DESIGN AND ANALYSIS OF BREADTH FIRST ONEPRONG

by

Anders Hansmats

A thesis submitted to the Faculty of the Graduate School of the University of Colorado in partial fulfillment of requirements for the degree of

Master of Science

University of Colorado

1997

This Thesis for the Master of Science degree by

Anders Hansmats

has been approved for the

Department of Computer Science

by

_____

Advisor C. Edward Chow

_____

Jugal K. Kalita

_____

Sudhansu K. Semwal

Date _____

University of Colorado

Abstract

DESIGN AND ANALYSIS OF BREADTH FIRST ONEPRONG

by Anders Hansmats

Thesis directed by: Professor C. Edward Chow
Department of Computer Science

Abstract.

Broadband optical networks require fast restoration from single-link failures, node failures and multiple-link failures. This report describes a distributed restoration algorithm based on the one prong algorithm. It uses a breadth first search approach and a time-out mechanism to explore restoration paths with shorter hop count first to reduce spare usage. It can handle single-link failures, node failures, multiple link failures and area failures.

TABLE OF CONTENTS

LIST OF FIGURES iv
LIST OF TABLES v
NETWORK RESTORATION 1
Centralized / Distributed restoration 1
Link based/ Path based approach 2
Definitions of network model 3
Performance Metrics for Network Restoration Algorithms 4
Functional Characteristics of Network Restoration Algorithms 7
ONEPRONG 10
SEARCHING 12
Regular Breadth first search 12
Regular Depth First Search 14
Iterative deepening 15
Iterative broadening 17
Heuristic search techniques 18
Searching from One Prong view. 18
DEPTH FIRST ONEPRONG WITH TIMER (DFOP) 20
Notification Phase 20
Path Trace-out Phase 21
Path Confirmation Phase 23
Clean Up Phase 23
BREADTH FIRST ONEPRONG WITH TIMER (BFOP) 24
Notification Phase 24
Path Trace-out Phase 25
Path Confirmation Phase 27
Clean Up Phase 28
DESIGN PLAN 29
Project Plan 29
INTERNAL DESIGN. 35
Event queue 36
Eventnode data structure 37
Eventrecord datastructure 38
Symbol node data structure 38
Node data structure 39
Table record datastructure 42
Confirm data structure 43
Message history data structure 44
Pack record data structure 44
Internal node data structure 45
List data structure 45

Inode datastructure 46
Linkdata datastructure 46
Timer record data structure 47
Path_ack_info datastructure 47
Up_info datastructure 48
Down_info datastructure 49
Program structure 50
ANALYSIS USING FUNCTIONAL CHARACTERISTICS 57
Path finding 57
Spare Channel Contention 58
Restoration Path Selection 59
Message volume control 59
Congestion control 62
Race Condition Control. 62
Restored Path Connection 63
PERFORMANCE ANALYSIS USING PERFORMANCE METRICS 64
Restoration level 64
Time to restoration 66
Spare channel utilization 67
Range of application 68
Message volume 69
Restoration level in 2 seconds 69
Conclusion and observations of performance analysis 69
OBSERVATIONS DONE DURING THIS PROJECT. 72
Reading/ learning existing one prong code 72
Porting one prong from UNIX GNU GCC to Win32 MSVC++ 73
Why Win32 74
Time needed for this project 75
Working abroad 76
BIBLIOGRAPHY 77
APPENDIX A 79
Input options 79
APPENDIX B 80
Output interpretation 80
APPENDIX C, PLOTS 82
Plot 1 82
Plot 2 84
Plot 3 86
Plot 4 88
Plot 5 90
APPENDIX D 92
Accumulated Bandwidth Problem 92

APPENDIX E, SIMULATION RESULTS 97
ONEP NJ3.NET 98
DFOP NJ3.NET 99
BFOP NJ3.NET 100
ONEP LATA3.NET 101
DFOP LATA3.NET 102
BFOP LATA3.NET 103
ONEP D_31.NET 104
DFOP D_31.NET 106
BFOP D_31.NET 108
ONEP DIS5_63.NET 111
DFOP DIS5_64.NET 115
BFOP DIS5_64.NET 119

LIST OF FIGURES

Number *Page*
Figure 1 Link Restoration 2
Figure 2 Path Restoration 3
Figure 3 Restoration level 5
Figure 4 Breadth First Search 13
Figure 5 Depth First Search 14
Figure 6 Iterative deepening 15
Figure 7 Iterative broadening 17
Figure 8 DFOP Notification phase 21
Figure 9 DFOP bw rejecting 22
Figure 10 BFOP Notification Phase 25
Figure 11 BFOP traceout information 26
Figure 12 Time Plan 32
Figure 13 Event queue 37
Figure 14 Node Data Structure 40
Figure 15 Program structure 50
Figure 16 Handling of CONF messages 55
Figure 17 DFOP PACK forwarding 57
Figure 18 BFOP PACK forwarding 58
Figure 19 Distribution of ACK/CONF messages 70
Figure 20 Plot of 25-6cut 82
Figure 21 Plot of 44-28cut 84
Figure 22 Plot of 44-28cut, zoom 85
Figure 23 Plot of 9-7cut 86
Figure 24 Plot of N00-N03cut, different delays 88
Figure 25 Plot of N00-N03cut 89
Figure 26 Plot of 14cut, different TTL 90
Figure 27 Accumulated Bandwidth Problem (1) 93
Figure 28 Accumulated Bandwidth Problem (2) 93
Figure 29 Accumulated Bandwidth Problem (3) 94
Figure 30 Accumulated Bandwidth Problem (4) 94
Figure 31 Accumulated Bandwidth Problem (5) 95
Figure 32 Accumulated Bandwidth Problem (6) 95

LIST OF TABLES

Table 1 BFOP Message Volume 61
Table 2 ONEP NJ3.NET 98
Table 3 DFOP NJ3.NET 99
Table 4 BFOP NJ3.NET 100
Table 5 ONEP LATA3.NET 101
Table 6 DFOP LATA3.NET 102
Table 7 BFOP LATA3.NET 103
Table 8 ONEP D_31.NET 105
Table 9 DFOP D_31.NET 107
Table 10 BFOP D_31.NET 110
Table 11 ONEP DIS5_64.NET 114
Table 12 DFOP DIS5_64.NET 118
Table 13 BFOP DIS5_64.NET 125

Chapter 1

NETWORK RESTORATION

There is an increasing dependency on today's communication networks, user demands for high-speed and economical communications service lead to the rapid deployment of high-capacity optical fibers in the network. The technological change to optical fiber based networks coincides with a time when telecommunications is a major part of a country's infrastructure. Business, medicine, air traffic control and reservations, and the governmental offices are among those dependent on telecommunications integrity. These demands for high-reliability services raise a network survivability problem. There have been many algorithms developed to restore networks [5] ,among the strategies there are centralized DCS-based network restoration and distributed DCS-based network restoration.

Centralized / Distributed restoration

In centralized network restoration a node is dedicated to be the Network Operation Center (NOC) and performs the diagnostics and restoration functions, it requires reliable links between the DCS nodes and the network restoration center. The NOC has knowledge of the topology of the entire network and uses this information along with messages that it receives from the nodes in the network to make diagnostics and set up alternate paths.

Distributed network restoration uses a simple but efficient flood search to quickly and automatically identify alternative paths. There is no need for network databases or control software as the network acts as its own database. Each node knows only its own unique identity and a set of rules that tell how to react when it sees an alarm or a message from a neighbor node.

It is usually the case that the distributed approach performs better on link failures and the centralized approach is better on node failures [5].

Link based/ Path based approach

There are two basic approaches to reroute the disrupted traffic due to a fiber cut. The first one is called Link restoration and it approaches the problem by replacing the affected link segment of a disrupted channel by a spare path between the two disrupted nodes. The restoration is initiated by nodes that are adjacent to the disrupted area.

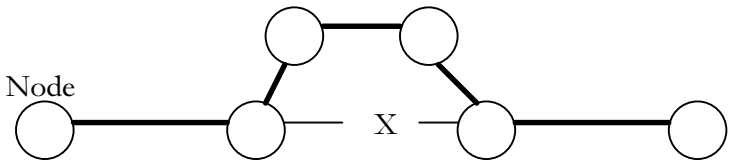


Figure 1 Link Restoration

The other approach is called Path restoration and it releases each disrupted channel and lets the source and destination nodes handle the establishment of the connection. Since it takes some time for the messages to propagate back to the end nodes of the path, the initiation will be started later, compared to the link based approach.

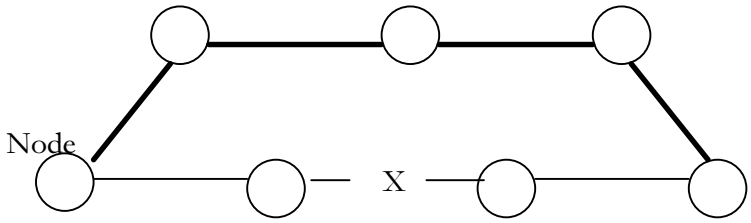


Figure 2 Path Restoration

Definitions of network model

To do analysis of network restoration algorithms C. Edward Chow defines the network restoration model like this in [5].

A network is defined as a system of switching nodes connected by communication lines, and can be represented as an augmented undirected graph with a set of nodes and a set of links. A link connects two nodes in the network and has an associated bandwidth. The bandwidth of a link is divided into basic units called channels. Each channel is in one of two states: working or spare. Each link in the network is labeled with two numbers, representing the number of working and spare channels in the link. A route is specified as an ordered set of concatenated channel Ids. The hop count of a route is the number of links of the

route. A path is specified by an ordered set of concatenated channel Ids. The hop count of a path is the number of channels in the path. A working path is a path where all channels are working channels while a spare path is a path where all channels are spare channels. A restoration path is a spare path that is designated for restoring a disrupted working channel due to network failure.

Performance Metrics for Network Restoration Algorithms

There are five performance metrics for evaluating Network Restoration algorithms described by John Bicknell, C.Edward Chow and Sami Syed in [8] .

1. Time to restoration
2. Level of restoration
3. Utilization of spare cannel resources
4. Range of application
5. Message volume

The first performance metric, time to restoration, refers to the time required by the algorithm to complete execution to whatever level of restoration it can achieve. This is an extremely important metric, since it is desirable to accomplish restoration as fast as possible to avoid call dropping. In the ideal case, an algorithm must achieve full possible restoration in less than two seconds.

The second performance metric, level of restoration, refers to how many of the lost working channels are restored. The ideal case is that all lost channels are restored. This may not always be possible, three situations can occur which limits the restoration. First, there may be not sufficient spare capacity in the network to support restoration of the lost working channels, even with an optimal algorithm. Second, there could be considerable spare capacity within the network, overall,

but it is distributed in a way that restoration cannot be achieved in a specific failure scenario. Third, even when complete restoration is possible, certain network topologies can create situations which are pitfalls for distributed algorithms using a heuristic approach to path finding or restoration path selection.

Performance metrics 1 and 2, when combined, are the most critical performance criteria for any network restoration algorithm. The ideal solution is a 100% restoration within two seconds. In situations in which an algorithm cannot accomplish full restoration within the two second limit, the rate at which it restores lost channels can be of importance.

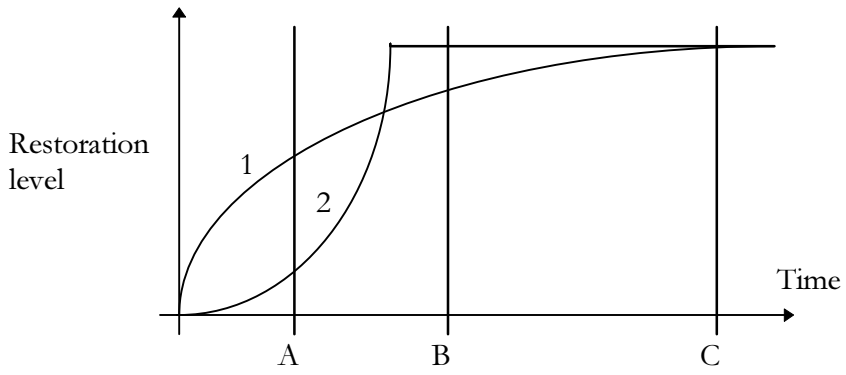


Figure 3 Restoration level

The above figure illustrates this. The two curves represent the rate at which two network restoration algorithms, 1 and 2, achieve increasing levels of restoration. There are three time marks shown, A, B and C. If two seconds of elapsed time occur at time mark C, then both algorithms have restored 100%. If two seconds of elapsed time occur at time mark A, then algorithm 1 is superior to

algorithm 2, since it has achieved a higher level of restoration. If, however, two seconds of elapsed time occurs at time mark B, algorithm 2 is superior to algorithm 1.

Performance metric 3, utilization of spare channel resources, refers to how many spare channels that are switched to working channels to replace the lost channels. Since bandwidth is a limited resource within a network, it is desirable that as few spare channels as possible is used in the restoration solution.

Performance metric 4, range of application, refers to what different kinds of failure scenarios the algorithm can be applied to affect restoration. A number of distributed algorithms can only address single link failures. A limited number of algorithms can be used to restore lost working channels in multiple link and node failure scenarios.

Performance metric 5, message volume, refers to how many network restoration messages that are generated by a restoration algorithm. It is desirable that the number of messages an algorithm generates should be as few as possible. Not only does message volume affect performance metric 1, it also limits other network message traffic flow during the restoration process which may also be of high or critical priority.

Functional Characteristics of Network Restoration Algorithms

The following seven functional characteristics which can be attributed to distributed network restoration is presented by C. Edward Chow in [5].

1. Find paths
2. Resolve spare channel contention
3. Select restoration paths
4. Control message volume
5. Control congestion
6. Counter race conditions
7. Connect restored paths

Functional characteristic 1, find path, relates to how an algorithm identifies possible restoration paths. Most distributed algorithms use some form of flooding to accomplish this. There are two approaches in finding paths. One is to limit, through some mechanism, path finding to a subset of all paths in the network, the other approach is to perform a tracing of all paths in the network.

Functional characteristic 2, Resolve spare channel contention , is the situation when there are multiple request for the same set of spare channels.

Since all the candidate restoration paths in a restoration solution usually are not disjoint paths, there is some competition among these paths for available spare channels. At some point in the execution of these distributed algorithms there exists a possibility that the same spare channel have more than one reservation or attempted reservation made to use it in the restoration solution.

For example, there are two links A and B with spares available. One request reserved all spares in Link A and continue to reserve spares in Link B. The other request reserved all spares in Link B and continue to reserve spares in Link A. Both requests run into dead end and either need to back off and release all the reserved spares and try again. A priority scheme can be used and have high priority request take over the spares reserved by the lower priority request.

Functional characteristic 3, select restoration paths, refers to the process the algorithm uses to select from among the identified candidate paths, which are to be used in the restoration process. Most of the distributed algorithms use a "first come, first serve" kind of approach .

Functional characteristic 4, control message volume, refers to what mechanisms the algorithm uses to reduce the number of messages produced during execution of the algorithm. It is desirable that an algorithm generates as few messages as possible, since this has an serious impact upon the time to restore metric and affects the networks capacity to process other time critical network restoration messages not involved in the restoration algorithm.

Functional characteristic 5, control congestion, refers to what techniques an algorithm uses to control message congestion at critical nodes. Critical nodes are nodes which, because of their location within the network, will process a larger number of messages than other nodes in the network.

Functional characteristic 6, counter race conditions, refers to how an algorithm control or respond to race condition during execution of the restoration algorithm. A network topology affects the speed with which messages propagate through a network. The timely arrival of a message or the relative arrival time of messages to a certain node may affect the performance of an algorithm. For example , if I use a random function for the message processing delay, and I modify it slightly, the message arrival patterns can change dramatically, and the outcome of the network restoration results are affected. The algorithm is subjected to the race condition.

Functional characteristic 7, connect restored paths, refers to the method used by the algorithm to make final restoration path connections. Most algorithms use a three phased approach: identify a path between the two end nodes, an acknowledgment is sent from one node to the other, the second node then sends a confirmation that the path has been connected.

Chapter 2

ONEPRONG

In the paper [6] C.Edward Chow, Jay Shah and Dave Allen describe the Adaptive One Prong algorithm. It is a distributed algorithm and it is based on an approach whereby aggregated bandwidth restoration requests are issued from one of the end nodes of the disrupted connections, the total bandwidth request for each disrupted connection is kept to be equal or under that of the lost bandwidth to allow more disrupted paths to be concurrently restored, and an intelligent retry logic is used to handle racing conditions and increase the restoration level.

The adaptive one prong lets the source node of a disrupted connection broadcast restoration messages which explore the potential paths between the two end nodes of the disrupted path. When the destination node receives the restoration request message, it recognizes that there is a potential restoration path and sends back an ACK message to reserve bandwidth. The bandwidth reserved will be equal or less than the lost bandwidth of the disrupted path.

The traditional approach aggressively reserves bandwidth on all outgoing branch with each branch up to that of the lost bandwidth during their initial restoration requests. The adaptive one prong on the other hand defers the bandwidth reservation until the destination receives a request message which indicates the hop count and therefore it has additional topology information which indicates the potential restoration paths. The adaptive one prong also apply a conservative approach whereby the sum of the reserved bandwidth on all outgoing links of a node will not exceed the bandwidth in the ACK messages it received or the available spare bandwidth.

Since multiple disrupted paths are reserving network spare bandwidth simultaneously, it is possible for a node to find that there are no more available spare on its outgoing links for a disrupted path. Some of the reserved bandwidth on those outgoing links may belong to the restoration process of the disrupted path that eventually leads to a dead end and will than be released. The adaptive one prong implements an retry mechanism which re-explore each of the outgoing links, it selects first a link with shorter hop count to the source node, and decreases a link's selection probability, if the link frequently leads to a dead end.

Chapter 3

SEARCHING

I looked at several different search methods that could perform the actual search when the destination node got its PACK message and is supposed to initiate its search[10] [12]. But since the search is not a regular "search" in the sense that the node that is supposed to initiate the search already got the paths to the destination, the search are only supposed to come up with a "path" from source to destination that can support the bandwidth that is needed. Note, the "path" can consist of one or more sub paths, for example, if the total bandwidth lost are 20 and sub path A can support 15bw and sub path B can support 5bw, the "path" from source to destination consists of sub path A and sub path B. The following descriptions of search methods are based on [12].

Regular Breadth first search

The breadth first search could be viewed as a queue, so that newly generated children are put on the end of the list and expanded only after nodes at shallower depths have been examined. The complete ordering for the search is shown below in figure 4.

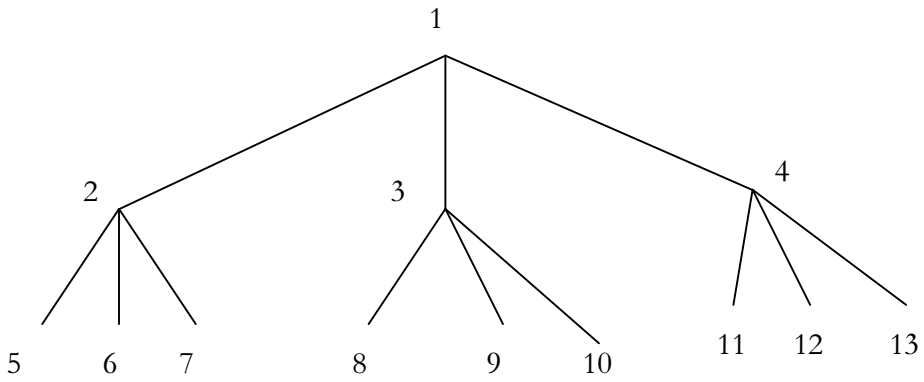


Figure 4 Breadth First Search

If the branching factor is assumed to be B, the total number of nodes at depth K would be B^K and all these have to be stored. To explore a tree like the above we need a search space of at least B^{K-1}. What about the amount of time needed, if we assume that it takes one unit of time to examine one node we would get the following. In order to reach a goal at depth K, the internal (non fringe) nodes that must be examined are all of the nodes at depths 0,1 ,2 ,....K-1 so we get

$$1 + B^1 + B^2 + \ldots\ldots B^{K-1} = \frac{B^K - 1}{B - 1}$$

Since we do not know how many fringe nodes we have to examine before we reach the goal (min 1 and max. B^K) we use the average = $(1 + B^K)/2$.

So the total amount of time

$$\frac{B^K - 1}{B-1} + \frac{B^K + 1}{2} = \frac{B^{K+1} + B^K + B - 3}{2B - 2}$$

Regular Depth First Search

This is the search technique used by the original one prong.

Depth first search the tree in the order described by the figure below.

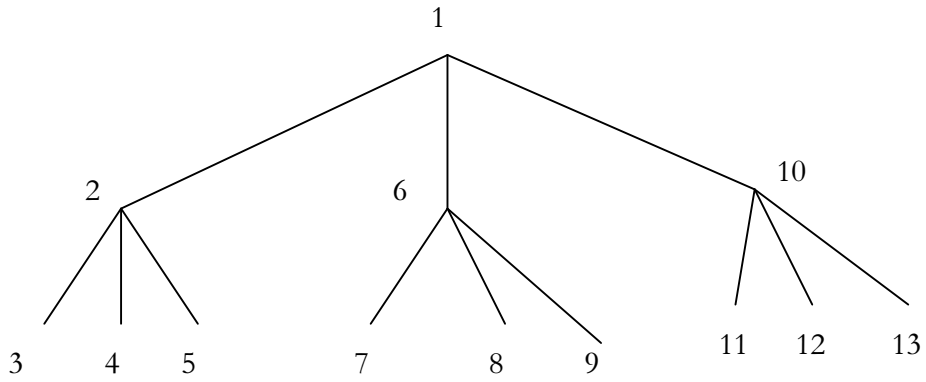


Figure 5 Depth First Search

The time needed can be calculated like this:

If the goal is at the far left of the tree, then depth-first search will proceed directly to it, examining a total of K + 1 nodes. If it is to the far right of the tree, the entire space will be examined, a total of

$$1 + 2 + \ldots . B^K = \frac{B^{K+1} - 1}{B-1}$$

nodes in all. If we average these two expressions, we could conclude that the number of nodes examined in the average case is

$$\frac{1}{2}\left[\frac{B^{K+1}-1}{B-1}+(K+1)\right]=\frac{B^{K+1}-1+(B-1)(K+1)}{2(B-1)}=\frac{B^{K+1}+BK+B-K-2}{2(B-1)}$$

Iterative deepening

The idea of iterative deepening is to search the tree initially with an artificial depth cutoff of 1, so that any node below depth is not examined. If this approach succeeds in finding a solution at depth 1, the solution is returned. If not, the tree is searched again with a depth cutoff of 2. Each of these iterative searches proceeds in depth first fashion. The memory space needed is the same as for regular depth first search , we need to store B-1 nodes at each depth. The search order is shown below.

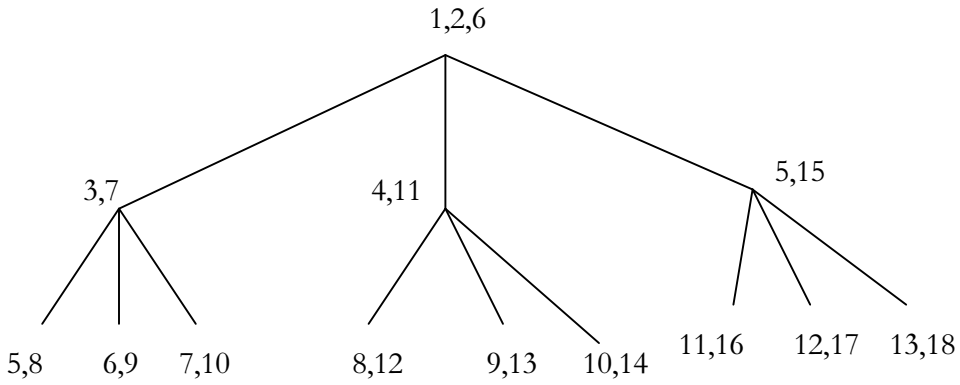


Figure 6 Iterative deepening

The time to search can be divided into the successful iteration and previous iterations.

