1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

PIONEERS IN METHYLENE BLUE TESTING SINCE 1974

MICRO

13000 COUNTRY ROAD 15, BUCHHEIM, NY 13027 USA
TEL: 518/461-7887 FAX: 518/461-7887 TLA 518/461-7887

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

A Diagnostic Expert System For Wide Area Networks

by

Robert Dussault, B. Eng.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements
for the degree of
Master of Engineering

Department of Systems and Computer Engineering,
Carleton University,
Ottawa, Ontario, Canada

April 16, 1992

© Robert Dussault

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-75967-4

Canada

The undersigned hereby recommend to the faculty of Graduate
Studies and Research acceptance of the thesis

A Diagnostic Expert System
for Wide Area Networks

submitted by Robert Dussault, B.Eng.
in partial fulfilment of the requirements for
the degree of Master of Engineering

Thesis Supervisor

Thesis Supervisor

Chair, Department of Systems and Computer Engineering

Carleton University

April 16, 1992

Abstract

The complexity and diversity of today's telecommunication systems demand experimentation with new approaches to solving problems in network management because traditional approaches have difficulty coping. Expert systems are emerging as a technology which can provide significant help to solve complex problems. Expert Network Diagnostician (END) has been developed to determine the role of expert system technology within a multi-paradigm approach to network fault diagnosis.

This thesis explores different approaches to network diagnosis, focusing on expert system technology. It describes END, its environment, development tool, evolution and place within the TRIO Pegasus research project - a multi-paradigm approach to network diagnosis. Future research directions are also discussed.

Acknowledgements

I would like to express special thanks to my thesis supervisors, Dr. N. Dawes and Dr. B. Pagurek for their enlightened approach to supervising my thesis research. Thanks also to Bell Northern Research management who provided access to their expert staff and to the Telecommunications Research Institute of Ontario (TRIO) for funding the project. Finally, I thank Danielle Brochu for her continuous help, support and love during the course of my studies. It would not have been the same without her.

Contents

1. Introduction and Organization	1
1.1 Introduction	1
1.2 Objective	2
1.3 Organization	4
2. Approaches to Automatic Fault Diagnosis	5
2.1 Introduction	5
2.2 Decision Tree	5
2.3 Model-based	6
2.4 Probabilistic	6
2.5 Multi-Paradigms	7
2.6 Neural Networks	8
2.7 Case-based Reasoning	10
2.8 Heuristic Search	11
2.9 Knowledge-based Systems	13
3. BNR Network Overview	17
3.1 Architecture	17
3.2 Work Groups	18
3.3 LAN Backbone	19
3.4 BNR Current Network Diagnostic Support	22
3.5 Network Diagnostic Problems	24
4. Pegasus Project	27
4.1 Introduction	27
4.1.1 Objective	27
4.1.2 Scope	28

4.2	Multi-Paradigm Approach	28
4.3	Architecture	28
4.4	Operational Overview	29
4.5	Blackboard	31
4.6	Simulator	31
4.7	Knowledge Sources	32
4.7.1	BANES	32
4.7.2	Neural Network	33
4.7.3	Expert Network Diagnostician (END)	34
4.8	Dynamic Data Structures	35
5.	Problem Descriptions	36
5.1	Introduction	36
5.2	Network Problems Simulation	37
5.3	Common Faults	38
5.3.1	Element - Station Problem Class	39
5.3.2	Element - Router Problem Class	39
5.3.3	Line Problem Class	40
5.3.4	Routing Problem Class	41
5.3.5	Monitoring Problem Class	42
5.4	Complex Problems	42
5.5	END Modules	43
6.	The COSMIC CLIPS Expert System Development Tool	45
6.1	Syntax and Organization	46
6.2	CLIPS Architecture and Reasoning	52
6.3	Truth Maintenance	59
6.4	Integrating CLIPS with External Functions	60
7.	Expert Network Diagnostician (END) Prototype Evolution	62

7.1	Introduction	62
7.1.1	Objective	62
7.1.2	Scope	63
7.2	END Development	63
7.2.1	Phase 1: Automation of Current Procedures	64
7.2.2	Phase 2: Augmenting END Diagnostic Capabilities	70
7.3	Interface to the Blackboard	74
7.4	Points of Interest in the Development of the END Prototype	77
7.4.1	Knowledge Acquisition	77
7.4.2	Integration with External C Programs	77
7.4.3	CLIPS Expert System Shell Limitations	78
7.4.4	Expert System Conversion to a Procedural Language	79
7.4.5	Path Generation Problem	80
8.	Research Results	81
8.1	Introduction	81
8.1.1	Initial Prototype	81
8.1.2	Final Prototype	86
8.2	BANES Output Verification Modules	88
8.3	END General Network Problem Locator Module	89
8.4	END Network Overload Tracer Module	90
8.5	Experimental Results Summary	97
9.	Conclusions and Future Research	99
9.1	Summary	99
9.2	The COSMIC CLIPS Development Tool	101
9.3	Future Research	102
9.4	Towards an Autonomous Network	103

References	105
Appendix A: END Network Overload Tracer Module	110
Appendix B: END Network Overload Tracer On-screen Result Scenario	115

List of Figures

2.1: A Typical Neural Network Architecture	9
3.1: Unshielded Twisted Pair (UTP) Ethernet Work Group	19
3.2: Work Group Connectivity to the Backbone	20
3.3: General LAN Backbone Connectivity	21
3.4: Magnet Architecture	23
6.1: Initial Stage of the Join Network Example	55
6.2: Second Stage of the Join Network Example	56
6.3: Third Stage of the Join Network Example	57
6.4: Fourth Stage of the Join Network Example	58
7.1: Context and Station Problem Class Control Flow Diagram	67
7.2: Routing Problem Class Control Flow Diagram	68
7.3: Router Problem Class Control Flow Diagram	69
8.1: END Interface Block Diagram	87
8.2: Network Data Storm Example	91
8.3: Data Storm Scenario Example	96

List of Abbreviations

ACE: Automated Cable Analysis
BNR: Bell Northern Research
CMOT: Common Management Information Protocol
DAD: Datapac Advisor
DC: Data Collector
DCD: Data Collector Daemon
END: Expert Network Diagnostician
GR: Global Resolver
IP: Internet Protocol
KEE: Knowledge Engineering Environment
KS: Knowledge Source
LAN: Local Area Network
MAN: Metropolitan Area Network
NDC: Network Diagnostic System
NOC: Network Operating Centre
NT: Northern Telecom
PC: Personal Computer
PING: Packet InterNet Groper
RDA: Remote Diagnostic Access
SNMP: Simple Network Monitoring Protocol
TRIO: Telecommunications Research Institute of Ontario
UTP: Unshielded Twisted Pair
WAN: Wide Area Network

Chapter 1. Introduction and Organization

1.1 Introduction

Traditional approaches to network management have difficulty coping with the ever-increasing complexity of current networks. The newly required functionality demands that new approaches be investigated. Artificial Intelligence (AI) is a prime candidate. The AI domain focuses on problem resolution techniques rather than solution algorithms. One example of a problem resolution technique is a knowledge-based system which consists of declarative (factual) knowledge and simple rules of inference for putting that knowledge to work in resolving specific problems. AI provides a new way, often successful, of attacking problems which have not been previously considered solvable by machines.

