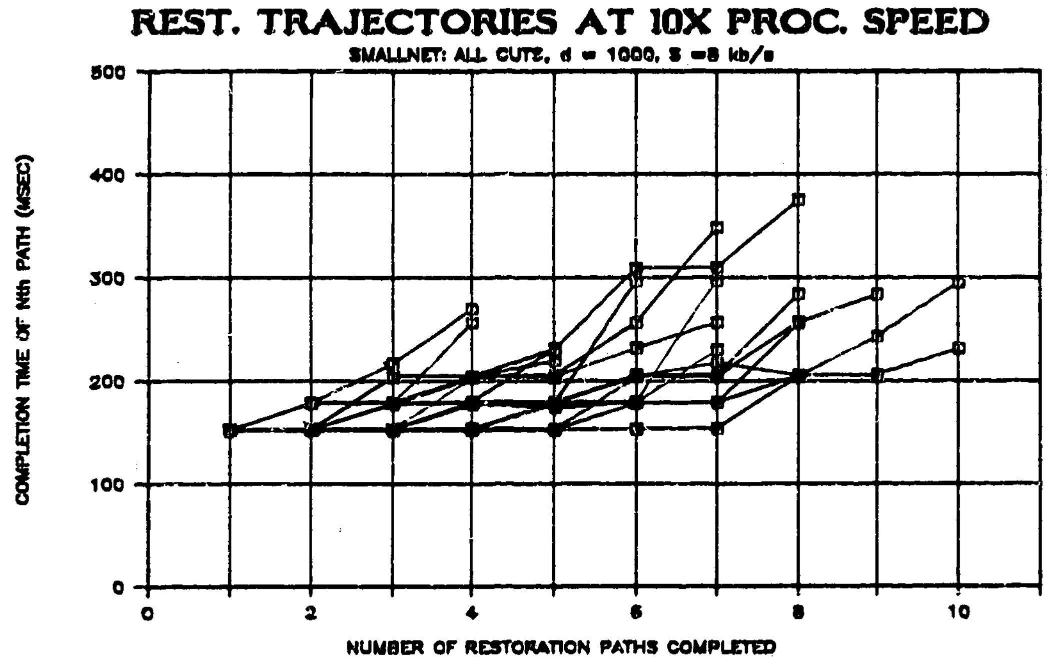


Fig. 12.4 All spancuts restoration space diagram for Smallnet with 10 times nominal processor speed (P=10.0).

mean reframe time but does not strictly truncate the probability tail on the pdf of reframe time. The S = 64 kb/s rate derives from two technical possibilities: (a) use of a reserved overhead byte in the SONET STS-1 format [SONET88] or, (b) use of a 64 kb/s overhead channel in the SYNTRAN DS-3 format [SYNT87]. P = 1.0 in all of the results where STT was varied.

Let us again use the approach of inspecting a set of sample trajectories to identify STT-dependent effects in Selfhealing dynamics. Fig. 12.5 shows the restoration trajectories of six span cuts from Smallnet at S = 8, 13, 64 and 80 kb/s. The results with S = 1 kb/s produced such high restoration times that we omitted them from these plots. With S=1 kb/s, the mean path completion times rise to 530 msec, the mean restoration time becomes 827 msec and several restoration times exceed 1 second. If the plots were scaled to fit the 1 kb/s results, we would lose all resolution on the trajectories of real interest. The reduction of S to 1 kb/s produced by far the greatest single increase in restoration times seen in all of the experiments conducted to date on Smallnet, strongly implying that in practice with P = 1.0 we probably are not interested in any signature modulation method that yields less than 8 kb/s, (the next lowest rate for which we have results).

In the trajectories of Fig. 12.5, increasing S (reducing STT) does not seem to change the basic form of any trajectory greatly but instead produces trajectories that are more or less multiplicative scalings of one another. This is particularly true in comparing the 8 kb/s and 13 kb/s trajectories. For example, multiplying the S=13 kb/s trajectory of spancut (7-8) by 1.17 produces a set of path completion times that very closely overlies the S=8 kb/s trajectory for the same spancut. This implies that, at least for S < 13 kb/s, processing times are tending to insignificance and restoration times are consequently exhibiting a pure dependence on the F, T, R dynamical sequence required for their construction and the signature transfer time required in each stage of these dynamics.

The trajectories for S = 80 kb/s and S = 64 kb/s are very nearly identical although except for a few regions where they cross over one another. Spancuts (8-2) (7-8) and (6-4) each exhibit regions where the S=80 kb/s trajectory *rises higher in time* than the corresponding S=64 kb/s trajectory. This interesting effect is due to topological shifting towards overlength path routings in the S=80 kb/s results, due to the path spreading effect. We already saw in the Bellcore network that when span distances were short and P = 1.0, even S = 8 kb/s was sufficient to allow a notable increase in instances where the forward expansion of an index mesh reaches the Chooser over an overlength route without TIB as the cause. In such *path spreading*, the time required for propagation in mesh expansion is so small that differential delays in nodes are significant enough that overlength paths may be created via smaller nodes which are typically faster in signature processing than large nodes in the desired path.

The cases in Fig. 12.5 in which the S=64 kb/s trajectory is *below* the S= 80 kb/s trajectory are caused when path spreading has occurred at 80 kb/s with respect to the S=64 kb/s paths.

The result is a longer net time for some of the S=80 kb/s paths because, when the Chooser is reached earlier via a 'long' path in the forward direction, Selfhealing can be thereafter destined to follow a reverse-linking process over that longer path. In the end this takes more time than via the shorter path. The onset of path spreading sufficient to actually reverse the benefits of a faster S in some cases suggests the notion of an *optimum S*. At *low S* (high STT in general), restoration times are high but *reduce* almost proportionally as S is increased. In this region, PLE also tends to be very high and no path spreading occurs because link delays dominate over processor delays. Then as S is increased path spreading begins to be quite significant. With the data points available this appears to occur between 40 kb/s and 64 kb/s. With significant path spreading a reversal in the trend to lower restoration times can occur because topologically longer paths tend to occur more frequently. The creation of longer paths inevitably requires greater reverse-linking time and so, overall, restoration times again begin to *rise* as S is further increased. The optimum is ultimately fairly shallow however in terms of speed of restoration because when S is high, the additional time for longer reverse linking is by definition reduced in proportion to the higher S.

We nonetheless want to identify the S value at which path spreading 'just begins' because that will also be optimal from the viewpoint of sustaining a high PLE by avoiding path spreading. Although there is some path spreading at 64 kb/s, Fig. 12.5 is a fairly clear indication that with P=1.0, the highest rate of signature transport worth pursuing is probably 64 kb/s. At 80 kb/s strong path spreading effects begin to show but overall restoration times do not improve any further.

Two measures were obtained to quantify the way in which path spreading and TIB effects collectively depend on S. The all-spans distribution of path lengths, obtained with S as a parameter, is shown in Fig. 12.6. The total path length, summed over all paths in the restoration of all spancuts, with S as independent variable, is shown in Fig. 12.7. The latter plot is a way of capturing all instances of excess links in any path for any spancut whether TIB-induced or due to path spreading. The six data points in Fig.12.7 required 132 restoration experiments in which a total of 858 paths were created. Fig. 12.8 shows the distributions of path time with varying S in the same experiments as the plots of Fig.s 12.6 and 12.7. Fig. 12.8 shows that across the range from S= 8 to 80 kb/s, average path time reduces from 164 msec to 123 msec. Thus a reduction of approximately 33% is potentially obtainable (with P=1.0) by increasing S from 8 to 64 kb/s. Fig. 12.8 also shows that beyond 64 kb/s there is almost no further speed improvement while path efficiency begins to drop noticeably as shown in Fig. 12.7.

Although restoration times suffer greatly at S= 1 kb/s, Fig. 12.6 shows that when Selfhealing is that greatly dominated by link delays, the ideal path length distribution (100% PLE) is almost perfectly reproduced. The point at S = 1 kb/s in Fig. 12.7 also shows that the lowest absolute TPL is also found there. However it is not as important in practice to obtain exactly 100% PLE as it is to

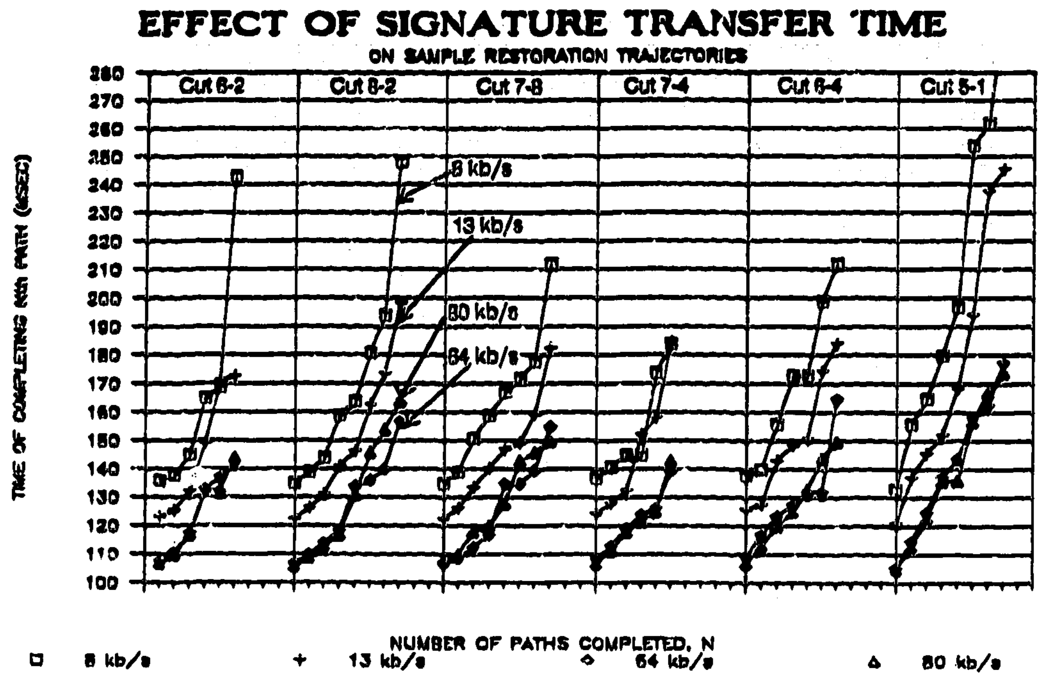


Fig. 12.5 Six sample restoration trajectories with signature transfer rate (S) as a parameter (S = 8, 13, 64 and 80 kb/s).

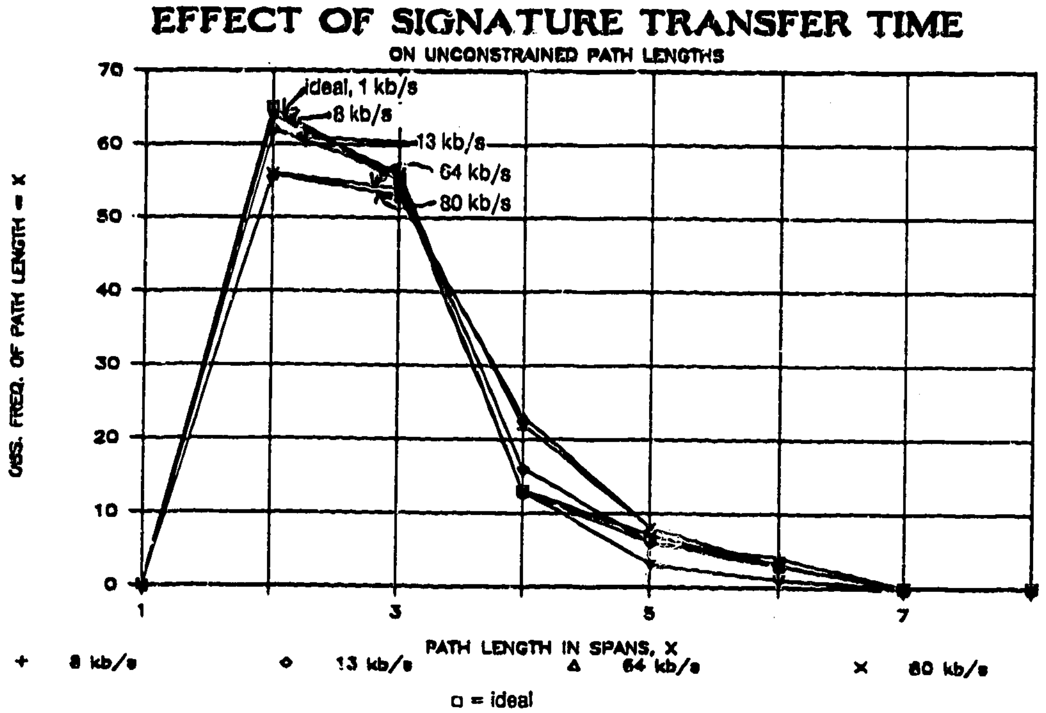


Fig. 12.6 Distribution of path lengths from all span cuts in Smallnet with varying S in Smallnet.

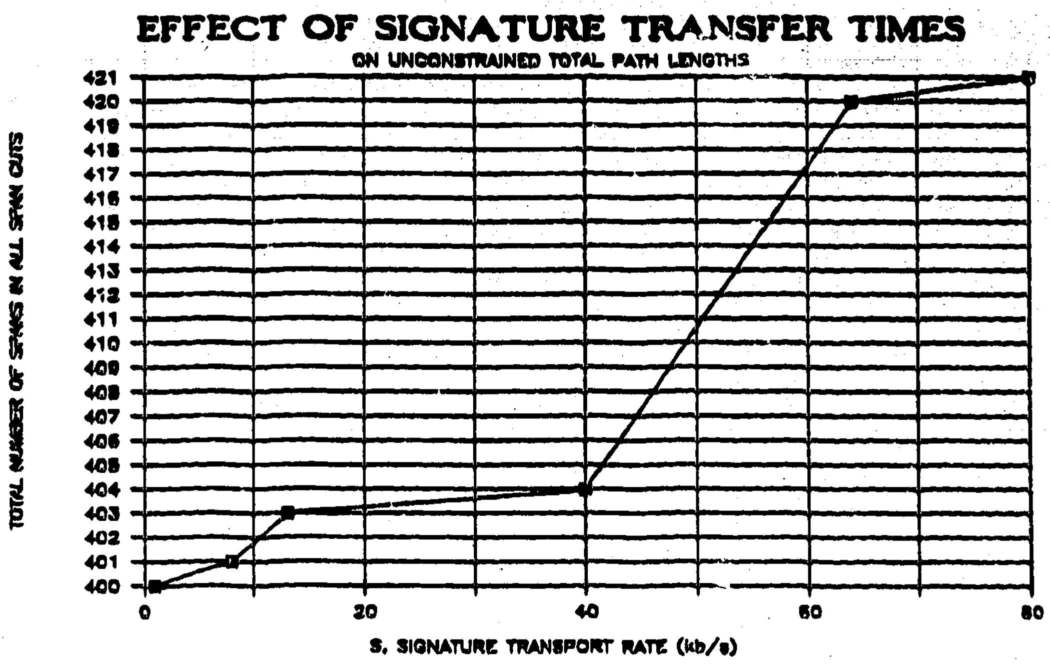


Fig. 12.7 Total path length of all span cuts in Smallnet as a function of S.

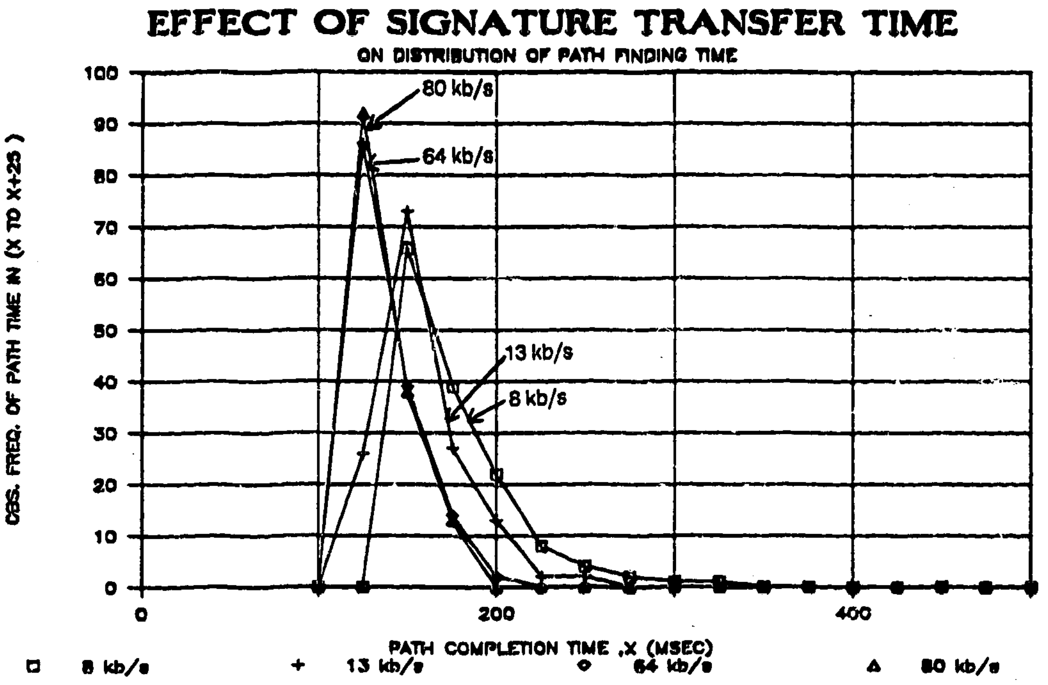


Fig. 12.8 Distribution of path completion times in Smallnet with S as a parameter.

obtain 100% PNE with high speed and reasonably high PLE. From Fig.s 12.7 we see that S= 8 kb/s and 13 kb/s both provide quite efficient path length characteristics but that a steep rise in excess length paths occurs between 40 and 64 kb/s. At 64 kb/s the overall PLE is however still 94.3% so that there is no major objection in practical restoration applications to using signature transport at rates of 64 kb/s if the required channel is available. To some extent however, the ability to predict the actual path routings for planning purposes without full emulation is compromised with S=64 kb/s. Results imply that with S below 40 kb/s (with P=1.0) the paths found by *Metadijkstra* will be good predictors of topology and the times predicted by emulation *without measuring processor times* will be good predictors of Selfhealing speed.