The time needed for the successful iterations is given by

$$\frac{B^{K+1} + BK + B - K - 2}{2B - 2}$$

If the goal is at the far left of the tree, then we have to examine K+1 nodes. If it is to the far right of the tree, the entire space will be searched.

$$1+....B^{K} = \frac{B^{K+1} - 1}{B - 1}$$

So if we take the average of these we get

$$\frac{B^{K+1} + BK + B - K - 2}{2B - 2}$$

The time needed for the previous iterations to depths 1,2,3..K-1 will need to examine the entire tree at these depths and the size of the tree at depth d is given by:

$$1 + B^{1} + B^{2} + ...B^{d} = \frac{B^{d+1} - 1}{B - 1}$$

So the total time for the previous iterations is therefor

$$\sum_{j=0}^{j=K-1} \frac{B^{j+1}-1}{B-1} = \frac{1}{B-1}\left[B\left(\sum_{0}^{K-1} B^{j}\right) - \sum_{0}^{K-1} 1\right] = \frac{1}{B-1}\left[B\left(\frac{B^{K}-1}{B-1}\right) - K\right]$$

$$= \frac{B^{K+1} - BK - B + K}{(B-1)^{2}}$$

Combining this with the above we get:

$$\frac{B^{K+2} + B^{K+1} + B^{2}K + B^{2} - 4BK - 5B + 3K + 2}{2(B-1)^{2}}$$

Iterative broadening

Just as iterative deepening imposes artificial depth limits on the search and gradually increases those limits until a solution is found, iterative broadening imposes artificial breadth limits, increasing them until a solution is found. The figure below shows a search tree with cutoff = 2.

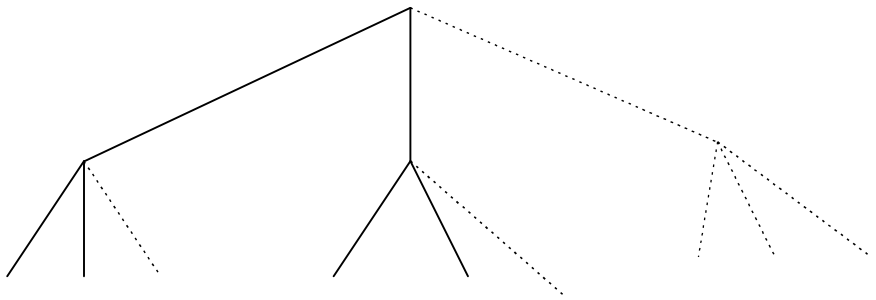


Figure 7 Iterative broadening

The maximum amount of time spent by the approach is approximately

$1+2^K+3^K+4^K+.....B^K$

which is approximately $\frac{B^{K+1}}{K}=\frac{B}{K}B^K$ for large B, and approximately B^K for large K.

Heuristic search techniques

In exploring search spaces the trick is to use additional information about the problem being considered to avoid examining most of the nodes that conceivably lead to solutions. Instead of selecting the next node to expand in domain-independent way, we need to use domain-specific information when we do the selection. What we usually do is to apply some sort of "rule of thumb" or heuristic to decide what to do next. This heuristic can be simple or very complex depending on what you would like to achieve. In [10] there are lots of heuristic search techniques mentioned and the use of a heuristic approach can open new ways to finding fast restoration algorithms.

Searching from One Prong view.

By the use of the timer function and the fact that the entire search space is included in the knowledge of the finish node, makes the search space irrelevant. When the timer expires the finish node has got information about all the paths the received PACK messages has traversed through. Based on this information and the information the finish node has about its neighbor nodes, the finish node needs to make a decision where it is going to send the ACK messages. All paths

lead to a goal and the only constraint is the bandwidth available on each link in the path. The factor of importance here is the time needed to find path/paths that can support the bandwidth of the disrupted path. This makes the iterative deepening and the iterative broadening a worse choice than the breadth first search. This leaves us with the heuristic search methods as an alternative to breadth first search. But since we are using the hop count to decide which path to try, we are actually using a simple heuristic. Using a complex heuristic search could perhaps make a good network restoration algorithm, but could also result in a very complex simulator. Even a rather simple heuristic search technique like the Best-First search [10] could become very complex in a network restoration environment. Since I want to minimize spare usage, breadth first search with the simple hop count heuristic is the natural step from the depth first approach used be the original one prong.

Chapter 4

DEPTH FIRST ONEPRONG WITH TIMER (DFOP)

The Depth First one prong with timer (DFOP) algorithm is based on the path based one prong distributed algorithm presented by C. Edward Chow, Jay Shah and Dave Allen in [6] . The one prong was built based on the reuse of Two Prong[7] software and implemented with a simplified logic, it uses a depth first approach to find restoration paths. It uses a three way message exchange, aggregated bandwidth request and a conservative bandwidth reservation for more concurrent path restoration.

DFOP is basically a one prong equipped with a timer mechanism to allow potential PACK messages to arrive before ACK is sent and a restoration path selection based on shorter hop count. The DFOP has got four phases: Notification Phase, Path Trace out Phase, Path Confirmation Phase, and Clean UP phase.

Notification Phase

On detecting a network failure, the nodes on each side of the failure forwards FAIL messages along the paths to the end nodes releasing the working channels of the disrupted paths. When the start node receives the fail message it broadcasts the network with PACK messages. Note, this is not a full broadcast, a tandem node will only broadcast PACK messages to downstream nodes for a certain up node once. This will reduce the size of the tables and the message volume. To reduce the number of messages and the protocol processing time, one PACK message can carry information about several disrupted paths.

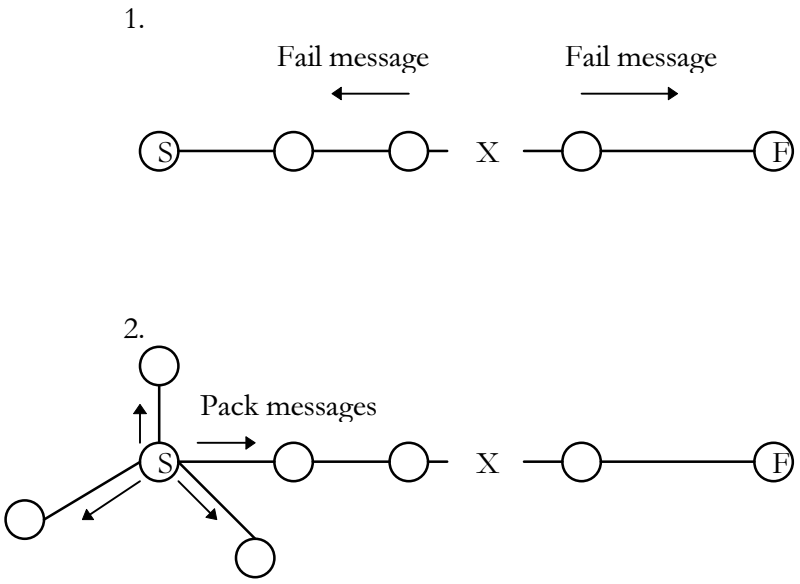


Figure 8 DFOP Notification phase

Path Trace-out Phase

On receiving the first PACK message for a path the finish node initiates the timer with a certain delay decided. The delay can be constant or random within up and down limits. If constant delay is used every finish node will wait a certain amount of time, decided by the constant timer delay switch at the command line, before it sends its first ACK. If random delay is used every finish node will wait a random amount of time within the down limit and up limit. When the timer expires after a certain amount of time, the finish node will search the whole table and find the entry with the smallest number of hop and send an ACK to the corresponding upstream node. The ACK message carries an ACKHOP field that indicates the hop count of its traversed path. When the sum of the ACKHOP count and the HOP count field in the table, which was created by the PACK message, exceeds the hop count limit set at the command line, the ACK gets

rejected. On receiving an ACK message, the tandem node updates the table indicating the path bandwidth now reserved and forwards the ACK message upstream. Note, that an ACK message can be split into several ACK messages with smaller bandwidth request. If a node can not handle the whole requested bandwidth or just a part of the requested bandwidth in an ACK message, a CONF/CONFX message is sent back to the downstream node. A CONF(bw=0) is sent if none of the requested bandwidth could be handled, a CONFX(bw=x) with the rejected bandwidth is sent if part of the request could be handled. When the downstream node receives the reject message sent by the upstream node, it will search its table to find another up node to fulfill the request. Which up node, is decided by looking at the hop count field in the table and pick the entry with minimum hop count.

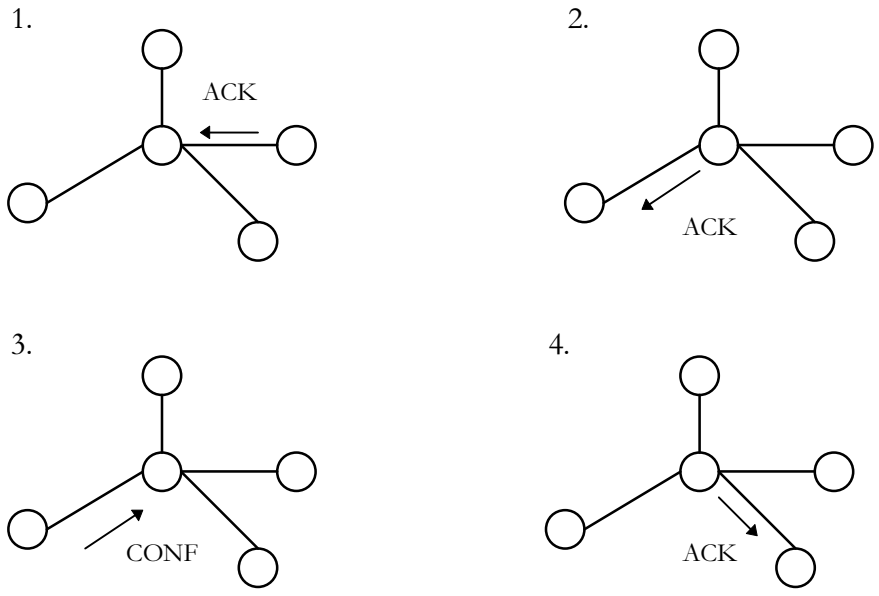


Figure 9 DFOP bw rejecting

When an ACK is rejected the node increases the retry number of the link by one, this is done because an ACK message is only allowed to be sent along a link a certain number of times.

Path Confirmation Phase

When the start node receives an ACK message it sends a CONF message along the restoration path and confirms the establishment of the restoration path. If all the bandwidth of the path is restored, a CONFF message is sent. On receiving a CONFF message, the tandem node will release the bandwidth that is not on the confirmed path and indicate in its table that the path has been fully restored. The exchange of CONFF messages is a mechanism to allow end nodes to inform tandem nodes to stop the bandwidth restoration process.

Clean Up Phase

When all bandwidth of a path is restored, a PDONE message is flooded out to speed up the release of unneeded bandwidth.

Chapter 5

BREADTH FIRST ONEPRONG WITH TIMER (BFOP)

The Breadth First one prong with timer (BFOP) algorithm is based on Depth First one prong with timer described above.

BFOP is basically a DFOP with a breadth first search approach to get a better spare usage and explore the restoration paths with shortest hop count first. The BFOP has got the same four phases as DFOP: Notification Phase, Path Trace out Phase, Path Confirmation Phase, and Clean UP phase.

Notification Phase

On detecting a network failure, the nodes on each side of the failure forwards FAIL messages along the paths to the end nodes releasing the working channels of the disrupted paths. When the start node receives the fail message it broadcasts the network with PACK messages. BFOP uses a full broadcast to explore every possible path. To reduce the number of messages and the protocol processing time, one PACK message can carry information about several disrupted paths. All PACK messages carry a trail field to record the path the PACK message traverse through. This packpath[1] will then be used by the finish node.

[1] The path the PACK message traversed through from start node to finish node

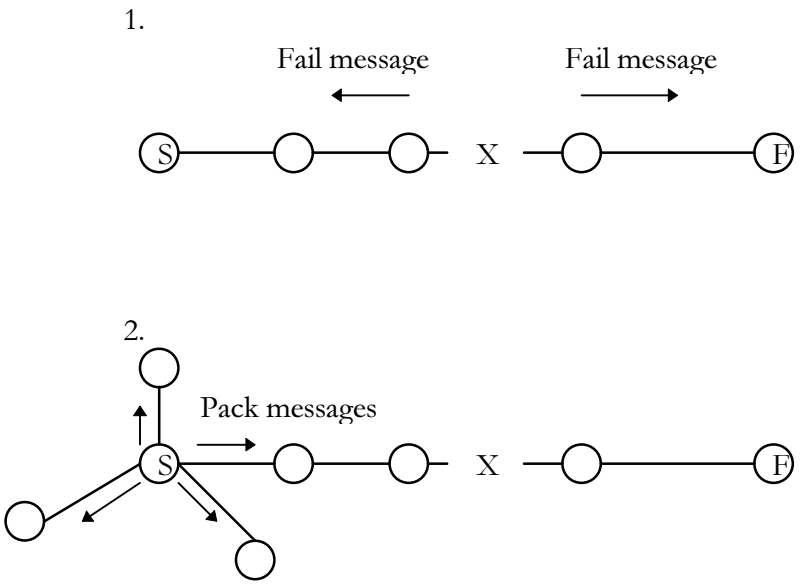


Figure 10 BFOP Notification Phase

Path Trace-out Phase

On receiving the first PACK message for a path the finish node initiates the timer with a certain delay. The delay can be constant or random within up and down limits. If constant delay is used every finish node will wait a certain amount of time, decided by the constant timer delay switch at the command line, before it sends its first ACK. If random delay is used every finish node will wait a random amount of time within the down limit and up limit. When the timer expires after a certain amount of time, the finish node will search the whole table and find the entry with the smallest number of hop and send an ACK along the corresponding packpath. The information the finish node has got when the timer expires can be viewed like this:

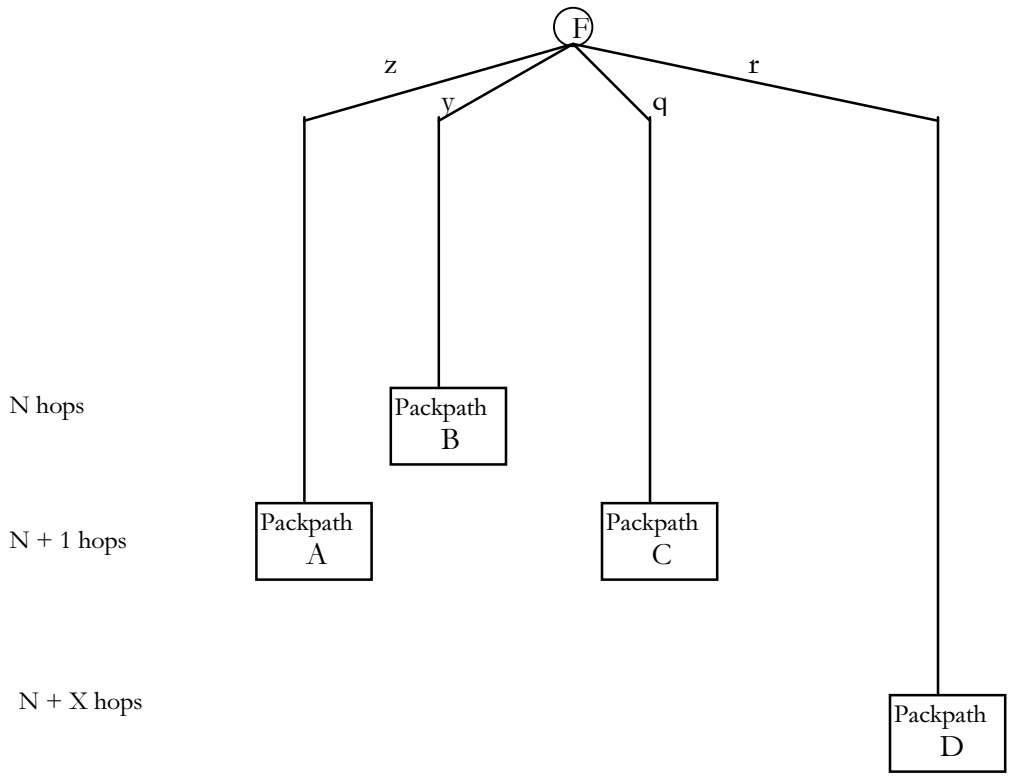


Figure 11 BFOP traceout information

The z, y, q, r represents the number of spares available on the first link of the packpath.

There exists several packpath with different number of hops and the finish node will send an ACK along the packpath with the minimum number of hops. Note, a restoration request for a path can be split into several ACK messages and sent along different packpaths if there is not enough spares available on the first link. The ACK message will then follow this packpath to the start node. The ACK message carries an ACKHOP field that indicates the hop count of its

traversed path. When the sum of the ACKHOP count and the HOP count field in the table, which was created by the PACK message, exceeds the hop count limit set at the command line, the ACK gets rejected. On receiving an ACK message, the tandem node updates the table indicating the path bandwidth now reserved and forwards the ACK message to the upstream node in the packpath.. Note that an ACK message can not be split into several ACK messages with smaller bandwidth request at a TANDEM node as in the DFOP, the only node that does the split is the finish node. If a node can not handle the whole requested bandwidth or a part of the requested bandwidth in an ACK message, a CONF/CONFX message is sent back to the downstream node. A CONF(bw=0) is sent if none of the requested bandwidth could be handled, a CONFX(bw=x) with the rejected bandwidth is sent if part of the request could be handled. When the downstream node receives the reject message sent by the upstream node, it will release the protected bandwidth and forward the CONF/CONFX message along the path the ACK message came back to the finish node. This means that much of the intelligence has been moved from the tandem nodes to the finish node. When the finish node receives a CONF/CONFX message it will search its table for another packpath based on the hop count and try that one instead. The retry number is not connected to a certain link like in DFOP, the retry number is connected to a certain packpath. An ACK message is only allowed to be sent along a certain packpath a certain number of times.

Path Confirmation Phase

When the start node receives an ACK message it sends a CONF message along the restoration path and confirms the establishment of the restoration path. If all the bandwidth of the path is restored, a CONFF message is sent. On receiving a CONFF message, the tandem node will release the bandwidth that is

not on the confirmed path and indicate in its table that the path has been fully restored. The exchange of CONFF messages is a mechanism to allow end nodes to inform tandem nodes to stop the bandwidth restoration process.

Clean Up Phase

When all bandwidth of a path is restored, a PDONE message is flooded out to speed up the release of unneeded bandwidth.

Chapter 6

DESIGN PLAN

I have decided to treat my Thesis as a programming project with a contract between me and my Thesis committee. This contract is an agreement between me and my Thesis committee specifying that I will do a certain job within specific constraints to get my final degree. In this case the Thesis Proposal is used as the contract, defining what I am supposed to do. I have decided to do my Thesis this way because it will refresh the phases in managing a project[9] and also give my Thesis committee better knowledge of what I have done. Since I am the only project member, all decisions are made my me, or if necessary in understanding with my major advisor.

Project Plan

The project plan describes how the project will be carried out, and it has got the following sections.

Phase Plan

The project has been divided into the following phases. The plan does not have to be followed in sequential order, phases can be carried out in parallel.

- Collect background information
- Understand / Learn "base" code
- Port "base" code from UNIX to WindowsNT.

- Design modified Depth First One prong.
- Implement modified Depth First One Prong.
- Design Breadth First One prong.
- Implement Breadth First One prong.
- Test the two simulators.
- Do analysis/performance testing on modified Depth First One Prong.
- Do analysis /performance testing on Breadth First One Prong.
- Design GUI
- Implement GUI
- Write report.

Collect background information. During this phase I will collect articles, books and other information needed to do this project, this phase also include getting the necessary software.

Understand / Learn "base" code. During this phase I am going to learn the "base" code, how the code is designed, what data structures are used and so on.

Port "base" code from UNIX to Win32. The "base" code is written for UNIX and compiled using GNU gcc and since I am going to work on a different platform using a different environment the code needs to be ported from UNIX, GNU gcc to WindowsNT, Microsoft Visual C++ , this is done during this phase.

Design modified Depth First Oneprong. This is the design phase for the modified Depth First One Prong.

Implement modified Depth First OneProng. This is the programming phase for Depth First One Prong.

Design Breadth First OneProng. This is the design phase for Breadth First One Prong.

Implement Breadth First OneProng. This is the programming phase for Breadth First One Prong.

Test the two simulators. This is the test phase for the two simulators. During this phase the two simulators will be tested and errors will be corrected.

Do analysis and performance testing on modified Depth First OneProng. During this phase the performance of the Depth First One prong will be measured and compared against the original one prong.

Do analysis and performance testing on Breadth First OneProng. During this phase the performance of the Breadth First One Prong will be measured and compared against the original one prong.

Design GUI. This is the design phase for the graphical user interface.

Implement GUI. This is the programming phase for the graphical user interface.

Write report. This is the final phase which includes writing the last documents and put all documents together in the Thesis report.

Time Plan

This plan show the different phases in the phase plan over time, when every phase is expected to start and finish.

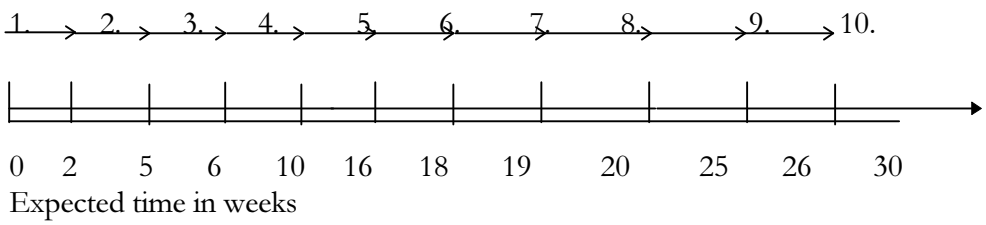


Figure 12 Time Plan

1. Collect background information
2. Understand / Learn "base" code
3. Port code from UNIX to WindowsNT.
4. Design modified Depth First One prong
5. Implement modified Depth First One prong
6. Design Breadth First One prong
7. Implement Breadth First One prong
8. Test the two simulators
9. Do analysis / performance testing on modified Depth First One prong
10. Do analysis / performance testing on Breadth First One prong

11. Design GUI
12. Implement GUI
13. Write report

Organization Plan

This plan is supposed to define the organization during the various phases of the project, and it should define the specific responsibilities of each group/person within the organization. Since the organization consists only of one person, me, and the whole job is done only by me this plan is not necessary.

Test Plan

This section describes the tools, procedures for conducting testing of the produced software.

- The debugging facilities of the program and Microsoft Visual C++ will be used to test the software.
- If I find it necessary, every new function with certain complexity will be tested in a separate program outside the major program.
- A major test will be done to verify the functionality of each simulator.
- The test phase ends when the major test has been done and I find the simulators stable enough.

Change Control Plan

This plan describes the how changes in the plan should be handled.

- All major changes will be reported to my major advisor and together we will proceed with suitable actions.
- All minor changes will only be reported to my major advisor if I find it necessary.

Document Plan

This should describe all the paperwork that is done during the project. Since the Thesis report is the only document produced and it is supposed to contain all important documents produced during this project, no document plan is needed.

Chapter 7

INTERNAL DESIGN.

Both the Depth First One prong simulator and the Breadth First Oneprong simulator are based on the one prong simulator developed at University of Colorado at Colorado Springs which is part of the NETRESTORE [2] simulation system used for evaluating DCS-based distributed network restoration algorithms. The one prong was developed using code from the Two Prong simulator[7], but was implemented with a simplified logic. The Depth First One prong (DFOP) uses a depth first approach to find the restoration paths , as the original one prong, but it has modifications to increase the performance. The Breadth First One prong (BFOP) uses a breadth first approach to find the restoration paths instead of the depth first approach used by the original one prong.. Both simulators have also got a more extensive error checking, preventing the simulators from producing wrong results. The original one prong had a Pascal to C core , which has been removed to simplify porting to other platforms. The documentation of the original one prong was very limited so by including this section in my report, it should be easier for another person to proceed with this work in the future.

The next parts describes the basic building blocks of the simulators, how it works, which data structures that are used and how they are connected to each other. The basic building blocks are the same for both simulators.

Event queue

The event queue holds every event that has happen or should happen during the simulation, when a new message is sent it is put in the event queue or the event when a timer for a node should expire is also put in the queue. Each enode data structure is a node in the event list and these nodes are sorted according to the index field, which contains the time when the event should occur. Each enode also contain a pointer to a event record, the erec data structure. This data structure contains the id of the next node that should process this event and other information about an event. The erec also contains a pointer to a list of symbol node data structures, these structures are used as a kind of associative array, used to convey different kinds of information from source to destination. The figure below shows a graphical representation of the event queue and its data structures.