One of the most promising AI approaches is expert system technology. It is the most widely spread AI implementation. An expert system, as suggested by its name, is a machine that captures some of the knowledge and problem-solving strategies of a

human domain expert and selectively applies them under specific circumstances to resolve complex problems [Liebowitz88]. Telecommunication network fault diagnosis represents a potential domain application for expert systems.

1.2 Objective

The intent of this thesis is to apply expert system technology as part of a multi-paradigm approach to a real engineering problem: network fault diagnosis. This thesis study has three goals:

1. To assess the feasibility of using expert system technology for the diagnosis of faults in telecommunication networks,
2. To assess the potential and value of using NASA COSMIC CLIPS as a tool for building such an expert system,
3. To develop a system to determine the role of expert system technology in a multi-paradigm approach to network fault diagnosis.

To accomplish these goals, a prototype Expert Network Diagnostician (END) expert system has been implemented to first model part of the fault diagnostic procedures used by the current BNR network operators and, secondly, to integrate END within a

multi-paradigm solution to perform network diagnosis. The END prototype has been built using NASA COSMIC CLIPS running on a SPARC station 2.

The development of END was initially undertaken to investigate the potential effectiveness of using expert system technology to automate the procedures used in detecting, isolating, and correcting network problems from a Network Operating Centre (NOC). The first version of END demonstrated that an expert system could automate the most common detection and fault isolation procedural techniques used by the current BNR network operators. The final version of the END prototype demonstrated the capabilities of expert system technology as a crucial part in a multi-paradigm approach to large Wide Area Network (WAN) fault diagnosis. There currently exist many candidate paradigms for network fault diagnosis. Most current and past industry developments in the field have been based on single paradigm approaches. Each paradigm has its strengths and weaknesses. By combining END with the BANES [Dawes91b] probabilistic approach for example, one paradigm complemented the other and superior capabilities and accuracy were achieved, as will be demonstrated in this thesis. END represents a crucial part of the Pegasus multi-paradigm research project [Dawes91c] because it is the only interactive paradigm: END waits on semaphores for specific pre-conditions to be triggered, performs live tests on the network, assesses the results, provides a diagnosis and in some cases, even repairs the faults (e.g. with the appropriate permission, END can reboot a station causing a data storm). The results obtained in this thesis research proves it is not only feasible to use expert system technology for network fault diagnosis but it can also serve

as an effective, enhanced tool as part of a virtually autonomous multi-paradigm approach to large WANs network fault detection, isolation and repair.

1.3 Organization

The remainder of this thesis is divided into nine chapters. Chapter 2 provides a review of the literature on different potential approaches to automatic fault diagnosis. In Chapter 3, a summary description of the BNR telecommunications network used as a test bed for the END prototype is given, covering its main components and its current diagnostic problems. Chapter 4 provides a description of where the END prototype fits in the overall Pegasus multi-paradigm research project. The problem domain targeted by this thesis research is described in Chapter 5. Chapter 6 provides an overview of the expert system building tool (COSMIC CLIPS). In Chapter 7, the evolution of the logical design of the END prototype is described followed by a detailed description of its final architecture, capabilities and research results in Chapter 8. Finally, the conclusions of this research are presented in Chapter 9.

Chapter 2. Approaches to Automatic Fault Diagnosis

2.1 Introduction

As today's telecommunication networks become increasingly complex, the significance of network element monitoring and diagnosis is paramount. In this chapter, some important potential approaches to network fault diagnosis are considered.

2.2 Decision Tree

The first approach likely to be considered is to write a decision tree in any conventional language. In this case, the burden of diagnostic reasoning lies primarily with the domain expert. Although coding the decision tree into the computer may seem mechanical and quite simplistic, developing the decision tree is quite labour intensive. Extracting knowledge from the domain expert is labour intensive simply because there are many items or parameters to consider at each decision point in the tree. Since true domain experts are very hard to find, the decision tree paradigm comes with a very high knowledge acquisition price tag. Other disadvantages of this approach or paradigm are its

lack of flexibility and explanation facilities, and limited fault coverage. Although decision trees are simple in nature, the many disadvantages mentioned above prevent it, in many cases, from becoming a potential candidate for problem resolution [Cebulka89].

2.3 Model-based

Another approach to network diagnosis is to enter a detailed physical (connectivity) model of the network combined with sophisticated behaviour models of the individual nodes and lines. The burden of diagnosis lies entirely on the computer (program). It must have detailed knowledge of the behaviour of the network elements to recognize abnormal behaviour from any element. Ideally, this should be performed without any false alarms. The model-based approach has the advantage of providing automated diagnosis with an explanation facility capability at a lower knowledge acquisition cost. It is also capable of diagnosing unexpected faults. However, model-based techniques used today are computationally complex and are unable to learn network behaviours to perfectly distinguish between abnormal and normal behaviours. False alarms are frequently generated and some faults are undetected because a certain behaviour threshold has not been exceeded [Kahn87].

2.4 Probabilistic

A probabilistic approach uses past history cases and/or a gathering of evidence to

produce diagnostic results. A good example of a probabilistic paradigm is BANES. This paradigm provides a diagnosis by reasoning in uncertain nested evidence spaces [Dawes91b] where the belief that an element is broken is propagated from the network elements which are the furthest away from the data collectors to the broken element (data collectors are nodes which periodically collect status information on a group of network elements). Chapter 4 provides a summary description of BANES. However, a stand-alone probabilistic paradigm is limited by its nature. Because this paradigm learns about the network only through status messages and statistical information such as MTBF (mean time before failure), its diagnosis would not find faults like a non-reporting broken node which never broke before nor was ever expected to break.

2.5 Multi-Paradigms

Most paradigms can be augmented by combining them with one or more different paradigms like the statistical or probabilistic approach. For example, in the model-based paradigm, if statistics on the different network elements would be available during the diagnostic process, behaviour thresholds could be subsequently adjusted, less false alarms would result, etc. The model-based paradigm diagnostic performance would be significantly improved. The idea of using multiple complementary paradigms within a single diagnostic system is the underlying structure of the Pegasus research project [Dawes91c]. Each network fault diagnostic paradigm has its strengths and weaknesses. As further described in Chapter 4, the Pegasus research team uses multiple paradigms for

every diagnosis, with one paradigm filling the gap of a weak area of another paradigm. Within a multi-paradigms approach, there needs to be an activation controller which provides overall control on the activation and hypothesis analysis of the different paradigms. In this approach, the strength of all paradigms are combined to provide a powerful diagnostic tool.