12.3 Dependencies on Relative STT and P values

We can further apply the preceding results by interpreting the observed behaviour in terms of *relative* processor speed to STT values. We can for instance ask what P value would produce equivalent performance, and an equivalent design centering, if SONET transport was used, instead of DS-3, in a metropolitan network application with typical d = 60 km which changes the STT. In SONET applications we assume that 64 kb/s is available for signature transfer. Combined with d = 60 km, the nominal STT becomes 64 bits/64 kb/s + 60 km * 5 microsec/km = 1.3 msec. Because this is 1/10th the STT in the previous long-haul DS-3 network, the S/P ratio (in general STT*P product) of Fig. 12.1 (P=1.0) is only preserved if we have P = 10 in the postulated metropolitan SONET network. The reduction in STT and a tenfold processor speed increase will of course reduce absolute restoration times but we can nonetheless say that these combined parameters are otherwise equivalent to P=1.0 in the longhaul network in the sense that the design lies at the same relative position between asymptotic limits in P and STT.

Therefore we infer that use of a SUN 3/60 processor running Selfhealing in a short-haul SONET network application would be equivalent to P = 0.1 in Fig. 12.1 in the sense that processor speed up efforts *would yield significant improvements*. One may go further and say based on our experience with path spreading in Bellcore (Ch.9) that without increasing P to about 10.0, or deliberately reducing S from 64 kb/s, the path properties of Selfhealing in any SONET metro application would be expected to suffer due to path spreading. This is because such conditions do not replicate the balance between P and STT which we have seen leads to quite efficient path length statistics which the combination of P = 1.0 and STT = 13 msec.

The two cases; (a) d=1,000 with S=8 kb/s (STT =13 msec) and (b) d=60 with S=64 kb/s (STT =1 msec) probably represent the extremes in both d and S for likely applications. We can therefore recommend that if a single crossconnect design is to perform Selfhealing in both long-haul and metro applications, then a processing speed of about 10 times a SUN 3/60 is recommended as a design requirement. We say this based on the knowledge that an *increase* in

P will have no harmful effects on the *longhaul* application, but is required to avoid path spreading in the future SONET metro application. Such a system design would be assured of executing Selfhealing in an STT-limited regime in both DS-3 longhaul applications while remaining well-biased to avoid path spreading effects in a SONET metro application. The P=10 target of could be approached in practice by a combination of code optimization and/or custom hardware to assist execution of the SH protocol, such as outlined in Ch.13.

It is not essential that near-term applications run in an STT-limited regime but seems natural as the appropriate eventual target to set. A side benefit of an STT-limited system design is that the network emulation required for planning studies could be based on simplified methods which need consider only relative link lengths and can omit the complexity of incorporating the effects of the processors own execution time in emulating Selfhealing. In fact in STT-limited operation, several functions of network planning could proceed using MetaDijkstra as a substitute for full emulation of Selfhealing, since path-spreading is assuredly eliminated in an STT-limited system design. Such a design objective also assures the avoidance of effects due to possibly varying processor speeds in DCS machines from different manufacturers.

12.4 Summary and Conclusions

Selfhealing's speed and topological performance both depend on the relative balance of signature transfer times (STT) and nodal processing delays, which are *O(1/P)*. We quantify this with the S/P ratio when all other factors are fixed. S/P indicates the relative speed of signature transfer with respect to normalized processing speed. When S/P is *low*, signature propagation times dominate, Selfhealing times tend to be inversely proportional to S and are determined primarily by the sequence of path finding dynamics. Low S/P ratio also makes for the best path efficiency results because path spreading and TIB are both counteracted. A *high* S/P ratio is the reverse situation in which link delays are insignificant compared to nodal processing times. In this case Selfhealing restoration *times* are faster than and vary primarily as 1/P, but PLE can suffer noticeably. At high S/P ratios restoration trajectories tend to be smoothed out and continuous whereas they exhibit abrupt flat sections of path accumulation when S/P is high, tending towards the size-invariant attractor trajectory when P = infinity. We find S= 8 kb/s, P= 1.0 to be a well-centred and adequate design combination for longhaul DS-3 networks in which increased P would be of little advantage. In a metropolitan network using S= 64 kb/s however, net speed improvements would be obtained for up to a tenfold increase in P and indeed such improvements in P would be desirable to control path spreading if S=64 kb/s in a short-haul network.

CHAPTER 13 CONCLUDING DISCUSSION

For practical reasons the preceding chapters have been limited to the central topic of Selfhealing for span restoration. A number of other results have been obtained however, and we will briefly review these so that in closing the reader will be aware of the overall scope of the results obtained in the area of study. We will then conclude the dissertation with a review of the main development, a summary of the research contributions which are thought to have been made and recommendations for further research.

13.1 Additional Results Obtained

13.1.1 Large-scale Telecom Canada Network Studies

As a result of the preceding work the author has obtained, and is now supervising, a major research contract funded by Telecom Canada to study the application and performance of Selfhealing in Telecom Canada's present and future network. It is not possible or desirable to report this work extensively here except to note that in the course of the Telecom study we have obtained an important further verification of Selfhealing on very large real network models provided by Telecom Canada. In the Telecom Canada study we have again consistently seen 100% PNE and PLE significantly above 95%. One sample result that is pleasing to show is in Fig. 13.1. This is the largest restoration space diagram we have yet obtained. It is based on the Telecom Canada network plan for the year 2005 and includes 93 nodes, 157 spans, 2532 working links and 4696 spare links. P=1.0, S=8 kb/s and RL=9 were the parameters used and span distances are actual geographical distances between nodes, spanning 7,000 km end-to-end. This very welcome result predicts that even without further improvements in protocol implementation, Selfhealing would essentially always beat the call dropping threshold in the Canadian network until at least the year 2005.

13.1.2 Profiling Optimization Studies

A significant activity of a developmental nature was not emphasized in the preceding chapters. This was the use of real time software *profiling* techniques for understanding of where the protocol was most computationally intensive. Using the insights obtained from profiling, several optimizations in our implementation were postulated, implemented and verified by profiling. The majority of real time execution profiling was performed on the IBM PC-based version of the emulator by way of a special memory-resident program which intercepted a real time clock interrupt. This program would keep a table of memory address ranges corresponding to the compiled and linked memory locations of each source-code level procedure in the protocol. Then each time the real time clock interrupt occurred, the profiling routine would sample the program counter and deduce the procedure that was running. In this way, a good picture could be built up over entire restoration events of how much relative time the protocol spent in each of its particular action blocks.

Three successive versions of the protocol were developed using this method of analysis. The most significant changes/improvements to the protocol implementation that were introduced with each version were as follows. The starting point was a version 1 protocol which functioned as intended but had none of the following improvements. At each stage of introducing incremental refinements side-effects and bugs had to be understood, removed, profiling repeated to see the effects and then a further stage of refinements planned.

The significant features of each version were: Version 0 used exhaustive Sender flooding (ie. it would flood all spares in every span, even if the number of signatures into a span exceeds *lostccts*), and deduced port to span associations and all precursor relationships from first principles on each entry. Version 1 introduced the use of *short circuit* IF evaluations of boolean condition test constructs. Version 2 introduced conservative Sender flooding (the flooding rule described here in Ch.6) and added the Assoc-port register to act as a scratchpad pointer to break out of the need to search the node every time a precursor relationship had to be found. Version 3 was a major rewrite of the entire protocol to introduce one important improvement, while keeping the above changes in effect. This was the introduction of transient linked lists (TLL) to accelerate all searches and scans performed by the protocol. These TLLs are built upon first entry of the SH task and discarded at the end of a Selfhealing event. They store, in the form of doubly-linked-lists, the identities of three subsets of ports associated with end logical span found at the node. The subsets (defined in Ch.6) are intended to reduce greatly the length of searches the protocol performs to find certain ports satisfying various conditions in the protocol. This last change made the greatest improvement but required the most sweeping changes because in addition to creating the lists initially, every search and action block of the protocol had to be rewritten in the context of operations on these lists including continual maintenance of the lists as processing proceeds.

Fig. 13.2 has been abstracted from these profiling optimization efforts to summarize the advances made in this activity. The version 0 protocol was not profiled, so the version 1 protocol can be treated as the starting point in Fig. 13.2. The overall average speedup factors obtained (using version 1 as 1.0) were x1.2 for version 2 and x4.75 for version 3. It can be seen in Fig. 13.2 that versions 1 and 2 are completely dominated by the time spent in BetterSig search and Repeatable-Sig processing. The BetterSig test was expensive because the Tandem node had to scan itself fully (all its spares) for each new RS at its site. Repeatable-Sig was also intensive because also involved scanning the whole node for outgoing free ports. By comparison, version 3 has a much more uniform distribution of execution time as a result of an $O(n^2)$ improvement in searching operations due to the use of span-oriented structured sublists of ports at the node.

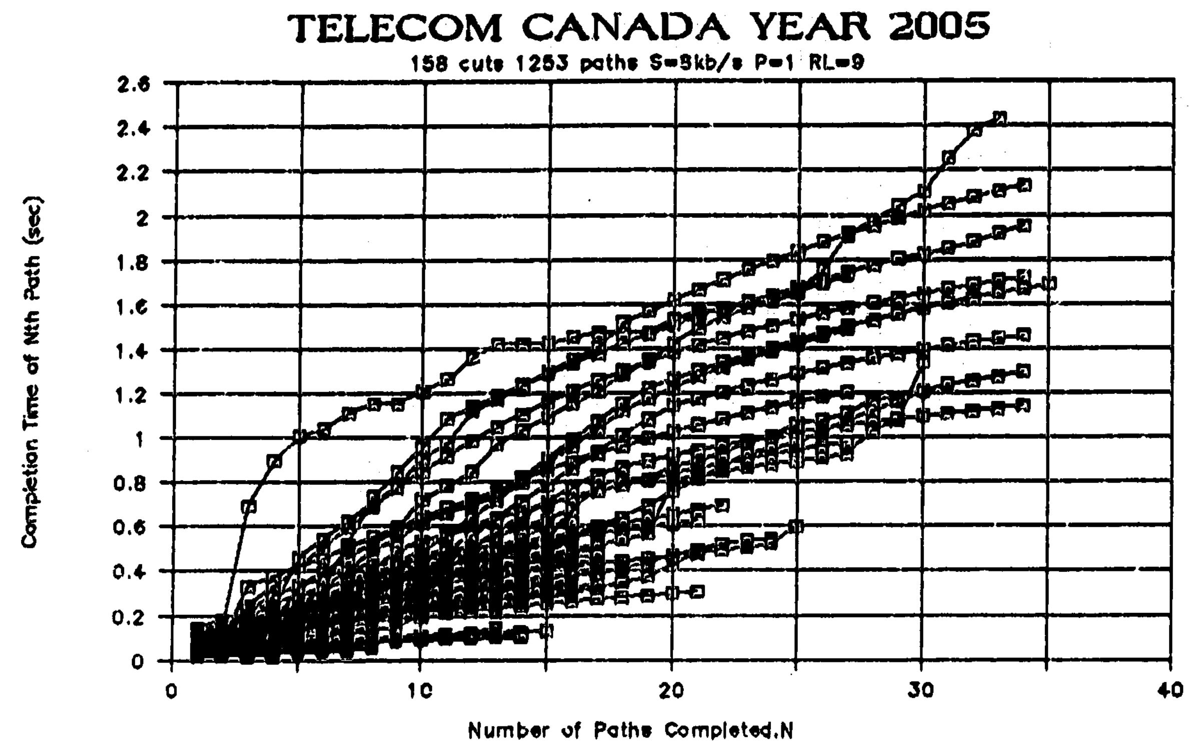


Fig. 13.1 Sample restoration space plot for all span cuts in large Telcom Canada network model from year 2005.

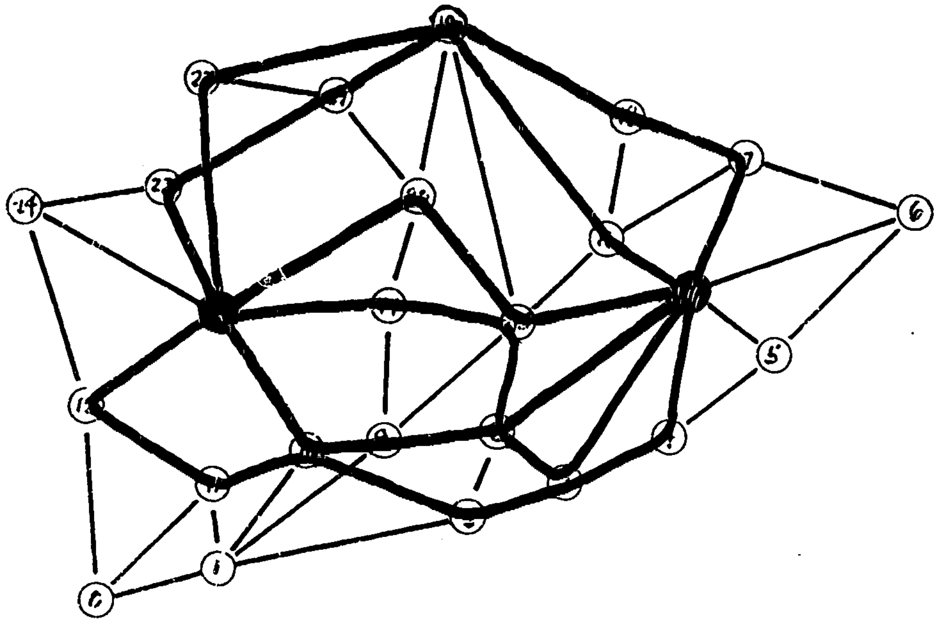


Fig. 13.4 An illustrative sample of *Bandwidth Scavenging*: the result for Scavenging between nodes 13 and 7 in Metro net (RL = 7).

13.1.3 Capacity Scavenging

In studies of span restoration we are by definition interested in diverse path creation only between pairs of nodes in the network that are directly connected by a span of the network. A related use of Selfhealing technology does not have this limited context however. *Capacity Scavenging* is the use of Selfhealing technology to find automatically all possible paths between *any two* nodes of the transport network. The main applications for this derivative of Selfhealing are thought to be in advanced trunk and service provisioning. Several distinctions exist between Selfhealing and Scavenging but the key relationship between them is that Scavenging is like Selfhealing triggered by an *artificial* pseudo-span failure between two nodes of a network that are not directly connected. The differences are that whereas Selfhealing is triggered by an alarm, seeks a number of paths equal to the number of failed circuits, and automatically substitutes traffic, Scavenging would be triggered only by an explicit request to the two nominated nodes and would only show the new paths possible, without automatically substituting any traffic. Scavenging is primarily seen as an advanced interactive provisioning tool which uses the real network as the database and as service layer for an advanced Self-traffic engineering network (to follow).

Two items are selected as exhibits to illustrate the Scavenging variant of Selfhealing which has been investigated. With Selfhealing there are $O(C^* | [spans] |)$ spancut experiments that can be performed in a network. With Scavenging however, there are $O(|N|^2)$ individual node-to-node Scavenging experiments that are possible. Fig. 13.3 shows a histogram of the all-possible-relations Scavenging experiments performed in Metro net with RL=9, using the PC emulator and v.3 protocol. There are (25*24)/2 = 300 unique node-pairs for Scavenging the the Metro network mode. Fig. 13.3(a) shows the observed statistical frequency of the time required for maximum Scavenging between nodes based on all 300 possible Scavenging events in that network. Fig. 13.3(b) shows the corresponding distribution of the number of provisionable paths found between each pair of nodes. Fig. 13.4 is an illustrative sample of the results from one individual Scavenging experiment. PNE=100% was observed in the Metro network although the path length efficiency of Scavenging has not been studied in detail. PLE should be similar to that of Selfhealing. From first principles there appears no reason why the PNE should not be 100% at all times in Scavenging (with RL >= TRL) given that Selfhealing exhibits this property. A note relevant to the provisioning context is that if Scavenging for only 1 path is performed at a time (as would seem most likely for a fast *customer request* driven provisioning scheme), then strictly assured shortest path properties are expected.