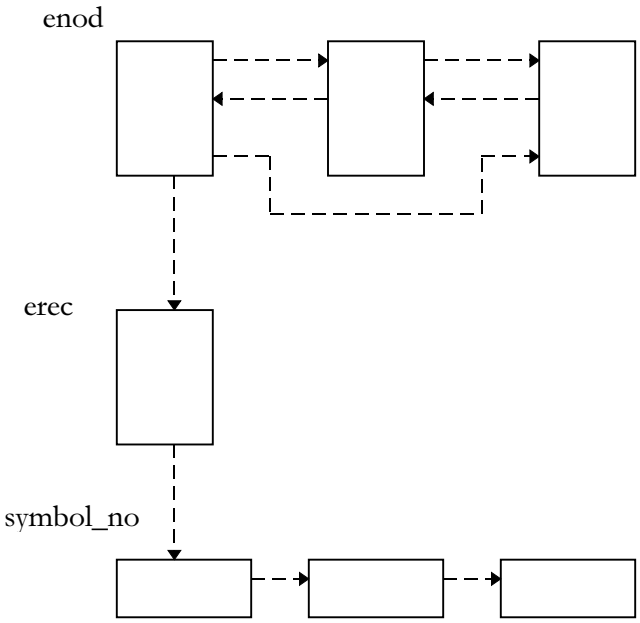


Figure 13 Event queue

Eventnode data structure

A linked list of these data structures becomes the event queue.

```
typedef struct enode
{
float       idx;
enode       *skip,*next,*prev;
erec        *e;
}enode;
```

idx	index field contains the time of the event, when it should occur.
*skip *next *prev	pointers used by the linked list.
*e	pointer to eventrecord.

Eventrecord datastructure

Eventrecord, used to keep information about a event.

```
typedef struct erec
{
float           time;
int             done;
long            next;
long            src;
long            dst;
symbol_node     *msg;
}erec;
```

time	the time of the event.
done	if it has been carried out.
next	the id of the next node that should process this event, for message sending this filed contains the receive id of the message. Note this field may be the same as the dst field, if the next node happens to be the destination of the message.
src	the id of the creator of this event.
dst	the id of the destination of this event.
*msg	a pointer to the associative array.

Symbol node data structure

A linked list of these data structures becomes an associative array .

```
typedef struct symbol_node
{
struct symbol_node *next;
char *sstr, *dstr;
}symbol_node;
```

*next	pointer to the next in list.
*sstr	stringpointer that points to the keystring
*dstr	strinpointer that points to the valuestring.

Node data structure

The node data structure is the most complex data structure in the simulator, it holds lots of information, and this information is basically gathered in linked lists of different kind, as the simulation proceeds. The description below gives information about every data structure connected to the node. The figure below shows a graphical representation of the data structures connected to each node.

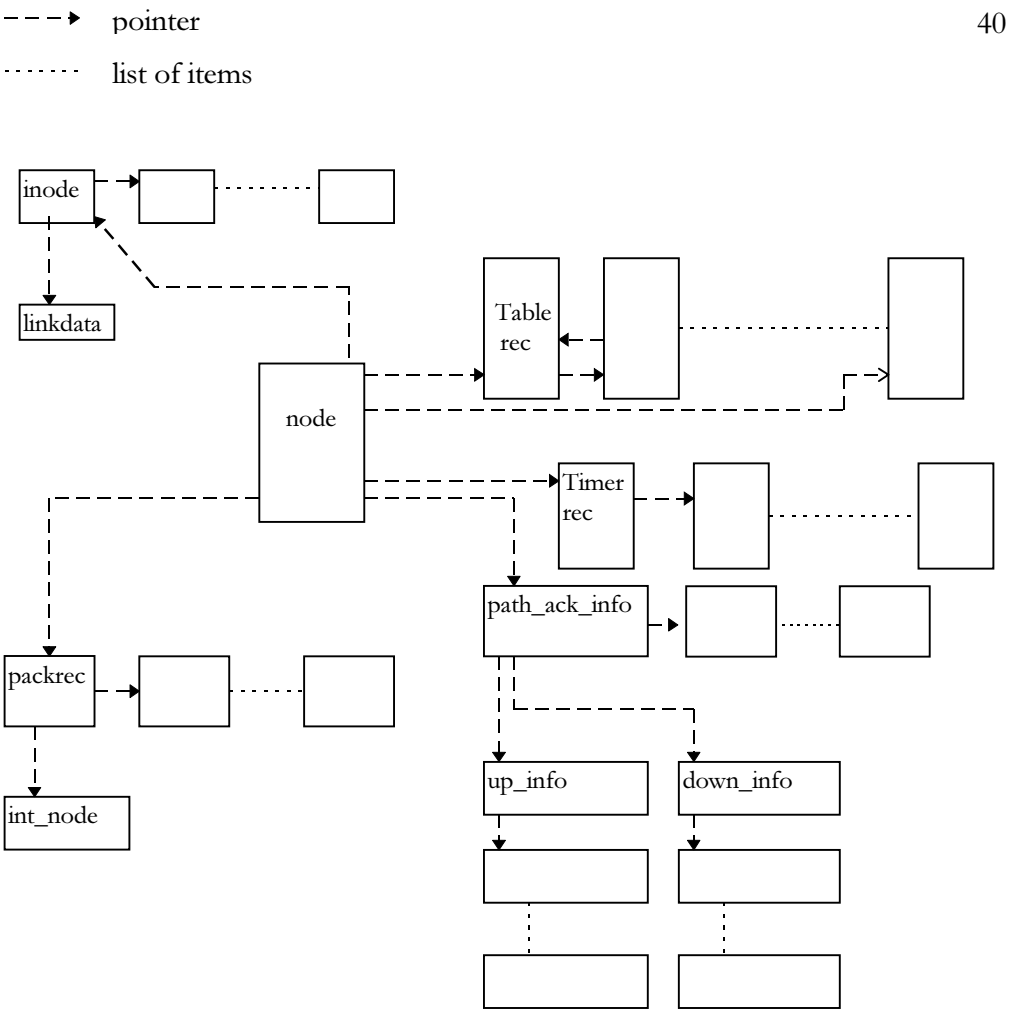


Figure 14 Node Data Structure

First there is the actual node data structure, it contains basic information about the node, most of this information is read from the network description

file. This data structure also contain other data structures, the list data structure and the TimerRecord data structure described below.

```
typedef struct node
{
int             id;
float           time;
float           dcx_time;
list            links;
char            name[4]
float           msg_process_delay;
float           lat,long;
tablerec        *tablehead;
tablerec        *tabletail;
packrec         *packlist;
path_ack_info   *ack_sent;
char            packbuf[MAX_PACKET_LEN+1];
TimerRecord     *RecordHead;
} node;
```

id:	Internal node identification
time:	Time that the node last processed a message.
dcx_time:	Time that DXC finishes last request.
links:	Linked list of linkdata
name:	Name of node as read in from network description file.
msg_process_delay:	Message processing delay, from network description file.
lat, long:	Latitude, Longitude of node.
tablehead:	Pointer to start of tablerec list.
tabletail:	Pointer to end of tablerec list.
packlist:	Pointer to list of packrec
ack_sent:	Pointer to list of path_ack_info.
packbuf:	Buffer for accumulate messages.
RecordHead:	Pointer to list of TimerRecords.

Table record datastructure

This data structure contains the information for one entry in the table. The whole table is a list of tablerec data structures. An entry is made when the node receives a FAIL message or a PACK message, these entries can then be copied or split as the simulation proceed and other messages arrive. Every tablerec contains information about a certain path. This data structure also contains another data structure, the confirm data structure described below.

```
typedef struct tablerec
{
int             path;
int             qos;
int             up;
int             hopcount;
int             bw;
int             down;
int             ackhop;
int             whichPart;
confirm         confirmed;
float           time_connected;
float           time_received;
struct tablerec *next, *prev;
int             path_bw;
char            PackPath[260]; [2]
int             num_ack_sent; [3]
} tablerec;
```

path:	Internal path identification number
qos:	Quality of service, 1 indicates highest priority
up:	Internal identification number of upstream node.

[2] Only used by Breadth First Oneprong to store the path the PACK message traversed through.

[3] Only used by Breadth First Oneprong to track how many ACK have been sent along a certain path.

hopcount:	Hop count from startnode via up node
bw:	If entry references a connection, bandwidth of connection else bandwidth requested.
down:	Internal identification number of downstream node.
ackhop:	Hop count from finish node via down node.
whichpart:	Location of node on 'path', START, MIDDLE or FINISH.
confirmed:	Used to keep track of confirmed paths.
time_connected:	If the connection has been acked but not confirmed, time connection was acked. If connection has been confirmed, time the connection was confirmed.
time_received:	Time when information in entry was received.
next, prev:	Used for linked list

Confirm data structure

This data structure is used to keep track of state of the connections and is located inside the tablerec data structure.

```
typedef struct confirm
{
int             value;
message_history ack;
int             conf_recv;
boolean         conf_sent;
}confirm;
```

value:	shows the state of the connection, 0 unconnected, CONFIRMED if entry has been confirmed, ACKED if entry has been acked but

	not confirmed.
ack:	used to keep record of acks received.
conf_recv:	FINAL if path has been completely restored, 0 if no CONFs received for this connection. > 0 message number of last CONF received.
conf_sent	TRUE if CONF/CONFF has been forwarded to the downstream node.

Message history data structure

This structure is used to maintain the information that allows a node to forward a CONF or CONFF to a downstream node with correct message number.

```
typedef struct message_history
{
int             recv;
int             down;
int             up;
} message_history;
```

recv:	Id of last ACK received.
down:	Id number of last ACK from the downstream node for which an ACK was forwarded.
up:	Id number of last ACK forwarded to the upstream node.

Pack record data structure

Used by node to keep track forwarded PACK messages.

```
typedef struct packrec
{
char            id[NodeNameLength*2];
int_node        *paths;
struct packrec  *next;
} packrec;
```

id:	PACK identification number.
paths:	Pointer to list of int_node.

next: Used by linked list.

Internal node data structure

Used by above data structure to keep track of which nodes it has forward PACK messages for certain paths.

```
typedef struct int_node
{
int             num;
int             up;
struct int_node *next;
} int_node;
```

num: Internal path identification number.
up: Internal node identification number of up node for which a PACK was sent.
next: Used by linked list.

List data structure

Linked list data structure, used as basic node in certain linked list.

```
typedef struct list
{
inode           *head, *tail, *next;
unsigned short count;
} list;
```

head,tail,next: Pointer to linked list
count: Number of items in list.

Inode datastructure

This is a container data structure, used for example by the node to keep track of links connected to it. Need special functions to retrieve data from list.

```
typedef struct inode
{
 long           idx;
 struct inode   *next;
 Anyptr         e;
} inode;
```

idx:	Index field
next:	Used by linked list
e:	Pointer of type void, points to whatever is used.

Linkdata datastructure

Used to keep track of links connected to a node.

```
typedef struct linkdata
{
int             dest;
boolean         active;
int             work, spare;
float           delay;
float           last_trans;
} linkdata;
```

dest:	Internal identification number of node at the other end of link.
active:	Used to keep state of link, if true it is usable NOT broken.
work,spare:	work-number of working channels, spare-number of spare channels.
delay:	Transmission time over link

last_trans: Ending time of last message transmission.

Timer record data structure

Used to keep track of the timers for each path at a finish node.

```
typedef struct TimerRecord
{
int                 initiated;
float               expire_time;
int                 path;
float               delay;
struct TimerRecord  *next, *prev;
} TimerRecord;
```

initiated: State of timer, 1 initiated, 0 not initiated.
expire_time: Time the timer expire.
path: Internal path identification number.
delay: Delay, number of seconds a path waits until it can send its first ACK.
next,prev: Used by linked list.

Path_ack_info datastructure

DFOP: Each node is allowed to forward an ack message for a given path to the same upstream node a certain number of times. This data structure is used to keep track of which nodes that have been requested to participate in which restoration paths.

BFOP: Only used to keep record of number of ack messages sent.

```
typedef struct path_ack_info
{
int                     path;
int                     num_acks;
up_info                 *uplist;
down_info               *downlist;
struct path_ack_info    *next;
} path_ack_info;
```

path:	Path id.
num_acks:	Not used.
uplist:	Pointer to list of up_info.
downlist:	Pointer to list of down_info.
next:	Used for linked list.

Up_info datastructure

This datastructure contains the information about the processing status of upstream link. [4]

```
typedef struct up_info
{
int             up;
int             ack_num;
boolean         retry;
struct up_info  *next;
} up_info;
```

[4] In BFOP the retry is connected to a certain "packpath" and therefor the BFOP tablerec datastructure has a field for number of ack messages sent for that "packpath", this is only used as a record in BFOP.

up:	Internal identification number of the upstream node for this link.
ack_num:	Number of ack messages, or retry we have done for the link.
retry:	Status of retry.
next:	Used by linked list.

Down_info datastructure

This datastructure contains the information about the processing status of an down stream link. This structure is not used in one prong, relic from Twoprong[7].

Program structure

The code is divided into 9 bigger blocks. Each block is described below. The event loop is located inside the main function, and every time a new event should be processed, the main( ) function calls the Process( ) function, which calls the proper processing function based on what type of event it is. The figure below shows a graphical overview of the program structure.

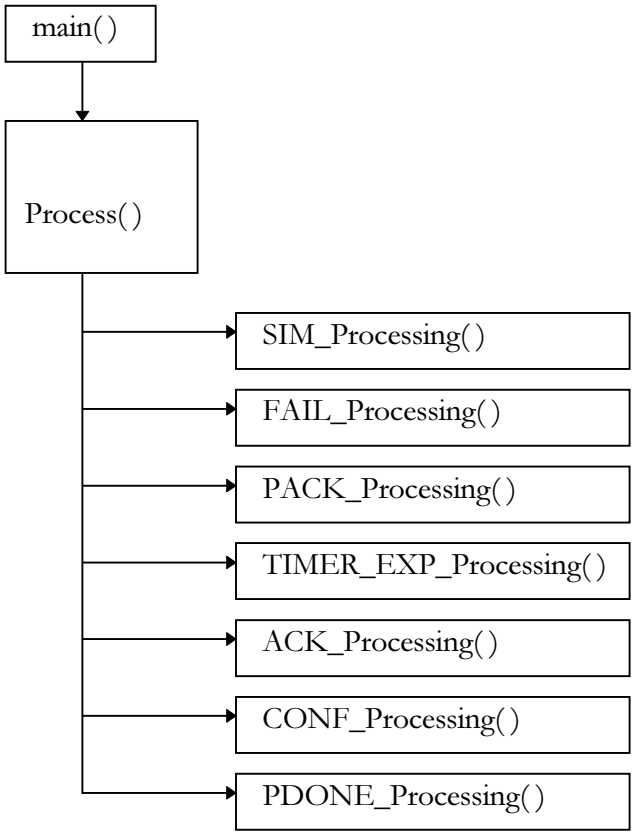


Figure 15 Program structure

Since both simulators are based on the one prong discrete event simulator , the main frame of the simulators are similar. Every action is viewed as an event and put in the event queue. The detection of an error is viewed as a message sent to the nodes detection the message. The expiration of a timer is also viewed as a message sent to the node in question. The below description is only a basic description describing the main operations carried out by each function, there exists of course many other operations in each function that have not been mentioned below.

main( ). This function handles the reading of the command line, the initialization and the event loop. On every loop a new event is retrieved from the event queue and sent to the Process function. When all events have been processed, the simulator is done and terminates.

Process( ). This function receives the next event to be processed. Based on the type of event the appropriate event handling function is called.

SIM_Processing( ). Handles the processing of failure detection messages. Failure detection messages are constructed to simulate the detection of a break, the nodes on each side of the break receives a failure detection message. If the failure detection message contain information about the loss of a node, new failure detection messages are constructed for each of the lost links and sent to the remaining active node on the end of each lost link. If the failure detection message contains information about the loss of a link, the lost channels are marked inactive. If it the node is a middle node on any of the lost paths, the channels of the lost path are released and a FAIL message is sent to the affected

nodes on the lost paths. If it is a start node on any of the lost paths, it initiates broadcast of PACK messages.

FAIL_Processing( ). This function handles the processing of every FAIL message. FAIL messages are sent by the nodes detecting a break A FAIL message can contain several packets, each packet containing information about the loss of a path. If a FAIL message contain several packets, the packets in the FAIL message will be considered one by one. If the node in the packet is a middle node, a FAIL message will be sent to the next node along the disrupted path. If the node in the packet is a start node, PACK messages will be broadcasted. Since a FAIL message can contain several packets, two data structures are used to keep track of which messages are to be sent. The "packs" data structure keep track of all PACK messages to be sent by the end of the FAIL processing and the "fails" data structure keep track of all FAIL messages to be sent by the end of the FAIL processing.

PACK_Processing( ). This function handles the processing of every PACK message. The PACK messages are initially sent by the start nodes for each path and flooded over the network towards the finish node. A PACK message can contain several packets, each carrying information about the loss of a path. If the lost path in the packet has already been maximally restored and the recipient of the PACK knows that the path has already been maximally restored, no PACK message is forward. The "trail" field of the PACK is checked to see if the message has created a loop, if so the PACK is thrown away. One entry is added, for each packet ,to the connection table with information about the lost path.

The BFOP forward the PACK message the downstream nodes using a full broadcast, the DFOP uses a heuristic to decide which downstream nodes the

PACK messages will be forwarded to. If the node receiving the PACK message is a finish node, there exists a path from start to finish. On receiving the first PACK message for a path, the timer is initiated with the delay given at the command line. If a node failure occurs and the lost node is a start/finish node the path can not be restored. To prevent a PACK message from running around the network forever looking for a dead finish node, a trail field is used containing a list of all nodes visited. A node may not forward a PACK to a node in the trail of a PACK message. This trail field is also used by BFOP as a packpath.

TIMER_EXP_Processing( ). This function handles the event when a delay timer expires for a node. The timer expire event for a path is initiated by the finish node when it receives the first PACK message for a path. When a timer expires the finish node initiates the path trace out phase, sending ACK messages to appropriate up stream nodes. Entries with shorter hop count will be tried first. The total amount of bandwidth protected by the ACK messages for a path, will not exceed the bandwidth of the lost path. This is done to make it possible for concurrent path restoration attempts. In BFOP, to ensure that the ACK will follow the path of the chosen PACK message from finish to start a packpath field containing the path of the PACK message is included.

ACK_Processing( ). This function handles the processing of every ACK message. The ACK message is sent out by the finish node when the timer has expired.

In BFOP, on receiving of a ACK message, the next node in the packpath is extracted. If there exists available spares on the link to the next node in the

packpath, the node will protect no more than the number of spares in the request and forward a ACK along the to that node. If there are no spares available, a CONF(bw=0) will be sent back to the node the request came from. If part of the request could be handled, a ACK is sent to the next node in the packpath and a CONFX(bw=x) is sent back with the rejected bandwidth in the bandwidth field of the message.

In DFOP, on receiving of a ACK message, the node search its table to find the entry with minimum number of hops. If there exist enough spares to handle the request on that uplink, a ACK is sent to that upnode. If there are not enough spares available, the table will be searched again to find the next entry with minimum number of hops. This procedure will continue until all entries for that path have been tried, if there still are bandwidth left in the request that could not be handled, a CONFX(bw=X) is sent back. If none of the bandwidth in the request could be handled, a CONF(bw=0) is sent back.

If a ACK message makes it from the finish node to the start node a restoration path have been found. The start node will confirm and reply with a CONF or a CONFF if the path has been maximally restored.

CONF_Processing( ). This function handles the processing of every CONF type of message. When a node receives a CONF(bw=X) or a CONFF(bw=X), the node confirms any additional bandwidth needed to bring its confirmed bandwidth for the path up to the bandwidth in the CONF or CONFF. When the finish node receives a CONFF the path has been maximally restored and the finish node will broadcast PDONE messages over the network.

In BFOP on receiving a CONFX, all protected bandwidth protected by the corresponding ACK is released and the CONFX is sent back along the packpath towards the finish node. When the finish node receives a CONFX message it will search its table to find an entry with minimum number of hops. Another ACK will then be sent along the packpath, retrieved from the entry, to replace the bandwidth that could not be handled.

In DFOP on receiving a CONFX, all protected bandwidth protected by the corresponding ACK is released and the node search its table to find another entry with minimum number of hops to replace the bandwidth that could not be handled. If all or part of the rejected bandwidth in the CONFX could not be handled, a CONFX is sent back along the path the ACK came.

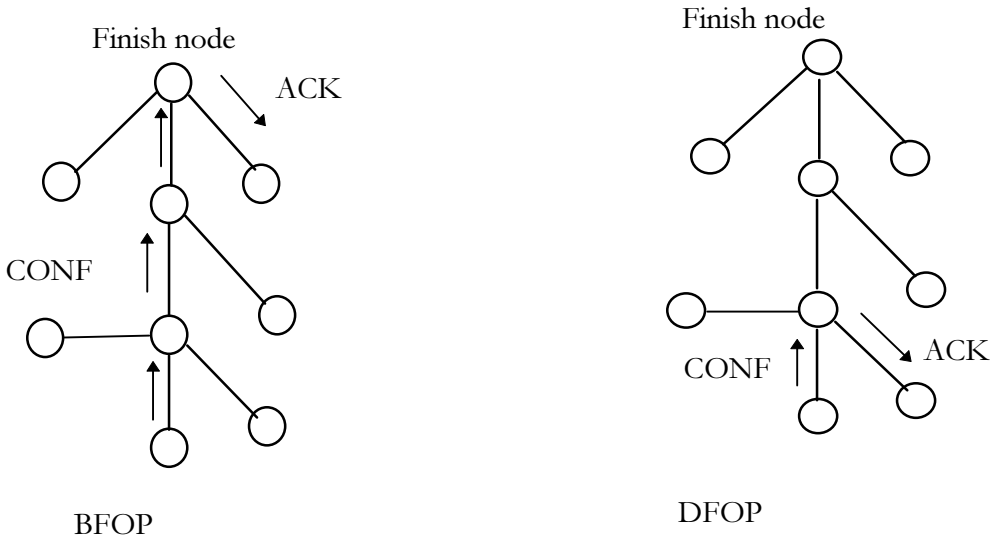


Figure 16 Handling of CONF messages

PDONE Processing( ). This function handles the processing of every PDONE message. When a node receives a PDONE, it marks all paths in the packets of the PDONE as maximally restored and forwards PDONE messages to all its immediate neighbors not fount in the trail in the PDONE. The forwarded PDONE will contain a packet for each path that the node has just discovered to be maximally restored.

Chapter 8

ANALYSIS USING FUNCTIONAL CHARACTERISTICS

Path finding

Both DFOP and BFOP finds paths by flooding PACK messages over the network. The path traveled by a PACK message from start node to finish node defines a path.

DFOP do not use a full broadcast, it uses a heuristic to reduce the message volume. A PACK is only forwarded from a certain upstream node to a downstream node once.

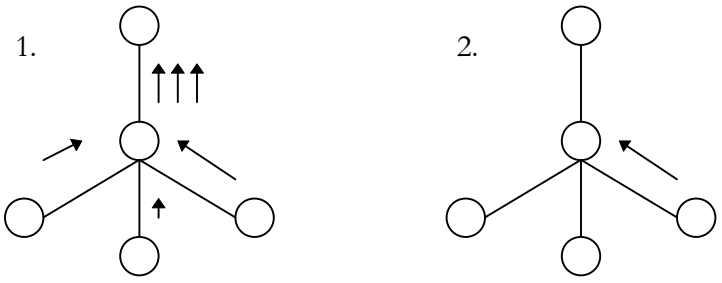


Figure 17 DFOP PACK forwarding

Node A is a downstream node and C, D, and E are upstream nodes. Node B will only forward a total of 3 PACK messages to A for a certain path, one from each upstream node. If node B receives another PACK from one of the upstream nodes it will not forward it to A , just make an entry in its table.

In BFOP much of the intelligence has been moved from the middle nodes to the finish node. It needs all information about the paths, so the approach used by DFOP is not possible. It uses a full broadcast, all PACK messages from a certain upstream node will be sent to the downstream node.