2.6 Neural Networks

Current AI techniques rely on proven knowledge-based techniques to try to replicate human expertise but they have a very narrow focus. Their implementations are for restricted domain applications. Even for small domain applications, hundreds or thousands of rules are sometimes required to cover the small domain. The knowledge-based approach is still a program with very limited sensory capabilities. The majority of the sensory ability a human learns during his or her lifetime is knowledge acquired before the age of two. No expert system has yet replicated the sensory and common sense of a two-year-old child. The basic idea behind neural networks is to try to structure modern parallel processing computational techniques in the same manner as the structure of the human brain functions [Cebulka89]. Neural network systems do not compete with knowledge-based systems. They complement each other because the knowledge-based techniques can only perform in well defined domains and tend to crash or simply do not give results in poorly defined domains. Neural networks are ideal in poorly defined domains. In fact, current neural networks are often used for classification of some input patterns. A typical

neural network architecture is shown at figure 2.1.

Figure 2.1: A Typical Neural Network Architecture

The typical neural network has 3 layers. The nodes or cells represent independent processing units. The output layer gives the classification results, the input layer is where the attributes of the different elements to be classified are entered. Each input cell is connected to one or more hidden cells which are potentially connected to more hidden cells before they are finally connected to an output cell. Some connections are stronger than others so all cells are given a weighting factor so that different cells are connected to the same cell with different strengths. This eventually translates to strong response(s) to one

or more output cells.

In a hypothetical example, we begin with a switch operating normally, generating a series of digital messages [Lippmann87]. The messages are captured in a buffer and then passed to a neural network. With the different inputs and expected outputs, the neural network learns to differentiate between normal and abnormal behaviours. Eventually, when the preconditions of a switch failure occur, the neural network sends an alarm message and actions can be undertaken to prevent the failure from occurring.

A neural network needs many training sessions to efficiently provide correct classification results. Neural networks are still at an experimental stage and the experts in the field are unable to determine the minimal training set required or even what constitutes good results. Once the neural network has learned the network behaviour, it can tolerate input variations or even specific processing unit cell failures and still produce correct results. Under similar circumstances, most conventional programs would fail or even crash. Neural networks appear to be very promising; this is why this paradigm is an integral part of the Pegasus research project.

2.7 Case-based Reasoning

In this approach, diagnostic information is stored as past case histories. Each case history can be selected with the conditions under which it was applicable and the diagnostic

treatments or results that arose in that situation can be used [Cebulka89]. This paradigm can be implemented where the domain is clearly defined and, most importantly, a domain which has a lot of past case histories. An example domain is medicine. It is common to use indexing or pattern matching techniques to correlate a current diagnostic problem with a past case history. The solution can be to simply take the past case history and apply it to the current problem or make modifications to accommodate the differences between the current case history and the most similar past case history. The major difficulty is to get a good match. This paradigm could be implemented for network fault diagnosis. However, the domain does not have sufficient past case histories to implement a complete case-based network diagnostic tool. This problem is accentuated by the speed at which technology is progressing, which would create the necessity to constantly update the case file to make it complete.

2.8 Heuristic Search

Most paradigms have to "search" for the best solution(s) from many possibilities to resolve a current problem. Different heuristic search techniques can be used to guide this search. Consider the following travelling salesman example [Cebulka89]:

A sales rep has a list of cities, each of which must be visited exactly once. There are direct roads between each pair of cities on the list. The task is to find the shortest possible route for a round trip, starting at any of the cities and then returning

there.

This problem has a simple but long solution: try all possible routes on a map and measure the travelling distance required for each route. The problem with this solution is a phenomenon called combinatorial explosion. If the sales rep has as few as 10 cities to visit, the number of possible paths to explore becomes quite large (3,628,800). These solutions can be computationally very expensive. To improve the search and avoid combinatorial explosion, heuristics guide the search in better directions. The nearest neighbour heuristic can be selected for a case like the travelling salesman example: during the search for the shortest path, only the closest unvisited city is selected as the next destination. The time spent for the search would be drastically reduced. The drawback is that the solution may not be the best solution (i.e. there exists a shorter path) but it will usually produce a good solution. Heuristic search is best utilized when there are many possible solutions to the problem and there is no clear algorithm or procedure to resolve it. One example of an application to network diagnosis is when a routing problem occurs and the network diagnostician is trying to reconstitute the most probable path under which the fault occurred to isolate the possible faulty route element. There exists many possible paths but the search for the most probable path can be guided with each link between two nodes having a cost factor relating its orientation, distance and line speed. For example, if the source node is located in Montreal and the destination node is located in Toronto, the heuristic search for the most probable path would not consider any path going east out of Montreal even if they are eventually connected to Toronto.

2.9 Knowledge-based Systems

Knowledge-based systems (often referred to as expert systems) are best suited for well-defined domains where there exist human or human-like expertise and the problem is complex and knowledge intensive. There currently exist many knowledge-based systems and their applications in the network management field and other fields are spreading. Expert system technology is the subject of this thesis. It is the first time that expert system technology has been used in a multi-paradigm approach to fault diagnosis of a large WAN such as the BNR network. The main disadvantage of expert system technology is its narrow focus. Its important advantages are:

- a. augments but does not replace human reasoning,
- b. resolves complex problems,
- c. facilitates the modification of programs,
- d. different sources can activate the programs,
- e. interacts with the network,
- f. integrates programs with other systems.

To better understand expert systems, refer to table 2.1: Typical Differences between Conventional Programs and Expert Systems [Giarratano89].

<u>Characteristic</u>	<u>Conventional Program</u>	<u>Expert System</u>
Control by ..	Statement order	Inference engine
Control strength	Strong	Weak
Solution by ...	Algorithm	Rules and Inference
Solution search	Small or none	Large
Problem solving	Algorithm is correct	Rules
Input	Assumed correct	Incomplete, incorrect
Unexpected input	Difficult to deal with	Very responsive
Output	Always correct	Correctness varies with problem
Explanation	None	Usually
Applications	Numeric, file, and text	Symbolic reasoning
Execution	Generally sequential	Opportunistic rules
Program design	Structured design	Little or no structure
Modifiability	Can be difficult	Reasonable
Prototyping	Normal	Fast and easy

Table 2.1: Typical Differences between Conventional Programs and Expert Systems
[Giarratano89]

Expert systems have been used in different fields in interpreting and identifying, predicting, diagnosing, designing, planning, monitoring, debugging and testing, instructing, training, and controlling but it was only recently that the use of expert systems in network management has mushroomed [Goodman89]. The types of applications that were developed thus far can be divided into two categories: The first one is concerned with network planning and design, and simulation. The second category includes those systems which monitor, diagnose and repair network faults. This second category is the topic of this thesis research.

One of the most "famous" current industry expert systems used for monitoring, diagnosing and repairing network faults is ACE [Miller85]. Automated Cable Analysis (ACE) was one of the first expert systems developed in the field. ACE operates routinely at night in different telephone companies to provide expert level diagnoses on cable faults. Other known expert systems include:

- a. Network Diagnostic System (NDS) used for WANs diagnosis [Williams83],
- b. Troubleshooter, developed by AT&T and used for troubleshooting Datakit VCS networks [Marques88],
- c. ESTA, used for analyzing trunk outage codes [Kahn87],
- d. NEMESYS, used by AT&T to display congestion areas in its long distance network [Guattery85],
- e. Datapac Advisor (DAD), a "well known" BNR expert system that monitors

the Canadian packet network [Rabie88].