13.1.4 Protection Switch Replacement

A fairly simple point which is theoretically implicit within the results reported for Selfhealing is that the method should transparently function just like a modern automatic protection switching

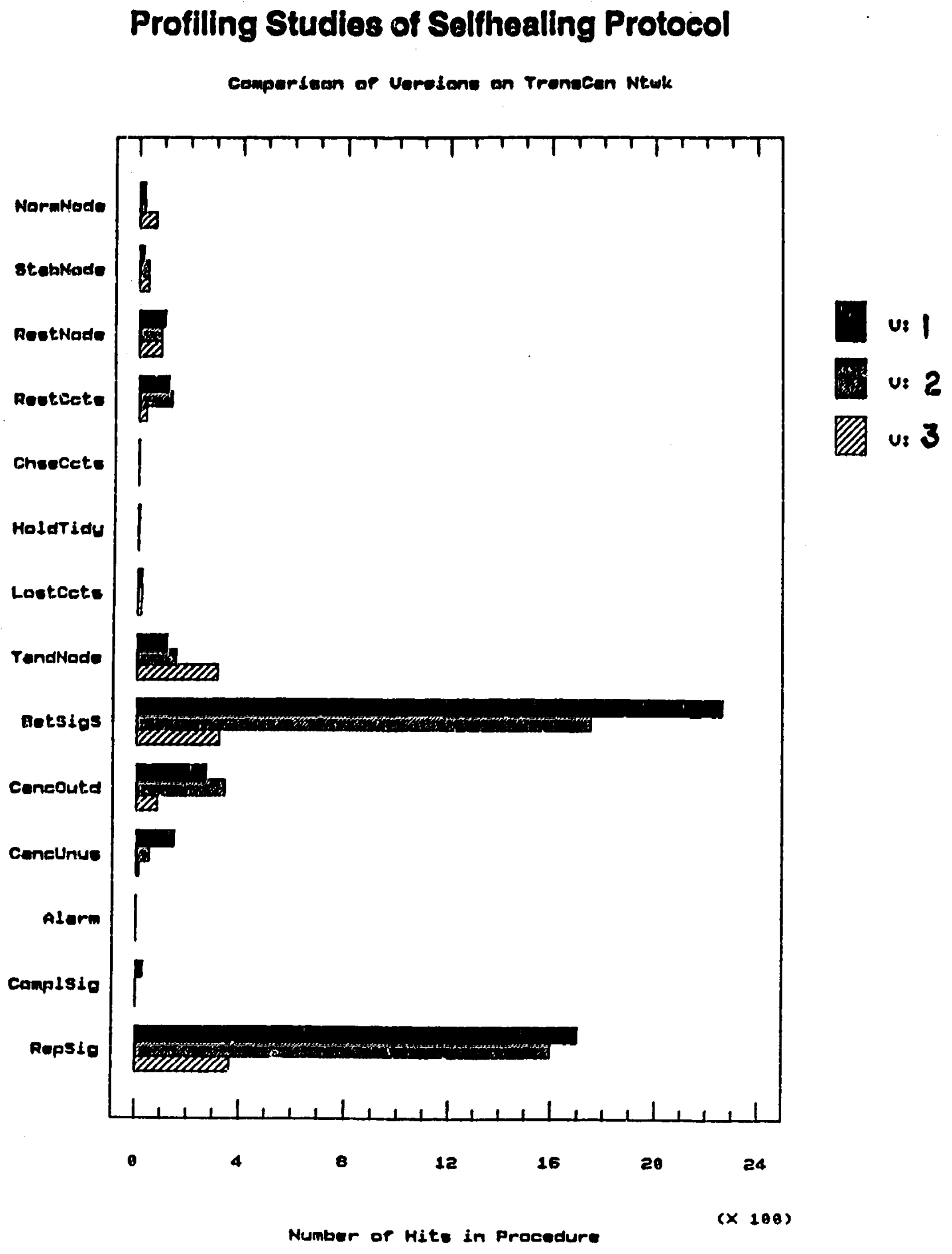
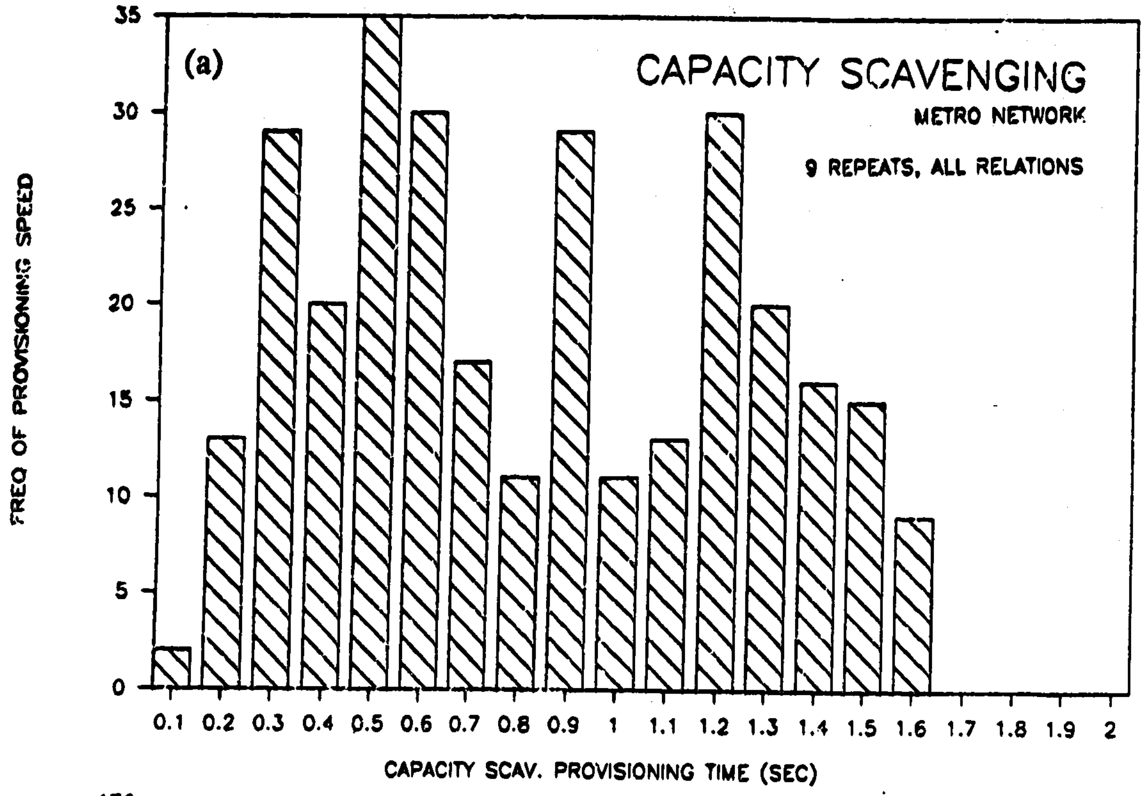


Fig. 13.2 Sample result from real time profiling and optimization studies showing relative distributions of execution time in three versions of the SH protocol.

system when required to do so. This section is included to confirm that we did explicitly verify this behaviour. For operation in APS mode, no change is made to the protocol in any way, only to the fault conditions. Instead of completely slicing the *alarm-span*, an APS mode event is emulated by placing an alarm state on only one direction of one or more working links of the specified span but not placing alarms on the spare links of the span. These conditions simulate a single fiber, laser, or regenerator failure. In all cases an almost trivial Selfhealing event occurs in which the Sender flooding automatically includes any surviving spares on the same span as the failure. These signatures directly reach the Chooser over one span which triggers a reverse linking signature, Sender traffic substitution and suspension of remaining signatures. The whole event is over in only two STTs plus a few msecs of processor delay. APS mode events are intriguing to watch on the emulator screen because the outgoing wave of signatures is comprised of *remnant indices* almost immediately after initiation. The net effect looks just like the ripples from a stone thrown into a pond.

13.1.5 Silicon Compilation Study

As a practical matter we were interested if, for the *ultimate* in speed of Selfhealing, it would be feasible to design a custom VLSI processor dedicated to hardware execution of the SH protocol. Because of its FSM structuring SH is in fact a good candidate for advanced techniques which map protocols or other functional specifications directly into hardware structures optimized to execute the protocol. One such *Silicon Compiler* is under development by Dr. E. Girczyc (U.of A. and Audesyn Ltd.). Using Audesyn's ELF silicon compiler, a study was performed to estimate the gate count of a VLSI implementation to execute the v.2 SH protocol in hardware. The results reported by Dr. Girczyc were that the protocol could be implemented in about 9400 gates with a 1200 word microcoded horizontal control ROM of 205 bit wide words. This result for the SH-task IC is essentially independent of DCS node size because all RS, TS, and PS registers were assumed to remain in the DCS port cards and the v.2 protocol does not need any dynamically sized data structures such as the TLL's in v.3. (v.2 is in this respect the most suitable for direct hardware implementation.) The resulting VLSI design would require 1 pin for an interrupt line, 10 pins for address bus and 16 pins for a bidirectional I/O bus for interactions with port cards and the host O/S. The pinout and gate count estimates found in this study imply that the protocol is ultimately amenable to dedicated hardware implementation in CMOS which would run with a 50 Mhz cycle rate. It is not known exactly what effective P value this would translate into but it clearly is an indication that P > > 1.0 is probably attainable if really required.



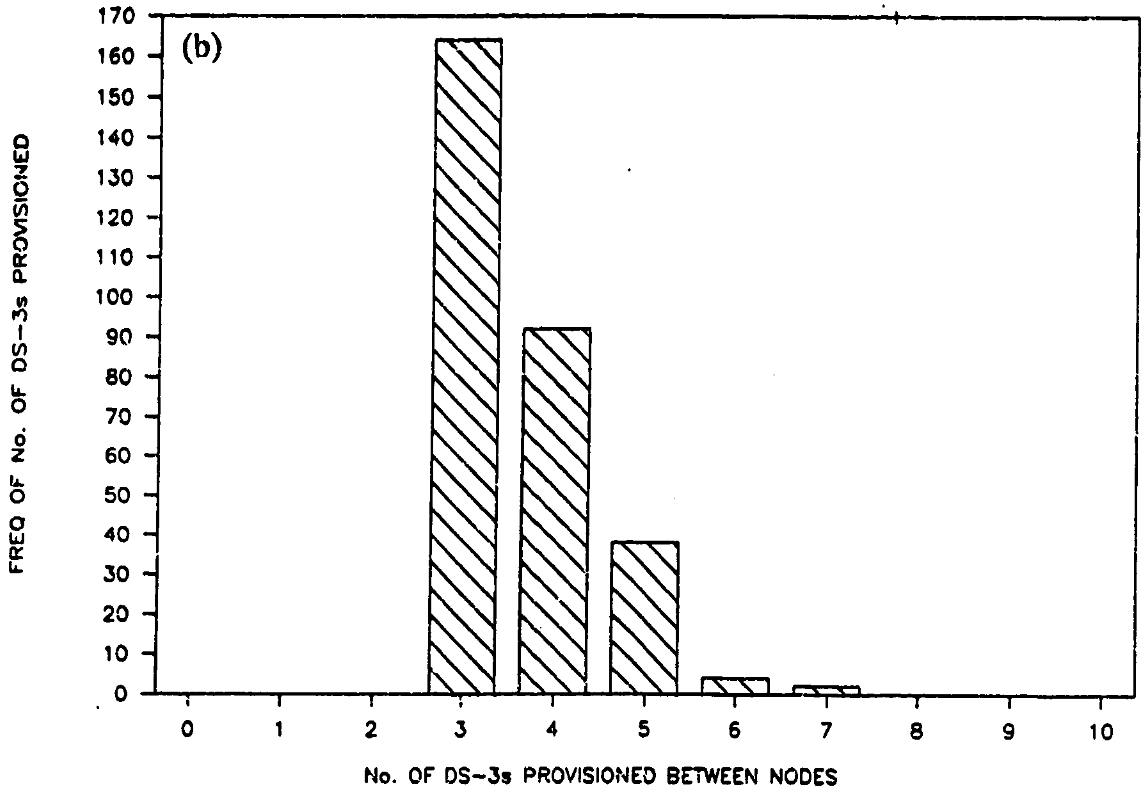


Fig. 13.3 Sample all-pairs *Bandwidth Scavenging* results from Metro net:
(a) statistical frequency of time for maximum Scavenging,
(b) statistical frequency of number of paths found by Scavenging.

13.2 Summary and Results

13.2.1 Review of Thesis

The central thesis of this work was that it should be possible to derive an advanced form of restoration system based on the new electronic DCS technology. In the envisaged system complex restoration plans would be computed and put into effect in real time in an entirely autonomous distributed manner by the massively parallel collective interaction of all the DCS-based processors of the network. In such a system the real network, at the moment of the fault, would be the only database required by the method.

In Ch.1 we saw that a new need for advanced restoration has arisen as an adjunct to the widespread deployment of vulnerable high-capacity fiber optic transport facilities. To date however, the idea of *real time* restoration (under two seconds) has not been seen as an achievable goal. Prior to this work all investigators were exclusively considering *centralized* control of DCS machines for the restoration problem. While centralized control will reduce restoration times from hours to tens of minutes, several drawbacks remain: primarily, dependence upon telemetry and central database integrity. In addition such schemes will continue to drop calls and to require continued use of separate span protection switching systems.

We have shown that we can skip the mid-range performance of centralized restoration systems and make the jump directly from manual reconfiguration to truly distributed, autonomous, real time restoration via DCS machines. Ch.1 showed that not only are there strong economic motivations for doing so but in addition a restoration scheme that beats the call-dropping threshold permits elimination of separate span protection switching equipment and has beneficial effects on the traffic-related dynamic processes associated with a major trunk group interruption.

Ch.2 proposed and defined a reference architecture for the DCS machine for use in developing and specifying the Selfhealing method and protocol. Ch.2 also looked at real transport networks for characteristics that were later reflected in our specific study network models. Significant features of these modern transport networks was the almost exclusive use of DS-3 based FOTS carrier systems in N. America, high average levels of redundancy and span topologies that are inherently closed restorable graphs. In addition, the rate of span failures is rising with the use of fiber, but multiple precisely simultaneous independent faults remain extremely improbable and methods are taken to avoid common routing of facilities on logically diverse spans. Therefore although Selfhealing is relatively easily extendible to multiple faults, and can deal with successive *series* of faults as is, the above means that the most meaningful and useful results are to be obtained at present by a focus on the single fault condition as in this study.

Ch. 3 took up the issue of formally presenting restoration as a new class of communications routing problem and contrasting it with the well-known packet data and call routing problems.

When formulated this way it was seen that restoration is the problem of finding k fully link-disjoint paths in a multigraph subject to hard collective constraints on the capacity of every span, requiring correct end-to-end mapping and with the objective of shortest possible paths. The most important difference between restoration and the prior routing problems is the aspect of full link-disjointness amongst k simultaneously found paths as opposed to finding one shortest path with no prohibition against reuse of the routing links. This requires specification of a 2*k times as many variables in a higher dimensional space with additional constraints that do not apply to the prior problems. Ch.3 then dealt with the literature of classical routing problems and the literature of the graph algorithmic community. The main finding was that no *distributed* algorithm for k-shortest paths appears to have been previously advanced. In addition in the literature of centralized graph algorithms no algorithm had been published on the very *specific* problem of k-shortest link disjoint paths in a multigraph.

Ch. 4 was devoted to development of quantitative measures of speed and topological efficiency for an advanced restoration scheme. No previously established set of such metrics existed and so it was necessary to develop and propose suitable measures. The measures defined in Ch. 4 are, in their proposed order of importance, as follows: (1) path number efficiency (PNE), (2) restoration time measures (t_H, t_{95}, t_{pav}, t_{p1}), (3) shortest path probability (SPP) and (4) path length efficiency (PLE). Ch.4 explains the use of these metrics in either *operational* or *intrinsic* assessment contexts. Each of the topological performance measures is computed with respect to an ideal restoration pattern and Ch.4 also dealt with how to obtain such reference solutions. The issue of k-shortest paths versus k paths of minimum total length as the target for ideal restoration was dealt with arguing that the former is the appropriate ideal engineering target for a fast restoration system largely because the latter formulation is NP-complete. To obtain the ideal restoration plans a centralized algorithm for k-shortest link disjoint paths, MetaDijkstra, was introduced and explained and this algorithm was used as the theoretical reference for subsequent measures of topological efficiency in this study.

Ch. 5 introduced the *Signatures* and the concept of indirect massively parallel interaction amongst DCS machines via quasi-static signatures embedded on every transport entity of the network. It was explained how this paradigm for interaction is significantly different from the ubiquitous interprocessor messaging notions which dominate today's distributed algorithm concepts. It was shown how the paradigm of indirect massively parallel collective interaction through isolated processing amidst a sea of signatures is ideally suited to the restoration problem and takes advantage of some properties unique to the problem of distributed control of the *transport network*. Amongst the advantages of this approach is the powerful result that the Selfhealing protocol effectively executes in an active memory space that is continually updated from outside the node in a manner that implicitly reflects the properties of the host network, even though no direct knowledge of that network is stored in the node. Because the Selfhealing protocol is cast as a an event-driven FSM for the processing of signature events, this chapter is also devoted to introducing the attributes of a Selfhealing signature and defining the logical structure and properties of the set of DCS port registers which comprise the active memory space within which Selfhealing executes and through which it indirectly interacts with the world outside its node. Ch.5 also reviews three related original works on the problem of signature transport in the DS-3 format and its variants to provide evidence of the practical viability of the idea of signature-driven network interactions.

Ch.6 is a companion to Ch.5 in that it gives the second half of the substance of the Selfhealing method which has been developed. The focus of Ch.6 was on conveying operation of the signature-processing protocol which effects Selfhealing. The claim, and the most important contribution of this work, substantiated by results in later chapters is that when this protocol is executed concurrently and asynchronously at every node of a network, responding solely to signature defined events, with no direct interaction with other nodes, and with no stored knowledge of network topology, each node will derive *isolated crosspoint operate decisions which collectively synthesize efficient coherent restoration plans* when viewed at the network level. The protocol that achieves this result in described in Ch.6 both in terms of the network-level processes which it effects (simultaneous competing mesh expansions which are mediated by rules in the Tandem node and collapse as a result of reverse linking) and in terms of specific signature-processing rules within the isolated context of a single node. *Signature diagrams* are introduced as a tool to describe and analyze this particular type of protocol. Appendix B accompanies Ch.6 as an integrated formal specification of the entire node-based signature processing FSM.

Ch. 7 is devoted to explaining the research methods employed both to discover and optimize the protocol which we hypothesized to exist and then to systematically characterize its speed and topological performance in the most realistic manner possible. The basic method employed was fully concurrent execution of a true implementation of the protocol module within a specialized form of multi-tasking O/S environment which emulates the parallel virtual-time interaction between nodes according to the defined links, spans, distances and nodes of the host study network. This chapter was necessary to stress the important aspects of methodology employed, including self-execution timing, virtual-time concurrent programming methods, and an interrupt-driven execution paradigm in which the protocol implementation cannot tell it is not inside a real DCS machine of the defined reference architecture. These special aspects of methodology support our assertion that the method employed essentially constitutes as a series of *experiments on a prototype implementation* as opposed to the often far less realistic connotation of results obtained by *simulation of the intended process*.