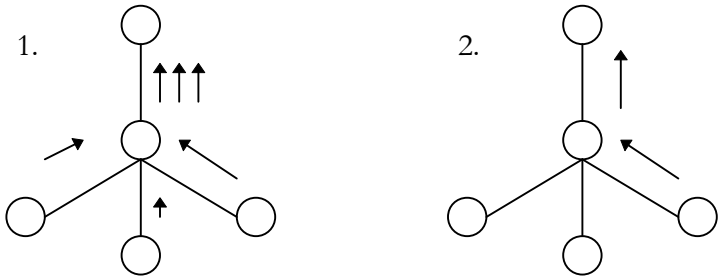


Figure 18 BFOP PACK forwarding

This produces a bigger volume of PACK messages, so BFOP needs other approaches to reduce the PACK message volume.

Spare Channel Contention

Both DFOP and BFOP use Floodgating [5] to reduce spare channel contention. Unlike most other distributed which will flood multiple messages over a link without regard to the total bandwidth which has been requested by

these messages. DFOP and BFOP keeps track of the total requested bandwidth over a single link and will not forward requests for more bandwidth than the link can support.

Restoration Path Selection

The usual approach to select restoration path is on a "first come, first served" basis and was used by the original one prong. This is not the approach used by DFOP and BFOP. By using a timer to delay the sending of the first ACK message, the finish node hopefully get more entries to choose from. Which entry it is going to try is based on the hop count, the entries with shorter hop count will be tried first. This approach gives the finish node in the BFOP approach, full control of which path will be tried and in what order.

In the DFOP approach the finish node does not have full knowledge about the paths that will be tried, the hop count gives the finish node a hint on which link to initiate the restoration attempt. If this attempt comes to a dead end, the finish node has no control until all possibilities on that certain link have been tried.

Message volume control.

The DFOP uses the above heuristic to decrease the volume of the PACK messages and a "trail" field to prevent looping, but it also has got other ways to control the total message volume. It uses a hop count limit on the ACK messages, preventing it from digging to deep in the tree. Floodgating [5], a mechanism used to ensure that the total amount of restoration bandwidth requested over any given link does not exceed the total number of available spares channels in that link, also serves to reduce the message volume, since many late arriving messages are

not flooded once all available spare channel capacity on all links from a node has already been reserved.

The first version of BFOP had no methods to reduce the volume of PACK messages except for the "trail" field preventing loops, causing the message volume to be very high. To reduce the volume of PACK messages, several methods were implemented. The table below shows the results for the different approaches on the NJ3 net.

cut	delay (sec)	msg vol no red	Hop limit	msg vol	TTL (sec)	msg vol	entry lim	msg vol
N00-N01	0.0	579	5	256	0.8	167	1	140
N00-N02	0.1	503	5	232	0.7	136	1	100
N00-N03	0.6	968	9	966	1.7	950	44	968
N00-N04	0.0	669	4	183	0.5	103	3	272
N00-N05	0.0	586	4	180	0.5	111	4	419
N00-N07	0.1	583	3	105	0.5	105	1	122
N01-N02	0.0	245	7	186	1.2	146	4	195
N02-N04	0.0	340	4	78	0.7	78	1	98
N03-N04	0.0	216	4	120	0.5	89	2	118
N03-N07	0.0	277	7	257	1.0	206	10	235
N03-N08	0.0	609	7	568	1.2	496	12	609
N04-N05	0.0	466	4	184	0.5	119	1	146
N04-N06	0.0	862	4	279	0.7	279	2	156
N04-N07	0.0	403	3	173	0.5	179	1	149
N04-N08	0.1	515	5	341	0.7	157	11	567
N04-N10	0.0	697	7	655	1.2	602	29	689
N05-N07	0.1	332	3	65	0.5	65	1	82
N06-N07	0.1	870	5	348	0.7	209	2	230
N07-N08	0.1	1217	3	201	0.5	201	1	224
N07-N10	0.1	769	7	665	1.2	555	15	765
N08-N09	0.1	1291	7	1181	1.2	959	14	1291
N08-N10	0.0	245	6	193	1.0	139	12	245
N09-N10	0.0	881	4	307	0.7	307	3	264
N00	0.0	1006	3	70	0.6	80	1	202
N02	0.0	78	4	29	0.7	29	1	45
N03	0.0	265	5	179	0.7	124	5	249
N04	0.0	560	5	344	0.8	249	2	248
N05	0.0	178	3	45	0.5	45	1	62
N07	0.1	655	4	297	0.5	180	2	302
N08	0.1	559	5	360	0.7	244	4	337
N10	0.0	429	6	396	1.0	338	7	417

Table 1 BFOP Message Volume

The first approach tried was to set a hop count limit on the PACK messages. The second approach was to use a TTL, time to live, on the PACK messages. The third approach was to set a limit on the number of unconnected entries in a table at each node. Of the three approaches the TTL worked best followed by the hop count limit. The current version of BFOP has got all three possibilities to reduce the PACK message volume. BFOP also uses the Floodgating approach mentioned above. For short restoration paths and smaller networks BFOP has got a smaller message volume than DFOP. But as the network and the restoration paths increases, so does the message volume. BFOP needs tuning to reduce the message volume, if wrong parameters are given the message volume can reach substantial volumes for bigger networks. The message volume is a big drawback of BFOP.

Congestion control

Both DFOP and BFOP uses message forwarding and Floodgating heuristics to control congestion. Even though the PACK messages of BFOP are distributed in an even way, they can cause trouble as the network size increases.

Race Condition Control.

Both DFOP and BFOP use tables and logic to control race conditions , each node keeps a table of all messages it has seen. A logic is used to act upon each message that arrives. This logic depends on the message type received, its source , the requested bandwidth, the spare channel capacities of all links connected to the node and the state of all entries in the message table in the node.

Restored Path Connection

For both BFOP and DFOP the connection is made on the receiving of an ACK message, but this connection can be canceled until it receives a confirmation.

Chapter 9

PERFORMANCE ANALYSIS USING PERFORMANCE METRICS

For all simulations the default parameters have been used unless stated otherwise. This means that they have all been tested with the same parameter settings, the only difference is that DFOP has a delay parameter and BFOP has a delay parameter and a TTL parameter. I did not find it necessary to do tests with different transmission speeds, message processing delays, etc. since the main thing here was to investigate the impact of using a breadth first approach and a minimum hop based restoration path selection.

It is important to point out here that, the BFOP could not be tested in a full way. The computational power / memory of the machine I had access to, limited the time to live parameter to 4.0 sec for the d_31.net and 3.0 sec for the dis5_64.net. This caused the search depth for BFOP to be limited.

Restoration level

For the NJ3.net, a 10 node network, the DFOP achieves equal or higher restoration level than ONOP in all but two cases. It achieves 100% restoration level in 26 of 31 cuts and so does the ONEP. The BFOP achieve equal or higher restoration level than both DFOP and ONEP in all cases. The 100% restoration level is achieved in 28 out of 31 cuts.

For the lata3.net, which also is a 10 node network, both DFOP and BFOP achieves equal or higher restoration level compared to ONEP in all cases. The

100% restoration level is equal for both DFOP and BFOP, 11 out of 21 cuts. ONEP reaches a 100% restoration level in 10 out of 21 cuts.

For the d_31.net, which is a 31 node network, the situation is a bit different. The DFOP achieves equal or higher restoration level than ONEP in all cases. In some cases the increase in restoration level is very high, more than a 100% increase. It achieves a 100% restoration level in 55 of 67 cuts, compared to 45 out of 67 for ONEP. The BFOP has equal or higher restoration level than ONEP in all cases and equal or higher restoration level than DFOP in all but one case. For this case, the 17-4 cut, DFOP achieves a 99.03 % restoration level compared to 98.00% restoration level for BFOP. BFOP has got the highest number of 100% restoration level cuts. It achieves a 100% restoration level in 59 out of 67cuts.

The dis5_64.net is a big network with 64 nodes, and in some cases the lost bandwidth is very high compared to the other three networks. The DFOP achieves equal or higher restoration level in all but two cases compared to ONEP. In one case DFOP increases the restoration level by 400% compared to ONEP. DFOP achieves a 100% restoration level in 95 out of 147 cuts, compared to ONEP which achieves a 100% restoration level in 42 out of 147 cuts. The Time to live parameter, TTL, of the BFOP limits the depth of a restoration path, for this case a maximum TTL of 3.0 sec was used. This limits the search depth for BFOP, and it is not possible to make a fair judgment for this net. When the search depth is enough BFOP achieves higher restoration level than ONEP. Based on previous tests I assume that BFOP will at least reach the same restoration levels as DFOP, or perhaps better results if a bigger TTL was used.

Time to restoration

Based on equal restoration level.

For the NJ3.net when equal restoration level was achieved, DFOP is faster than ONEP in all but 3 cases. Overall the DFOP is faster than ONEP, in one case the restoration time has decreased to 50% of the ONEP restoration time. The BFOP is faster than ONEP in 20 out of 31 cases. Compared to DFOP it is only faster in 3 cases.

For the lata3.net DFOP is faster than ONEP in all cases. Here the increase in speed is more evident, it reaches 100% restoration level in less than 2 sec on several cuts and in one case the restoration time is only 26% of the ONEP restoration time. BFOP is faster than ONEP in all but 2 cuts and the scenario is the same for BFOP as for DFOP, it reaches a 100% restoration level in less than 2 sec on several cuts. Compared to DFOP, it is faster on 3 occasions. In one case BFOP achieves 100% restoration level in half the time of DFOP.

For the d_31.net when equal restoration level was achieved, DFOP was faster than ONEP in all but 4 cases. In some cases the time is only half the time needed by ONEP to achieve the same restoration level. BFOP is faster than ONEP in 22 of 34 cases when a 100% restoration level is achieved, but BFOP can also be very slow in some cases. Generally, when the restoration paths are short BFOP is faster. A comparison between BFOP and DFOP shows that BFOP is faster than DFOP in one case.

For the dis5_64.net, DFOP achieves 100% restoration level faster than ONEP in all but 4 cuts. The only cut that ONEP performs much better is the 53-52cut, DFOP need 14.7 sec to complete a 100% restoration level but ONEP does it in 10.9 sec. For the other three the difference is less severe. Overall the DFOP is faster than ONEP, the restoration time for DFOP range from 0 to 50%

of the ONEP restoration time. The BFOP was not designed for speed, but it is faster than ONEP in some cases. DFOP is faster than BFOP in most cases, but in some rare cases BFOP is equally fast as DFOP.

Spare channel utilization

The comparison is based on equal restoration level

When comparing the spare usage between different cuts, the number of spares used is not best measurement. A better measurement would be to calculate the quote between the number of spares used and the number of bandwidth lost, since the spare usage is dependent on the lost bandwidth.

$$Sparequote = \frac{Number_of_spares_used}{Number_bandwidth_lost}$$

This quote will show how many spares that were used for each lost bandwidth.

For the NJ3.net DFOP has better or equal spare usage compared to ONEP in all but 6 cuts. The occasions when ONEP has got the best spare usage, the difference is not that big compared to the occasions when DFOP has got the best spare usage. BFOP was designed for a small spare usage, and it has equal or better spare usage on all cuts but 1 cut compared to ONEP. There is overall a bigger difference in spare usage between BFOP and ONEP. On one occasion ONEP reaches a spare quote of 3.62 compared to BFOP which has got a spare quote of 2.04 for the same cut. A comparison between DFOP and BFOP shows that BFOP has equal or better spare usage than DPOP in all but 1 cut.

For the lata3.net DFOP has equal or better spare usage on all but 2 cuts. The difference here is not that big as it was on the nj3.net. The BFOP has got equal or better spare usage on all cuts compared to both ONEP and DFOP.

For the d_31.net , DFOP has equal or better spare usage than ONEP in all but 1 cut. Overall DFOP has got a better spare usage and DFOP, but the difference is not that big. BFOP has equal or better spare usage than ONEP in all but 2 cuts. Overall BFOP has better spare usage than both ONEP and DFOP. The difference in spare usage is bigger for BFOP and ONEP, compared to DFOP and ONEP. When equal restoration level is achieved, DFOP has equal or better spare usage in all but 2 cuts. Overall the spare usage is better for DFOP. When equal restoration level is achieved, BFOP has equal or better spare usage than ONEP in all but 1 case. Overall when the restoration level is equal, the spare usage is better for BFOP. If we compare BFOP and DFOP on a equal restoration level basis, the spare usage is almost identical. In 4 cases BFOP has better spare usage and in 2 cases DFOP has better spare usage, on all other cuts the spare usage is equal.

Range of application

The range is equal for all three algorithms. They can handle single link, multiple link, node and area failures.

Message volume

Overall the message volume for the DFOP and ONEP are similar since they both use the depth first approach. The BFOP uses a breadth first approach and has not got the message volume reduction schemes that are possible to use in the depth first approach. This causes the message volume to be high for BFOP in

most cases, except when the restoration paths are short. In these cases the message volume is lower than for the depth first approaches.

Restoration level in 2 seconds

This is not a performance metric mentioned in the earlier chapter, but it can give a hint at which rate an algorithm restores channels.

Overall both DFOP and BFOP achieve a higher restoration level than ONEP within 2 sec. It is difficult to do a overall comparison between DFOP and BFOP, sometimes BFOP is faster and in other cases, DFOP is faster.

Conclusion and observations of performance analysis

From the results I have received I would like to point out the following important observations:

- The DFOP has outperformed ONEP in most cases.
- BFOP good choice if we want a economical spare usage or a high restoration level, if the restoration time is not that important.
- Overall DFOP has better spare usage than ONEP but not as good as BFOP.
- It is possible that the big message volume for BFOP is not as bad as it seems, since the message volume is spread over a longer time and the messages are distributed in a more even way. The PACK messages contribute to most of the messages, but they are distributed in an even

way for both BFOP and DFOP. The ACK/CONF messages for BFOP are distributed in a more even way than for DFOP. The DFOP can cause a big concentration of ACK/CONF messages in certain areas of the network. The BFOP searches each restoration path in an increasing hop count based kind of way and does not produce these high concentrations of messages.

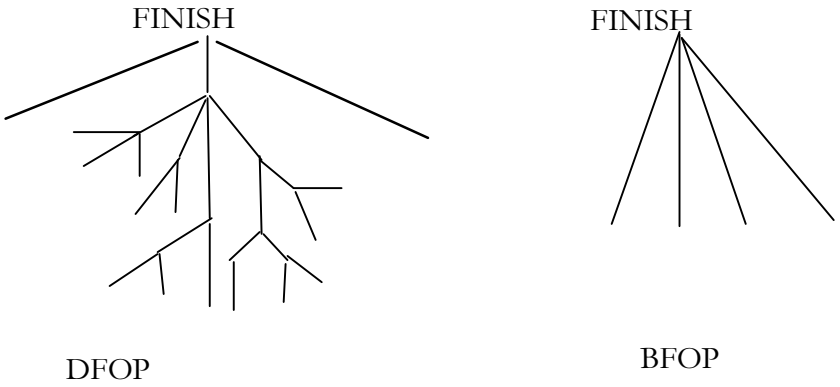


Figure 19 Distribution of ACK/CONF messages

- The use of a time-out mechanism can increase the performance of an algorithm
- The use of a minimum hop based restoration path selection can increase the performance of an algorithm.
- The use of random delay was also tested during the early phases of the performance testing phase, but was abandon because of several reasons, including : hard to track the delays, the performance was not good. It is important to point out one thing, at one occasion the random delay

performed better than the constant delay. In all these tests all nodes have used the same delay, but by using different delays for different nodes based on path or failure it is possible to increase the performance.

- Breadth First Approach can be a good choice for a link based type of algorithm, since the restoration paths that need to be found are shorter.
- For smaller network BFOP has got a better performance, the reason for this is that to reach a restoration path with a depth K, it has to check all possible restoration paths with a depth < K. This causes the algorithm to be slow when the restoration paths are long.
- When the network size increases it is very important that the TTL parameter is set to a suitable value.

Chapter 10

OBSERVATIONS DONE DURING THIS PROJECT.

Reading/ learning existing one prong code

The second phase of this project was to read and learn the one prong code from scratch. This basically included the work of going through every line of code and every detail of the code. This was a necessity if I wanted to use the code as a base. This turned out to be a very tedious and time consuming process, and the expected time was multiplied several times until I felt I had the necessary knowledge to start with the modifications. If I had been an experienced programmer the time needed to reach this level of knowledge had probably been less, but even though I have been studying computers for many years and done some programming I felt that I had a lack of experience. The problem of this phase was not related to the syntax of the program, it was more likely related to my limited experience in connecting code to the things the program is supposed to do. The syntax was written in a easy read kind of way, but the connection between the code and what the algorithm was doing was sometimes very difficult to see. If I had been more used to read someone else code, this problem had been less severe.

One thing that I have learned from this, it can sometimes be easier or faster to build your program from scratch than try to use someone else code as a base and the importance of a good documentation.

Porting one prong from UNIX GNU GCC to Win32 MSVC++

To be able to work under Windows NT, I needed to port the software from UNIX GCC to Microsoft Visual C++ (MVC++). This included removing the Pascal-To-C (p2c) core that the UNIX version needed, rewrite functions that were not included in MVC++ or try to find suitable functions to replace the UNIX functions that was not available in MVC++, construct new data structures that was not available in MVC++ and recompile the whole thing in MVC++. The Pascal-To-C core was a set of library files, that was included at compile time.

When the one prong got through the compiler and the syntactical errors were corrected, a rather time consuming debugging phase followed. The whole simulator acted very unstable and many strange errors occurred and several error checking functions were added before I could find the errors.

Later during this project, after I have ported the code, I found GNU GCC for Windows NT at http://www.cygnus.com. I tried to compile the original code, using GNU GCC instead of Microsoft Visual C++ to see if I would have the same problems. The porting procedure was much easier this time.

It seems like it is not possible just to take C code from one operating system and recompile it on another operating system using a different compiler and think that it is going to work. This code did not have any GUI so I thought that it should be easy, but it took a lot more time than I expected. If one has to port code from one platform to another it is a good idea to try to use the same compiler if it is possible.

Why Win32

Since this project was done back home in Sweden and with no access to any UNIX workstations, I had to choose between LINUX (UNIX for PC) and Windows NT. The environment under LINUX was the text based GNU gcc compiler and its debugger, for Windows NT I managed to borrow a copy of Microsoft Visual C++ environment with a fully graphical user interface.

I decided to work on a Windows NT based system because it gave me the possibility to use a more professional , easy to use, programming environment, instead of the GNU environment on LINUX, and since it included a visual tool for building graphical user interface it was the natural choice. By using Windows NT I also got access to better word processors, spreadsheets and the Internet. I have also included some other reasons why I wanted to work under windows NT.

- The development of better operating systems for the PC like Windows NT and Win95 combined with lower prices and growing computational power has increased the industrial interest in the PC. This has made it possible for the PC to move in to areas that before was only accessible to UNIX workstations.
- By start porting research software like the NETRESTORE [2] simulation system to PC, we have also made it more accessible to students to make their research at other places then the university.
- Windows NT is a fast growing operating system

Time needed for this project

When I started this project I estimated the time needed to finish this project to 26 weeks, perhaps 30 weeks of full time work. This was a poor judgment from my side. During phase 2 and 3 I noticed that I was not able to follow the expected time plan, and it got worse during the later phases. If I had taken different approaches to certain problems, some time could have been spared.

During the programming phase the first big mistake was done. I tried to construct the simulator by taking different parts from the original one prong, mix it with my own parts, put it together and then test it in a "big bang" kind of way. This was of course the wrong approach, I ended up with a complex simulator that did not behave as I had expected in any way. After a long time of debugging I decided to start over and try a different approach. Instead of taking parts from the original code and mix with my own code, I used the original code and transformed it in small steps, testing the code after each step. By using this approach the code had to written in respect to the original code, resulting in many strange solutions but I ended up with two working simulators.

I only had access to one computer so I had to do all simulations, using that computer. This was also a time consuming part, 200 - 300 hours were needed. By running the simulations 24 hours a day the simulation phase took about 2 weeks. During this period I could not use my computer so my possibilities to work on the project was limited.

As a result of all these delay, the GUI had to be abandoned. After an agreement with my major thesis advisor we decided to abandon the GUI part and concentrate on the other parts.

Working abroad

All of this work was done at home in Orsa, Sweden and the only contact I had with my university in Colorado Springs, USA was by using email. This of course limited my resources in many ways. I had no access to a academic library, and if I needed information I had to search for it on the Internet or travel 120 km to my old university in Borlänge, Sweden. The supply of more advanced computer books is also very limited compared to the United States. The only computer I had access to was my own Pentium 90 machine and all work, including simulations, were done using this machine.

Overall, if one has access to a computer at home and a Internet connection it should be possible to work this way. Of course this demands an active email reading advisor, that can answer your questions quickly. In my case it worked out very well, if I had any questions I sent them in the evening and most of the time I had the answers I needed the next morning.

BIBLIOGRAPHY

[1] C. Edward Chow "CS622 Distributed Networks Lecture Notes"

[2] C. Edward Chow "User Guide of Netrestore2.0", MCI report

[3] C.-H. E. Chow, J. Bicknell, S. McCaughey and S. Seyd " A fast Distributed Network Restoration Algorithm" Proceedings of the 12th International Phoenix Conference on Computers and Communications, page 261-267, Scottsdale, Arizona, March 24-26, 1993.

[4] C.-H. E. Chow, J. Bicknell and S. Seyd "RREACT: A Distributed Protocol for Rapid Restoration of Active Communication Trunks", Network Management and Control Volume 2, page 391-406.

[5] C. Edward Chow "The final Research Report on Inter Exchange (IPX) Network Restoration", draft, MCI-IPX Network Restoration Project Final Report.

[6] C. Edward Chow, Jay Shah and Dave Allen "System and Method for Restoring a Telecommunications Network Based on an Adaptive One Prong Approach", draft of a patent application, page 1-14.

[7] C. Edward Chow, J. Bicknell and S. Seyd "PERFORMANCE ANALYSIS OF FAST DISTRIBUTED LINK RESTORATION ALGORITHMS" , International journal of communication systems vol 8, page 325-345, 1995.

[8] J. Bicknell , C. Edward Chow and Sami Seyd " Performance Analysis of Fast Distributed Network Restoration Algorithms", draft, to be published in Proceedings of GLOBECOM 1993.

[9] Philip W. Metzger "Managing a Programming Project", book, second edition, Prentice-Hall, ISBN 0-13-550772-3.

[10] Elaine Rich, Kevin Knight "Artificial Intelligence", book, second edition, International edition, McGraw-Hill, ISBN 0-07-100894-2.

[11] Microsoft Press "Visual C++ User's Guide Version 4 Volume one", book, ISBN 1-55615-915-3.