The DAD system is implemented in KEE [Kunz84], a popular, powerful but complex expert system shell running in a LISP environment, on a high-resolution graphic workstation. The DAD monitoring station aids the network operator in identifying and solving network faults by providing advice, reminders and background information [Rabie88]. DAD is a decision support system and the ultimate decision-making must be performed by the network operator. The DAD implementation cannot be directly used for WAN network fault diagnosis because it has been developed solely for the Datapac packet network and it would be difficult to integrate it in a multi-paradigm approach to network fault diagnosis. However, because of its successful implementation, some of DAD's concepts were considered for the development of the END system.

The current industry expert system implementations are mostly used as stand-alone systems. However, expert system technology has its own strengths and weaknesses. The expert system paradigm can be used concurrently with other paradigms within a single network diagnostic system to provide better, more accurate results. As described in section 2.5, multiple paradigms can be used concurrently for every diagnosis, with one paradigm filling the gap of a weak area of another paradigm. This thesis research investigates this new approach.

Chapter 3. BNR Network Overview

This chapter is intended to provide a summary description of the Bell Northern Research (BNR) network to appreciate the current diagnostic problems. The BNR Wide Area Network (WAN) serves as a test bed for the development of the END prototype and other diagnostic paradigms of the TRIO network fault diagnostic project (Pegasus).

3.1 Architecture

The BNR network comprises a hierarchy of Local Area Networks (LANs) which interconnect at the city level into a Metropolitan Area Network (MAN) and further interconnect into a Wide Area Network (WAN) to other BNR and Northern Telecom (NT) sites between cities, across the country, between countries and around the world. The BNR WAN spans 3 continents: America, Europe and Asia.

3.2 Work Groups

The term "work group" at BNR refers to the collection of users of a particular LAN [BNR90]. These users are located within the same area (usually same building), bound by the physical limitations (reach) of the Unshielded Twisted Pair (UTP) ethernet technology. The UTP technology allows the use of telephone cables to extend the LAN to the workstation limited to a distance of 100 meters. Therefore, to connect all workstations, multiple LANs must be installed. In most circumstances, the users of a work group make up a directorate. Most of the communications take place within a work group. Figure 3.1 makes up a typical "work group" illustration [BNR90].

Each workstation is connected to a primary hub which consists of 12 ports with one port typically dedicated for diagnosis or emergency use. If more than 11 users are to be connected, a second primary hub is required. Up to 16 primary hubs can be connected to a secondary hub. The secondary hub connects workstations of different primary hubs and provides a potential access to a central file server. As described in the next section, the secondary hub also provides communication connectivity outside the work group.

Figure 3.1: Unshielded Twisted Pair (UTP) Ethernet Work Group

3.3 LAN Backbone

At BNR, each work group forms a LAN (including workstations, and primary and secondary hubs). Communications within a work group and outside the work group LAN is provided via a bridge onto a LAN backbone. The backbone interconnects different LANs (see figure 3.2) [BNR90].

Figure 3.2: Work Group Connectivity to the Backbone

The bridge is a two-port device with one port connected to a secondary and the other port connected to a fanout device which interconnects different LANs and forms the backbone. A backbone is a group of LANs typically serving one group within a building (e.g. SUN users backbone, MacIntosh users backbone, etc.). The fanout is an 8-port or 16-port device which can be interconnected in a hierarchy to provide up to 256 possible backbone connections. Figure 3.3 illustrates how access to other LAN backbones is made [BNR90]. Each BNR Ottawa site has multiple backbones. The main reasons for using

multiple backbones are traffic management, UTP technology distance restrictions, separation of different workstation technologies (ex: Sun local LANs versus Macintosh local LANs) and network protocol separation.

Figure 3.3: General LAN Backbone Connectivity.

Routers are interconnected via serial lines at speeds ranging from 56 Kb to T1 (1.544 Mb). Typically, there are two routers per backbone. Each BNR IBM mainframe is connected to a backbone. Macintosh workstations use Fastpath devices where appletalk protocols piggyback onto ethernet (appletalk protocol is carried out on the data portion of the ethernet packets). This allows two distant Macintosh workstations to communicate

using ethernet.

3.4 BNR Current Network Diagnostic Support

BNR has a central network monitoring desk referred to as the Network Operating Centre (NOC) from which the entire Wide Area Network (WAN) is monitored and managed. The TCP/IP community has defined two network monitoring protocols for network management: SNMP (Simple Network Monitoring Protocol) and CMOT (OSI's Common Management Information Protocol running over TCP/IP). BNR has adopted SNMP as its network monitoring standard since CMOT, a more advanced and complex protocol, is not yet fully developed. For some nodes like the IBM mainframes, only the basic PING (Packet InterNet Groper) network management protocol can be used. The BNR network monitoring system is called Magnet in which a central network management application called Xnetmon is the network surveillance tool (see figure 3.4)

Xnetmon displays network elements (workstations, bridges, routers, etc.) as objects in which the object's shape defines its type and colour defines its status. The objects are organized in a hierarchical manner whereby a single, higher level object can represent a city or a whole country. The highest level view shows three continents. The state of a lower level network element such as a workstation is propagated upwards to the highest level view. The NOC operator can create, modify or delete objects. A network element can be active, inactive, in a test mode or in a user-defined troubled state.

Figure 3.4: Magnet Architecture

The Magnet monitoring system uses data collector daemons located at different points in the network. The data collectors (DCDs) use SNMP or PING to query the different elements periodically and this information is reported back to and displayed on the Xnetmon workstation. The DCD serves as a manager application which requests the network element's management function (called the agent) to perform the management functions. When the agent does not reside on the network element itself, it is called a proxy agent. This condition will occur when the data collector is unable to communicate with the network element via the management protocol. Each DCD daemon resides on a UNIX machine and typically connects to five ethernet. The Global Resolver (GR)

provides name service and resolving functionality. A Remote Diagnostic Access (RDA) tool enables the Magnet operator to analyze most workstation to workstation traffic from a remote RDA station. The RDA system consists of a PC and a modem located at the Magnet operator site. At the other end, a modem, PC and matrix are connected to most network secondary hubs.

3.5 Network Diagnostic Problems

There are a wide variety of faults which can occur in the network. Some faults can be diagnosed easily but a good number of faults require the high expertise of the Magnet operator, and there are faults that Magnet operators simply cannot pinpoint. In Chapter 7, some scenarios that Magnet operators follow during the diagnosis of different faults are described. The following paragraph describes some of the current BNR network diagnostic problems.