Ch.'s 8 through 12 are all devoted to reporting results experimentally obtained with the protocol implementation with varying parameters and varying network models and to interpretation and understanding of those results. The most important single result in the four networks treated under a range of conditions in Ch.'8 through 12 is that 100% path number efficiency was always observed. Ch.8 was devoted to an exhaustive consideration of every spancut in Smallnet obtained under assumed standard parameter values. By analysis of cases where overlength paths resulted (PLE < 100%), the *temporary index blocking* phenomenon of Selfhealing dynamics was discovered. On the hypothesis that TIB effects were responsible for the overlength path instances in Smallnet, a prediction was made that reducing the repeat limit would restore PLE=100% and this was verified. By introducing the *restoration trajectory* plot to display and interpret the dynamics of a Selfhealing event several insights into the properties of Selfhealing were obtained, including an ability to make a characteristic comparisons to other classes of restoration schemes. Results from a specific closely instrumented Selfhealing trace experiment were presented to help illustrate and consolidate the concepts of simultaneous parallel index mesh expansion and mesh collapse and the dynamic transient nature of a Selfhealing event.

Ch.9 went on to explore a larger portfolio of networks in which to verify and characterize Selfhealing. These were the Bellcore, US and Metro study networks. In the Bellcore network we discovered the *path spreading* effect which can occur when signature transfer times are relatively small compared to nodal processing times. Path spreading is related to TIB in its effect but is attributable to processor induced differential mesh expansion rather than link index blocking induced distortions in mesh expansion. In the Bellcore net we explored the effects of both RL and S/P ratio and by first reducing the RL to the network TRL value and then re-biasing the S/P ratio we found that 100% PLE was possible (as well as 100% PNE). All restoration events in all study networks were complete in under 1 second.

Ch.10 reported a substudy dedicated to the parametric effects of repeat limit (RL). In this we discovered the important role of RL in determining lifetime of *remnant index meshes*, beyond the obvious role of RL as a geographic limiter. The concept and importance of the threshold repeat limit (TRL) was introduced and observed performance above and below the TRL was explained. A recommended policy for determining RL was given for real network operations based on first determination the maximum TRL of any span in a centralized approximate network model. Results show that operation suitably above the TRL of a network does not steeply degrade the speed of operation, permitting RL to be set realistically above the TRL so as to ensure 100% PNE in the face of actual uncertainty of network topology over time. We found that the time of restoration is quadratically dependent on RL but with a relatively small coefficient of dependency. RL-induced topology shifting was observed and some interesting effects explained with the aid of trajectory plots.

Ch.11 pursued the question of the effective time complexity of Selfhealing. From the viewpoint of the protocol inspected simply as a collection of algorithmic action blocks and from the stance of worst case dynamical sequence of events in path synthesis, the method is characterized as *$O(n^2)$* where n is a uniform variation in the overall size of every span in the network. A distinction was made between two notions of time dependency on size: uniform growth in the depth of every span and isolated growth in the size of one span within a network. The combined manner in which the time for restoration depends on *both* the intrinsic dynamic sequence predetermined by network topology and the sheer uniform size of the network was explained with the principle of a size-invariant *attractor trajectory* for each spancut and the separate effect of finite processor speeds. With these two factors it is possible to explain how individual RT's may rise as *$O(n^2)$* where n is the size of one span, while simultaneously rising as only *$O(n^{1.3})$* or less when the entire network is uniformly scaled up. This is because the intrinsic shape of an RT is determined by the intrinsic sequence of F,T,R dynamic events required for route-establishment. This basic shape may rise worst-case as *$O(n^2)$* but in principle any number of paths can be accumulated without time penalty at each plateau within this basic underlying *attractor trajectory*. The theory of an intrinsic size-independent *attractor trajectory* for every spancut of a network was verified by repeating certain experiments with $P=100$. The prediction was confirmed that in the presence adequate processing speed, Selfhealing has *essentially no penalty* for uniform growth in network size. Related analysis led to the insight, however, that when one span grows in isolation with respect to its network, that *$O(n^2)$* worst-case growth can arise if the network topology is so as to force a *pure cascade* of increasing-length route establishments.

Ch.12 was devoted to exploring the form in which Selfhealing speed and topological efficiency is dependent on signature transfer times (STT) and processor speeds in an absolute sense as well as in a relative sense. It was found that reducing STT (with $P=1.0$) was only beneficial up to about $S=40$ kb/s. Beyond that, the restoration times could actually begin to increase with further increases in S due to path-spreading longer route establishment effects. Regarding P, it was found that no significant gain in topological efficiency or speed would be gained by increasing P in the DS-3 longhaul application but that in a SONET-based metropolitan application it would be advisable to increase P up to about $P=10.0$ to re-center the relative balance between STT and processor times and counteract path spreading effects. It was concluded that a Selfhealing DCS designed with effective $P=10.0$ would result in STT-limited operation in either extreme of longhaul DS-3 or shorthaul SONET applications. Such STT-limited operation is recommended as a reasonable and useful design target.

13.2.2 Summary of Main Research Results

It is thought that the following are the major original contributions established by this work:

1. the method of signatures as a new paradigm for massively parallel indirect distributed system interaction,

2. the conception, development, and demonstration of the Selfhealing protocol through which efficient complex restoration plans are derived in real time by the isolated database-free asynchronous action of a self-selecting subset of DCS nodes in a network using the paradigm of (1.),

3. the development and analysis of three technologies for signature transport (and other general purpose auxiliary signalling applications) in the DS-3 signal format and its variants,

4. characterization of the system-level behavicur and properties of the Selfhealing protocol including identification, and explanation of the following phenomenon and concepts: temporary index blocking, path spreading, remnant index meshes, index congestion, threshold repeat limit, size-invariant attractor trajectory, dependence on STT and P, topology shifting effects.

In addition it is thought that the following aspects of the work, although of lesser import and technical difficulty, are nonetheless original to the present effort:

1. first demonstration of a traffic-dynamic motivation for fast span restoration in addition to the usual economic and availability motivations,

2. first work setting the objective of truly real time (< CDT) restoration as the appropriate next goal for restoration technology,

3. origination and definition of a set of metrics suitable for the systematic comparative quantitative study of advanced restoration technologies and for structured management or specification of speed - topology tradeoffs (PNE, t_R, PLE, SPP),

4. origination of the *restoration trajectory plot* and *restoration space plot* as widely applicable tools for analysis and characterization of restoration methods and their interaction with a given host network,

5. first formal presentation and development of span restoration as a problem in the field of routing algorithms routing and clear illustration of its significant differences from previous well-known routing problems,

6. origination of the MetaDijkstra algorithm for centralized calculation of k-shortest link disjoint paths in a multigraph, for use both as a reference model in distributed restoration systems research and for direct use in certain network planning activities such as intended by Hasegawa et al. in [HaKa87],

7. explanation of the use of signatures outside of the Selfhealing context as a standalone technology for network-wide hardware path audit trace as a means to increase and/or monitor database integrity in centralized control schemes.

13.2.3 Publications Resulting from these Investigations

In the course of this work, papers resulted which were directly related to Selfhealing, to address the issue of signature transport in the DS-3 format, and in one case simply as a useful spin-offs from the work. We will here list those papers and explain briefly how they relate to the body of work as a whole. Other publications by the author during the course of these studies but *not* related to Selfhealing are listed in the vita. The central publication to date on Selfhealing is:

1. W.D.Grover, "The Selfhealing network: A fast distributed restoration technique for networks using digital cross-connect machines",*Proc. IEEE Global Conf. Commun*, Tokyo, 1987, vol.2, pp.1090-1095.

This paper will be revised in 1989 to include key results from this thesis and to include complete disclosure of the protocol for submission to the *IEEE Trans. Comm.* in late 1989. This step was not taken during the course of the work because of the time required to assure ATRC patent protection prior to publication.

The requirement for signature transport in the DS-3 format, while not central to the theoretical soundness of Selfhealing, was a point of considerable practical concern to the author and resulted in the following papers. Collectively these works establish several technical alternatives for DS-3 signature transport, removing that possible obstacle to practical deployment of Selfhealing, and providing generally useful mechanisms for other applications requiring a new form of DS-3 overhead channel. These papers are :

2. W.D. Grover, "A Frame-Bit Modulation Technique for Addition of Transparent Signalling Capacity to the DS-3 Signal", *IEEE Pacific Rim Conf.on Commun., Computers and Sig. Proc.*, June 1987, Victoria, B.C., pp. 319-322.

3. Grover,W.D. and W.A. Krzymien, "A Proposal to Use the Justification Bits of a DS-3 Stream for Signalling" *accepted July 1989 for publication in IEEE Trans. Comm.*

4. W.D. Grover, "On The Design of an M13 Multiplex to Support the Proposed DS-3 C-bit Format", *Proceedings of Queen's Fourteenth Biennial Symposium on Communications*, Kingston, May 1988.

A revised and extended version of paper 4. entitled "On the Design of a DS-3 Multiplex with Signalling Channel Derived by C-bit Liberation" has been accepted for publication in *IEEE Trans. Comm.*

A fifth paper arose as a spin-off related to the idea of replacing the repeat field of a Selfhealing signature with an actual time-stamp of the real clock time at which a signature was originated (ie. clock-face absolute time information). If every DCS machine had microsecond-regime absolute time synchronization, such time stamps could be used to measure true time-of-flight rather than logical repeat distance in Selfhealing, as well as having several other uses in the control and coordination of DCS-based transport network. This paper was originally presented as

5. W.D. Grover and T.E. Moore, "Precision time transfer in transport networks using digital cross-connect systems", *Proc. IEEE Global Conf. Commun*, Hollywood,Florida, 1988, pp.1544-1548.

and has been accepted for publication in the *IEEE Trans. Comm.*

Three patents have also been filed by ATRC relating to the above and they are presently in the course of examination. The central patent on Selfhealing has been examined and approved, but not yet issued. A patent submission on DS-3 F-bit modulation has been examined and replies to the reviewers questions were returned in March 1989. A patent on DCS-based precision time-transfer has not yet been examined.

13.3 Topics for Further Research

13.3.1 Advanced Path Length Measurement Strategies and Repeat Limit Considerations

Several advanced strategies pertaining to the RL field of the basic Selfhealing method were identified in the course of the work. One is to replace the present integer-valued repeat field (or possibly add a new field to the SH signature format) with an actual time-stamp of sufficient global precision that subsequent DCS machines can measure the actual *time-of-flight* of a signature wavefront in arriving at its site. This could in principle be developed into a method for converting the path properties of Selfhealing from shortest *logical* path length properties to shortest *geographic* path length properties. This would require prior absolute time-synchronization of all DCS nodes of the network as in [GrMo88a].

Another area of investigation retains an integer-valued repeat field disciple but would let the IRV of signatures initiated by the Sender be some decreasing function of the amount of time after first performing signature flooding so that network behavior tends to find all shortest routings first and then is allowed to probe successfully longer reroutings only if needed.

The repeat field could also remain an integer but be used with other real-valued and/or nonlinear cost functions computed by each Tandem node to achieve path solutions with certain properties other than strictly shortest-path, such as perhaps a node-packing strategy or to selectively avoid or use preferred portions of the network etc.

13.3.2 Comparison to Globally-minimal Total Path Length

It would be of some interest to quantitatively investigate the frequency with which the set of paths derived according to the k-shortest link disjoint paths criterion actually differ from the set of k paths that produces globally minimum total path length. This would involve writing a permutation/search program to find the strictly minimum TPL path combinations in a network like Smallnet and comparing them to the TPL of corresponding path sets from MetaDijkstra. Although strictly perfect PLE is not a top priority in practice, it is a point of some interest for academic thoroughness to find out how different these two measures actually are in sample networks of the type we use. Experience and intuition from the present work and amongst the team now performing the Telecom Canada study suggests that k-shortest paths is in practice quite close to globally minimal total path lengths.

13.3.3 Further Research with MetaDijkstra Algorithm

Two ideas for the enhancement of MetaDijkstra might be pursued. These are the exploration of node-packing heuristics and the use of automatic random dithering methods to improve MetaDijkstra's path sets when *exactly equal* early path choices arise in networks using logical or integer path length measures. By *node packing* we mean finding k-shortest link disjoint paths subject to also using the fewest total number of distinct nodes in the solution. In practice there may be some real-world reasons to involve as few nodes as possible in any given event, as long as 100% PNE is preserved. Therefore amongst paths of otherwise equal lengths (or perhaps even if PLE suffers) it is of interest to consider finding restoration plans that use the fewest nodes. Two approaches could be pursued: (a) systematically and artificially reduce the weights of links coming into a node, in proportion to the number of paths already through that node. (b) split all actual nodes into two pseudo-nodes with a pseudo-span of infinite depth between them and reduce the weight of links on this internal span in response to the number of paths through the node.

13.3.4 Self-traffic Engineering Networks

This further research area is presently being pursued under the authors supervision. It is based on the principle that the Scavenging variant of Selfhealing offers an automatic form of provisioning new trunk groups or augmenting the size of existing trunk groups in the network whenever any two voice switches experience high blocking on the logical trunk group(s) between them. It is clear how given excess blocking between any two voice-level switches of the network Scavenging could automatically find a required number of additional paths but the research challenge lies in deriving policies or algorithms that over long periods of time will ensure approximate global optimality in the mapping of trunks to facility routings, even though every individual scavenging event was in itself an isolated expedient shortest path routing at the time it was made. Other research questions pertain to a near-optimal strategy for deallocation of facility resources when group utilizations drop, as well as automatic allocation when they rise. The ultimate goal is a network which continually and near-optimally adjusts the mapping and sizing of logical trunk groups of the network into the actual facility resources present thereby automating a planning cycle that takes about two years today. Automatic adaptation of trunk groups sizes to focused overloads, travelling busy hours and massively mis-engineered loadings such as occur on Mother's Day or due to radio call-in shows would be a highly useful addition to today's networks including those already using dynamic call routing within fixed trunking architectures.

13.3.5 Node Recovery via Interactive Scavenging

Usually it is the outside plant transmission facilities that experience failures since they are much more exposed to hazard. Far rarer but even more serious than a spancut is the actual loss of a node of the network. Although serious spancuts are occurring nearly every month, somewhat ironically from the viewpoint of the timing of this work, the loss of a *node* has recently received enormous attention because of the spectacular recent Hinsdale telephone C.O. fire [Zorp89]. Although nothing short of physical repair can restore the actual source/sink traffic to the destroyed node it does seem that Selfhealing technology could be adapted to relatively quickly reestablish a maximum amount of continuity to the transport network as a means of restoration of the facilities that physically transited the destroyed node.

The principal idea of this approach to restore the transport network between remaining nodes by a series of successive Scavenging events between all pairs of nodes that were adjacent to the failed node. This would achieve a 'knitting' of restoration routes bypassing the failed nodes as a result of a semi-automated series of Scavenging events interactively conducted under the guidance of a central control site. While not a fully real-time reaction, it would still be much faster than the time required to find new routings for transiting facilities in the Hinsdale disaster. The basic interactive sequence of events with the control centre is envisaged as:

```
Repeat
    1. nominate two nodes x,y who are neighbours of the failed nodes and previously had one or
       more TE's transiting the failed node.
    2. command a Scavenging event between them x,y for a number of paths equal to all or an
       allowed portion of the capacity lost between x,y.
    3. receive reports from cooperating DCSs self-determined to be involved by the x,y
       Scavenging action with the crosspoints they would operate if requested to realize the
       potential paths found.
    4. Command implementation of all or a selected subset of the paths volunteered by the
       network in the x,y Scavenge.
    5. pick next two surviving neighbours to be reconnected in whole or part
Until all required combinations Scavenged
```

Assuming the total redundancy remaining in facility routes in the vicinity may not permit 100% equivalent bypass connectivity, it would be probably be beneficial to repeatedly pass through the above loop rationing only a certain number or portion of TEs at each stage until no further Scavenges can find additional paths for any x,y pair.

— *END* —

FOOTNOTES

chapter 1...

1 Why does the network drop all connections when a facility fails for over 2 seconds? The practice traces back to the introduction of carrier transmission systems, when it was found that when a carrier system failed, the call state signalling bits of the individual trunks (eg, off-hook, ringing, answerback, etc.) could chatter spuriously or go through state sequences that swamped the switching machine central control, seizing resources of the switch which are provisioned on a traffic-engineered basis; i.e., signalling units, dial-tone senders, digit registers, power-feed units, call detail records, translation registers and so on. This could seriously threaten the switches' operational capability altogether. For these reasons, the carrier group alarm (CGA) and associated call-dropping criteria were adopted. In most administrations the CDT has been increased to (2.5 ± 0.5) seconds, by CCITT recommendation, from an earlier value of about 300 msec.

2 This sub-study was carried out in collaboration with M. MacGregor who adapted the LANSF discrete-event simulation system to conduct the intended experiments.

3 Richards [Rich88] has proposed one approach to restoration which does *not* involve DCS machines at all. He advocates the modified use of 1:1 protection switching subsystems in existing transmission terminal equipment by placing *100% redundant physically separate fiber cables on every route of the network.* This approach has extreme simplicity but is highly inefficient, inflexible and expensive. In a polygrid of single-cable routes, each having a small fraction of spare circuits and each warranted in its own right for service, every route can enjoy full restoration if needed via multiple reroutings over the spare portion of other intact routes. Far less than 100% redundancy can suffice and no new routes need be established.

4 Coordination amongst multiple administrations is particularly a problem in North America. Unlike British Telecom which operates a single network with one operating company, North American DS-3 networks were built in many stages by many separate companies, over a period about 15 years. Telecom Canada particularly faces the problem of multi-administration coordination with its 10 independent operating companies and no unified database of sufficient accuracy to permit automatic centralized control of DCS machines. The creation of a single centralized database describing the whole network, and maintaining it in real time is a very difficult co-ordination problem. Selfhealing is, by its nature, free from this requirement: its database is the network in which it is embedded.