[12] Ginsberg, Matt "Essentials of Artificial Intelligence", book, Morgan Kaufman Publishers, San Mateo, California 1993

APPENDIX A

Input options

Command Format:
BFOP netfile [options] for Breadth First one prong.
DFOP netfile [options] for modified Depth First one prong

Options

/s=XXX Stoptime (default /s=300.0 sec)
/l=XXX LCS (default /l=0.05)
/i=XXX Index (default /i=1.5) fiber refractive index, 1.5 equivalent to 5 μs/km
/n=XXX Noise (default /n=0.00001)
/o=filename, where filename=Outputfilename (default /o=NETSIM.OUT)
/b=XX-YY linkbreak between XX and YY, /b=ZZ nodebreak node ZZ
/x=XXX Transmission speed (default /x=8.0kbps)
/p=XXX Port CPU Delay (default /p=0.100 sec)
/m=XXX msg/signature processing delay (default /m=0.004 sec)
/c=XXX dcs connectd time (default /c=0.02 sec)
/y=X parallel DCS mode, default =1 for sequential mode
/d=XXX Debug 100=debug,110=debugPath,101=debugNode (default /d=000)
/r=XXX Hoplimit (default /r=25)
/a=XXX Acklimit (default /a=1)
/t=X Algorithm type (default /t=1)
/k=XXX Timer Delay=XXX (Constant) Turns off Random Delay(which is default)
/z=XXX RandomTimerDelay (UP)Limit, TimerDelay 0-XXX sec (default=1.0 sec)
/q=XXX RandomTimerDelay (DOWN)Limit, TimerDelay XXX-1.0 sec (default=0.0 sec)
The /z and /g switches are ignored if /k=XXX is set
/u=X PACK Time to live (default /u=30.0 sec) **
/v=X Unconnected Table Entry Limit (default = 100) **
/e=X Table Entry Limit (default=100) **
Example: BFOP nj3.net /b=N00-N01

** Not available for DFOP

APPENDIX B

Output interpretation

```
E:\timer_onep>BFOP nj3.net /b=N04 > out
N04  path=20/10/3/2/5=30.00%/20.00%/50.00%  bw=232/31/74=41.89%
in2p=1/0/9=10.00%/0.00%/90.00%
in2bw=9/74=12.16%  lrt=3.033093  wam=2.296818  ham=2.186093  msgs=405
spares=0/82/82
```

```
E:\timer_onep>BFOP nj3.net /b=N04 > out
N04          path=A/B/C/D/E=F%/G%/H%          bw=I/J/K=L%
in2p=M/N/O=P%/Q%/R%
 in2bw=S/T=U% lrt=V wam=W ham=X msgs=Y spares=0/Z/XZ
```

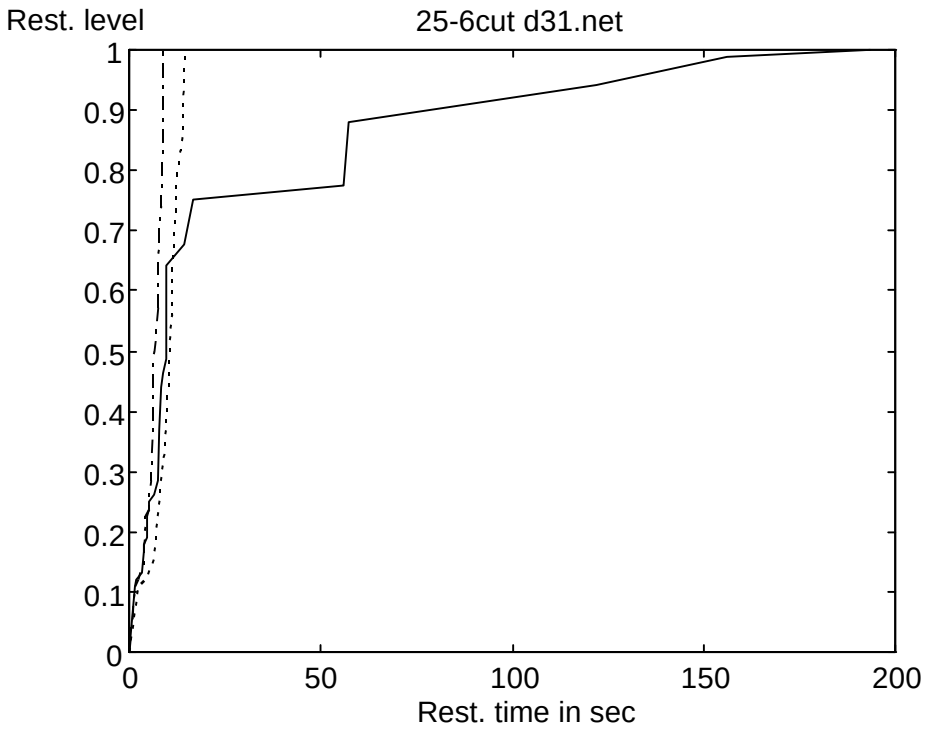
A:	number of lost path
B:	number of non restorable paths
C:	number of fully restored paths
D:	number of partially restored paths
E:	number of paths that could not be restored
F:	percentage of restorable paths that got fully restored (C / (A-B))
G:	percentage of restorable paths that partially restored (D / (A-B))
H:	percentage of restorable paths that could not be restored (E / (A-B))
I:	total bw of non restorable paths :
J:	bw of restored paths:
K:	bw of restorable paths
L:	percentage that got restored (of restorable paths)
M:	The number of paths that are fully restored within 2 seconds.
N:	The number of paths that are partially restored within 2 seconds
O:	The number of paths that are not restored within 2 seconds
P:	The percentage of full restored paths within 2 seconds . (M/(A-B))
Q:	The percentage of partially restored paths within 2 seconds (N/(A-B))

R:	The percentage of not restored paths within 2 seconds. (O/(A-B))
S:	bw restored in 2 seconds.
T:	bw of restorable paths
U:	restoration level after 2 seconds
V:	time of last restoration
W:	weighted arithmetic mean of restoration time
X:	harmonic mean of restoration time
Y:	message volume
Z:	spares
XZ:	spares

APPENDIX C, PLOTS

The following section contains restorationlevel / restorationtime plots of some interesting cuts.

Plot 1

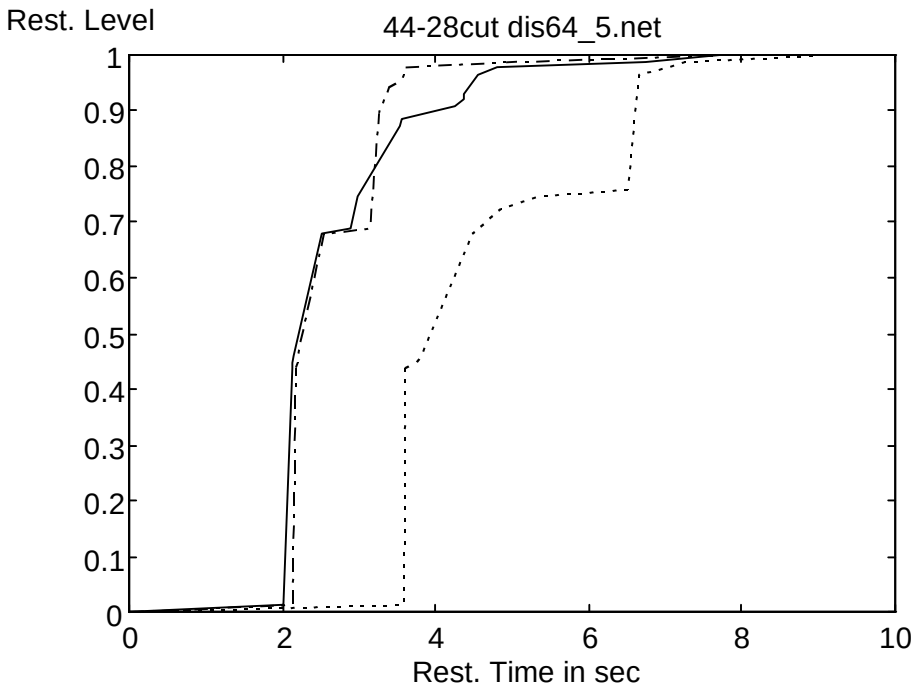


_._ ._ DFOP, ... ONEP, --------- BFOP

Figure 20 Plot of 25-6cut

This plot shows the 25-6 cut from the d31 net. In this case one can clearly see the big difference in restoration time between BFOP and the other two. BFOP needs almost 200 seconds to reach a 100% restoration level, compared to DFOP and ONEP which both reach a 100% restoration level within 20 seconds. The reason for this big difference has to do with the hop count of the restoration paths and the breadth first approach. If a certain restoration path has a hop count of X hops, the breadth first approach need to search every path that has got a hop count smaller than X hops. So when the restoration paths gets long, the time needed to find them increases rapidly. The BFOP uses 771 spares (sparequote=9.18) with a message volume of 22446 messages and it reaches a 10.71 % restoration level within 2 seconds for this case. DFOP uses 770 spares (sparequote=9.17) with a message volume of 1636 messages and it also reaches a 10.71 % restoration level within 2 seconds. The original One Prong, ONEP uses 743 spares (sparequote=8.85) with a message volume of 1600 messages but it reaches 0.0 % restoration level within 2 seconds. This is an interesting case because this is one of the few cases when ONEP uses fewer spares than both BFOP and DFOP even though both BFOP and DFOP are designed for economical spare usage.

Plot 2



_._._._ DFOP, ... ONEP, _____ BFOP

Figure 21 Plot of 44-28cut

This plot shows a case when both BFOP and DFOP reaches a 100 % restoration level within 8 seconds and ONEP within 9 seconds. This is shown at the figure below. In this case BFOP and DFOP follow each other closely up to a 70% restoration level and then DFOP speeds up and reach a 95-98% restoration

level within 4 seconds, while BFOP needs at least 6 seconds to reach the same restoration level. In the end they both reach a 100% restoration level within 7.8 seconds (DFOP 7.81 sec, BFOP 7.71 sec). ONEP is slower than both DFOP and BFOP and needs over 9 seconds to reach 100 % restoration level. BFOP uses 273 spares (sparequote=3.14) with a message volume of 8103. DFOP uses 269 spares (sparequote=3.09) with a message volume of 1854. ONEP also uses 269 spares (sparequote=3.09) with a message volume 1850.

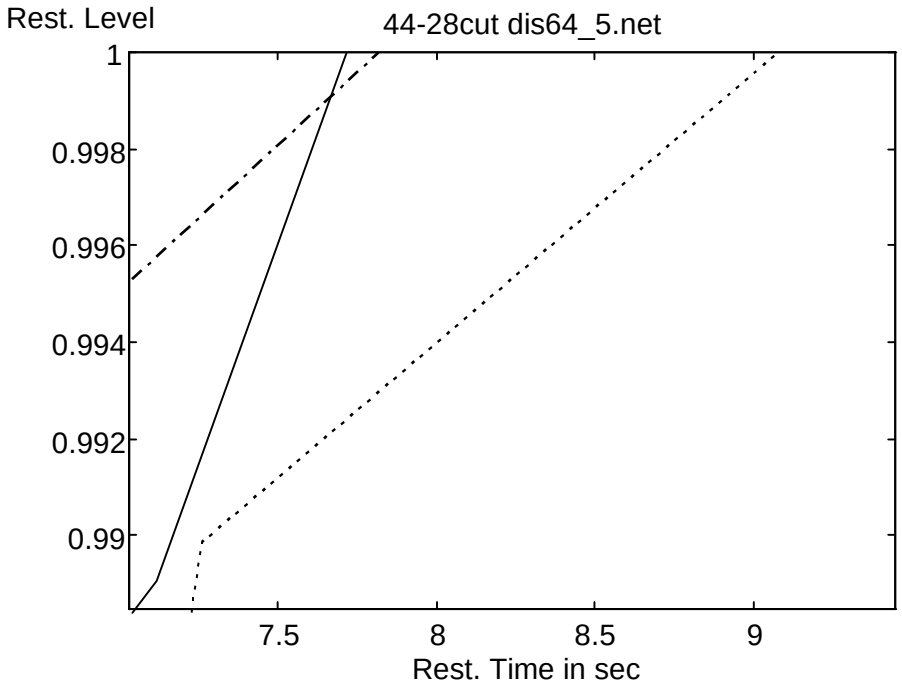
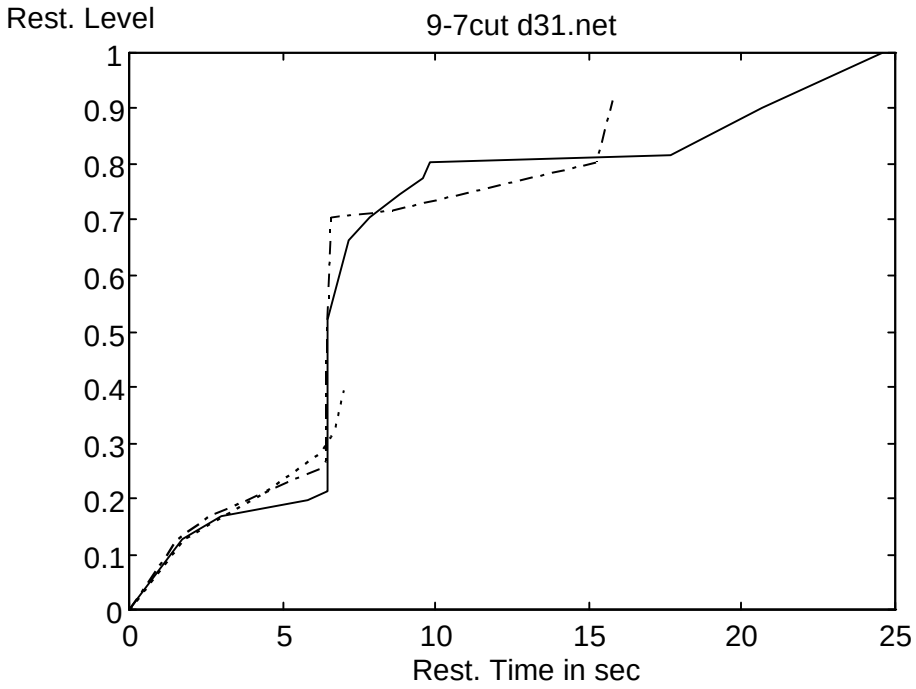


Figure 22 Plot of 44-28cut, zoom

Plot 3



_._._._ DFOP, ... ONEP, _____ BFOP

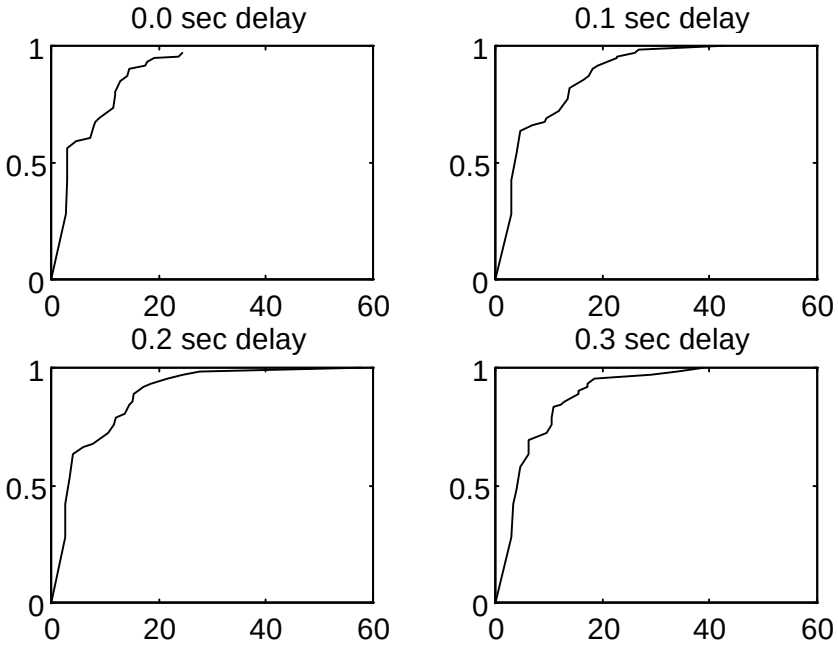
Figure 23 Plot of 9-7cut

This case shows a situation when both DFOP and BFOP achieves a much higher restoration level than the original One Prong (ONEP). It also shows that BFOP can reach a higher restoration level than both DFOP and ONEP within an acceptable restoration time. ONEP only reaches a 40 % restoration level, DFOP reaches a 90 % restoration level while BFOP reaches a 100 % restoration level. ONEP uses 162 spares and needs 526 messages to achieve a 39.44 % restoration level within 7.0 sec, DFOP uses 447 spares and needs 772 messages

and achieves 91.55% restoration level in 15.8 sec and BFOP uses 563 spares and 8879 messages to achieve a 100 % restoration level within 24.6 sec.

Plot 4

N00-N03 NJ3.net



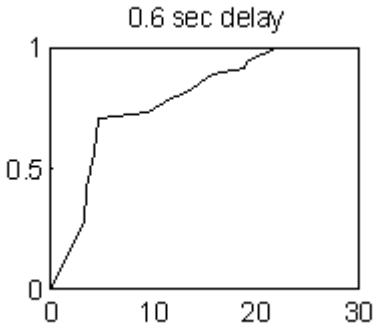


Figure 24 Plot of N00-N03cut, different delays

These plots shows the restoration level / restoration time for different delays. The 0.0 sec delay only reaches a 95 % restoration level, the 0.2 sec delay needs almost 60 seconds to reach a 100 % restoration level while the 0.6 sec delay reaches a 100% restoration level within 25 seconds. These plots shows the importance of choosing a suitable delay. The plot below shows all delay in one plot.

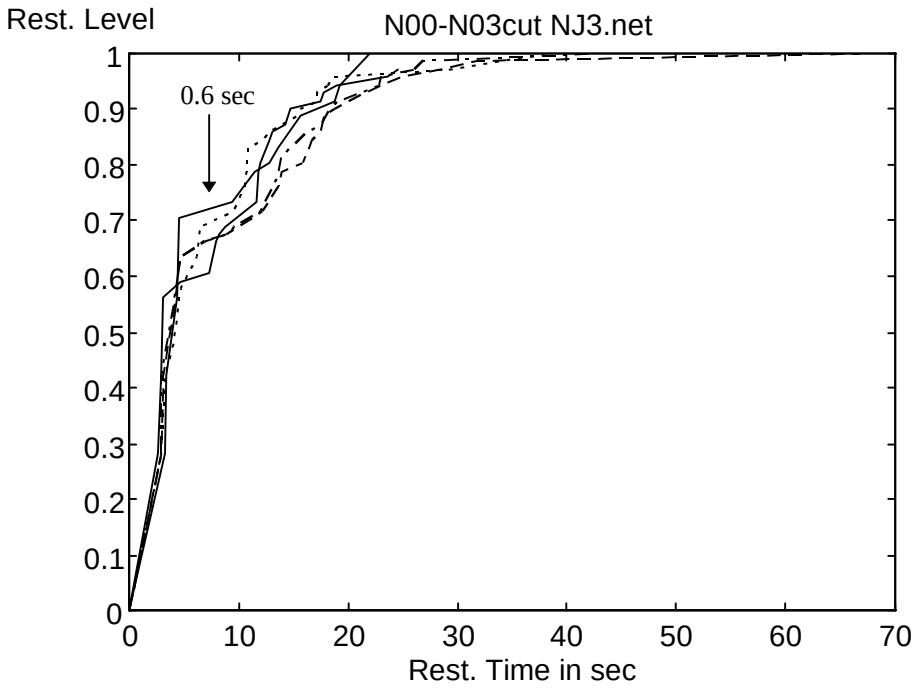


Figure 25 Plot of N00-N03cut

Plot 5

14 cut d31.net

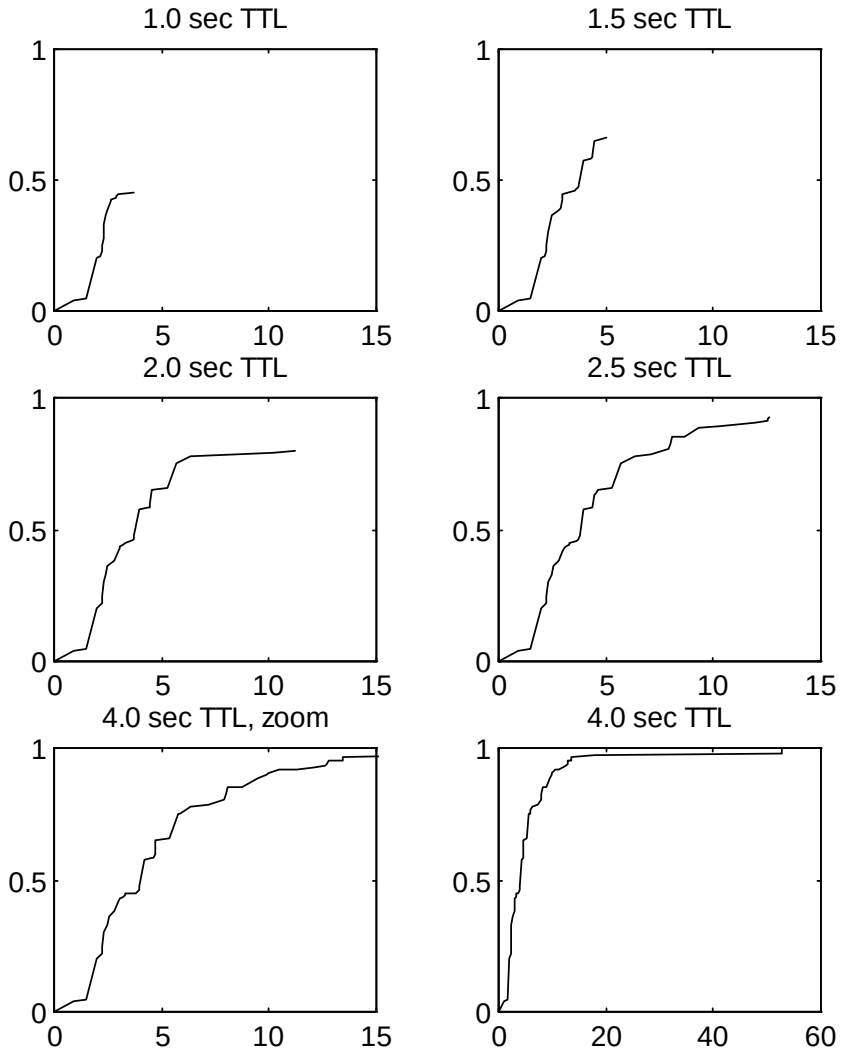


Figure 26 Plot of 14cut, different TTL

The above plots shows the restoration level / restoration time plots for different TTL. If one look at the 1.0 sec TTL and the 1.5 sec TTL there is no big difference, it only looks like the plot has been cut off. If one on the other hand look at the 2.0 sec TTL and the 2.5 sec TTL, there is an interesting difference. In this case one can see that they are very similar up to a 75 % restoration level, but then the 2.5 sec TTL increases the restoration level more rapidly than the 2.0 sec TTL. The reason for this is the fact that the 2.5 sec TTL has got "more" restoration paths to choose from, while the 2.0 sec TTL has got a lack of suitable restoration paths compared to the 2.5 sec TTL.

APPENDIX D

Accumulated Bandwidth Problem

The use of accumulated bandwidth in the ACK message can in certain cases be a danger, causing the simulator to produce wrong result. Since both DFOP and BFOP are based on the Oneprong software, this problem was not discovered until late in the project when both simulators were going through the test phase. It is a problem that does not occur that often, but when the message exchange gets high in a certain area it can occur. It took many hours of searching and debugging before the problem was identified and what caused the problem. I have called it the "accumulated bandwidth problem". This problem has been solved in DFOP and BFOP.

Assume the network area below

An ACK has already traversed through A-B-F-H, requesting 15 bw.

Another ACK arrives from D requesting 6 bw and forwards to F. The accumulated bw is now 15+6 =21 to F.

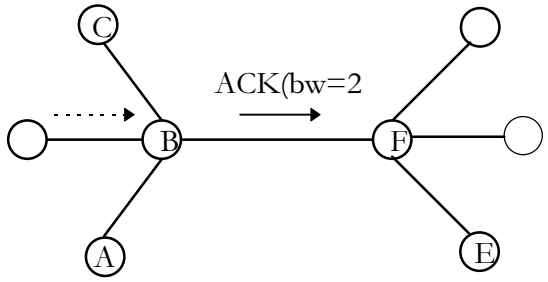


Figure 27 Accumulated Bandwidth Problem (1)

Another ACK arrives at B from C requesting 5 bw. An ACK is forwarded to F. The accumulated bw is now 15+6+5=26 to F

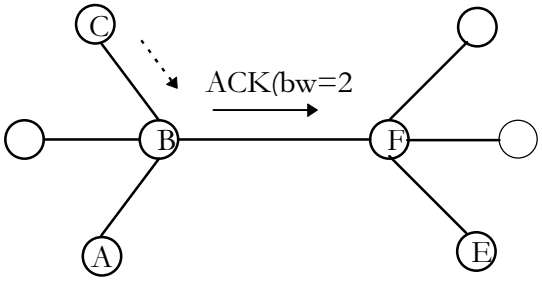


Figure 28 Accumulated Bandwidth Problem (2)

The first ACK arrives at F, but can not be handled by F, a CONF(bw=0) is sent back to B. The accumulated bw =15 down to B.

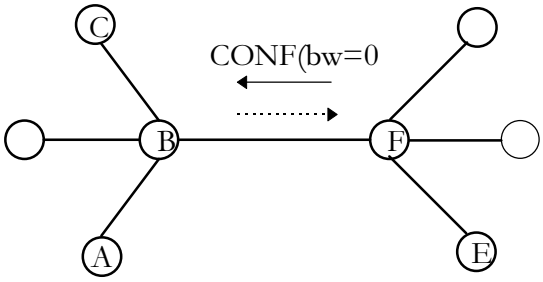


Figure 29 Accumulated Bandwidth Problem (3)

A CONF message arrives from E releasing 2 bw down to E.