Some network faults occur when workstation wires are loose or disconnected. These can be permanent or intermittent faults. When Xnetmon reports the occurrence of these types of faults, it cannot correlate the fault with a loose wire. Only an experienced operator is capable of finding the source of such a problem. Some faults occur when configuration changes are made or when maintenance is being performed without reporting it to the NOC. BNR is trying to improve its configuration control because at any point in time, the Xnetmon representation of the network is approximately 10% incorrect. When

a user reports that he cannot reach a destination from his workstation, the Magnet operator begins a series of lengthy steps to try to isolate and possibly correct the fault using status messages, RDA sessions and multiple test queries. The fault can be anything from an inactive bridge or router to a wrong IP address. There is extensive work required of the Magnet operator in such a situation. Similarly, when a user phones in to report intermittent faults between two workstations, the Magnet operator can be kept busy for a lengthy period of time. In this case, the Magnet operator usually dials up from his RDA PC to the RDA PC and matrix of the secondary under which the reported troubled workstation resides. The Magnet operator usually asks the user who reported the problem to repeat the operations which led to the intermittent fault. From his remote PC, the Magnet operator can analyze the traffic while the user replicates the scenario. More complex faults occur when, for example, a line driver goes down. If, for example, three router serial lines are implemented on three different channels of the same physical line whose line driver is faulty, Xnetmon would report this as three independent faults. The correlation of these faults with the single line driver fault is difficult to ascertain because the monitoring system does not know the three router serial lines are on the same physical line.

An ideal network monitoring system should be capable of viewing the network and the problems at different communication protocol layers. Other complex faults occur at the router level where dynamic rerouting and other router algorithms only add to its complexity. Yet another type of fault occurs when faults are masked as other faults. For

example, if a fault occurs at the DCD level, all network elements that the faulty DCD is polling will be reported to Xnetmon as being down when, in reality, only the DCD itself is down. The problem is the faulty network is used to report its own faults.

On a typical day, many interventions by the Magnet operator are required. The Magnet operator has many available tools but must undertake many procedures. Complex diagnostic algorithms could be automated to make a system like Magnet a better, more autonomous diagnostic system. The BNR network serves as a test bed environment for the automation of network fault diagnosis for the Pegasus research project.

Chapter 4. Pegasus Project

4.1 Introduction

4.1.1 Objective

The short-term objective of the Pegasus research project is to develop a multi-paradigm prototype for the monitoring and diagnosis of a wide range of faults occurring within a very large wide area network (WAN) computer communication network. Its long-term objective is to serve as an autonomous prototype system capable of monitoring, providing fault diagnosis at both lower and higher communication protocol layers and possibly repairing some classes of faults of not only WANs but also other types of networks such as broadband and photonic networks.

4.1.2 Scope

The scope of the system is not limited to computer networks but could be adjusted to accommodate other types of telecommunication networks such as broadband ISDN or even totally different applications, e.g. the diagnosis of component failures within an electronic device. The Expert Network Diagnostician (END) prototype is a sub-system of the Pegasus system. The aim of this chapter is to briefly describe the Pegasus system.

4.2 Multi-Paradigm Approach

As mentioned in Chapter 2, many candidate paradigms currently exist for network diagnosis. Most past and current industry developments in the field have been based on single paradigm approaches. Each paradigm has strengths and weaknesses. If two or more paradigms are combined within a single diagnosis system, better capabilities and performance may be achieved by one system. The idea of using multiple complementary paradigms concurrently is quite recent and is very promising. A multi-paradigm approach to network monitoring and diagnosis is the basis of the Pegasus research project.

4.3 Architecture

The Pegasus system prototype is being developed on a UNIX Sparc 2 platform. The underlining programming language is C but some sub-systems have been written in

different programming languages. The current Pegasus system is divided into five sub-systems: the blackboard, the simulator, the Knowledge Sources (KSs), the scheduler and the dynamic data structures. The Kss currently consist of an expert system (END), a probabilistic system (BANES) and a neural network.

4.4 Operational Overview

The basic Pegasus system operation can be summarized as follows:

1. The simulator replicates the entire BNR network by simulating its topology and its normal and abnormal network behaviour. This is translated into network nodes being polled for their status, traffic being generated and problems being reported on different network components or groups of components.
2. The simulator posts these status messages and reported problems on the blackboard.
3. The scheduler controls the activation of all Knowledge Sources (Kss). The Kss are the different diagnostic paradigms. The Kss which are well suited for the current class of problems posted on the blackboard are given priority in the computer resource allocation. Intermediate results of a KS

(single paradigm) can be used to improve the diagnosis of another KS.

The scheduler is only in its initial stage of development and the control and setting of KS priorities have not been designed thus far.

4. The Kss post their hypothesis results into the hypotheses area of the blackboard. A separate, higher-level KS would analyze the hypotheses to produce correct and consistent results.
5. Currently, the different Kss are triggered by status messages and problems posted on the blackboard. The only interactive KS is the END (Expert Network Diagnostician) expert system. Its diagnostic knowledge base does not rely solely on status messages. The END system is interactive in the sense that it can query most network devices, obtain IP addresses, request a station to reboot or even force traffic data from one station to another. These manipulations are translated into function requests sent to the simulator via the blackboard. The END system function call waits on a semaphore for the reply from the simulator which is to be posted on the blackboard. These capabilities make the END system one of the most powerful Kss.

4.5 Blackboard

The blackboard can be compared to a central database with its own built-in access control functions. Its purpose is to provide asynchronous communication between the different Kss, the scheduler and the simulator. The blackboard makes use of the UNIX environment shared memory segments and semaphores. It is divided into several areas (e.g. status messages, requests, replies, problems and hypotheses). Each area consists of an array of records with control data structures to manage the array. Each KS can retrieve or post information on any appropriate blackboard area. This information can be shared with other KSs. The KSs wait on their own semaphores for changes to specific blackboard areas. KSs can be written in any programming language but must be capable of calling C functions to access the blackboard data structures.

4.6 Simulator

The simulator simulates a very large WAN so that the different KSs can be extensively tested. Premature testing by the different KSs or the whole Pegasus system on a live network (alpha tests) would be a nightmare for the network operators. Real topological, operational and traffic data were taken from the BNR network to create the simulator. The system supports at least 100 routers, 255 LANs, 500 bridges and several thousand workstations. Simulation is performed much faster than real time speeds [Dawes91a]. The simulator provides for most of the real network functionality such as the

use of PING and SNMP network monitoring protocols, regular polling of network elements to obtain their status, and dynamic rerouting. The simulator has an interface from which a user can break or repair certain devices, change the error rate on serial lines, interrogate a device via a SNMP query, or reboot a device. In such a case, the simulator user is a KS designer who induces specific faults, observes its effects and tests his or her KS. The simulator also has a random generator which, when selected, generates failures and repairs on different devices at different times. The simulator has been designed to run the equivalent of 25,000 years in simulation time. This allows for comprehensive network reliability analysis. To accelerate testing, the simulation runs about 20 times faster than real time in certain modes.