5 The radio isotope analogy for signatures in their role as a DS-3 path audit trace was first put forward by Dr. W. Krzymien.

chapter 2...

1 The DCS switching core is generally not directly compatible in an electronic sense with DSX-3 inputs because the latter are bipolar B3ZS encoded and not frequency synchronous to the switching core. If the internal switching matrix uses synchronous time-switching techniques to achieve a large non-blocking core with physically reduced space requirements, then another function of the port interface is to perform pulse-stuffing justification of plesiochronous DS-3 inputs, adapting them to the synchronous time base of the DCS core.

2 This meaning of the term *circuit* is different from the popular understanding in telephony work. In telephony a circuit implies the end-to-end telephone connection established by the call routing and establishment process. *Circuit* here pertains only to the carrier transmission system *between switching sections*, not a telephone circuit. A *path* is the entity that goes end-to-end across the transport network in this context.

3 In reality other states exist in addition to *working* and *spare* : out-of-service for maintenance testing, protection lockout, extra traffic, unequipped FOTS tributary interface states etc. but these are all equivalent to not existing from the viewpoint of the restoration problem. *Spare circuits* exist either because they are deliberately set aside to create redundancy for restoration of working circuits, or due to the redundancy resulting from normal optimal provisioning interval considerations. The latter effect is particularly true in fiber networks, where the economic provisioning module is quite a large capacity increment; i.e., optimal planning typically dictates installation of a 135 Mb/s (3 DS-3's on one fiber) or even a 565 Mb/s (12 DS-3s on one fiber) system if only 1/4 of that total capacity is required at the time of installation. At any one time, a modern fiber network can therefore have up to 100% aggregate redundancy.

4 Note that this is different from the alternative definition of redundancy: (Σspare/ (Σspare + Σworking).

5 Telecom Canada is a collaborative organization in which ten of Canada's major telecommunications companies cooperate to provide a voice, data and image network spanning six times zones (7000 km coast to coast). While Telecom Canada allows the present report to use general statistical properties of their network, they consider the network maps from which these data were extracted to be sensitive information which they do not wish to be reproduced.

6 This thesis primarily treats the single failure case but extension to multiple simultaneous faults is fairly straightforward by application of multi-tasking software methods. (ie. each node can have several independent SH tasks.) Chapter's 5 and 6 elaborate to some extent on the extension to multiple faults and, although not included here, the Selfhealing network emulator was successfully modified in August 1989 at ATRC for dual simultaneous faults. This was necessary to conduct part of the 1989 ATRC/Telecom Canada research contract on Selfhealing networks.

chapter 3...

1 This observation holds for most real-world routing problems that can be thought of such as aircraft, shipping, mail, data packets, call attempts and routing through administrative or managerial flowcharts. By comparison, restoration is more like a problem of routing multiple continuous fluid flows through a 3 dimensional network of pipelines.

2 An extensive electronic search was designed to find relevant works in the literature of network graph and routing algorithms. The search included an on-line scan of INSPEC with the wide keywords: (shortest paths, k-shortest paths, distributed routing, multiple shortest paths, restoration, etc.) This resulted in examination of over 200 summaries and recovery in full of about 100 papers covering the relevant efforts in algorithms for graph problems.

3 Topkis's algorithm was studied at some length in this work, including an exploratory implementation to attempt to adapt Topkis's method for a k-shortest link disjoint paths reference algorithm. The basic idea was to try and see what would be required for Topkis paths to in fact remain link disjoint *throughout their routing* when *launched* in a link disjoint manner. This was ultimately not successful, but created the germs of the separate MetaDijkstra approach that eventually was successful.

4 **Selfhealing, which works in a *distributed* computational model, can also be considered to be a *parallel* algorithm, without any loss of generality. Synchronizing the processors has no detrimental effect and one can disallow communication between processors (which represent nodes) that do not have a corresponding span between them in the network graph. Similarly, because Selfhealing is capable of dealing with multigraphs it is a trivial reduction for Selfhealing to work on simple graph problems (although the reverse adaptation of algorithms for simple graphs is generally more difficult). This means that Selfhealing may also constitute a first algorithm for k-shortest link-disjoint paths on parallel (MIMD) computers. The parallel all-pairs shortest usual path algorithms that are reported in [QuDE84] all have need of n^2 or more processors. Selfhealing could solve the same problem with only n processors but would require n(n/2-1) simulated Selfhealing events for all-pair problems. This may be a particular combination of characteristics not yet reported in a parallel shortest paths algorithm.**

chapter 4...

1 [HoSa78] and [AHU75] both devote significant attention to explaining the concept of "greedy algorithms" and the desirability in practice of problem formulations which allow a greedy solution as opposed to a combinatorial solution.

2 This is the usual approach to demonstrating NP-completeness in practice: ie. to decompose or map the actual problem suspected of being so into a form where its solution is expressible in terms of one of the relatively few basic problems which have already been proven NP-complete from first principles. NP-completeness does not mean that correct solutions for the problem cannot be found. It means only that *any possible algorithm for the problem* inherently grows in computational time requirements faster than any polynomial-time function of problem size.

3 In fact, when the author has demonstrated Selfhealing to an audience using the graphical display of the network emulator, before explaining the problem in its technical depth, it is sometimes judged immediately upon seeing the emulator conduct Selfhealing that such routing is a simple problem. People watching the emulator find it hard to divorce themselves from the global network view seen on the computer screen and remember that no node in the Selfhealing network has any such knowledge of the network it is in and once the resulting path routings are visible they sometimes seem irresistibly obvious in hindsight.

4 The approach used in *MetaDijkstra* avoids some serious limitations explained by the authors of the algorithm outlined in [HaKa87] and only one other related algorithm was found in the literature. The latter is a more complicated algorithm by Suurballe [Suur74] for finding *node*-disjoint paths in $O(n^3)$ time in simple graphs. By comparison, MetaDijkstra finds *link*-disjoint paths in a multi-graph and has $O(n^2(k+1)) = O(n^2k)$ time complexity where n is the number of nodes in the network and $k = \min [W(u,v), |[P_t]_{uv}|]$). ((k+1) because it takes one extra iteration after finding k paths to realize there are no more)

chapter 5...

1 (on game design analogy) The analogy of distributed system design to the design of isolated-player games for their by-product provides a non-traditional approach to the formulation of distributed system problems and goes in hand with the indirect interaction paradigm of signatures. The game design analogy asks: 'Can a game be formulated which has a *system-level by-product* which *is the desired* large scale behaviour?' If the problem of designing the game to have the desired by-product can be solved, a protocol (the game rules) is more likely to be found which has *no need of global information* because the apparent goal of the nodes is not cast in terms which involve any other nodes. In Selfhealing for instance, the nodal rules for signature processing, taken by themselves give no hint or flavour of path finding. They appear to be arbitrary rules for the manipulation of signatures. The intended result happens as a by-product of the simultaneous distributed application of these rules.

Many real games do have such a *self-organizing side-effect* on the objects of the game. In 'Scrabble' for instance the by-product of the game is the creation of a crossword puzzle although this is not the direct goal of any player. Similarly, most card games have a self-organizing side-effect on the deck of cards. (This is why shuffling is required.) Other games can similarly be found or invented that involve a set of operations on some domain or medium and have a self-organizing (or entropy reducing) side-effect on the medium in which the game is played. The notion of a game that has a self-organizing effect on the medium within which it is played is much more the lineage of Selfhealing than is the usual line of distributed algorithm development.

2 (role of source-target fields) With pure-code and FSM methods it is relatively easy to manage one instance of the Selfhealing protocol at a node so that it is re-entrably utilized to concurrently process signature events pertaining to different problems. By separately identifying all signatures by the fault to which they pertain, (current-class) we keep the ability to deal simultaneously with multiple faults, and even to perform different roles in each fault.

3 (on the status of spares used by Selfhealing) When Selfhealing converts a spare port into a working port in the cause of restoration, higher-level non-real time software applications are assumed to deal afterwards with the distinction between a spare pressed into service and a normally engineered working circuit. This extends to the whole process of reversion after physical fault repair. That is a non-real time problem which we leave (for present purposes) to conventional centralized control means. It is feasible in principle however to implement automatic reversion after fault repair in Selfhealing.

chapter 7...

1 The operating system real time clock interrupt in the SUN 3/60 gives a resolution of only 20 msec. This was not accurate enough to study Selfhealing without resorting to repeated runs of every cut to obtain further precision through sample averaging. The total time for such averaging runs would however have been prohibitive on the large networks we wanted to study on the SUN-based network emulator.

chapter 8...

1 This approach could be of interest as a background process through which a Selfhealing network could continually determine its own optimal span-specific repeat limits. Whereas Selfhealing effectively conducts a simultaneously multi-path search, some other recent proposals only work by virtue of the simplifying principle of converting an n-path parallel problem into n time-consecutive single-path individual search problems [YaHa88].

from chapter 9...

1 In the present results the metro network is topologically as it appeared in [Grov87b] but the US network has been changed by the addition of two extra spans to more closely match the connectivity in the Eastern seaboard area implied in Kessler's 1986 map with respect to his 1984 map. In addition spare circuit quantities in the US network have been re-randomized to be less regular and modular than they were in the [Grov87b].

2 Random x value offsets are applied in plots of the type in Fig. 9.7. For example all of the points logically associated with an integer path length of 2 will actually be plotted at a random x value between 1.8 and 2.2. This is done to counteract the loss of information in the intended scatterplot when hundreds of points fall exactly on top of each other. Ordinate values are plotted exactly.

chapter 10...

1 The expectation of an approximately square-law rise in restoration time with RL is based on the principle that RL specifies, in its geographical sense, the *radius* of a region within which rebroadcasts can occur. Each index emitted into the network will attempt to consume a total volume of space-time resource (links x STT) which is approximately given by:

$$F(C,RL) = \sum_{j=1}^{RL} (C-1)^j$$

where C = the average number of spans connected to each node. With typical network connectivities between 3 and 4, a modest repeat count of 6 allows this finite geometric series to reach a values between 363 and 1364. Although the function F(C,RL) is far stronger than any polynomial, the rise of restoration times with RL still follows a square-law characteristic because F(C,RL) rapidly saturates the networks *area* within the spatial limit set by RL. With more or less uniform networks, the assured saturation of all spans by the strength of the above flooding effect, implies that total time may be expected to rise as the area function because as radius increases the number of signature rebroadcasting nodes that are within range to re-affect the central area of the fault grows as the *area* of the involved region.

2 These experiments were conducted with the PC stepped emulator using the index trace mode and an override to force node 0 to act as Sender for the (3-0) spancut and seek only one path. Although the real behavior of Selfhealing is asynchronous, the artificial stepped synchronism is not grossly unrealistic for networks with equal length spans and it serves well for inspection of effects such as this.

chapter 11...

1 Garey and Johnson give an excellent treatment and quantitative illustration of this principle in the first chapter of [GaJo79].

2 This $O(n^2)$ functional requirement in the protocol was in fact a major motivator behind the introduction of an implementation based on linked-list representations of the ports in each span organized by their signature state. This has the effect of greatly reducing the lengths of the searches required in each iteration of the unavoidably $O(n^2)$ function. The search lengths required are reduced from a search of the entire node to as little O(1) accesses to single data items.

3 In the span-specific scaling method a previous spancut on the base network (ie. Smallnetx1) was used to determine exactly which spans would be involved in the solution for a given spancut. Then the lesser of the two end-node spare capacities was taken as the number of working circuits for the span which would be required to force maximum path finding stress (topology-limited results). Only one spare circuit was placed on the span destined to be cut in the experiment and only one working circuit on the other spans. Then, instead of scaling the entire network, only the subject span to be cut and the neighbouring spans known to be involved would be scaled by the factor xF. This creates a scaled-up psuedo network that is in all respects accurate for the one intended spancut only. All these measures were aimed at effecting the largest possible scale-up subject to a hard limit on the total number of links that the emulator could handle. At the time of writing however, this limitation has been circumvented and all-spancut experiments in Telecom Canada network models of up to 6,000 links have been emulated directly.

Bibliography

[Abe88a] G. Abel, "When networks go down hard, recovery companies go to work",*Communications Week*, Feb 22, 1988.

[Abe88b] G. Abel, "AT&T Speeds drive to digital net",*Communications Week*, Dec. 12, 1988.

[AbRh82] J.M.Abram and I.B.Rhodes, "Some shortest path algorithms with decentralized information and communication requirements",*IEEE Trans. on Automatic Control*, vol. AC-27, no.3, June 1982, pp.570-582.

[AbRi86] E. Abdou and P.Richards, "Fiber network design for survivability", *Proc. FiberSat Conf.*, Vancouver, 1986, pp.302-305.

[AHU74] A. Aho, J.E. Hopcroft and A.V. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, Reading, Mass., 1974.

[AIDy88] A. Aimie, S.Dyer, D.Habibi, R.Hanke and J.Teilen, "Design issues in the development of a national fiber-optic transmission network", *Proc. IEEE Int'l Conf. Commun.*, 1988, pp.48-52.

[ALT88] Automated Light Technologies, "Cable monitoring systems (advertisement for)", *Lightwave magazine*, vol.5 no.12, Dec 1988, p.31.

[Amir88] H. Amirazizi, "Controlling synchronous networks with digital cross-connect systems", *Proc. IEEE Global Conf. Commun*, Hollywood,Florida, 1988, pp.1560-1563.

[Anis85] A.V. Anisimov, "A local algorithm for the shortest path problem with a single source", *Trans. in Cybernetics (USA)*, [translated from Kibernetika (USSR)], vol.22, no.3, May-June 1986, pp.327-332.

[AoYo87] K. Aoyama, N. Yoshikai and S. Nishi, "Advanced operation and maintenance concepts of optical fiber transmission systems", *Proc. Int'l Conf. Commun.*, 1987, pp.428-432.

[AsCa81] G.R.Ash, R.H.Cardwell and R.P. Murray, "Design and optimization of networks with dynamic routing", *BSTJ*, vol.60, no.8, 1981, pp.1787-1820.

[AT&T82] American Telephone and Telegraph Co., "Digital multiplex requirements and objectives',*Bell System Technical Reference PUB 43802*, July 1982.

[AT&T87] AT&T Fiber Optics, "Fast and easy fiber restoration", *AT&T Lightguide Digest*, no. 3, 1987.

[AT&T88] AT&T Communications , "C-bit Parity - A Proposed Addition to the Applications in the DS-3 Standard", *Contribution to T1 Standards Project, T1X1.4/88-003*, February, 1988.

[AvDu86] A. Aveneau and R. Dupuis, "A versatile data multiplexer and cross-connector", *Proc. IEEE Global Conf. Commun.*, 1986, pp.1185-1189. (illustrates basic architecture of DCS.)

[BCor85] Bellcore, "Digital Cross-connect system (DCS) requirements and objectives", *Tech. Ref. TR-TSY-000170*, Issue 01, Nov. 198.

[Bear88] D. Bear, *Principles of telecommunication traffic engineering*, Third edition, IEE telecommunication series, Peter Peregrinius, London, U.K., 1988.

[Bell58] R. Bellman, "On a routing problem", *Quart. Appl. Math*, vol.16, 1958, pp.87-90.

[Bell78] Bell Canada, *Guidelines for specifying reliability, maintainability and availability requirements in system performance specifications*, Bell Canada ref. no. DS-8580, June, 1978.

[Bell84] Bell Communications Research, *Digital Crossconnect System (DCS) Preliminary Requirements*, TA 365-23221-84-01, April 1984.

[Bell86] Bell Communications Research, *Asynchronous Digital Multiplexer Requirements and Objectives*, TR-TSY-000009, Issue 1, May 1986.

[BiAl88] P.A. Birkwood, S.E. Aidarous and R.M.K. Tam, "Implementation of a distributed architecture for intelligent network operations", *IEEE J. Sel. Areas Commun.*, vol.6, no 4, May 1988.

[Bour86] J. Bourne, "Technology as service enabler: balancing the strategy", *Proc. FiberSat Conf.*, Vancouver, 1986, pp.377-381. (shows synchronous fiber transport network on which the 'metro' study network in this work is based)

[Brad75] G.H. Bradley, "Survey of deterministic networks", *AIIE Trans*. vol. 7, 1975, pp.222-234.

[Bran85] P.Brant, "Managing Network Dynamics", *Proc. Nat. Commun. Forum*, Oct. 1985, pp.507-509.

[BTL71] Bell Telephone Laboratories, *Transmission Systems for Communications*, 4th edition, 1971, pp.612, pp. 608-615.

[Carr79] B.Carre, *Graphs and Networks*, Clarendon Press, Oxford, 1979

[CCIT85] The international Telegraph and Telephone Consultative Committee, G.703 Specification for Interface at 44736 Kbs (DS3), CCITT 'Red Book' Fascicle III.3, Geneva,1985.

[Chan82] K.M. Chandry and J.Misra," Distributed computation on graphs: shortest path algorithms", *Comm. Assoc. Comput. Mach.*, vol.25, no.11, Nov. 1982, pp.833-837.

[Char88] V.E. Charleton, "Future network and survivability", *TE&M* , Oct. 1, 1988.

[Chau87] P. Chaudhuri, "Algorithms for some graph problems on a distributed computational model", *Information Sciences*, vol. 43 (1987) pp.205-228.

[ChCa86] S.Cheung, J.W. Carlyle, W.J. Karplus, "Asynchronous distributed simulation of a communication network", *Proc. 1986 Summer Computer Simulation Conference*, Reno, July 1986, pp. 147-152.

[Chen82] C.C.Chen, "A distributed algorithm for shortest paths", *IEEE Trans. Comput.*, vol C-31, no.9, Sept 1982, pp.898-899.