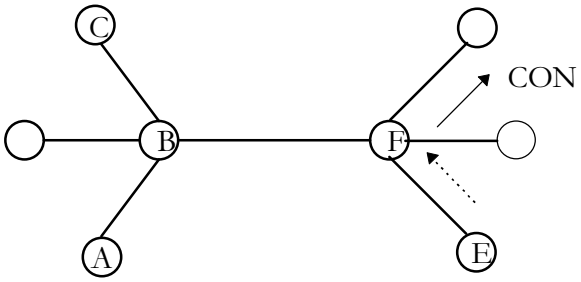


Figure 30 Accumulated Bandwidth Problem (4)

The second ACK(bw=26) arrives at F, F calculates the actual bw request to 26-15=11 bw, but only 4 bw could be handled, a CONFX(bw=7) is sent back to B and ACK messages are sent to E and G.

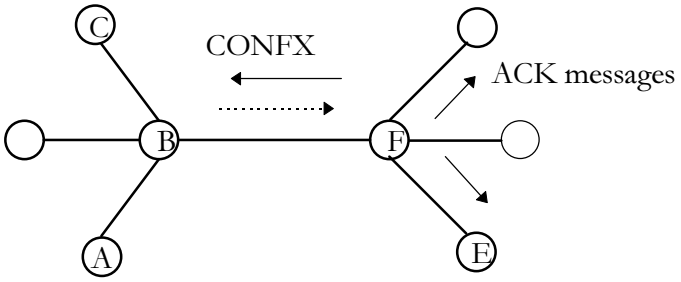


Figure 31 Accumulated Bandwidth Problem (5)

Some time later the CONFX message arrives at B, and node B calculates how much bw that could be handled by F. The corresponding entry in the table is equal to5 bw and the bw field in the message is equal to 7, results in 5-7= -2 a negative and wrong result.

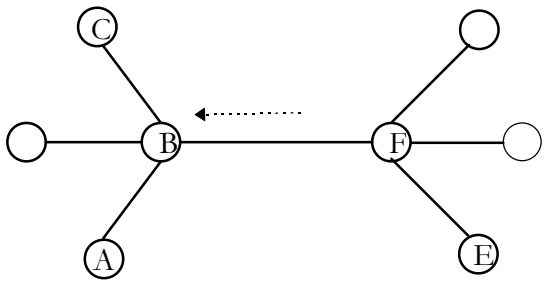


Figure 32 Accumulated Bandwidth Problem (6)

The reason is that node F calculates the actual requested bw to 11 when it is supposed to be 5, resulting in a requst that is bigger than it is supposed to be. This is just one of the cases that can occur because of this problem, there could be other situations messing up the network.

Assume that there initially were accumulated bw =16 bw to F.

1. A request for 4 bw arrives at B and B forward it to F (acc_bw=20)
2. F receives the this request, calculates can not handle it, forwards a CONF back to B.

3. B receives another request for 4 bw and forward it to F (acc_bw=24)
4. F receives a message that releases 2 bw on F-E link and 2 bw on F-G link.
5. F receives the request from B (acc_bw=24) and calculates the actual request to 24-16=8. Forward an ACK with actual request for 2 bw along link F-E, an ACK with actual request of 2 bw along F-G and forward a CONFX(bw=4) back to B.
6. Some time later B receives the CONFX message, finds the entry in its table with bw=4, calculates that 0 bw could be handled. B now tries another link, requesting the 4 bw that B think could not be handled.

Now there are bw protected from F to G and F to E, and they can later be fully/partially rejected causing a reject message to be sent back to B. The messages can also traverse all the way to the start node, and get confirmed, causing situations that can not be handled

APPENDIX E, SIMULATION RESULTS

This Appendix contains the best simulations results for DFOP, BFOP and ONEP for the NJ3 net, the LATA3 net, the D_31 net and the DIS5_64 net.

The complete simulation results in Microsoft Excel 4 format can be downloaded from URL:

ftp:// owl.uccs.edu/~chow/pub/master/aohansma/doc/results.zip

ONEP NJ3.NET

nj3.net cut	bw lost	ONEP Final rest.level	Time (sec)	Rest.lev 2 sec	sprs	Msg vol
N00-N01	74	100.00%	4.505	0.00%	238	149
N00-N02	71	100.00%	4.403	19.72%	192	129
N00-N03	71	94.36%	4.946	0.00%	308	195
N00-N04	53	100.00%	4.573	18.87%	158	167
N00-N05	55	100.00%	3.889	16.36%	157	172
N00-N07	52	100.00%	3.190	0.00%	128	145
N01-N02	53	100.00%	4.155	50.94%	170	101
N02-N04	16	100.00%	3.503	25.00%	52	125
N03-N04	68	100.00%	3.735	0.00%	209	106
N03-N07	59	93.22%	5.971	0.00%	249	107
N03-N08	51	92.16%	4.536	0.00%	200	172
N04-N05	48	100.00%	4.120	0.00%	135	189
N04-N06	16	100.00%	3.370	0.00%	52	138
N04-N07	81	100.00%	4.138	0.00%	209	106
N04-N08	50	100.00%	3.887	0.00%	181	146
N04-N10	48	100.00%	4.679	0.00%	256	185
N05-N07	47	100.00%	3.969	19.15%	145	113
N06-N07	41	100.00%	4.519	0.00%	114	208
N07-N08	57	100.00%	4.517	0.00%	155	264
N07-N10	64	100.00%	3.909	0.00%	198	175
N08-N09	78	100.00%	4.734	0.00%	239	237
N08-N10	65	100.00%	6.251	0.00%	233	91
N09-N10	34	100.00%	3.794	0.00%	119	181
N00	110	46.36%	3.766	0.00%	124	191
N02	8	100.00%	2.896	0.00%	24	58
N03	26	100.00%	2.937	0.00%	104	76
N04	74	100.00%	4.555	0.00%	158	252
N05	27	100.00%	2.720	0.00%	81	83
N07	98	74.48%	5.860	0.00%	212	311
N08	38	100.00%	3.935	0.00%	133	162
N10	26	100.00%	3.404	0.00%	103	116

Table 2 ONEP NJ3.NET

DFOP NJ3.NET

nj3.net cut	bw lost	DFOP Final rest.level	Time (sec)	Rest.lev 2 sec	dly (sec)	sprs	Msg Vol
N00-N01	74	100.00%	3.727	41.89%	0.00	224	165
N00-N02	71	100.00%	3.045	46.48%	0.00	184	123
N00-N03	71	100.00%	6.012	0.00%	0.30	335	240
N00-N04	53	100.00%	2.182	62.26%	0.00	132	145
N00-N05	55	100.00%	2.707	34.55%	0.00	157	172
N00-N07	52	100.00%	2.306	53.85%	0.10	128	151
N01-N02	53	100.00%	3.738	7.55%	0.00	170	94
N02-N04	16	100.00%	2.767	25.00%	0.00	52	116
N03-N04	68	100.00%	6.214	5.88%	0.10	234	134
N03-N07	59	100.00%	6.365	15.25%	0.00	240	130
N03-N08	51	82.35%	3.661	0.00%	0.00	187	173
N04-N05	48	100.00%	2.569	18.75%	0.00	135	176
N04-N06	16	100.00%	2.850	37.50%	0.00	64	142
N04-N07	81	100.00%	3.163	0.00%	0.00	210	183
N04-N08	50	100.00%	3.037	0.00%	0.10	133	144
N04-N10	48	100.00%	6.801	0.00%	0.10	220	220
N05-N07	47	100.00%	2.359	38.30%	0.10	123	106
N06-N07	41	100.00%	2.401	34.15%	0.10	114	212
N07-N08	57	80.70%	2.482	22.81%	0.00	126	275
N07-N10	64	100.00%	3.064	37.50%	0.10	198	177
N08-N09	78	100.00%	4.080	0.00%	0.00	273	258
N08-N10	65	100.00%	4.230	0.00%	0.00	223	70
N09-N10	34	100.00%	2.758	52.94%	0.00	99	177
N00	110	46.36%	2.316	17.27%	0.00	124	560
N02	8	100.00%	2.752	50.00%	0.00	24	58
N03	26	100.00%	2.858	0.00%	0.00	104	76
N04	74	72.97%	2.785	32.43%	0.00	154	253
N05	27	100.00%	2.342	0.00%	0.00	81	83
N07	98	80.61%	3.967	42.86%	0.10	200	348
N08	38	100.00%	2.858	0.00%	0.00	133	162
N10	26	100.00%	3.661	0.00%	0.00	115	126

Table 3 DFOP NJ3.NET

BFOP NJ3.NET

nj3.net cut	bw lost	BFOP Final rest.level	Time (sec)	Rest.lev 2 sec	dly (sec)	TTL (sec)	sprs	Msg vol
N00-N01	74	100.00%	3.713	41.89%	0	0.8	220	167
N00-N02	71	100.00%	2.682	26.76%	0.1	0.7	180	136
N00-N03	71	100.00%	21.962	0.00%	0.6	1.7	337	950
N00-N04	53	100.00%	2.423	50.94%	0	0.5	132	103
N00-N05	55	100.00%	3.344	16.36%	0	0.5	156	111
N00-N07	52	100.00%	2.306	53.85%	0.1	0.5	128	105
N01-N02	53	100.00%	6.356	50.94%	0	1.2	162	146
N02-N04	16	100.00%	2.766	25.00%	0	0.7	52	78
N03-N04	68	100.00%	4.340	5.88%	0	0.5	200	89
N03-N07	59	100.00%	10.139	15.25%	0	1.0	242	206
N03-N08	51	100.00%	5.186	0.00%	0	1.2	235	496
N04-N05	48	100.00%	2.611	18.75%	0	0.5	135	119
N04-N06	16	100.00%	3.203	0.00%	0	0.7	52	279
N04-N07	81	100.00%	5.362	51.85%	0	0.5	180	179
N04-N08	50	100.00%	5.486	54.00%	0.1	0.7	130	157
N04-N10	48	100.00%	23.241	0.00%	0	1.2	203	602
N05-N07	47	100.00%	2.358	38.30%	0.1	0.5	123	65
N06-N07	41	100.00%	2.841	34.15%	0.1	0.7	114	209
N07-N08	57	100.00%	2.542	28.07%	0.1	0.5	155	201
N07-N10	64	100.00%	7.770	37.50%	0.1	1.2	193	555
N08-N09	78	100.00%	9.599	34.62%	0.1	1.2	273	959
N08-N10	65	100.00%	6.500	0.00%	0	1.0	189	139
N09-N10	34	100.00%	2.757	23.53%	0	0.7	99	307
N00	110	46.36%	2.260	17.27%	0	0.6	124	80
N02	8	100.00%	2.752	50.00%	0	0.7	24	29
N03	26	100.00%	4.525	0.00%	0	0.7	104	124
N04	74	79.73%	3.226	5.69%	0	0.8	172	249
N05	27	100.00%	2.342	0.00%	0	0.5	81	45
N07	98	80.61%	2.470	53.16%	0.1	0.5	195	180
N08	38	100.00%	4.404	0.00%	0.1	0.7	133	244
N10	26	100.00%	7.140	0.00%	0	1.0	103	338

Table 4 BFOP NJ3.NET

ONEP LATA3.NET

cut	bw lost	ONEP Final rest.level	Time (sec)	Rest.lev 2 sec	sprs	Msg vol
N00-N01	18	27.78%	3.492	0.00%	10	200
N00-N02	18	38.89%	4.245	16.67%	14	290
N00-N03	18	61.11%	5.068	0.00%	35	200
N00-N04	13	100.00%	6.100	0.00%	29	168
N00-N05	14	78.57%	4.922	0.00%	29	240
N00-N07	13	100.00%	5.885	0.00%	26	146
N01-N02	13	38.46%	2.581	0.00%	10	176
N02-N04	4	100.00%	3.575	0.00%	8	68
N03-N04	17	100.00%	5.609	0.00%	40	159
N03-N07	15	100.00%	6.315	0.00%	36	174
N03-N08	13	100.00%	5.899	0.00%	34	173
N04-N05	12	66.67%	6.197	0.00%	16	172
N04-N06	4	75.00%	2.290	50.00%	6	70
N04-N07	20	85.00%	4.853	0.00%	38	174
N04-N08	12	100.00%	4.117	0.00%	24	128
N04-N10	12	100.00%	3.597	0.00%	24	130
N05-N07	11	91.67%	6.107	0.00%	31	202
N06-N07	10	20.00%	3.181	0.00%	4	98
N07-N08	14	100.00%	4.747	0.00%	29	148
N07-N10	16	100.00%	5.316	0.00%	39	172
N09-N10	9	77.78%	3.919	0.00%	14	128

Table 5 ONEP LATA3.NET

DFOP LATA3.NET

lata3.net cut	bw lost	DFOP Final rest.level	Time (sec)	Rest.lev 2 sec	dly (sec)	sprs	Msg Vol
N00-N01	18	27.78%	1.538	27.78%	0.00	10	200
N00-N02	18	38.89%	1.578	38.89%	0.00	14	117
N00-N03	18	61.11%	3.115	22.22%	0.30	34	230
N00-N04	13	100.00%	2.401	69.23%	0.00	30	170
N00-N05	14	78.57%	2.300	35.71%	0.30	28	404
N00-N07	13	100.00%	1.699	100.00%	0.00	26	146
N01-N02	13	38.46%	1.539	38.46%	0.00	10	176
N02-N04	4	100.00%	1.435	100.00%	0.00	8	68
N03-N04	17	100.00%	5.556	94.12%	0.00	37	164
N03-N07	15	100.00%	1.961	100.00%	0.00	34	168
N03-N08	13	100.00%	2.677	61.54%	0.00	34	172
N04-N05	12	66.67%	1.599	66.67%	0.00	16	172
N04-N06	4	75.00%	1.415	75.00%	0.00	6	170
N04-N07	20	100.00%	2.836	85.00%	0.00	53	184
N04-N08	12	100.00%	1.679	100.00%	0.00	24	128
N04-N10	12	100.00%	1.679	100.00%	0.00	24	130
N05-N07	11	91.67%	2.347	83.88%	0.00	28	196
N06-N07	10	20.00%	1.479	20.00%	0.00	4	98
N07-N08	14	100.00%	1.819	100.00%	0.10	28	144
N07-N10	16	100.00%	3.115	81.25%	0.30	39	172
N09-N10	9	77.78%	1.565	77.78%	0.00	14	293

Table 6 DFOP LATA3.NET

BFOP LATA3.NET

		BFOP						
cut	bw lost	Final rest.level	Time (sec)	Rest.lev 2 sec	dly (sec)	TTL (sec)	sprs	Msg vol
N00-N01	18	27.78%	1.538	27.78%	0.00	0.30	10	120
N00-N02	18	38.89%	1.577	38.89%	0.00	0.30	14	54
N00-N03	18	61.11%	5.298	22.22%	0.30	0.90	31	528
N00-N04	13	100.00%	2.822	69.23%	0.00	0.60	29	220
N00-N05	14	78.57%	2.642	35.71%	0.30	0.60	28	206
N00-N07	13	100.00%	1.699	100.00%	0.00	0.30	26	104
N01-N02	13	38.46%	1.539	38.46%	0.00	0.30	10	60
N02-N04	4	100.00%	1.435	100.00%	0.00	0.30	8	45
N03-N04	17	100.00%	2.362	88.24%	0.00	0.60	36	152
N03-N07	15	100.00%	2.402	73.33%	0.00	0.60	34	246
N03-N08	13	100.00%	3.566	61.54%	0.00	0.90	34	332
N04-N05	12	66.67%	1.599	66.67%	0.00	0.30	16	122
N04-N06	4	75.00%	1.415	75.00%	0.00	0.30	6	20
N04-N07	20	100.00%	2.019	95.00%	0.00	0.30	40	142
N04-N08	12	100.00%	1.679	100.00%	0.00	0.30	24	80
N04-N10	12	100.00%	1.679	100.00%	0.00	0.30	24	74
N05-N07	11	91.67%	4.565	50.00%	0.00	0.90	28	532
N06-N07	10	20.00%	1.479	20.00%	0.00	0.30	4	68
N07-N08	14	100.00%	1.819	100.00%	0.10	0.30	28	92
N07-N10	16	100.00%	13.815	81.25%	0.00	0.90	37	346
N09-N10	9	77.78%	1.505	77.78%	0.00	0.30	14	33

Table 7 BFOP LATA3.NET

ONEP D_31.NET

		ONEP				
d_31 cut	bw lost	Final rest.level	Time (sec)	Rest.lev 2 sec	sprs	Msg vol
3-2	35	100.00%	4.294	22.86%	151	282
6-4	204	57.84%	16.932	0.00%	790	2356
8-1	176	85.80%	8.388	0.00%	865	542
8-3	147	94.56%	15.073	0.00%	953	1861
9-7	71	39.44%	7.006	12.68%	162	526
11-3	90	100.00%	12.158	0.00%	625	1671
13-3	57	100.00%	8.182	14.04%	249	986
14-11	158	87.97%	15.025	4.43%	606	1526
14-12	112	83.04%	13.418	6.25%	509	1509
15-13	29	100.00%	3.393	0.00%	102	222
15-2	7	100.00%	2.123	0.00%	21	107
16-6	129	100.00%	9.548	0.00%	753	1368
16-14	180	75.00%	17.649	0.00%	795	1977
17-4	206	61.65%	18.777	0.00%	901	2398
20-19	11	100.00%	1.533	100.00%	22	102
21-8	81	96.30%	14.299	0.00%	550	1562
21-17	170	100.00%	16.611	0.00%	1072	2297
21-19	265	100.00%	18.797	0.00%	1304	2600
21-20	12	100.00%	5.765	91.67%	30	210
22-7	89	61.80%	11.524	14.61%	291	874
23-5	45	100.00%	8.084	0.00%	268	321
23-17	24	100.00%	7.962	0.00%	193	273
23-10	33	100.00%	3.139	3.03%	118	209
24-11	138	99.28%	13.687	5.07%	705	1852
24-22	102	100.00%	18.324	0.00%	777	1884
24-12	18	100.00%	6.784	0.00%	69	767
25-6	84	100.00%	14.379	0.00%	743	1600
26-9	52	100.00%	6.127	5.77%	241	454
26-25	24	100.00%	6.852	0.00%	86	228
27-22	79	100.00%	8.822	15.19%	299	1183
27-25	144	94.44%	12.863	0.00%	590	1659
27-9	134	85.82%	9.331	8.96%	369	1106
28-1	92	100.00%	7.717	0.00%	398	482
28-16	80	100.00%	9.678	0.00%	379	1244
29-19	255	98.04%	16.059	0.00%	1155	2406
29-10	153	96.08%	8.353	0.65%	509	884
29-18	81	76.54%	9.112	0.00%	259	480
29-20	16	100.00%	3.658	0.00%	80	107
30-5	39	48.72%	2.637	2.56%	56	206
30-10	120	100.00%	9.115	1.67%	427	855

30-18	23	100.00%	3.404	4.35%	83	317
31-28	37	100.00%	7.653	0.00%	156	825
31-17	64	98.44%	9.114	0.00%	294	1102
1	10	100.00%	2.246	0.00%	30	104
2	0	0.00%	0.000	0.00%	0	0
3	158	52.53%	11.064	5.06%	530	1364
4	178	75.28%	18.005	0.00%	980	2252
5	0	0.00%	0.000	0.00%	0	0
6	133	79.70%	17.644	0.00%	870	2052
7	0	0.00%	0.000	0.00%	0	0
8	173	91.33%	15.815	0.00%	1000	1704
9	64	98.44%	7.694	18.75%	395	771
10	136	86.03%	6.813	0.74%	344	746
11	113	100.00%	14.653	6.19%	701	1907
12	14	100.00%	5.962	0.00%	49	525
13	15	100.00%	3.130	0.00%	60	114
14	179	85.47%	15.749	3.91%	714	1803
15	0	0.00%	0.000	0.00%	0	0
16	167	82.63%	12.686	0.00%	742	1785
17	152	69.08%	16.225	0.00%	824	1981
18	0	0.00%	0.000	0.00%	0	0
19	230	96.52%	16.204	0.00%	1101	2137
20	0	0.00%	0.000	0.00%	0	0
21	171	30.99%	11.263	0.00%	294	1890
22	95	100.00%	11.371	13.68%	533	1234
23	24	100.00%	7.962	0.00%	193	266
24	92	97.83%	14.807	0.00%	624	1592
25	32	100.00%	10.004	28.12%	210	976
26	0	0.00%	0.000	0.00%	0	0
27	166	92.17%	13.732	16.87%	627	1527
28	94	100.00%	10.486	0.00%	439	1134
29	64	98.44%	9.381	1.56%	409	867
30	37	100.00%	4.832	5.41%	158	305
31	23	100.00%	7.341	0.00%	100	802

Table 8 ONEP D_31.NET

DFOP D_31.NET

		DFOP					
d_31 cut	bw lost	Final rest.level	Time (sec)	Rest.lev 2 sec	dly (sec)	sprs	Msg vol
3-2	35	100.00%	4.185	22.86%	0.00	151	283
6-4	204	93.14%	32.653	0.00%	0.00	1716	2689
8-1	176	100.00%	10.565	0.00%	0.50	1101	608
8-3	147	100.00%	11.056	0.00%	0.00	1048	1946
9-7	71	91.55%	15.781	12.68%	0.00	447	732
11-3	90	100.00%	7.202	0.00%	0.00	612	1661
13-3	57	100.00%	4.819	14.04%	0.00	247	982
14-11	158	100.00%	11.163	5.06%	0.00	697	1618
14-12	112	95.54%	13.825	33.04%	0.00	633	1574
15-13	29	100.00%	3.007	0.00%	0.00	102	222
15-2	7	100.00%	2.133	0.00%	0.00	21	107
16-6	129	100.00%	5.700	6.98%	0.00	743	1330
16-14	180	86.11%	29.029	5.00%	0.30	1045	2604
17-4	206	99.03%	24.260	0.00%	0.10	1852	2833
20-19	11	100.00%	1.536	100.00%	0.00	22	102
21-8	81	100.00%	7.557	0.00%	0.10	587	1549
21-17	170	100.00%	10.788	0.59%	0.50	1044	2368
21-19	265	100.00%	23.427	0.38%	0.00	1371	2837
21-20	12	100.00%	5.516	91.67%	0.00	30	210
22-7	89	100.00%	9.498	14.61%	0.10	646	929
23-5	45	100.00%	7.910	0.00%	0.00	268	317
23-17	24	100.00%	7.943	0.00%	0.00	193	278
23-10	33	100.00%	3.079	3.03%	0.00	118	209
24-11	138	100.00%	7.580	12.32%	0.00	680	1850
24-22	102	100.00%	8.519	0.00%	0.00	743	1834
24-12	18	100.00%	4.143	0.00%	0.00	68	765
25-6	84	100.00%	8.912	10.71%	0.10	770	1636
26-9	52	100.00%	5.911	5.77%	0.30	241	456
26-25	24	100.00%	7.012	0.00%	0.00	86	228
27-22	79	100.00%	5.824	25.32%	0.00	299	1161
27-25	144	94.44%	7.703	0.00%	0.50	582	1623
27-9	134	97.01%	9.064	20.90%	0.10	478	1097
28-1	92	100.00%	6.373	0.00%	0.00	398	483
28-16	80	100.00%	4.922	0.00%	0.00	364	1237
29-19	255	100.00%	27.347	0.39%	0.00	1337	2863
29-10	153	100.00%	7.488	0.65%	0.10	588	878
29-18	81	100.00%	7.003	0.00%	0.00	324	468
29-20	16	100.00%	3.694	0.00%	0.00	80	107
30-5	39	100.00%	4.511	2.56%	0.10	171	217
30-10	120	100.00%	7.818	0.83%	0.00	427	855