4.7 Knowledge Sources

4.7.1 BANES

The BANES diagnostic approach is a probabilistic paradigm based on nested evidence space data structures. It is divided into two areas: one is the nested evidence space representation and the other is its simple belief approximation calculations [Dawes91b]. The network elements are polled regularly by a few data collectors located at different points across the network. The status of the different network elements is taken from the perception that the data collector has of its own polled elements. Elements closer to the data collector are used to carry monitoring (status) information of elements farther

away from the data collector. If this closer element breaks down, monitoring information of other, farther elements would not be transmitted to the data collector. The data collector would therefore report that the broken element and all elements underneath it are inoperable. The BANES paradigm tackles this problem by providing a hierarchical nested evidence space representation of the network [Dawes91b]. The assumption that an element is broken is propagated from the elements which are the furthest away from the data collector (which are usually workstations) to the closest element(s) (i.e. bridges, routers). For example, if a bridge and ten workstations beneath it are all reported as being inactive, the original belief that the workstations are broken would be propagated to the bridge. There would be a strong belief that only the bridge is broken. Propagation of belief would end at the bridge if no other element between the bridge and the data collector, on any route to the data collector, are reported as being inactive.

The BANES paradigm recognizes faults for which there is status message evidence with 99% accuracy [Dawes91b]. The main disadvantage of the BANES paradigm is that it not interactive (i.e. it cannot perform live tests or queries on the network); its only sources of diagnostic information are the network topology and the status messages.

4.7.2 Neural Network

As mentioned in chapter 2, neural networks are well suited for problem classification. As part of the Pegasus research project, a neural network is being developed

to serve as a monitoring system. The neural network inputs the traffic data of some router lines for a certain period of time and learns to differentiate between normal traffic and abnormal traffic. This paradigm presents an important application to network management: it is proactive, i.e. it can prevent node and line failures by reporting abnormal behaviour. Modifications to the network can be made before an element breaks down.

4.7.3 Expert Network Diagnostician (END)

The END prototype is the demonstration product of this thesis. As mentioned in Chapter 2, the expert system paradigm capabilities make it a very powerful and flexible network monitoring and diagnostic tool. Its main advantage over other paradigms is that it is interactive. It can perform human-like functions such as querying network devices, obtaining IP addresses, requesting a station to reboot or even forcing traffic data from one station to another. The END system is a rule-based system which can diagnose and, in some cases, repair faults autonomously. The knowledge acquisition phase required a great deal of time to extract the procedural diagnostic information used by the BNR network operators. These scenarios or procedures were then coded into rules. The rules represent the current network operator diagnostic knowledge. The initial END system diagnosis was limited to the station, router and routing problem classes. The END system was then further developed to cover other types of faults with different diagnostic procedures by making use of other paradigms like BANES. Two modules of the END system are used concurrently with BANES to improve diagnostic accuracy. END takes advantage of the

ability of BANES to correlate multiple reported faults to a single fault source. END verifies the BANES hypotheses using its interactive capabilities such as Telnet queries to validate the hypotheses. This concurrent use of two KSs results in an improved diagnostic accuracy. The END system has therefore been developed one step further than the automation of current network operator procedures to take full advantage of the different technologies (paradigms) available today.

4.8 Dynamic Data Structures

For most conventional programs, data structures are fixed once the program has been compiled. The usual procedure to modify a data structure (e.g. add an attribute to an object) is to change the source code, re-compile it and run the system again. With dynamic data structures, the blackboard is compiled in such a way that its data structures can be modified while the program is still running. The Pegasus research team has a particular interest in dynamic data structures because it does not confine the different KSs to a single blackboard data structure. Each KS can have its own data structure view of the blackboard to fully exploit its own capabilities.

Chapter 5. Problem Descriptions

5.1 Introduction

The aim of this chapter is to introduce and describe the different types of faults and problems encountered in the operation of a WAN for which the END system provides a diagnosis. The different classes of faults and problems handled by END represent the most common faults occurring in the current BNR WAN. Some occasional, complex faults are also described. Chapter 6 provides a description of the development tool: COSMIC CLIPS, which was used to develop the END diagnostic prototypes. The Expert Network Diagnostician (END) prototypes were developed to demonstrate the potential role of expert system technology within a multi-paradigm approach to network fault diagnosis. The evolution of the logical design of the END prototype is described in Chapter 7, followed by a detailed description of its final architecture, capabilities and research results in Chapter 8.

The BNR network fault analysis and problem classification presented in this chapter and the remainder of this thesis has been produced by myself. The expert network diagnostic information analysis (knowledge acquisition phase) and expert system modules (including all interfaces to the Blackboard and indirect interfaces to the Dragon simulator and other KSs) have also been produced by myself. The other components of the Pegasus project, i.e. the Dragon simulator, the Blackboard, BANES and the neural network KS, have been developed by other members of the Pegasus research team.

5.2 Network Problems Simulation

The Dragon simulator [Dawes91a] allows the simulation of a busy, very large, datagram WAN. As previously described in Chapter 3, the network elements are periodically polled by data collectors located across the network using SNMP or PING network management protocols to obtain their status (active, inactive, etc.). This status information is reported back to the central Xnetmon station in the form of status messages. The simulator only displays the status message if the element has incurred a status change. A typical status message example is "bCARc1 14" which means that router "BCARC1" is reported as being inactive (state 14). Status messages serve as symptoms of faults which require further investigation. The reason for the occurrence of this former symptom is that router "bCARc1" failed to reply to periodical SNMP queries (polling) after a certain period of time.

The simulator also simulates problems being generated by a network user reporting a problem via a telephone call. The different reported problems are described in the following sections. These problems can be randomly generated by the simulator or generated by the simulator user. The simulator user can write directly to the problems area of the blackboard using the "p1" blackboard interface program to report a particular problem. Other symptoms are obtained from interactive network queries (i.e. bCARc1 does not respond to a Telnet query). Interactive queries are currently only used by END. Symptoms of faults are therefore obtained from status messages, reported problems or interactive queries. These symptoms trigger the different END diagnostic expert system modules.

5.3 Common Faults

A wide variety of faults can occur within a large network such as the BNR WAN. Permanent and intermittent faults can be caused by simple node or line failures, loose or disconnected wires, undeclared configuration changes, wrong routing tables, network congestion and other types of problems occurring at different communication protocol layers. Most of the common network faults can be divided into five problem classes. The following sections list the five problem classes, the main symptoms associated with each fault and the most probable reasons why the fault occurred.