[Chen86] M.S. Chen,"Shortest paths in communication networks", *Proc. IEEE Global Conf. on Commun.*, Houston, 1986, pp. 1251-1257.

[ChFr72] W. Chou and H.Frank,"Routing strategies for computer network design", 22nd Int'l Symposium on computer-communications networks and teletraffic", New York, April 1972, pp. 301-309.

[Chri75] N.Christofides, *Graph Theory: An Algorithmic Approach*, Academic Press, London, 1975, pp. 150-186.

[ClKR63] S. Clarke, R.Krikorian and J.Rausen, "Computing the N best loopless paths in a network", *J. Soc. Indust. Appl. Math.*, vol.11, 1963, pp.1096-1102.

[CoIs86] D. M. Cohen and E.J. Isganitis, "Automatic generation of a prototype of a new protocol from its specification", *Proc. IEEE Global Conf. Commun.*, 1986, pp.73-80. (cite in research methods - ie. notion of '*executing the specification*'

[Dant60] G.B. Dantzig, "On the shortest route through a network", *Management Sci.*, vol.6, 1960, pp.187-190

[DeFo79] E.V. Denardo and B.L. Fox, "Shortest route methods", *Opertations Res.*, vol.27, 1979, pp.161-186 (part 1) and pp.548-566 (part 2)

[DeKa88] F. Deguchi, Y. Katsuyama, M. Matsushita and H. Yamaguchi, "Data communication network and database implementation strategy for a transmission network operations system", *IEEE J. Sel. Areas in Commun*, vol. SAC-6, no.4, May. 1988, pp. 722-726.

[DeMa87] K. DeMartino, " A network architecture that exploits fiber optic capabilities", *IEEE Montech '87 Conf. on Commun.*, Montreal, 1987, pp.104-107.

[DePa80] N. Deo, C.Y. Pang and R.E. Lord, "Two parallel algorithms for shortest path problems", *Proc. Int'l Conf. on Parallel Processing*, 1980, pp.244-253.

[DePa84] N. Deo and C. Pang, "Shortest Path Algorithms: Taxonomy and annotation",*Networks*, vol.14, 1984, pp.275-323.

[DGKK79] R.Dial, F.Glover, D.Karney, D.Klingman, "A computational analysis of alternative algorithms and labelling techniques for finding shortest path trees, *Networks*, vol. 9 (1979) pp.215-248.

[Dijk59] E.W. Dijkstra, " A note on two problems in connection with graphs", *Numer Math.* vol.1, 1959, pp.269-271.

[Dirk88] P.Dirke, "Dependability planning of telecommunication networks", *Proc. IEEE Int'l Conf. Commun.*, 1988, pp.245-249.

[DiSi86] N.F.Dinn, L.P. Sinha, J.A. McEntree and R.S.Libenschek, "IFROPS - International facilities relief optimization planning system", *Proc. IEEE Global Conf. Commun.*, 1986, pp.871-876. (source of Pacific basin network graph)

[Dora86] D.G Doran, "Restoration of broadband fiber facilities",*Proc. FiberSat Conf.*, Vancouver, 1986, pp.298-301.

[Drey69] S.E. Dreyfus, " An appraisal of some shortest-path algorithms", *Operations Res.*, vol. 17, 1969, pp.395-412.

[Dutt72] D.L. Duttweiler, "Waiting Time Jitter", *Bell System Technical Journal*, vol 51, January 1972, pp. 165-207.

[ECSA85] Exchange Carriers Standards Association, *Proposed American National Standard for Synchronous DS-3 Format (SYNTRAN)*, ANSI Document T1X1.4/85-050, Final Draft, Sept. 1985.

[ECSA87] Exchange Carriers Standards Association, *Proposed Draft American National Standard on Digital Hierarchical Formats*, T1X1.4/87-702R3, Aug., 1987, pp.25-27.

[Elma70] S.E.Elmaghraby, "The theory of networks and management science", Part 1, *Managment Sci.*, vol. 17, 1970, pp.1-34.

[FlFl87] B. Fleury and S.Fleming, "Digital broadband restoration (DBBR) implementation with digital broadband cross-connect systems",*Proc. IEEE Montech Conf. Commun.*, 1987, pp.169-172.

[Fole87] J.Foley, "Severed AT&T cable interrupts service",*Communications Week* , Sept. 28, 1987.

[Fox73] B.L. Fox, "Calculating kth shortest paths", *INFOR - Can. J. Operational Research and Information Processing.*, vol.11, 1973, pp. 66-70.

[Fred87] G.N. Frederickson, "Fast algorithms for shortest paths in planar graphs, with applications". SIAM J. COMPUT., vol.16, no.6, Dec. 1987, pp.1004-1022.

[Frie77] A.M.Frieze, "Minimal Paths in directed graphs", *Operational Res. Quart*, vol.28, 1977, pp.339-346.

[GaJo79] M.R. Garey and D.S.Johnson, *Computers and Intractability*, W.H. Freeman, New York, 1979.

[GbRu88] P. Gburzynski and P. Rudnicki, *LANSF - A Structural Modeling System for Local Area Networks*, Technical Report, Dept. of Computer Science, University of Alberta.

[GlGl84] F. Glover, R. Glover and D. Klingman, "Computational study of an improved shortest path algorithm", *Networks,* vol.14, 1984, pp. 25-36.

[GrKr87] Grover,W.D. and W.A. Krzymien, "Computation of Available Signaling Rate Using The Stuffed Bits of a DS-3 Signal", *1987 Canadian Society for Information Theory Workshop*, St. Jovite, May 1987. (presented by W.A. Krzymien)-submitted to IEEE Trans. Comm.

[GrLi87] A. Graves, P.Littlewood, S. Carleton, "An experimental crossconnect system for metropolitan applications", *IEEE J. Sel. Areas in Commun,* vol. SAC-5, no.1, Jan. 1987, pp. 6-17.

[GrMo88] W.D. Grover and T.E. Moore, "Precision time transfer in transport networks using digital cross-connect systems", *Proc. IEEE Global Conf. Commun*, Hollywood,Florida, 1988, pp.1544-1548.

[Gar88a] J.J. Garcia-Luna-Aceves, "A distributed,loop-free,shortest path routing algorithm",*Proc. IEEE Infocom*, 1988, pp. 1125-1137.

[Gar88b] J.J. Garcia-Luna-Aceves, "Distributed routing using internodal coordination",*Proc. Computer Networking Symposium*, April 1988, pp. 412-421.

[GiCo84] A.Girard and Y.Cote, "Sequential routing optimization for circuit switched networks, *IEEE Trans. Commun.*, vol. COM-32, Dec. 1984, pp.1234-1242.

[Gros87] R.J. Grosvenor, "Techniques of remote monitoring and control applied to public telecommunications networks as an aid to network management", *First IEE Nat. Conf. on UK Telecommun. Networks - Present and Future*, London, June 1987, pp.84-91.

[Grov86] Bell Canada, "Availability Requirements for DCS Machines in a Dynamic Network", June 1986.

[Grov87a] W.D. Grover, "A Frame-Bit Modulation Technique for Addition of Transparent Signaling Capacity to the DS-3 Signal", *IEEE Pacific Rim Conf.on Commun., Computers and Sig. Proc.*, June 1987, Victoria, B.C., pp. 319-322.

[Grov87b] W.D.Grover, "The Selfhealing network: A fast distributed restoration technique for networks using digital cross-connect machines",*Proc. IEEE Global Conf. Commun*, Tokyo, 1987, vol.2, pp.1090-1095.

[Grov87c] W.D. Grover, Method and Apparatus for Transmission of Auxiliary Information in the DS3 Format, US and Canadian Patents Pending.

[Grov88] W.D. Grover, "On the design of a DS-3 multiplex with signalling channel derived by C-bit liberation", accepted May 1989 for publication in *IEEE Trans. Comm.*

[GrRo85] W.D. Grover and K. Roberts, "Preliminary System Requirements for DCS 3/3 Crossconnect", BNR Report prepared for Bell Canada, TR85-0107, March 1986.

[Grub88] J. Gruber, "Performance monitoring and surveillance in integrated services networks", *IEEE J. Sel. Areas in Commun*, vol. 6, no.8, Oct. 1988, pp. 1378-1383.

[GTE88] GTE Corp., "Restolink cable splice kit (advertisement for)", *Lightwave magazine*, vol.5 no.5, May 1988.

[HaKa87] S. Hasegawa, A.Kanemasa, H.Sakaguchi and R.Maruta, "Dynamic reconfiguration of digital cross-connect systems with network control and management", *Proc. IEEE Global Conf. Commun*, Tokyo, 1987, pp.1096-1100.

[Hans87] R.C. Hansen, "AT&T's digital network evolution", *Proc. IEEE Int'l Conf. Commun.*, 1987, pp.641-645.

[HiEv73] M.T. Hills and B.G. Evans, *Transmission Systems*, Allen & Unwin, London, 1973, pp.30-75.

[Hill79] M.T. Hills, *Telecommunications Switching Principles*, Allen & Unwin, London, 1979. pp.86-103.

[Holl86] P.M. Holland, "The impact of reconfigurable networks", *Proc. Int'l Conf. on Wideband Commun.*, London, England, Oct. 1986, pp.113-125.

[HoSa78] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Chapter 4, Computer Science Press, Rockville, Maryland, 1978.

[IEE87] Institution of Electrical Engineers, *First IEE National Conference on 'UK Telecommunications Networks -Present and Future'*, Short Run Press, Exeter, 1987.

[ImIr84] H. Imai and M. Iri, "Practical efficiencies of existing shortest-path algorithms and a new bucket algorithm", *J. Operations Res. Soc. Japan*, vol. 27, no.1, March 1984, pp. 43-57.

[John77] D.B. Johnson, "Efficient algorithms for shortest paths in sparse networks", *J. Assoc. Comput. Mach.*, vol.24, 1977, pp.1-13.

[Kess88], [Kess86] Kessler Marketing Intelligence, *Fiber Optic Long Haul Systems Planned and In Place*, Newport, Rhode Island, USA, Issue 1 : 1986, Issue 2, 1988.

[Koji87] N. Kojima,"Construction and maintenance on optical fiber cable networks", *Proc. Int'l Conf. Commun.*, 1987, pp.437-443.

[KoTs88] D.J. Kolar and Tsong-Ho Wu, "A study of survivability versus cost for several fiber network architectures", *Proc. IEEE Int'l Conf. Commun.*, 1988, pp.61-66.

[Krei88] J. Kreidl, "British begin work on sharkproof link of TAT-8",*Lightwave*, October, 1988. (Also discusses damage to cables by ship's anchors and trawler nets.)

[KuMo87] M.G. Kulkarn and N.R. Mokhariwale, "Service protection network for transmission systems", *Telecommunications (India)*, vol.37, no.3, June 1987, pp.11-17.

[Lapa88] M. Lapadula, "Microwave to the rescue",*TE&M* , Oct. 1, 1988.

[Lawl72] E.L.Lawler,"A procedure for computing k best solutions to discrete optimization problems and its application to shortest path problems", *Management Sci*, vol.18, 1972, pp.401-405.

[Lawl76] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976, pp.98-108.

[LeeC62] C.Y. Lee, "A note on the N-th shortest path problem", *IRE Trans Comput.*, vol.10, 1962,, pp.572-573.

[LeGu88] F. Lenoir, M.Guiheneuf, A.Monteillet and H. Fava, "Protection switching in the trunk transmission network (140 Mbit/s digital link switching equipment), *Commut. Transm. (France)*, vol.10, no.1, 1988, pp.67-74.

[LiCa88] P. LiCalzi, "Sprint joins other T1 players, announces deployment of SS7",*Communications Week*, Jan 18, 1988.

[MaNa87] P. Mars, "Routing, flow control and learning algorithms", *First IEE Nat. Conf. on UK Telecommun. Networks - Present and Future*, London, June 1987, pp.78-83.

[McRo80] J.M.McQuillan,I.Richter and E.C. Rosen, "The new routing algorithm for the ARPANET", *IEEE Trans. Commun.*, vol COM-28,no.5,pp.711-719, May 1980.

[MaDe82] P.Mateti and N.Deo, "Parallel algorithms for the single source shortest path problem", *Computing*, vol.29, 1982, pp.31-49.

[MaGr89] M. MacGregor, W. Grover, " Simulation experiments of traffic dynamics, with reattempts, on failed trunk groups subject to varying restoration models", *ATRC seminar*, Dec 1988.

[MaLi87] W. C. Marra, J.S. Linnel, N.M. Demfin and K.Ogawa, "FT series G lightwave digital transmission system architecture and upgrade capacity", *Proc. Int'l Conf. Commun.*, 1987, pp. 650-654.

[Marr87] L.J. Marra, "The sharkbite threat to the TAT-8 Cable System",,*Proc. IEEE Global Conf. Commun*, Tokyo, 1987, pp.1293-1296.

[Mars73] J. Marshall, "On Lawler's K best solutions to discrete optimization problems", *Management Sci.* vol.19, 1973, pp.834-835.

[Mayo88] J.S.Mayo, "Photonics technology offers speed-of-light",*Communications Week*, Feb. 22, 1988.

[McGo88] R.E. McGorman, "A methodology for designing survivable telephone networks",*Proc. IEEE Int'l Conf. Commun.*, 1988, pp.1172-1176.

[Meye88] D.Meyer, "How to exploit T1 savings",*Communications Week*, Aug. 22, 1988.

[Mini78] E.Minieka, *Optimization Algorithms for Networks and Graphs,* New York: Dekker, 1978, pp.65-77.

[Mint58] G.J. Minty, "A variant on the shortest-route problem", *Operations Res.* vol.6, 1958, p.882.

[MoBh86] H.Mohanty and G.P. Bhattacharjee, "A distributed algorithm for the shortest path problem", *Proc. IEEE Global Conf. Commun*, Houston, 1986, pp.1272-1276.

[MoMe81] F.H. Moss and P.M. Merlin, "Some theoretical results in multiple path definition in networks", *Networks*, vol.11 (1981) pp.401-411.

[MoGr87] T.E. Moore and W.D. Grover, "Suggested applications of reserved 'Future Use' overheads in SONET", contribution to T1X1.4 Optical Rates and Formats sub-working group, T1X1.4/87-541, November 1987.

[Moor59] E.F. Moore, "The shortest path through a maze", *Proc. Intl. Symp. Switching Theory,* Cambridge, Mass.,1959, pp.285-292.

[NaGr86] J.A. Nagel and J.R. Gray, "Digitizing the AT&T communications network", *Proc. IEEE Global Conf. Commun.*, 1986, pp.852-856.

[Nat88] Nature, article on "Travelling Salesman Problem", Nov 1988. *(temporarily misplaced- full citation to follow)*

[Nell86] J.G. Nellist, "Fiber optic system availability", *Proc. FiberSat Conf.*, Vancouver, 1986, pp.367-372.

[Pall84] S. Pallottino, "Shortest-path methods: complexity, interrelationships and new propositions", *Networks,* vol.14, 1984, pp. 257-267.

[PaRa85] S. Pawagi and I.V. Ramakrishnan, "On using multiple inverted trees for parallel updating of graph properties", *Proc. 23rd Allerton Conference on Communications, Control and Computing,* 1985, pp. 251-8.

[PeLe83] I.M. Pesic and D.W. Lewis, "Three heuristics for improving centralized routing in large long-haul computer communication networks", *Nat. Computer Conf.*, Anaheim, 1983, pp. 691-704.

[Perk86] A. Perko, "Implementation of algorithms for k-shortest loopless paths", *Networks*, vol. 16 (1986) pp.149-160.

[Pier75] A.R. Pierce, "Bibliography on algorithms for shortest path shortest spanning tree and related circuit routing problems", *Networks,* vol.5, 1975, pp.129-149.

[Plam85] Y. Plamondon, "Structure availability study for FOTS in the Intertoll Network", Bell Canada Technical Report, Jan. 1985.

[Poll61] M. Pollack, "Solutions of the kth best route through a network - a review", *J.Math Anal. Appl.*, vol.3, 1961, pp. 547-559.

[Poll61] M. Pollack, *'The kth best route through a network'*, Operations Res., vol.9, 1961, pp.578-580.

[PoOl72] R.B. Potts and R. Oliver, *Flows in Transportation Networks*, Academic Press, New York, 1972.

[PoWi60] M.Pollack and W.Wiebenson, "Solutions of the shortest-route problem - a review", *Operations Res.*, vol 9. 1960, pp.411-412.

[QuDe84] M.J.Quinn and N. Deo, "Parallel Graph Algorithms", *Computing Surveys*, vol. 16, no.3, Sept 1984, pp.319-348.

[QuYo84] M.J. Quinn and Y.B. Yoo, "Data structures for efficient solution of graph theoretic problems on tightly-coupled MIMD computers", *Proc. Int'l Conf. on Parallel Processing*, Bellaire, MI, 1984, pp. 431-438.

[RaGh86] Ratan, K.Ghosh and G.P. Bhattacharjee,"Parallel algorithm for shortest paths", *IEE Proc.*, vol.133, Pt.E, no.2, Mar. 1986, pp.87-93.

[RiPa86] P.Richards and R.Pandya, "Towards dynamic network control", *Proc. FiberSat Conf.*, Vancouver, 1986.

[RoKa88] Y.Rokugo, Y.Kato, H.Asano, K. Hayasi et al, "An asynchronous DS3 cross-connect system with add/drop capability", *Proc. IEEE Global Conf. Commun.*, Hollywood, Florida, 1988, pp.1555-1559.

[RoPe84] D. Ronen and Y.Perl, "Heuristics for finding a maximum number of disjoint bounded paths", *Networks*, vol. 14 (1984) pp.531-544.

[RoRo86] E.Roohy-Laleh and N.Ross, "Network design for survivability: Procedure and case study in a dynamic network architecture, "*Proc. IEEE Global Conf. Commun.*, Toronto, Ont., 1986, pp.923-930.