30-18	23	100.00%	2.964	4.35%	0.00	83	317
31-28	37	100.00%	4.880	0.00%	0.00	155	820
31-17	64	100.00%	5.757	0.00%	0.00	302	1106
1	10	100.00%	2.232	0.00%	0.00	30	104
2	0	0.00%	0.000	0.00%	0.00	0	0
3	158	52.53%	7.202	5.06%	0.00	517	1351
4	178	100.00%	11.404	0.00%	0.00	1449	2252
5	0	0.00%	0.000	0.00%	0.00	0	0
6	133	100.00%	9.665	6.77%	0.00	1092	2097
7	0	0.00%	0.000	0.00%	0.00	0	0
8	173	100.00%	10.782	0.00%	0.50	1163	1838
9	64	100.00%	26.756	18.75%	0.30	401	865
10	136	100.00%	16.160	0.74%	0.10	492	806
11	113	100.00%	7.607	15.04%	0.00	668	1895
12	14	100.00%	4.108	0.00%	0.00	48	523
13	15	100.00%	3.007	0.00%	0.00	60	114
14	179	97.77%	13.120	3.91%	0.50	941	2071
15	0	0.00%	0.000	0.00%	0.00	0	0
16	167	100.00%	29.212	5.39%	0.10	1122	2342
17	152	72.37%	11.390	0.00%	0.00	764	2800
18	0	0.00%	0.000	0.00%	0.00	0	0
19	230	100.00%	24.357	0.43%	0.00	1160	2254
20	0	0.00%	0.000	0.00%	0.00	0	0
21	171	38.60%	12.699	0.58%	0.30	512	4295
22	95	100.00%	7.475	13.68%	0.00	527	1248
23	24	100.00%	7.943	0.00%	0.00	193	271
24	92	100.00%	7.579	0.00%	0.00	625	1576
25	32	100.00%	7.737	28.13%	0.00	200	991
26	0	0.00%	0.000	0.00%	0.00	0	0
27	166	98.19%	13.402	16.87%	0.00	694	1813
28	94	100.00%	4.880	0.00%	0.00	402	1081
29	64	100.00%	7.003	1.56%	0.00	421	889
30	37	100.00%	4.383	5.41%	0.00	158	305
31	23	100.00%	4.880	0.00%	0.00	99	801

Table 9 DFOP D_31.NET

BFOP D_31.NET

		BFOP						
d_31 cut	bw lost	Final rest.level	Time (sec)	Rest.lev 2 sec	dly (sec)	TTL (sec)	sprs	Msg vol
3-2	35	100.00%	4.185	22.86%	0.00	2.00	151	484
6-4	204	64.71%	19.481	0.00%	0.00	2.50	883	11318
6-4	204	93.14%	203.962	0.00%	0.30	4.00	1672	34305
8-1	176	100.00%	13.418	0.00%	0.10	4.00	1069	2615
8-3	147	95.24%	27.748	0.00%	0.10	2.50	909	7595
8-3	147	100.00%	127.110	0.00%	0.00	4.00	1018	19002
9-7	71	90.14%	15.560	0.00%	0.50	2.50	448	3099
9-7	71	100.00%	24.602	12.68%	0.20	4.00	563	8879
11-3	90	100.00%	13.438	0.00%	0.00	2.50	612	6336
13-3	57	100.00%	4.919	14.04%	0.10	1.50	247	1716
14-11	158	100.00%	7.937	5.06%	0.10	2.00	714	2772
14-12	112	89.29%	17.122	7.14%	0.10	2.50	514	6477
14-12	112	100.00%	84.959	7.14%	0.10	4.00	652	20418
15-13	29	100.00%	3.007	0.00%	0.00	1.00	102	107
15-2	7	100.00%	2.133	0.00%	0.00	1.00	21	37
16-6	129	100.00%	9.864	6.98%	0.10	2.00	737	2761
16-14	180	86.67%	39.010	5.00%	0.20	2.50	935	7322
17-4	206	73.30%	25.240	0.00%	0.00	2.50	1014	11436
17-4	206	98.06%	98.945	0.00%	0.00	4.00	1760	34627
20-19	11	100.00%	1.536	100.00%	0.00	1.00	22	54
21-8	81	100.00%	11.143	0.00%	0.10	2.50	535	5727
21-17	170	100.00%	24.503	0.59%	0.10	2.50	1009	7997
21-19	265	99.62%	20.085	0.38%	0.50	2.50	1248	9866
21-19	265	100.00%	41.071	0.38%	0.10	4.00	1260	30795
21-20	12	100.00%	5.516	91.67%	0.00	2.00	30	350
22-7	89	98.88%	7.590	14.61%	0.10	2.50	549	2424
22-7	89	100.00%	7.780	14.61%	0.10	4.00	559	8402
23-5	45	84.44%	6.460	0.00%	0.00	2.50	191	942
23-5	45	100.00%	13.158	0.00%	0.10	4.00	268	3707
23-17	24	100.00%	7.942	0.00%	0.00	2.50	193	889
23-10	33	100.00%	3.079	3.03%	0.00	1.00	118	83
24-11	138	99.28%	9.482	12.32%	0.00	2.50	670	6881
24-11	138	100.00%	9.606	12.32%	0.00	4.00	680	17858
24-22	102	98.04%	14.996	0.00%	0.10	2.50	703	6930
24-22	102	100.00%	30.834	0.00%	0.00	4.00	731	20330
24-12	18	100.00%	4.243	0.00%	0.10	1.50	68	1033
25-6	84	75.00%	14.220	10.71%	0.00	2.50	482	7820
25-6	84	100.00%	193.687	10.71%	0.00	4.00	771	22446
26-9	52	100.00%	6.090	5.77%	0.20	2.00	261	1148
26-25	24	100.00%	7.007	0.00%	0.00	2.50	86	1422

27-22	79	100.00%	5.900	25.32%	0.10	2.00	299	2909
27-25	144	97.92%	51.745	6.25%	0.10	2.50	645	7668
27-9	134	98.51%	15.486	20.90%	0.10	2.50	488	4881
28-1	92	100.00%	6.371	0.00%	0.00	2.00	392	455
28-16	80	100.00%	5.018	0.00%	0.10	1.50	364	1379
29-19	255	100.00%	32.563	0.39%	0.50	2.50	1228	10801
29-10	153	100.00%	14.960	0.65%	0.10	2.00	553	1953
29-18	81	100.00%	6.999	0.00%	0.00	2.50	324	1773
29-20	16	100.00%	3.694	0.00%	0.00	1.00	80	57
30-5	39	100.00%	5.740	2.56%	0.00	1.50	171	125
30-10	120	100.00%	17.802	1.67%	0.10	2.50	425	4295
30-18	23	100.00%	2.962	4.35%	0.00	1.00	83	124
31-28	37	100.00%	4.980	0.00%	0.10	1.50	155	1057
31-17	64	100.00%	5.373	0.00%	0.10	1.50	297	1391
1	10	100.00%	2.232	0.00%	0.00	1.00	30	42
2	0	0.00%	0.000	0.00%	0.00	2.50	0	0
3	158	52.53%	11.206	5.06%	0.10	2.50	517	5291
4	178	79.21%	17.924	0.00%	0.10	2.50	878	10068
4	178	100.00%	119.510	0.00%	0.20	4.00	1408	31167
5	0	0.00%	0.000	0.00%	0.00	0.00	0	0
6	133	84.21%	23.176	6.77%	0.20	2.50	805	7726
6	133	100.00%	114.919	6.77%	0.30	4.00	1099	18925
7	0	0.00%	0.000	0.00%	0.00	0.00	0	0
8	173	95.95%	22.525	0.00%	0.10	2.50	1028	5568
8	173	100.00%	107.539	0.00%	0.30	4.00	1115	12513
9	64	100.00%	6.466	18.75%	0.20	2.00	401	1983
10	136	100.00%	26.075	0.74%	0.10	2.50	494	3136
11	113	99.12%	10.251	15.04%	0.10	2.50	658	6000
11	113	100.00%	11.634	15.04%	0.10	4.00	668	15243
12	14	100.00%	4.208	0.00%	0.10	1.50	48	792
13	15	100.00%	3.007	0.00%	0.00	1.00	60	69
14	179	92.74%	12.580	20.11%	0.00	2.50	797	4822
14	179	100.00%	52.893	4.47%	0.00	4.00	952	12634
15	0	0.00%	0.000	0.00%	0.00	2.50	0	0
16	167	85.03%	20.095	5.39%	0.10	2.50	788	5938
16	167	100.00%	101.931	5.39%	0.10	4.00	1102	12719
17	152	72.37%	14.471	0.00%	0.10	2.50	688	6428
18	0	0.00%	0.000	0.00%	0.00	2.50	0	0
19	230	100.00%	16.571	0.58%	0.10	2.50	1150	8743
20	0	0.00%	0.000	0.00%	0.00	2.50	0	0
21	171	38.60%	16.571	0.58%	0.10	2.50	461	9264
22	95	98.95%	9.644	13.68%	0.10	2.50	517	4406
22	95	100.00%	11.091	13.68%	0.10	4.00	527	13041
23	24	100.00%	7.942	0.00%	0.00	2.50	193	760
24	92	98.91%	10.450	0.00%	0.30	2.50	615	5432
24	92	100.00%	11.900	0.00%	0.30	4.00	625	13731

25	32	100.00%	7.837	28.12%	0.10	2.50	198	5089
26	0	0.00%	0.000	0.00%	0.00	2.50	0	0
27	166	100.00%	15.281	16.87%	0.00	2.50	683	6526
28	94	100.00%	4.982	0.00%	0.10	1.50	402	1097
29	64	100.00%	10.049	1.56%	0.00	2.50	467	3488
30	37	100.00%	4.823	5.41%	0.00	2.50	158	929
31	23	100.00%	4.980	0.00%	0.10	1.50	99	1015

Table 10 BFOP D_31.NET

ONEP DIS5_63.NET

		ONEP				
	bw	Final	Time	Rest.lev		Msg
cut	lost	rest.level	(sec)	2 sec	sprs	vol
2-1	123	91.06%	12.863	4.07%	536	1154
4-3	129	97.67%	16.277	0.00%	723	2392
10-8	549	34.43%	34.295	0.00%	1684	8900
13-2	25	100.00%	5.208	20.00%	82	524
14-5	60	100.00%	6.415	23.33%	207	852
15-1	278	93.53%	15.925	0.00%	1443	2401
15-4	217	83.87%	27.338	0.00%	1419	5374
16-13	30	100.00%	2.705	93.33%	62	766
17-13	25	100.00%	4.979	60.00%	72	527
17-9	117	96.58%	22.683	0.00%	1209	5408
17-16	130	95.38%	23.576	0.00%	1154	5031
19-7	276	22.46%	19.313	0.72%	449	4439
21-8	357	56.58%	26.013	0.00%	1651	7078
21-11	258	89.15%	25.545	0.00%	1794	6705
23-4	111	100.00%	13.063	21.62%	517	3059
24-12	135	100.00%	13.512	0.00%	568	2097
25-20	316	84.18%	25.964	2.53%	1993	7699
25-22	236	60.17%	23.877	3.39%	952	5444
27-8	290	90.34%	26.377	0.00%	2917	6890
27-25	394	53.55%	43.103	0.00%	2272	10540
28-16	95	100.00%	8.666	0.00%	334	1664
28-17	0	0.00%	0.000	0.00%	0	0
29-10	73	98.63%	5.546	2.74%	290	942
30-23	49	100.00%	10.115	0.00%	199	1912
30-3	18	100.00%	2.363	0.00%	54	226
32-26	158	100.00%	21.962	0.00%	1235	4113
32-20	158	98.73%	19.974	0.00%	1264	3413
33-14	469	29.64%	42.602	0.00%	1440	9039
34-31	57	87.72%	12.688	14.04%	262	1815
35-21	110	100.00%	5.972	0.00%	235	1124
35-11	32	96.88%	5.553	0.00%	78	920
35-29	37	100.00%	4.640	0.00%	148	704
36-31	349	91.12%	33.938	0.86%	2128	9812
39-33	329	53.80%	32.719	0.61%	1796	8974
40-36	21	100.00%	2.619	0.00%	63	224
41-36	341	91.50%	33.447	0.88%	2296	10325
42-40	144	93.06%	13.738	0.69%	1110	3167
42-39	336	48.81%	28.685	0.60%	1463	8826

42-41	263	76.81%	28.885	0.76%	1851	8501
43-9	33	100.00%	1.986	100.00%	66	222
44-15	176	100.00%	20.858	0.00%	1319	4406
44-43	156	99.36%	18.362	0.00%	1153	4554
44-28	87	100.00%	9.081	0.00%	269	1850
45-12	212	97.17%	22.347	5.66%	1433	4624
45-37	165	98.18%	16.619	7.27%	991	4715
46-19	57	100.00%	6.573	3.51%	301	1207
47-20	262	95.42%	26.056	4.58%	2019	6890
47-22	36	100.00%	12.461	0.00%	233	2425
48-18	240	89.17%	16.519	5.00%	1059	3925
49-37	31	100.00%	7.556	0.00%	130	1393
49-24	24	100.00%	2.977	0.00%	74	456
50-11	282	87.23%	28.797	0.00%	2920	6977
50-48	283	92.58%	89.125	0.00%	3259	7539
51-6	87	26.44%	8.821	0.00%	114	997
52-18	81	100.00%	12.900	0.00%	244	1630
53-18	164	93.29%	14.136	7.32%	793	2950
53-12	155	95.48%	14.535	0.00%	1068	2759
53-52	56	100.00%	10.884	0.00%	201	930
54-6	86	97.67%	11.649	0.00%	536	647
54-46	64	96.88%	11.574	0.00%	435	644
54-51	6	100.00%	2.906	16.67%	19	456
55-46	96	80.21%	19.894	1.04%	714	1838
55-33	78	97.44%	19.662	1.28%	748	1979
56-1	47	100.00%	7.058	0.00%	246	827
56-26	176	99.43%	21.583	0.00%	1360	4174
56-4	145	100.00%	22.065	2.07%	976	3673
56-3	4	100.00%	1.397	100.00%	8	217
57-37	151	82.78%	16.155	7.95%	801	4423
57-18	66	100.00%	10.947	6.06%	291	2042
57-48	123	95.12%	17.843	9.76%	612	4199
58-43	166	87.95%	18.836	0.00%	1163	4759
58-41	312	84.62%	59.712	0.00%	1776	7887
58-42	29	100.00%	12.434	0.00%	233	1574
58-9	115	93.91%	24.381	0.00%	1148	5212
59-1	162	98.15%	11.089	0.00%	1028	2169
59-38	76	64.47%	13.182	0.00%	351	2735
59-27	213	96.24%	28.271	0.00%	2051	7203
59-16	81	95.06%	20.160	0.00%	1007	4253
60-47	230	94.78%	34.784	0.00%	2595	6901
60-45	240	95.42%	33.684	0.00%	2795	6773
61-31	308	95.78%	30.342	0.97%	2014	9085

61-7	448	13.84%	18.788	0.45%	356	5358
61-40	136	86.76%	13.741	0.00%	1039	2830
61-34	88	42.05%	3.591	37.50%	86	491
62-19	219	71.69%	16.032	0.00%	992	4404
62-51	175	84.00%	12.888	0.00%	997	2705
62-34	44	100.00%	4.839	0.00%	202	928
63-14	376	57.71%	34.015	0.27%	1939	8802
63-10	533	21.76%	33.617	0.00%	1224	8246
63-5	43	97.67%	7.098	4.65%	183	465
64-38	191	32.98%	15.557	0.00%	433	3496
64-33	168	94.64%	17.327	0.00%	1183	4571
1	39	100.00%	13.045	0.00%	266	1461
2	23	100.00%	5.000	21.74%	75	286
3	0	0.00%	0.000	0.00%	0	0
4	298	51.68%	20.864	1.01%	647	5178
5	0	0.00%	0.000	0.00%	0	0
6	0	0.00%	0.000	0.00%	0	0
7	219	31.51%	18.561	0.91%	501	4145
8	573	50.44%	36.669	0.00%	2714	9548
9	107	89.72%	24.470	0.00%	1017	4764
10	229	74.24%	36.048	0.00%	2361	8231
11	199	89.45%	26.061	0.00%	2085	6247
12	65	100.00%	13.421	0.00%	345	2064
13	13	100.00%	5.151	38.46%	45	491
14	420	37.14%	37.927	0.00%	1618	8461
15	329	82.37%	28.979	0.00%	1615	5577
16	80	100.00%	21.454	0.00%	1075	4546
17	111	94.59%	22.030	0.00%	1154	5116
18	160	93.75%	16.396	8.12%	1008	3379
19	241	20.33%	14.648	0.83%	294	4022
20	235	95.74%	27.328	5.11%	1983	7158
21	299	64.21%	23.897	0.00%	1572	6268
22	20	100.00%	11.713	0.00%	149	1937
23	33	100.00%	9.422	0.00%	151	1677
24	22	100.00%	2.437	0.00%	66	225
25	384	78.91%	32.393	2.08%	2462	9129
26	149	98.66%	19.408	0.00%	1114	3550
27	406	78.08%	40.799	0.00%	3478	10955
28	83	100.00%	9.522	0.00%	260	1641
29	0	0.00%	0.000	0.00%	0	0
30	0	0.00%	0.000	0.00%	0	0
31	318	87.11%	32.752	0.94%	1743	9325
32	132	98.48%	18.963	0.00%	985	3282

33	300	86.33%	42.952	0.33%	3028	9027
34	0	0.00%	0.000	0.00%	0	0
35	4	100.00%	4.160	0.00%	16	663
36	289	95.16%	34.187	1.04%	2009	9263
37	166	80.12%	15.848	7.23%	739	4272
38	76	64.47%	13.182	0.00%	351	2734
39	309	58.58%	30.404	0.65%	1881	8581
40	106	100.00%	15.127	0.00%	1056	2329
41	226	93.36%	34.627	1.33%	1929	10221
42	386	53.89%	32.677	0.52%	1606	9167
43	113	99.12%	15.841	0.00%	938	3683
44	198	99.49%	19.880	0.00%	1257	4505
45	199	84.42%	24.218	6.03%	1706	5533
46	81	95.06%	19.835	1.23%	740	1538
47	209	96.17%	32.420	0.00%	2342	6461
48	118	88.14%	20.346	10.17%	881	4749
49	0	0.00%	0.000	0.00%	0	0
50	218	90.37%	27.571	0.00%	2268	6728
51	86	100.00%	9.193	0.00%	435	1420
52	0	0.00%	0.000	0.00%	0	0
53	152	96.71%	14.052	0.00%	1056	2637
54	64	96.88%	11.574	0.00%	435	637
55	39	100.00%	14.249	0.00%	475	872
56	175	90.29%	21.516	0.00%	1239	4056
57	151	82.12%	16.253	7.95%	786	4352
58	233	72.96%	30.009	0.00%	2116	7618
59	248	85.89%	28.190	0.00%	2187	7065
60	205	92.20%	31.937	0.00%	2254	6242
61	236	42.37%	18.183	0.85%	783	4446
62	169	91.72%	13.823	0.00%	993	3112
63	391	39.13%	36.216	0.00%	1805	7637
64	63	85.71%	13.958	0.00%	373	3092

Table 11 ONEP DIS5_64.NET

DFOP DIS5_64.NET

		DFOP					
	bw	Final	Time	Rest.level	dly		Msg
cut	lost	rest.level	(sec)	2 sec	(sec)	sprs	vol
2-1	123	100.00%	34.762	0.00%	0.50	666	1285
4-3	129	100.00%	11.294	29.46%	0.00	729	2336
10-8	549	59.74%	54.729	0.00%	0.50	4443	22964
13-2	25	100.00%	5.001	20.00%	0.00	82	524
14-5	60	100.00%	5.629	23.33%	0.00	207	864
15-1	278	100.00%	76.950	1.44%	0.50	1907	4627
15-4	217	100.00%	64.044	13.36%	0.00	1579	7301
16-13	30	100.00%	2.051	93.33%	0.00	62	766
17-13	25	100.00%	4.960	60.00%	0.00	72	521
17-9	117	100.00%	14.862	0.00%	0.10	1208	5452
17-16	130	100.00%	13.719	20.00%	0.00	1134	5003
19-7	276	37.68%	12.402	0.72%	0.00	1084	7740
21-8	357	98.32%	106.343	0.00%	0.50	3730	12855
21-11	258	99.61%	15.791	0.00%	0.50	2041	7165
23-4	111	100.00%	7.771	21.62%	0.00	511	3044
24-12	135	100.00%	12.068	0.00%	0.00	557	2079
25-20	316	100.00%	37.307	4.11%	0.00	2204	8022
25-22	236	86.86%	102.090	0.00%	0.50	1690	7987
27-8	290	100.00%	12.083	0.00%	0.00	3002	6767
27-25	394	99.75%	69.656	0.25%	0.30	5212	14515
28-16	95	100.00%	7.817	0.00%	0.10	331	1650
28-17	0	0.00%	0.000	0.00%	0.00	0.00	0
29-10	73	100.00%	7.031	2.74%	0.00	300	937
30-23	49	100.00%	7.068	0.00%	0.00	198	1910
30-3	18	100.00%	2.353	0.00%	0.00	54	226
32-26	158	98.73%	10.248	0.63%	0.50	1160	4079
32-20	158	97.47%	9.933	0.63%	0.10	1179	3821
33-14	469	60.55%	45.584	0.00%	0.10	3996	17182
34-31	57	100.00%	15.357	14.04%	0.10	354	1883
35-21	110	100.00%	4.140	0.00%	0.00	235	1124
35-11	32	100.00%	5.524	78.13%	0.00	85	919
35-29	37	100.00%	4.190	0.00%	0.00	148	700
36-31	349	98.28%	51.488	0.86%	0.10	2443	10562
39-33	329	100.00%	28.764	0.61%	0.50	3548	9849
40-36	21	100.00%	2.404	0.00%	0.00	63	224
41-36	341	100.00%	14.189	0.88%	0.10	2569	10437
42-40	144	100.00%	11.293	2.08%	0.00	1190	3143
42-39	336	98.51%	67.253	0.60%	0.00	3626	10662
42-41	263	100.00%	48.490	1.90%	0.10	2223	9506
43-9	33	100.00%	1.979	100.00%	0.00	66	222

44-15	176	100.00%	10.062	0.00%	0.00	1143	4317
44-43	156	100.00%	10.006	0.00%	0.00	1160	4521
44-28	87	100.00%	7.820	0.00%	0.10	269	1854
45-12	212	98.58%	16.400	8.49%	0.00	1406	5312
45-37	165	100.00%	10.325	10.91%	0.10	934	4707
46-19	57	100.00%	6.348	3.51%	0.00	295	1222
47-20	262	100.00%	13.729	5.34%	0.00	2044	6878
47-22	36	100.00%	10.203	0.00%	0.00	230	2413
48-18	240	95.83%	54.602	5.42%	0.10	1260	4409
49-37	31	100.00%	7.015	0.00%	0.00	130	1393
49-24	24	100.00%	2.701	0.00%	0.00	74	456
50-11	282	100.00%	51.131	0.00%	0.00	3510	8116
50-48	283	97.53%	51.359	0.00%	0.30	3464	9083
51-6	87	100.00%	8.432	0.00%	0.00	439	1017
52-18	81	100.00%	12.320	0.00%	0.00	244	1628
53-18	164	100.00%	11.201	7.32%	0.10	841	2902
53-12	155	99.35%	14.942	0.00%	0.10	958	2747
53-52	56	100.00%	14.697	42.86%	0.00	205	973
54-6	86	100.00%	12.848	0.00%	0.00	570	699
54-46	64	100.00%	12.605	0.00%	0.00	469	650
54-51	6	100.00%	2.732	16.67%	0.00	19	456
55-46	96	100.00%	20.169	1.04%	0.50	1034	1960
55-33	78	100.00%	20.524	1.28%	0.00	771	1935
56-1	47	100.00%	6.948	0.00%	0.00	246	827
56-26	176	98.86%	141.658	0.57%	0.50	1338	5855
56-4	145	100.00%	8.921	11.72%	0.00	851	3635
56-3	4	100.00%	1.400	100.00%	0.00	8	217
57-37	151	98.68%	11.287	11.92%	0.10	967	4670
57-18	66	100.00%	10.072	36.36%	0.00	282	2043
57-48	123	100.00%	20.335	21.14%	0.10	690	4313
58-43	166	90.36%	9.915	5.42%	0.00	1124	4757
58-41	312	100.00%	50.998	0.00%	0.50	2357	9366
58-42	29	100.00%	7.922	0.00%	0.00	223	1554
58-9	115	100.00%	26.685	3.48%	0.10	1167	5383
59-1	162	100.00%	8.091	0.00%	0.10	1070	2200
59-38	76	64.47%	9.230	0.00%	0.00	343	2640
59-27	213	100.00%	50.296	0.00%	0.00	2058	7582
59-16	81	100.00%	13.719	0.00%	0.00	983	4263
60-47	230	98.70%	14.657	0.00%	0.00	2662	7306
60-45	240	98.75%	14.657	0.00%	0.00	2817	7107
61-31	308	96.75%	94.567	0.97%	0.00	2025	9999
61-7	448	23.21%	10.729	0.45%	0.00	943	19637
61-40	136	94.12%	11.293	0.00%	0.00	1130	2831
61-34	88	42.05%	3.073	0.00%	0.10	86	1277
62-19	219	73.06%	10.019	0.00%	0.00	1017	5854
62-51	175	100.00%	11.708	0.00%	0.00	1208	2747