5.3.1 Element - Station Problem Class

PROBLEMS SYMPTOMS

REASONS

a. station is down

unresponsive to a
PING/SNMP query

disconnected
undergoing maintenance
unit H/W or S/W malfunction
undeclared configuration change

unresponsive to a Telnet
query

same as above

state 14 status message

same as above, and
data collection error

reported problem

same as above

b. station performance degradation

intermittent state 14
status messages and
intermittent responses
to a PING/SNMP query

same as above, and
traffic overload
loose station connection

5.3.2 Element - Router Problem Class

PROBLEMS SYMPTOMS

REASONS

a. router is down

unresponsive to a
PING/SNMP query

disconnected
undergoing maintenance
unit H/W or S/W malfunction
undeclared configuration change

unresponsive to a Telnet
query

same as above

state 14 status message	same as above, and data collection error
reported problem	same as above
b. router performance degradation	
intermittent state 14 status messages and intermittent responses to a PING/SNMP query	same as above, and traffic overload congestion station X is producing high traffic on router
c. router error rate	
all of the above	traffic overload congestion

5.3.3 Line Problem Class

PROBLEMS SYMPTOMS

REASONS

a. physical link is down

state 14 status message
reported problem
cannot force traffic
on line

line disconnected
line interference
noisy line

b. line driver is down

same as above

line driver failure

5.3.4 Routing Problem Class

PROBLEMS SYMPTOMS

REASONS

a. source/destination station

(i.e. for a routing problem, the fault may be located at the source or destination nodes)

reported problem
status messages
unresponsive to a
Telnet query

see station problem class, and
routing source station has the
wrong IP address for the
destination station

b. a router on the path between the source and the destination

(i.e. for a routing problem, the fault may be a router on the most probable routing path)

reported problem
status messages
unresponsive to a
Telnet query

see router problem class

c. a line on the path between the source and the destination

(i.e. for a routing problem, the fault may be a line on the most probable routing path)

reported problem
status messages
cannot force traffic on line

see line problem class

d. network overload (data storm)

reported problem
status messages

some network elements were
overloaded, exceeding their
traffic handling capabilities

5.3.5 Monitoring Problem Class

PROBLEMS SYMPTOMS

REASONS

a. data collector is down

a whole domain of
network elements have
state 14 status messages

data collector S/W failure

b. Xnetmon application is down

most or all network
elements have a state 14
status message

Xnetmon application S/W failure

A routing problem occurs when a user at station A cannot communicate with node B, located at some distant point in the network. There are other possible reasons for such a problem. Some faults occur and are masked by other faults. For example, if a fault occurs at the data collector (DCD) level, all network elements that the faulty DCD is polling will be reported to Xnetmon as being down when, in reality, only the DCD itself is down. The problem is that the faulty network is used to report its own faults.

5.4 Complex Problems

More complex faults occur when, for example, a line driver goes down. If, for example, three router serial lines are implemented on three different channels of the same physical line whose line driver is faulty, Xnetmon would report this as three independent faults. The correlation of these faults with the single line driver fault is difficult to

ascertain because the monitoring system does not know the three router serial lines are on the same physical line. An ideal network monitoring system should be capable of viewing the network and the problems at different communication protocol layers. Other complex faults occur at the router level where dynamic rerouting and other router algorithms only add to its complexity. Yet another complex but typical problem occurs when a novice or inconsiderate network user sends multiple huge files across the network to a distant destination node in the middle of the afternoon. Most nodes and lines in the path of the data storm would overload, causing all kinds of problems to other users wanting to use the same network resources. Chapter 8 describes this example in more detail and the program logic used by END to diagnose such a problem.

5.5 END Modules

The final Expert Network Diagnostician (END) prototype is composed of four independent expert systems (or modules), each one being an expert in specific classes of problems. END handles all the different faults of the five problem classes described in section 5.3 with a high accuracy (research results are described in Chapter 8). As will be further described in Chapters 7 and 8, two END modules, Veri-element and Veri-line, are used concurrently with the BANES probabilistic paradigm for simultaneous network diagnosis. END's interactive capabilities (e.g. it can send a Telnet query, force some traffic on a line, etc.) are used concurrently with BANES to provide more accurate diagnostic results. Each paradigm has its strengths and weaknesses. By combining two

or more paradigms within a single diagnosis system, better capabilities and accuracy may be achieved by one system.

Chapter 6. The COSMIC CLIPS Expert System Development Tool

The "C" Language Integrated Production System (CLIPS) made its debut at the USA NASA Johnson Space Centre in 1984 [Giarratano91a]. At that time, a great majority of the expert system development projects at NASA were using the LISP language. For this reason, one cannot help but notice that CLIPS has a certain LISP flavour. To build a highly portable expert system tool, NASA developed CLIPS using a conventional language: C. CLIPS can be used on virtually any platform. For NASA internal interoperability reasons, CLIPS was made compatible with the ART source code. ART is a large commercial expert system shell and development tool. The development of CLIPS was therefore influenced by both LISP and ART. In fact, CLIPS syntax is very similar to that of ART.

CLIPS is an expert system tool based on the RETE algorithm designed to facilitate

the modelling or representation of human expert knowledge. RETE is a very efficient pattern-matching algorithm developed by Charles Forgy of Carnegie-Mellon University for his OPS shell. Functionally, CLIPS is a forward chaining rule-based language with similar capabilities to OPS5 [Brownston85]. Today, CLIPS has received widespread acceptance among the academic, public and private sectors because of its low-cost, portability and fundamental capabilities.

The main reason why CLIPS was selected as the main expert system tool for the Pegasus project is its relative ease of integration with other C programs as compared with other expert system packages. The blackboard, which is the centre of the overall Pegasus project to which the END system must interface continually with to obtain its facts base is a C program providing C I/O data structures.

6.1 Syntax and Organization

The objective of this section is to provide a summary of the CLIPS tool syntax and organization. The CLIPS shell provides the three basic components of an expert system:

- a. facts list: global memory for data,
- b. knowledge base: rules,
- c. inference engine: control of execution.

Just like other languages, CLIPS has reserved key words. For example, if you type:

```
(assert (beat it))
```

the following fact, consisting of two fields will be added to the facts list:

```
(beat it)
```

All commands, facts, rules and other constructs are always within parentheses. In this example, we can see that the developers of CLIPS were influenced by LISP. Facts can be removed from the list with the "(retract)" command. Facts and rules can be directly typed in CLIPS or loaded from a file, using an ASCII editor. The general format of a rule is:

```
(defrule <rule name> [optional comment]
  < <patterns> > ; LHS of the rule (pre-conditions)
  =>
  < <actions> > ) ; RHS of the rule
```

CLIPS has been designed in such a way that the program can be selectively set to see all facts (default) or only see the facts that have been entered after the rules (so that rules are activated on new facts only). Once a rule has all its preconditions satisfied, it is put on the agenda, the collection (stack) of all activated rules. Rules with the same preconditions are fired using the LIFO (last in first out) scheme or the rules can be prioritized using the "salience" command. One way to avoid conflict resolution is to

implement the rule-based system where only one rule is activated at a time. However, this would defeat the purpose of writing the application as an expert system; it would only prove that the application should have been written using a procedural or sequential program in the first place. Any string preceded by the symbol "?" is a variable. The address of a fact is specified using the left arrow "< -". The following is a simple example to demonstrate CLIPS rule syntax:

```
(defrule walk
  ?activation < - (activate rule walk)
  (or   (police says walk)
        (and (walk_sign walk)
              (light green)))
  =>
  (retract ?activation)
  (printout t "Walk" crlf))
```