[Rose80] E.C. Rosen, "The updating protocol of ARPANET's new routing algorithm", *Comput. Networks*, vol. no. 1, pp.11-30, Feb. 1980.

[RoSe85] N.Robertson and P.D.Seymour, "Disjoint paths - A survey", *SIAM J. Algebraic Discrete Methods,* vol.6, Apr. 1985, pp.300-305.

[Rudi76] H.Rudin, "On routing and 'delta routing': A taxonomy and performance comparison of techniques for packet-switched networks", *IEEE Trans. Commun.*, vol. COM-24, Jan. 1976, pp.43-59.

[Saka68] M. Sakarovitch, "The k shortest routes and k shortest chains in a graph", *Transport. Res.,* vol.2, 1968, pp.1-11.

[SaOm88] V. Sahin, C.G.Omidyar,T.M. Bauman, "Telecommunications management network (TMN) architecture and interworking designs", *IEEE J. Sel. Areas in Commun,* vol. 6, no.4, May 1988, pp. 685-695.

[Saul86] D.F. Saul, "Constructing Telecom Canada's coast to coast fibre optic network",*Proc. IEEE Global Conf. Commun.*, Toronto, Ont., 1986. pp.1684-1688.

[Schl87] M.J.Schikner, "Evaluation and operation of an automatically switched digital service protection network",*First IEE Nat. Conf. on UK Telecommun. Networks - Present and Future*, London, June 1987, pp.72-77.

[ScSt80] M. Schwartz and T.E. Stem, "Routing techniques used in computer communication networks", *IEEE Trans. Commun.*, vol. COM-28, no.4, April 1980, pp.539-552.

[ScTh86] U. Schwiegelshohn and L Thiele, "On the systolic detection of shortest routes ", *Proc. Int'l Conf. on Parallel Processing*, St. Charles IL, Aug. 1986, pp.762-764.

[Sega79] A. Segall, " Failsafe distributed loop-free routing in communication networks", *4th Int'l Conf. Perf. of Computer Systems*, Vienna, 1979, pp. 541-564.

[Sega83] A.Segall, "Distributed network protocols", *IEEE Trans. Inform. Theory*, vol. IT-29, pp.23-35, Jan. 1983.

[Shei79] D.R. Shier, "On algorithms for finding the k-shortest paths in a network", *Networks*, vol. 9 (1979) pp.195-214.

[Shie74] D.R. Shier, "Computational experience with an algorithm for finding the k shortest paths in a network", *J. Res. Nat. Bur. Standards.*, vol. 78B, 1974, pp.139-165.

[Shie76] D.R. Shier, "Iterative methods for determining the k-shortest paths in a network", *Networks*, vol.6 (1976) pp. 205-229.

[Sipe88] C.M. Siperko, "WILTEL - A Rapidly growing private line network", *Proc. Int'l Conf Commun.*, 1988, pp. 37-42.

[Smit85] D. R. Smith, *Digital Transmission Systems*, Van Nostrand Reinhold Co., New York, 1985, pp. 171-181.

[SnKa87] R.K. Snelling and K. Wilson-Kaplan, "All fiber networks - a reality by 1988-89", *Proc. IEEE Int'l Conf. Commun.*, 1987, pp.1521-1523.

[SoAg81] I.M. Soi and K.K. Aggarwal, "On shortest-path algorithms in the topological design of computer networks : a comparative study", *Int. J. Systems Sci.*, vol.12, no.11, 1981, pp. 1379-1387.

[SONE88] American National Standard Draft, *Optical Interface Rates and Formats (SONET)*, ANSI Doc. T1.105-1988, March 1988.

[SuTa84] J.W. Suurballe and R.E. Tarjan, "A quick method for finding pairs of disjoint paths", *Networks*, vol. 14 (1984) pp.325-336.

[Sutt86] P.J. Sutton, "Service protection network",*Proc. IEEE Global Conf. Commun.*, Houston, 1986, pp.862-865.

[Suur74] J.W.Suurballe, "Disjoint paths in a network", *Networks*, vol. 4 (1974) pp.125-145.

[SYNT87] American National Standard, *Digital Hierarchy - Synchronous DS-3 Format Specifications (SYNTRAN)*, ANSI Doc. T1.103-1987, August 1987.

[Szen86] O.I. Szentesi, "Reliability of optical fibers, cables and splices", *IEEE J. Sel. Areas Commun.*, vol. SAC-4, no.9, Dec 1986, pp.1502-08.

[Tatu87] AT&T Communications (Tatulis, V.C), "DS-3 C-bit Parity - A Network Maintenance Application of the M-frame C and X-bits in the Asynchronous DS-3 Format", *Contribution to T1 Standards Project, T1X1.4/87-705*, February, 1987.

[TelC83] Telecom Canada Technical Staff, *Digital Network Notes*, Telecom Canada, Ottawa, 1983.

[TelC88] Information made available under ATRC - Telecom Canada Research contract on Selfhealing Networks.

[Thom76] R.E. Thomas, "On Optimum path problems", *Networks*, vol.6 (1976) pp.287-305

[Titc88] S.Titch, "Users, State investigate Bell CO (Central Office) Fire",*Communications Week*, Sept, 1988. (Note: this is the so-called 'Hinsdale disaster'.)

[Topk88] D.M. Topkis, "A k shortest path algorithm for adaptive routing in communications networks", IEEE Trans. Commun., vol. 36, no.7, July 1988, pp.855-859.

[ToUe86] I. Tokizawa, H. Ueda, H. Tsuji and M. Sumida, "Large capacity non-blocking digital cross-connect system using high speed optical links", *Proc. IEEE Global Conf. Commun.*, 1986, pp.1174-1178.

[ToUe87] I. Tokizawa, H. Ueda, H. Tsuji, "A synchronous DS4 multiplexer with cross-connect function", *IEEE J. Sel. Areas in Commun*, vol. SAC-5, no.1, Jan. 1987, pp. 19-25.

[Turn87] E. Turner, " Fiber optic maintenance", *Proc. Int'l Conf. Commun.*, 1987, pp.454-461.

[Vish83] U.Vishkin, " An efficient distributed orientation algorithm algorithm", *IEEE Trans. Infor. Theory*, vol. IT-29, no.4, July 1983, pp.624-629.

[ViVi87] R.Vickers and T. Vilmansen, "The evolution of telecommunications technology", *IEEE Communications Magazine*, vol.25, no.7, July 1987, pp.6-18.

[Vlie78] D. Van Vliet, "Improved shortest path algorithms for transport networks", *Transportation Research*, vol.12, 1978, pp.7-20.

[Wein73] A. Weintraub, "The shortest and the k-shortest routes as assignment problems",*Networks*, vol. 3 (1973) pp.61-73.

[Whit88] A.L.White, "US SPRINT's high capacity, long-haul fiber network", *Proc. Int'l Conf Commun.*, 1988, pp. 32-36.

[Wong76] A.Wongseelashote, "An algebra for determining all path-values in a network with application to k-shortest paths problems", *Networks*, vol. 6 (1976) pp.307-334.

[YaHa88] C. Han Yang and S. Hasegawa, "FITNESS: A failure immunization technology for network service survivability", *Proc. IEEE Global Conf. Commun*",Hollywood,Florida, 1988, pp.1549-1554.

[YaIw86] H. Yamamoto, M. Iwashita and H. Nakanishi, " A high reliability and high usage subscriber optical fiber cable network ", *Proc. IEEE Global Conf. Commun.*, Toronto, Ont., 1986, pp.376-380.

t[Yan86] J. Yan, "Impact of metropolitan fiber systems on special services digitization", *Proc. Fibersat Conf.*, Vancouver, 1986, pp. 292-297.

[Yen71] J.Y. Yen, "Finding the k-shortest loopless paths in a network", *Management Sci.*, vol 17, July, 1971, pp.712-716.

[YuKa86] Y.Okano, T.Kawata and T.Miki, "Designing digital paths in transmission networks", *Proc. IEEE Global Conf. Commun.*, 1986, pp.857-861.

[YuNo86] J. Yukawa, H. Nomura and M. Shimodaira, "Fiber optics system applications for private network communication systems", *Proc. IEEE Global Conf. Commun.*, Toronto, Ont., 1986, pp.1216-1221.

[Zane88] P. Zanella,"Customer network reconfiguration applications utilizing digital crossconnect systems", *Proc. IEEE Global Conf. Commun*, Hollywood,Florida, 1988, pp.1538-1543.

[Zorp89] G. Zorpette, "Keeping the phone lines open", *IEEE Spectrum* special report, June 1989, pp. 32-36.

APPENDIX A: SUMMARY OF SHORTEST PATHS LITERATURE REVIEW

Reference	Problem Type	Network Type	Path Type	Computational Model	Comments/ Emphasis
[Thom 76]	s, a	s	e	c	• MST, SP & max reliability path problems - overview
[DiGR 79]	s, a	s	e	c	• computational methods for label setting algorithms
[MaNa 87]	s, a	s	e	d	• learning automata for packet network routing
[Anis 85]	s	s	e	d	• non-deterministic algorithms
[Fred 87]	s, a	s	e	c	• preprocessing of data structures to enhance SP algorithms
[Dijk 59]	s, a	s	e	c	• basic label-setting single-source SP algorithm
[SaSt 80]	s, a	s	e	c	• survey of packet network routing algorithms
[SoAg 81]	s, a	s	e	c	• comparative study of SP algorithms for packet networks
[ImIr 84]	s, a	s	e	c	• empirical efficiencies of common algorithms & a new algorithm
[GlGl 84]	s, a	s	e	c	• comparative study of a hybrid label-setting/ correcting algorithm
[Pall 84]	s, a	s	e	c	• priority queues in SP algorithms
[Chen 86]	s	s	e	n/a	• theoretical study of distribution of elementary paths in a network
[ChFr 72]	s, a	s	e	c	• survey of algorithms for packet network routing
[PeLe 83]	s, a	s	e	c	• heuristics for improving centralized algorithms for packet network routing
[Vlie 78]	s, a	s	e	c	• computational efficiency of common SP algorithms on road networks
[PaRa 85]	a	s	e	p	• all pairs shortest paths algorithm for MISD processor using $O(\log_n)$ time & $O(n^3)$ processors

APPENDIX A: SHORTEST PATHS LITERATURE - (continued)

Reference	Problem Type	Network Type	Path Type	Computational Model	Comments/ Emphasis
[MoBh 86]	s	s	e	d	* distributed algorithms for finding all-pairs shortest paths in packet network context
[Gar 88a]	s	s	e	d	* distributed algorithm for all-pairs shortest paths in packet network context
[Chau 87]	s	s	e	d	* algorithms for distributed graph searching & finding lengths of shortest paths to other nodes
[Sega 79]	s	s	e	d	* loop-free distributed shortest path route finding algorithms for packet networks
[Gar 88b]	s	s	e	d	* loop-free distributed shortest path algorithms for packet networks
[ScTh 86]	s	s	e	s	* all-pairs shortest path on pure systolic array: O(n) time, $O(n^2)$ processors
[DePa 80]	s	s	e	p	* parallelization of well-known s.p. algorithms for MIMD parallel processors
[QuDe 84]	s	s	e	p	* parallel graph algorithms for MST, SP & TS problems
[MaDe 82]	s	s	e	p	* parallelized Dijkstra and Moore SP single source algorithms
[QuYo 84]	s	s	e	p	* parallelization of MST & single-source SP algorithms with improved data structures for MIMD processors
[AbRh 82]	s	s	e	d	* single source SP algorithm in distributed asynch model using only local information
[ViSh 83]	o	s	n/a	d	* an algorithm for distributed asynch nodes to acquire global topology knowledge in $O(/E/^2)$
[Topk 88,86]	k	a	o	x	* k-shortest paths with the 1st links disjoint only

APPENDIX A: SHORTEST PATHS LITERATURE - (continued)

Reference	Problem Type	Network Type	Path Type	Computational Model	Comments/ Emphasis
[Perk 86]	k	s	e	c	* k-shortest by path distances only, loopless, no disjointness constraints
[Shie 76]	k	s	e	c	* three centralized algorithms for single source k shortest loopless paths not satisfying any disjointness
[Wein 73]	k	s	e	c	* use of "assignment problem" methods for same non-disjoint k-paths problem as [Shie 76]
[Wong 76]	k	s	c	c	* an algebra for k-shortest path problems
[Shie 79]	k	s	e	c	* review of non-disjoint k-shortest path methods
[Yen 71]	k	s	d	c	* k length distinct loopless paths in a simple graph
[Lawl 76]	k	s	d	c	* k node-order distinct paths in a simple graph
[Shie 74]	k	s	e	c	* empirical computational results with some k-shortest path (non-disjoint algorithms)
[Poll 61]	k	s	e	c	* survey of algorithms on (non-disjoint) k-shortest paths
[Suta 84]	o	s	ed	c	* k=2 (pairs) of edge-disjoint paths, centralized algorithm
[Suur 74]	k	s	nd	c	* K node disjoint paths, extension to edge-disjoint discussed
[RoPe 84]	k	s	nd	c	* heuristics to assist in finding max. # of node-disjoint paths with length bound
[MoMe 81]	k	s	d	c	* theoretical conditions on a graph to provide k different paths (not shortest) between all node pairs

APPENDIX B: SELFHEALING PROTOCOL SPECIFICATION

PART I. DATA STRUCTURES, VARIABLES AND CONSTANTS

```
CONST  nports      = /*DCS size};{might be built with sweep as well*/
    nmaxspans   = /*maximum number of spans this node could have*/
    maxrepeats  = /* repeat value above which no signature is broadcast further*/;
    maxspansize = /*largest span at this node*/
    thisnode    = /*assign network wide ID of this node};
    timeout     = /* Sender node timeout value */
    IRV         = /* initial repeat value for this node or entire network*/

TYPE  nodeid = /*enumerated list of valid node names plus a nul value*/;
    current-states = (Normal, Setup-tandem, Sender, Chooser);
    portID  = 1..nports, nul;
    signalids = /* Definition according to network-wide ID scheme, plus a nul value*/

    Sig-register = RECORD
            NID, source, target  :nodeid;
            index  :0..nports, nul;
            repeat :0..maxrepeats+1;
            END;

    Status-reg = RECORD
            alarm, spare, conn, SIE, SCIE, AIE, RSdel, IDdel :boolean;
            SigID   :signalids;
            assoc-port  :portid;
            END;

    interrupt = /* set of valid interrupt vectors in this OS};

VAR   RS, TS : ARRAY [1..nports] OF Sig-register ;
    PS: ARRAY [1..nports] OF Status-reg;
    Int-port, scanport: portid;
    Affected-Ports, Alarmed-Ports : SET OF portid;
    alarm-span, int-span : nodeid;
    Spans : SET OF nodeid; /* nodeids are used for enuneration of logical spans at a node*/
    var indices-choosen : SET of valid index numbers
    SH-interrupt-vector: interrupt /*interrupt assigned for SH task invocation*/
    state : current-states;
```

PART II. UTILITY PROCEDURES AND FUNCTIONS USED BY STATE IMPLEMENTATIONS

Procedure Initialize-Span-Sets

1. Identifies logical spans at this site using principle of common RS.NID for all links of same span.
2. Creates a set [Spans] *which includes each span found, identified by the NID of adjacent node common to links in that span.*
3. For each span found, three subsets are formed from port numbers in the span which have (SPARE and not ALARM) = true (ie. available for Selfhealing use):

[emp-set@span] = set of ports with nul TS
[send-set@span] = set of ports with an active TS
[rec-set@span] = set of ports with an active RS

4. Returns a variable alarm-span *which is set to the span number of the failed span in a node which is attached to the failed span, otherwise it is nul. Note alarm-span is the nodeid of the node on the other end of the failed span.*
Notes: 1. A single port number may appear in more than one subset for the same span.
2. These sets are addressed as [<subset> @ span#]

3. Elements of these sets are accessible by an operator next (set) *that advances cyclically to another unique member of the set each time it is used. When all elements of the set have are exhausted* next(-) *returns* nil. reset(set) *is used to reinitialize the* next(-) *operator whenever a search on the whole set is desired.*

```
Procedure move(port, set1, set2)
/*Moving ports from one of the sets created above to another during signature processing is
performed by this function */
{[set1] := [set1] - [port];
 [set2] := [set2] + [port];}

Function avail-empties : boolean
/* see if any free ports remain to be had in any span other than the present int-span*/
avail-empties := false; int-span := RS.NIS @ int-port;
FOR span:= 1 to nspans DO WHILE avail-empties:=false;
        {reset [emp-set@span];
         IF (next[emp-set@span] <> nil) AND (span <> int-span) THEN avail-empties :=true};

Fun     Test-for-Involved: boolean
invol[illegible] := false;
FOR every span IN [Spans] DO IF (rec-set@span <>nil) THEN Involved := True;}

Function Sending-to-span(span, index): boolean
/* returns true iff a TS already exists in given span of specified index */
testvar := false;
reset[send-set@span]
repeat
        scanport := next[send-set@span]
        If(TS.index@scanport = index) THEN testvar :=true
until (sending or scanport = nil)
sending-to-span := testvar

Procedure Sender-Flood;
index-stamp := 1
for every span in ([Spans] NOT alarm-span)) DO
        For i := 1 to min( lostccts, |span|) DO
                { reset[emp-set@span]; scanport := next [ emp-set @ span];
                 TS.index @ port = index-stamp
                 TS.repeat @ port := IRV
                 TS.source @ port := thisnode
                 TS.target @ port := alarm-span
                 TS.mode @ port := 0
                 move (port, emp-set @ span, send-set @ span)
                 index-stamp := index-stamp + 1 }

Procedure Sel-Broadcast (int-port);
rootspan := RS.NID @ int-port
rootindex := RS.index @ int-port
for every span IN ([Spans] NOT rootspan)) do
        if ((|emp-set @ span| > 0) AND NOT Sending-to-span(span,rootindex) THEN
                { reset[emp-set@span];
                 scanport := next [ emp-set @ span]
                 TS.index @ port = RS.index @ int-port
                 TS.repeat @ port := RS.repeat @ int-port +1
                 TS.source @ port := RS.source @ int-port
                 TS.target @ port := RS.target @ int-port
                 move (port, emp-set @ span, send-set @ span)} }
```