62-34	44	100.00%	4.380	0.00%	0.00	202	928
63-14	376	93.09%	36.380	0.27%	0.30	4101	11620
63-10	533	51.59%	46.513	0.00%	0.50	4170	17597
63-5	43	100.00%	8.200	4.65%	0.00	194	480
64-38	191	32.98%	9.230	0.00%	0.00	429	3393
64-33	168	100.00%	9.230	0.00%	0.00	1236	4540
1	39	100.00%	7.802	0.00%	0.00	256	1427
2	23	100.00%	5.001	21.74%	0.00	75	286
3	0	0.00%	0.000	0.00%	0.00	0.00	0
4	298	54.36%	8.564	17.79%	0.00	678	23974
5	0	0.00%	0.000	0.00%	0.00	0.00	0
6	0	0.00%	0.000	0.00%	0.00	0.00	0
7	219	47.49%	10.809	0.91%	0.00	975	6573
8	573	68.06%	44.373	0.00%	0.00	4514	17132
9	107	100.00%	14.646	0.00%	0.00	1116	4802
10	229	100.00%	19.386	0.87%	0.10	3132	8634
11	199	100.00%	51.231	0.00%	0.10	2309	7314
12	65	100.00%	14.697	0.00%	0.00	347	2084
13	13	100.00%	4.952	38.46%	0.00	45	491
14	420	67.86%	37.445	0.00%	0.50	4006	16979
15	329	100.00%	80.499	4.56%	0.30	2573	9321
16	80	100.00%	13.719	0.00%	0.00	965	4480
17	111	100.00%	13.719	0.00%	0.00	1144	5043
18	160	91.25%	20.043	8.13%	0.00	1039	3404
19	241	43.57%	14.215	0.83%	0.00	1197	7187
20	235	100.00%	13.839	6.38%	0.00	1988	7141
21	299	95.32%	103.487	0.00%	0.10	2686	9549
22	20	100.00%	10.105	0.00%	0.00	148	2057
23	33	100.00%	7.068	0.00%	0.00	150	1675
24	22	100.00%	2.425	0.00%	0.00	66	225
25	384	100.00%	66.015	3.39%	0.00	3790	13232
26	149	100.00%	9.848	0.67%	0.10	1082	3554
27	406	99.01%	57.950	0.00%	0.30	5077	17993
28	83	100.00%	7.820	0.00%	0.10	249	1619
29	0	0.00%	0.000	0.00%	0.00	0.00	0
30	0	0.00%	0.000	0.00%	0.00	0.00	0
31	318	98.43%	50.976	0.94%	0.10	2220	9854
32	132	100.00%	9.162	0.76%	0.10	942	3250
33	300	95.33%	22.531	0.33%	0.00	3454	10314
34	0	0.00%	0.000	0.00%	0.00	0.00	0
35	4	100.00%	4.190	0.00%	0.00	16	663
36	289	100.00%	14.175	1.04%	0.00	2130	9251
37	166	98.80%	10.722	7.83%	0.50	1023	4517
38	76	64.47%	9.230	0.00%	0.00	343	2639
39	309	99.35%	24.717	0.65%	0.50	3170	9244
40	106	100.00%	11.185	0.00%	0.00	1026	2297

41	226	100.00%	14.109	2.65%	0.00	2052	10253
42	386	66.32%	12.300	0.78%	0.50	1943	18184
43	113	100.00%	9.915	0.00%	0.00	930	3632
44	198	100.00%	10.010	0.00%	0.00	1232	4466
45	199	98.49%	47.445	6.53%	0.50	2052	6626
46	81	100.00%	19.589	1.23%	0.00	769	1604
47	209	98.56%	14.657	0.00%	0.00	2393	6868
48	118	100.00%	50.406	11.86%	0.00	1043	5502
49	0	0.00%	0.000	0.00%	0.00	0.00	0
50	218	100.00%	52.073	0.00%	0.10	2501	7941
51	86	100.00%	8.432	0.00%	0.00	435	1424
52	0	0.00%	0.000	0.00%	0.00	0.00	0
53	152	100.00%	92.423	0.00%	0.00	942	3024
54	64	100.00%	12.605	0.00%	0.00	469	643
55	39	100.00%	14.441	0.00%	0.00	475	866
56	175	91.43%	11.418	0.57%	0.10	1284	6711
57	151	98.68%	11.287	11.92%	0.10	967	4562
58	233	78.54%	19.881	0.00%	0.10	2364	10334
59	248	89.11%	50.812	0.00%	0.00	2130	7125
60	205	98.54%	14.657	0.00%	0.00	2352	6603
61	236	44.49%	11.757	0.85%	0.10	824	11425
62	169	100.00%	11.605	0.00%	0.00	1102	3135
63	391	71.10%	38.687	0.00%	0.50	4091	17871
64	63	100.00%	9.230	0.00%	0.00	429	3058

Table 12 DFOP DIS5_64.NET

BFOP DIS5_64.NET

		BFOP						
	bw	Final	Time	Rest.lev	dly	TTL		Msg
cut	lost	rest.level	(sec)	2 sec	(sec)	(sec)	sprs	vol
2-1	123	99.19%	16.351	4.07%	0.00	2.50	597	2376
4-3	129	82.17%	23.242	29.46%	0.00	2.50	441	7895
10-8	549	30.97%	15.250	0.00%	1.00	2.50	1148	27020
		37.89%	48.6852	0.00%	1.00	3.00	1515	57229
13-2	25	100.00%	5.012	20.00%	0.00	2.00	82	754
14-5	60	100.00%	5.629	23.33%	0.00	2.00	207	1828
15-1	278	96.76%	14.201	10.43%	0.00	2.50	1207	5067
		98.92%	24.559	10.43%	0.00	3.00	1271	10437
15-4	217	76.04%	25.537	13.36%	0.10	2.50	802	17606
		86.18%	28.674	13.36%	0.00	3.00	1034	32515
16-13	30	100.00%	2.051	93.33%	0.00	1.00	62	216
17-13	25	100.00%	4.930	60.00%	0.00	1.50	72	423
17-9	117	52.99%	23.894	0.00%	0.00	2.50	448	19962
		65.81%	76.682	0.00%	0.00	3.00	630	37559
17-16	130	58.46%	7.913	0.00%	0.10	2.50	394	18786
		69.23%	13.462	20.00%	0.00	3.00	560	38071
19-7	276	37.32%	14.098	0.72%	0.10	2.50	774	20875
		37.68%	20.103	0.72%	0.00	3.00	807	46665
21-8	357	43.70%	14.446	0.00%	0.00	2.50	725	18539
		60.22%	48.964	0.00%	0.00	3.00	1374	39930
21-11	258	71.32%	15.339	0.00%	0.30	2.50	1003	19317
		79.07%	38.478	5.04%	0.00	3.00	1237	37794
23-4	111	100.00%	8.750	21.62%	0.00	2.50	511	15585
24-12	135	97.78%	6.350	0.00%	0.00	2.50	513	6824
25-20	316	73.73%	10.753	4.11%	0.00	2.50	1123	23179
		82.59%	45.803	4.11%	0.00	3.00	1438	46297
25-22	236	66.53%	14.103	5.08%	0.00	2.50	837	22544
		68.64%	14.174	5.08%	0.00	3.00	900	58030
27-8	290	41.38%	14.828	0.00%	0.00	2.50	1010	17294
		60.00%	19.746	0.00%	0.00	3.00	1570	34526
27-25	394	40.61%	9.256	0.25%	0.00	2.50	1100	26769
		58.88%	21.152	0.25%	0.00	3.00	1923	59345
28-16	95	100.00%	10.405	0.00%	0.00	2.50	318	6504
28-17	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
29-10	73	100.00%	6.994	2.74%	0.00	2.50	300	2869
30-23	49	100.00%	7.059	0.00%	0.00	2.50	198	9631
30-3	18	100.00%	2.353	0.00%	0.00	1.00	54	75
32-26	158	75.32%	7.790	0.63%	0.00	2.50	728	14597

		93.67%	44.252	0.63%	0.00	3.00	1041	30597
32-20	158	74.68%	7.379	0.63%	0.00	2.50	750	11680
		91.14%	44.153	0.63%	0.00	3.00	1044	24215
33-14	469	25.16%	7.746	0.00%	0.00	2.50	895	24939
		35.61%	43.481	0.00%	0.00	3.00	1444	60928
34-31	57	82.46%	6.936	14.04%	0.00	2.50	198	9086
35-21	110	100.00%	4.140	0.00%	0.00	1.50	235	754
35-11	32	100.00%	5.524	78.13%	0.00	2.00	85	980
35-29	37	100.00%	4.164	0.00%	0.00	1.50	148	344
36-31	349	78.51%	9.180	0.86%	0.00	2.50	1421	33429
		81.38%	28.233	0.86%	0.00	3.00	1548	72836
39-33	329	59.27%	9.819	0.61%	0.00	2.50	1452	22153
		68.69%	34.840	0.61%	0.00	3.00	1842	51952
40-36	21	100.00%	2.404	0.00%	0.00	1.00	63	50
41-36	341	74.49%	8.009	0.88%	0.00	2.50	1311	30008
		80.35%	32.752	0.88%	0.00	3.00	1540	58700
42-40	144	71.53%	11.109	2.08%	0.00	2.50	688	8882
		84.03%	11.346	2.08%	0.00	3.00	894	20086
42-39	336	58.04%	11.944	0.60%	0.00	2.50	1465	21355
		69.35%	100.902	0.60%	0.00	3.00	1928	51423
42-41	263	73.00%	27.979	1.90%	0.10	2.50	1036	24225
		80.23%	86.229	1.90%	0.00	3.00	1257	52324
43-9	33	100.00%	1.979	100.00%	0.00	1.00	66	69
44-15	176	85.23%	10.924	0.00%	0.10	2.50	853	11826
		99.43%	13.9147	0.00%	0.10	3.00	1063	24794
44-43	156	82.05%	7.437	0.00%	0.10	2.50	820	12473
		96.15%	11.026	0.00%	0.00	3.00	1063	21210
44-28	87	100.00%	7.715	0.00%	0.00	2.50	273	8103
45-12	212	79.72%	7.730	8.49%	0.00	2.50	791	14476
45-37	165	90.30%	14.429	10.91%	0.00	2.50	756	16803
		95.76%	14.506	10.91%	0.00	3.00	864	37734
46-19	57	100.00%	5.004	3.51%	0.00	1.50	290	649
47-20	262	69.85%	10.753	5.34%	0.00	2.50	1019	20036
		84.73%	10.813	5.34%	0.00	3.00	1467	46805
47-22	36	91.67%	7.688	0.00%	0.00	2.50	193	11154
		97.22%	8.429	0.00%	0.00	3.00	216	28672
48-18	240	83.75%	13.465	5.42%	0.00	2.50	762	13015
		84.58%	27.158	5.42%	0.00	3.00	786	32429
49-37	31	100.00%	7.011	0.00%	0.00	2.50	130	5516
49-24	24	100.00%	2.701	0.00%	0.00	1.00	74	79
50-11	282	26.95%	14.505	0.00%	0.00	2.50	585	18948
50-48	283	26.86%	14.483	0.00%	0.00	2.50	585	18252
		61.13%	26.086	0.00%	0.00	3.00	1697	42970

51-6	87	98.85%	7.784	0.00%	0.00	2.50	427	4725
		100.00%	8.432	0.00%	0.00	3.00	439	10813
52-18	81	96.30%	5.523	0.00%	0.00	2.50	196	7814
53-18	164	87.20%	6.976	7.32%	0.00	2.50	542	7186
		88.41%	8.201	7.32%	0.00	3.00	564	17721
53-12	155	88.39%	7.420	0.00%	0.00	2.50	668	8846
		89.03%	8.291	0.00%	0.00	3.00	679	23018
53-52	56	92.86%	2.847	42.86%	0.00	2.50	141	3374
54-6	86	84.88%	8.987	0.00%	0.00	2.50	376	1578
		89.53%	9.013	0.00%	0.00	3.00	424	3831
54-46	64	79.69%	6.479	0.00%	0.00	2.50	275	1512
		85.94%	8.491	0.00%	0.00	3.00	323	3607
54-51	6	100.00%	2.732	16.67%	0.00	1.00	19	83
55-46	96	68.75%	6.870	1.04%	0.00	2.50	463	5884
55-33	78	78.21%	7.672	1.28%	0.00	2.50	491	8354
		82.05%	8.760	1.28%	0.00	3.00	527	20878
56-1	47	100.00%	6.737	0.00%	0.00	2.50	231	2736
56-26	176	73.86%	36.733	0.00%	0.50	2.50	823	15260
		94.32%	97.922	0.57%	0.00	3.00	1197	31083
56-4	145	90.34%	7.790	11.72%	0.00	2.50	693	12826
56-3	4	100.00%	1.400	100.00%	0.00	1.00	8	30
57-37	151	86.75%	22.376	11.92%	0.10	2.50	737	16949
		93.38%	15.863	11.92%	0.00	3.00	879	37656
57-18	66	92.42%	7.004	36.36%	0.00	2.50	220	7775
		93.94%	9.213	36.36%	0.00	3.00	233	19234
57-48	123	90.24%	17.946	21.14%	0.10	2.50	509	17667
58-43	166	74.70%	34.931	5.42%	0.10	2.50	833	14062
		86.14%	35.066	4.82%	0.00	3.00	1050	24064
58-41	312	64.74%	15.326	0.00%	0.10	2.50	834	27174
		77.24%	106.887	0.32%	0.00	3.00	1275	57387
58-42	29	96.55%	7.013	0.00%	0.00	2.50	212	4223
		100.00%	7.875	0.00%	0.00	3.00	223	9733
58-9	115	49.57%	13.771	3.48%	0.00	2.50	341	17828
		65.22%	60.021	3.48%	0.00	3.00	560	37574
59-1	162	93.21%	16.297	0.00%	0.00	2.50	932	5020
		100.00%	26.614	0.00%	0.00	3.00	1042	9615
59-38	76	60.53%	7.809	0.00%	0.00	2.50	305	13010
		61.84%	8.573	0.00%	0.00	3.00	317	22859
59-27	213	59.62%	10.463	0.00%	0.00	2.50	889	21038
		75.12%	11.388	0.00%	0.00	3.00	1246	38338
59-16	81	32.10%	7.813	0.00%	0.00	2.50	238	15275
		50.62%	9.254	0.00%	0.00	3.00	415	29954
60-47	230	29.13%	7.566	0.00%	0.10	2.50	565	16353

		56.96%	19.8379	0.00%	0.00	3.00	1234	42153
60-45	240	27.92%	13.032	0.00%	0.10	2.50	563	16344
		47.50%	17.3024	0.00%	0.00	3.00	1055	41375
61-31	308	79.55%	7.781	0.97%	0.00	2.50	1168	33835
61-7	448	23.21%	8.878	0.45%	0.00	2.50	734	20274
61-40	136	76.47%	8.052	0.00%	0.00	2.50	825	9727
		80.15%	9.326	0.00%	0.00	3.00	888	20986
61-34	88	42.05%	4.812	0.00%	0.10	2.50	88	2492
62-19	219	71.69%	11.440	0.00%	0.00	2.50	913	22020
		73.06%	16.922	0.00%	0.00	3.00	950	56232
62-51	175	95.43%	11.982	0.00%	0.30	2.50	1073	12531
		96.00%	13.285	0.00%	0.00	3.00	1130	31760
62-34	44	100.00%	4.378	0.00%	0.00	1.50	202	505
63-14	376	47.34%	11.612	0.27%	0.10	2.50	1097	25933
		61.70%	63.352	0.27%	0.00	3.00	1726	64630
63-10	533	19.70%	7.746	0.00%	0.00	2.50	796	21958
		26.08%	16.732	0.00%	0.00	3.00	1159	48316
63-5	43	97.67%	7.316	4.65%	0.00	2.50	183	319
		100.00%	8.065	4.65%	0.00	3.00	194	616
64-38	191	31.41%	7.809	0.00%	0.00	2.50	391	16258
		31.94%	8.573	0.00%	0.00	3.00	403	30224
64-33	168	98.21%	7.809	0.00%	0.00	2.50	1198	25743
		100.00%	9.295	0.00%	0.00	3.00	1236	59660
1	39	97.44%	7.027	0.00%	0.00	2.50	246	2038
		100.00%	7.398	0.00%	0.00	3.00	256	4306
2	23	100.00%	5.012	21.74%	0.00	2.00	75	406
3	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
4	298	51.68%	8.323	1.01%	0.50	2.50	587	16564
		54.36%	8.561	17.79%	0.00	3.00	678	30264
5	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
6	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
7	219	47.03%	16.112	0.91%	0.10	2.50	757	20167
8	573	31.94%	8.108	0.00%	0.10	2.50	1005	19441
		41.01%	9.31828	0.00%	0.00	3.00	1540	43384
9	107	48.60%	16.918	0.00%	0.00	2.50	358	14833
		62.62%	57.070	0.00%	0.00	3.00	540	28096
10	229	34.06%	7.921	0.87%	0.00	2.50	717	23226
		45.41%	15.181	0.87%	0.00	3.00	1011	45425
11	199	33.67%	14.483	0.00%	0.00	2.50	506	18578
		61.31%	14.567	0.00%	0.00	3.00	1097	41159
12	65	89.23%	6.350	0.00%	0.00	2.50	239	7884
13	13	100.00%	4.952	38.46%	0.00	2.00	45	599

14	420	28.10%	7.746	0.00%	0.00	2.50	895	24138
		39.76%	43.481	0.00%	0.00	3.00	1444	59169
15	329	80.55%	19.864	8.81%	0.00	2.50	1232	13834
		90.27%	28.746	8.81%	0.00	3.00	1568	26814
16	80	31.25%	7.813	0.00%	0.00	2.50	220	13530
		50.00%	9.254	0.00%	0.00	3.00	397	25570
17	111	50.45%	23.774	0.00%	0.00	2.50	394	17390
		63.96%	87.986	0.00%	0.00	3.00	573	32444
18	160	74.38%	8.464	8.13%	0.00	2.50	589	10511
		76.25%	23.826	8.13%	0.00	3.00	624	27446
19	241	37.76%	18.024	0.83%	0.50	2.50	676	19736
		42.74%	26.530	0.83%	0.00	3.00	835	50633
20	235	64.68%	10.753	6.38%	0.00	2.50	921	19438
		78.72%	10.774	6.38%	0.00	3.00	1291	39534
21	299	47.49%	14.409	0.00%	0.00	2.50	607	16007
		62.88%	19.855	0.00%	0.00	3.00	1090	35342
22	20	85.00%	7.688	0.00%	0.00	2.50	111	10070
		95.00%	8.429	0.00%	0.00	3.00	134	25833
23	33	100.00%	7.059	0.00%	0.00	2.50	150	8281
24	22	100.00%	2.425	0.00%	0.00	1.00	66	46
25	384	53.39%	7.322	3.39%	0.00	2.50	1089	26669
		68.49%	71.624	3.39%	0.00	3.00	1729	53992
26	149	80.54%	7.790	0.67%	0.00	2.50	744	12618
		93.96%	44.252	0.67%	0.00	3.00	964	26512
27	406	29.06%	7.404	0.00%	0.00	2.50	826	21043
		54.68%	17.972	0.00%	0.00	3.00	1954	40805
28	83	100.00%	7.718	0.00%	0.00	2.50	253	6398
29	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
30	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
31	318	77.04%	8.733	0.94%	0.10	2.50	1266	32258
		81.76%	8.99278	0.94%	0.00	3.00	1436	67040
32	132	89.39%	7.828	0.76%	0.00	2.50	775	11271
		96.21%	44.314	0.76%	0.00	3.00	881	24150
33	300	39.67%	7.292	0.33%	0.00	2.50	845	21978
		55.33%	34.326	0.33%	0.00	3.00	1360	54202
34	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
35	4	100.00%	4.122	0.00%	0.00	1.50	16	279
36	289	74.39%	7.793	1.04%	0.00	2.50	1139	29099
		79.93%	8.684	1.04%	0.00	3.00	1321	64326
37	166	89.76%	14.709	7.83%	0.30	2.50	808	16186
		92.77%	14.529	10.84%	0.00	3.00	909	36834
38	76	60.53%	7.809	0.00%	0.00	2.50	305	13009
		61.84%	8.573	0.00%	0.00	3.00	317	22858

39	309	58.90%	32.458	0.65%	0.00	2.50	1349	20554
		74.43%	66.530	0.65%	0.00	3.00	1894	50209
40	106	77.36%	7.982	0.00%	0.00	2.50	723	6635
		82.08%	9.297	0.00%	0.00	3.00	786	14513
41	226	62.39%	7.396	2.65%	0.00	2.50	920	24715
		72.12%	8.770	2.65%	0.00	3.00	1168	50934
42	386	56.22%	13.853	1.30%	0.10	2.50	1456	16907
		65.54%	45.8657	1.30%	0.00	3.00	1869	37805
43	113	76.99%	7.316	0.00%	0.00	2.50	640	9174
		94.69%	8.726	0.00%	0.00	3.00	859	15614
44	198	85.86%	10.924	0.00%	0.10	2.50	918	10732
		99.49%	13.9147	0.00%	0.00	3.00	1219	20054
45	199	46.23%	7.719	9.05%	0.00	2.50	536	15437
		55.28%	21.725	9.05%	0.00	3.00	762	39261
46	81	80.25%	7.423	1.23%	0.00	2.50	479	5067
		81.48%	8.756	1.23%	0.00	3.00	491	13914
47	209	32.06%	7.566	0.00%	0.10	2.50	565	14983
		59.81%	17.0617	0.00%	0.00	3.00	1242	38118
48	118	66.95%	14.372	11.86%	0.00	2.50	493	15529
		76.27%	14.413	11.86%	0.00	3.00	614	36181
49	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
50	218	34.86%	14.483	0.00%	0.00	2.50	585	18154
		64.68%	14.700	0.00%	0.00	3.00	1283	42326
51	86	98.84%	7.784	0.00%	0.00	2.50	423	5972
		100.00%	8.432	0.00%	0.00	3.00	435	14646
52	0	0.00%	0.000	0.00%	0.00	0.00	0.00	0
53	152	88.16%	9.064	0.00%	0.00	2.50	652	8745
54	64	79.69%	6.479	0.00%	0.00	2.50	275	1461
		85.94%	8.491	0.00%	0.00	3.00	323	3497
55	39	56.41%	6.807	0.00%	0.00	2.50	198	1909
		64.10%	8.676	0.00%	0.00	3.00	234	5031
56	175	80.00%	13.029	0.57%	0.00	2.50	934	11378
		90.86%	44.252	0.57%	0.00	3.00	1148	22577
57	151	88.08%	14.909	8.61%	0.50	2.50	754	16316
		94.04%	14.9087	8.61%	0.00	3.00	862	37022
58	233	25.75%	38.377	0.00%	0.00	2.50	559	17332
		45.92%	38.434	0.00%	0.00	3.00	1069	27850
59	248	58.47%	10.519	0.00%	0.00	2.50	1040	15973
		66.94%	23.332	0.00%	0.00	3.00	1278	29722
60	205	32.68%	16.475	0.00%	0.10	2.50	567	15451
		52.68%	17.200	0.00%	0.00	3.00	1040	39043
61	236	44.49%	8.261	0.85%	0.10	2.50	803	14730
62	169	95.27%	12.325	0.00%	0.00	2.50	978	13671

		95.86%	12.339	0.00%	0.00	3.00	990	34265
63	391	26.60%	7.746	0.00%	0.00	2.50	787	20969
		37.85%	50.150	0.00%	0.00	3.00	1277	51465
64	63	95.24%	7.809	0.00%	0.00	2.50	391	13984
		100.00%	9.295	0.00%	0.00	3.00	429	32510

Table 13 BFOP DIS5_64.NET

INDEX

A
Aristotle,3