Procedure Build-Affected-Ports-Table;

```
/*Scans all DCS ports to find any new alarms on working ports, counts them (lostccts), stores the
portnumbers of the ports these failed facilities were connected to through the matrix and resets
the interrupt line RSdelta for all ports it has found alarmed. NOTE: Not span specific; will pick up
and lump all alarmed ports, assuming only one span failure.}
lostccts:=0; [affected-ports] := nul; [alarmed-ports] := null;
FOR scanport := 1 TO nports DO
        { IF RS.alarm@scanport AND NOT (PS.spare@scanport) AND RS[index].RSdelta THEN
        /*New alarm on working link found*/
         {lostccts = lostccts +1; /*add port to list*/
         [affected-ports] := [affected-ports] + [PS.Assoc-port@scanport]
         [alarmed-ports] := [alarmed-ports] + scanport }}
```

Function BetterSig(int-port: PortID):boolean;

```
/*Looks to see if an incoming sig on ScanPort is the best of its kind coming in for a given
indexcount and source-target relation. An incoming signature with a lower repeatcount or already
complemented is considered to be better.*/
var better : boolean
better := false ; ;
FOR Span := 1 TO NSpans DO { /*compare current RS on ScanPort against all other ports*/
        { reset[rec-set@Span]; Scanport := next [rec-set@span];
         WHILE (Scanport < > nil) AND (Scanport < > intport) AND (NOT Better) DO
         { IF ((RS.index@Scanport = RS.index@intport)
          THEN IF (RS.repeat@scanport < RS.repeat@intport)) THEN better := true
          ELSE IF (RS.repeat@ScanPort = RS.repeat@intport AND Complement(Scanport))
          THEN Better:=true;}
          Scanport := next [rec-set@span] }}
Bettersig:=better
```

Procedure Cancel-TX-Sigs-From (rootport, protport): portid; VAR NewEmptyPort:boolean);

```
/*cancels all TXsigs stemming from a given precursor RS at rootport with protection given to TS's
which lie in the span specified by protport. These properties make it useable for normal flood
canceiling (call with rootport = protport) or in reverse linking, (call with rootport being reverse
linking precursor and protport having the respective complement condition */
var skip, newemptyport :boolean;
rootspan := RS.NID@rootport; protspan := RS.NID@protport;
FOR Span:=1 TO NSpans DO
        IF span < > protspan THEN
                {reset [send-set@span];
                repeat
                        scanport := next[send-set@span];
                        IF (PS.assoc-port@scanport = rootport ) THEN
                                { Skip := true;
                                 nullout(TS@scanport);
                                 move (scanport, send-set@span, emp-set@span);
                                 newemptyport := true;}
                until (scanport < >nil) or skip }
```

Function Sending-To-Span (Span:integer; Port1:PortID):boolean;

```
/*Tests specified span to see if there is presently any TS signature of the same index family as the
RS found at Port1. This function has the by-product of revising TS precursor association where
improvements are possible */
var sending : boolean;
testindex:=RS.index @port;
sending := false;
reset[send-set@span];
repeat
        scanport:=next[send-set@span];
        IF (RS.index@scanport = testindex)
        THEN {sending := true;
                IF (TS.repeat @ scanport > RS.repeat@port1 +1) THEN
                { TS.repeat@scanport := RS.repeat@port1 +1;
                  PS.assoc-port@scanport := port1; } /*IF*/
until sending OR (scanport = nil);
sending-to-span := sending
```

PART III. ONE-TIME INITIALIZATION AND MAIN INTERRUPT-DRIVEN FSM

```
{  /* one time installation initializations*/
[Affected-Ports] := nul;  [Alarmed-ports] := nul; state := normal;
FOR port := 1 TO nports DO
  {     nullout(RS @ port);  nullout(TS @ port);
        SIE @ port := false; AIE @ port := true;
        SCIE @ port := true; TS.NID @ port := thisnode;
        TS.mode @ port := 0; RDdel := false; IDdel: =false} ;

REPEAT   /*infinite repeat loop representing interrupt-driven execution of SH task */
Wait-Suspend (SH-interrupt-vector, int-port);
/*Suspend the SH task until an interrupt occurs on the interrupt which is reserved for Selfhealing.
When awakening SH-task, supply identity of int-port, the port that caused the interrupt*/
    CASE state OF
          Normal:  Normal-node;
          Sender :  Sender-node;
          Chooser:  Chooser-node;
          Tandem:  Tandem-node
    END;
    RS[int-port].RSdel: = false;/*clear signature interrupt*/
UNTIL false; /*Repeat infinitely*/
END.
```

PART IV. Selfhealing FSM STATE IMPLEMENTATIONS

Procedure Normal-Node;

```
type events = (senderalarm, chooseralarm, repeatable-sig, nonrepeatable-sig);
{initialize-span-sets;
IF PS.alarm @ int-port then
   IF (ord(thisnode) > ord(RS.NID @ int-port) THEN event: =senderalarm
                                              ELSE event: =chooseralarm; }
   ELSE IF (RS.repeat @ int-port < maxrepeat)
               then event: =repeatable-sig
               else event: =nonrepeatable-sig;
CASE event OF
        senderalarm:
               Build-Affected-Ports-Table; Restccts: =0;
               Sender-Flood;  Start-Timer(timeout);  state: =sender;  ;
        chooseralarm:
               Build-Affected-Ports-Table; Restccts: =0;
               [indices-chosen] := [nul];  state: =Chooser;  ;
        repeatable-sig:
               sel-broadcast (int-port);
               [rec-set@(RS.NID@int-port)] := [rec-set@(RS.NID@int-port)] + [int-port];
               state:= Tandem;  ;
        nonrepeatable-sig:
               state:= normal ;
end ; /*CASE*/
```

Procedure Sender-node;

```
type events = (Timer-int, Return-sig, late-Alarm);
{ IF (interrupt-source = timer) then event: =timer-int
 else if (RS.target @ int-port = thisnode AND PS.spare@int-port AND NOT PS.alarm@int-port)
    then event: =Return-sig
 else if (PS.alarm @ int-port AND NOT PS.spare@int-port) then event := late-Alarm
```

```
CASE event OF
timer-int :
        for every span in ([Spans] NOT alarm-span)) do
                { for every port in (send-set @ span) if not complement(port) do
                  nullout (TS @ port)  move(port, send-set @ span, emp-set @ span)  }
return-sig:
        if complement (int-port) then {restccts:=restccts+1;
        Xpoint (Affected-ports[restccts],Int-Port);
        /*forwarding of SigID is implicit with operation of the crosspoint*/
         if (lostccts = restccts) then
           { for every span IN ([Spans] NOT alarm-span) do
                { for every port in (send-set @ span) if not complement(port) do
                  { nullout (TS @ port);  move(port, send-set @ span, emp-set @ span)}}}
late-Alarm:
        if int-port NOT IN [Alarm-ports] then { lostccts := lostccts + 1;
         [affected-ports] := [affected-ports] + [TS.assoc-port @ int-port];
         [alarm-ports] := [alarm-ports] + [int-port] };
end /*case*/
state:=Sender
```

Procedure Chooser-node;

```
type events=(New-Forward-Sig, SigID-Arrive, Late-Alarm, No-Sig);
if (PS.alarm @ int-port AND NOT PS.spare@int-port) THEN event := late-Alarm
        ELSE IF ((PS.SCIE @int-port) and (RS.TARGET @ int-port = thisnode)
        AND (RS.SOURCE = alarmspan )) then event:= New-Forward-Sig
        ELSE IF (PS.SIE @ int-port ) then event:=SigID-Arrive
CASE event OF
New-Forward-Sig:
        int-span := RS.NID @ int-port
        move (int-port, emp-set @ span, rec-set @ span)
        if (restccts<lostccts) then
        {  IF NOT ((RS.INDEX @ int-port) IN [indices-choosen])then
                { [indices-chosen] := [indices-chosen] + RS.INDEX @ int-port
                  TS.TARGET @ int-port := alarm-span
                  TS.SOURCE @ int-port := thisnode
                  TS.REPEAT @ int-port := IRV
                  TS.INDEX  @ int-port := RS.INDEX @ int-port
                  move (int-port, emp-set @ span, send-set @ span)
                  PS.SIE @ int-port := true
                  PS.SCIE @ int-port := false
                  restccts:=restccts+1; } };
SigID-Arrive:
        find scanport in [affected-ports] such that (PS.SIGID @ scanport = PS.SIGID @ int-port);
        Xpoint (port, int-port) ;
        TS.assoc-port @ int-port := scanport ;
        [Affected-ports] := [Affected-ports] - [scanport];
        PS.SIE @ int-port := false
late-Alarm:
        IF int-port NOT IN [Alarm-ports] then { lostccts := lostccts + 1;
         [affected-ports] := [affected-ports] + [TS.assoc-port @ int-port];
         [alarm-ports] := [alarm-ports] + [int-port] }};
end /*case*/
state:=Chooser ;
```

Procedure Tandem node;

```
VAR involved, avail-empties, newemptyport:boolean;
involved:=false; newemptyport:=false;
/* Primary invocation of Tandem FSM Logic caused by RS signature event on int-port...*/
TandemFSM (int-port,involved,newemptyport);
```

```
/* Secondary applications of Tandemnode FSM with psuedo int-ports to maximize application of
rebroadcast rules iff one or more TS ports became free as a result of primary invocation above */
IF newemptyport AND involved THEN
{minrepeat:= maxrepeat ; /* find globally lowest repeat count at the node */
FOR span:= 1 to nspans DO
        {reset[rec-set@span]; scanport := next[rec-set@span];
        WHILE (scanport <> nil) { IF (RS.repeat@scanport < minrepeat)
         THEN minrepeat := RS.repeat@scanport; next[rec-set@span]}/*while*/ } /*for*/
/* now prioritize access on basis of repeat fields of existing RS's*/
FOR repeat-order := minrepeat to maxrepeat-1 DO
 /* If there are empty TS ports in any spans other the current span, then if there is an RS in the
current
 span that has repeat = the current repeat-order value, the rexecute Tandemlogic from the
viewpoint
 of that port...*/
 reset[rec-set@span];
 WHILE (avail-empties AND scanport <> nil ) DO
    {scanport := next[rec-set@span];
    IF (RS.repeat@scanport = repeat-order AND NOT PS.RSDEL)
    THEN TandemFSM (scanport,newemptyport,involved)}/* WHILE */;

IF involved THEN state := Tandem;
            ELSE {state := Normal; (and release all data structures)}
/* END of Tandem-node Procedure */
```

Procedure TandemFSM(int-port, newemptyport, involved)

```
type events = (new-complement-sig, repeatable-sig, nonrepeatable-sig, sig-vanish, overrange-sig,
update-repeat);
int-span := RS.NID@int-port;
/* event parsing...*/
/* 1. see if int-port previously had an RS before this interrupt */
        was-empty := false;
        if (int-port NOT IN [rec-set@int-span]) then was-empty := true;
/* 2. see if this int-port has a physically new signature event or we are revisting an existing RS */
        new-sig := false;
        if (PS.RSDEL@int-port) then new-sig := true;
/* 3. see if the RS port is now empty */
        is-empty := false;
        if (RS.source@int-port = null) then is-empty := true;
/* 4. see if this port was previously a precursor.  If so note its index */
        was-precursor :=false
        was-index := null
        new-index := RS.index@int-port;
        IF NOT was-empty then
                { WHILE span IN [Spans] NOT int-span DO
                {reset[send-set@span]
                Repeat
                scanport:= next(send-set@span)
                        If (PS.Assoc-port@scanport = int-port) THEN {was-precursor :=true;
                                                was-index := TS.index@ scanport};
                Until was-precursor OR (scanport = nil)}}
/* now sort out which physical (or psuedo) signature event applies */
event:=non-repeatable-sig
IF (NOT was-empty AND is-empty) THEN event:= sig-vanish
IF (NOT is-empty AND NOT Bettersig(int-port))) THEN event:= repeatable-sig
IF (new-sig AND Complement(int-port)) THEN event:= new-complement-sig
IF (NOT is-empty AND NOT was-empty AND (new-index = was-index)) THEN event := update-
repeat
IF (RS.repeat@int-port >= maxrepeats) THEN event= overrange-sig

/* event processing*/
```

```
involved:=false; newemptyport:=false;
CASE event of
sig-vanish:
        [rec-set@int-span] := [rec-set@int-span] - [int-port];
        cancel-tx-sigs(int-port, int-port, newemptyport);
        test-for-involved ;
new-complement-sig:
        other-port := PS.Assoc-port@int-port;
        other-span := RS.NID@other-port;
        cancel-tx-sigs(int-port, other-port, newemptyport);
        TS.Source@(other-port) := RS.Source@int-port ;
        TS.Target@(other-port) := RS.Target@int-port ;
        TS.Index @(other-port) := RS.Index@int-port ;
        TS.Repeat@(other-port) := RS.Repeat@int-port +1;
        TS.Assoc-port@(other-port) := int-port;
        [send-set@other-span] :=  [send-set@other-span)] + [Other-port];
        Xpoint(int-port, other-port);
        involved:=true;
repeatable-sig:
        IF (was-precursor AND (was-index <> is-index) THEN
        cancel-tx-sigs(int-port,int-port, newemptyport);
        IF was-empty THEN [rec-set@int-span]:=[rec-set@int-span]+int-port
        Sel-broadcast(int-port);
        involved:=true;
update-repeat:
        for every span IN [SPANS] NOT int-span DO
        {reset[send-set@span];
        repeat
                scanport:=next(send-set@span)
        until (PS.Assoc-port@scanport = int-port or scanport = nil);
        IF scanport <> nil THEN TS.repeat@scanport:=RS.repeat@int-port+1}
non-repeatable-sig:
        IF (was-precursor AND (was-index <> is-index) THEN
        cancel-tx-sigs(int-port,int-port, newemptyport);
        IF was-empty THEN [rec-set@int-span]:=[rec-set@int-span]+int-port
        involved:=true;
overrange-sig :
        test-for-involved
end /*case*/
/* End of TandemFSM */
```

APPENDIX C: MEASURED EXECUTION TIMES FOR 'C' LANGUAGE STATEMENTS (SUN 3/60)

Category Name	Example	Execution Time (μs)	Time Averaged Over(s)
z_if	if (better) { }	0.711	71.08
s_if1	if (index < 0) { }	0.705	70.50
s_if2	if (index < CONST) { }	1.028	102.80
d_if	if (index = = int_link) { }	1.360	13.44
z_ass	better = TRUE;	0.557	55.72
z_ass2	index = 0;	0.437	436.98
z_ass3	index = CONST;	0.862	861.88
s_ass	index = int_link;	0.973	97.30
s_ass2	n_spans = newestspan + 1;	1.356	135.60
z_p_if	if (tempport != NULL) { }	0.760	7.60
s_p_if	if (index = = tempport->linkid) { }	1.734	173.58
z_p_ass	tempport = NULL;	0.624	62.44
s_p_ass	index = tempport->linkid;	1.355	135.48
d_p_ass	tempport = searchport;	1.085	216.90
z_pc_ass	tempport->last = NULL;	1.170	117.02
d_pc_ass	tempport = tempport->next;	1.622	162.16
d_pcc_ass	tempport->last->next = tempport->next;	2.649	264.92
z_s_if	if (! RS[int_link].rsdelta) { }	2.378	237.81
z_s_if2	if (RS[int_link].targetnode = = NONE) { }	2.664	266.44
s_s_if	if (TS[index].assoc_port = = index) { }	3.574	257.44
d_s_if	if (RS[index].indexcount = = RS[*portno].indexcount) { }	4.862	470.56
z_s_ass	RS[index].indexcount = NONE;	3.047	304.72
z_s_ass2	TS[index].repeatcount = 1 - MAXREPEATS;	3.216	321.60
s_s_ass	TS[index].assoc_port = int_link;	3.333	333.34

d_s_ass	TS[index].repeatcount = RS[int_link].repeatcount - 1;
	TS[index].indexcount = RS[int_link].indexcount;
	TS[index].sourcenode = RS[int_link].sourcenode;
	TS[index].targetnode = RS[int_link].targetnode;
		4.934	493.40

z_pa_if	if (fempty[i] != NULL) { }	1.193	119.28
s_pa_ass	tempport = ffullin[spannum];	2.376	23.76
s_as_if	if (spannum != SPANMAP[RS[index].nid]) { }	4.071	407.12
s_as_ass	spannum = SPANMAP[RS[index].nid];	3.805	380.46
z_a_ass	affected_ports[int_link] = 0;	1.733	163.34
d_asi_ass	affected_ports[++int_link] = TS[index].assoc_port;	5.329	532.94
s_a_ass	affected_ports[index] = int_link;	2.130	213.04
s_a_if	if (affected_port[int_link] < CONST) { }	2.322	232.18
pcall0	check_and_tidy();	2.658	265.76
pcall1	bettersig(&int_link);	3.560	2394.76
pcall3	cancel_tx_sigs_from(&int_link,&int_link,&newemptyport);	4.870	736.06
pcall4	llins(&fempty,&lempty,&tempport,&index);	6.124	61.24
pcall5	lldel(&fempty,&lempty,&tempport,&index,&error);	6.698	66.98
enum_ass	event = repeatable_sig;	0.437	437.16
sw	switch (event) { }	4.050	809.96
inc	i++;	0.659	65.92
ainc	affected_ports[int_link]++;	1.953	195.34
loop	for (;FALSE;) { }	0.375	17.52