Example 6.1: CLIPS simple rule example

This rule is called "walk". The rule can be activated if the fact "activate rule walk" is asserted in the knowledge base and one of the following events occur: if the fact "police says walk" is asserted or if the "walk_sign walk" and "light green" are both asserted. If the pre-conditions are satisfied, the rule is activated and the word "Walk"

is printed out to the screen. The left arrow symbol "<-" is used to bind the address of the fact "(activate rule walk)" to the variable "?activation" so that the fact can be removed from the list after the rule is activated (retract ?activation). The following example demonstrates the use of variables:

```
(defrule multiplication
  (initial-fact)
=>
  (printout t "Enter first number of the multiplication" crlf)
  (bind ?first (read))
  (printout t "Enter second number of the multiplication" crlf)
  (bind ?second (read))
  (bind ?result (* ?first ?second))
  (printout t "Result :" ?result crlf))
```

Example 6.2: CLIPS variables manipulation

This rule has as a pre-condition the fact "initial-fact" which is always asserted upon activation of CLIPS. The rule requests the first and second numbers of the multiplication, reads and binds them to the variables "?first" and "?second" respectively. The two variables are then multiplied and the result is bound to the variable "?result" and printed out to the screen. Let's consider a more complex example taken from the Expert Network Diagnostician (END) prototype to demonstrate some of the important capabilities of CLIPS. This next rule is not complete in itself but performs the main tasks required to isolate the fault in a routing problem.

```

11:(defrule routing_fault_locator
12:  ?path1 <- (path1 ?sst)
13: =>
14:  (retract ?path1)
15:  (printout t "PLEASE ENTER THE PATH SOURCE NODE NAME" crlf)
16:  (bind ?source (read))
17:  (printout t "PLEASE ENTER THE PATH DESTINATION NODE ID"
    crlf)
18:  (bind ?destination (read))
19:  (bind ?routerslist (routers_of_path ?source ?destination))
110: (bind ?count 1)
111: (bind ?router ao)
112: (while (neq ?router f)
113:   (bind ?router (nth ?count ?routerslist))
114:   (bind ?count (+ ?count 1))
115:   (if (neq ?router f) then
116:     (bind ?value vv)
117:     (bind ?element-closest-router na)
118:     (bind ?protocol pd)
119:     (bind ?rr (get_state2 ?router))
120:     (bind ?state (nth 1 ?rr))
121:     (if (neq ?state 2) then
122:       (assert (net-mon-prot ?router ?element-
        closest-router ?protocol ?value ?sst))
123:       (assert (routerproblem))))))
124:  (assert (path2 ?source ?destination ?sst)))

```

Example 6.3: END demonstration of CLIPS capabilities

This rule performs multiple tasks. The first task is to request the source node name and destination node id of a routing problem reported by a user. Secondly, it retrieves the list of all routers of the most probable routing path. Certain rules are then activated if one or more of the routers are reported as having an inactive state. Yet another set of rules is activated if all routers of the path have an active state.

Rule "routing_fault_locator" is preceded by other rules (not shown in this example) which initialized access to the blackboard and waited on a semaphore until a problem was reported (posted on the problems array of the blackboard). Interface to the blackboard is described in section 7.3. Rule "routing_fault_locator" is activated and fired upon assertion of the fact "(path1 X)" by a preceding rule where the number X is bound to variable "?sst" which is a problem counter incremented after a diagnosis on a specific element or path is completed. The function of this variable is to allow a rule to be fired more than once. For example, if a previous rule asserted the fact "path1 2", rule "routing_fault_locator" would be automatically activated for the second time with the variable "?sst" bound to number 2. In a simple CLIPS procedure, a rule is not fired more than once. At line 12, the address of the fact "(path1 ?sst)" is bound to the variable "?path1" so that it can be retracted at line 14, making it possible to use the rule again.

At line 15, a message is printed to the screen, requesting the source node name of the path which is to be diagnosed. The source node name entered by the user then becomes the value of the variable "?source" (see line 16). At lines 17 and 18, the same process is repeated to obtain the destination node id. At line 19, a function call is made to an external C function called "routers_of_path", passing the two variables "?source" and "?destination" as arguments. The external function returns a list of routers which represents the most probable path between the source and the destination nodes. This list of routers terminates with the character "f" to signal the end of the list.

As shown in example 6.3, CLIPS provides for conventional procedure loop control like the "while" and the "if then" loops. Entire CLIPS programs could be written in a conventional, procedural way but it would defeat the purpose of using an expert system. Lines 110 and 111 are variable initializations. Starting at line 112 is a loop to retrieve every router in the routers list, one at a time, until the character "f" is encountered. The "f" character signals the end of the list. At line 113, a CLIPS function called "nth" is used to retrieve the nth element of a list. If the element is not the end of the list (line 115), then some variables are initialized (lines 116 to 118). An external function call "get_state2" (line 119) is made with the nth router in the list as its single argument to get its current state. At line 121, if the router's current state is not equal to 2 (which signifies it is not active), then the rule to start the routers's diagnosis is activated by asserting the fact at line 122. The assertion at line 123 is used as a flag-like control. Line 124 is always asserted, but only once since it is outside the "while" loop. This assertion is used in addition to the line 123 flag to control the activation of the next rule.

6.2 CLIPS Architecture and Reasoning

In the latest CLIPS version 5.0, there are three ways of representing knowledge. The first method uses rules to capture the human or other types of expertise or heuristic knowledge gained with experience. The second way to represent knowledge in CLIPS is by using the construct functions "deffunctions" or "generic functions" to implement knowledge procedurally. Finally, there is also object-oriented programming which is also

intended for procedural knowledge. CLIPS object-oriented programming features include inheritance, classes, message-handlers and abstraction. A program in CLIPS can be written using only objects, only rules or a mixture of both. CLIPS initial object-oriented programming capability has been implemented in its latest version and it does not yet handle pattern matching with objects. This main drawback directed the development of the END system towards a strict rule-based system. The CLIPS architecture consists of high-level and low-level modules where the former need the latter to operate [Giarratano91b]. The lower-level modules handle such functions as memory allocation and maintenance, input/output, storage management and other low-level functions. The low-level to mid-level modules handle expression evaluation and the CLIPS command line interface. The mid-level modules are used for building data structures for the "defrule" and other constructs. The core of the inference engine is made up of six middle-level modules:

1. The Drive Module joins a network when a new fact has been asserted. The term "joins a network" means that a fact is compared to a rule pre-condition for pattern matching. An example is demonstrated in the next page.
2. The Engine Module maintains the agenda. The agenda is a list of the rules which are ready to fire because all their rules pre-conditions have been satisfied.

3. The Match Module determines, upon a new fact being asserted, which network pattern(s) (pre-conditions), if any, have been satisfied.
4. The Retract Module updates the join network when a fact is removed.
5. The Utility Module provides powerful utilities for other modules.
6. The Fact Manager maintains the facts list.

The higher-level modules are used for execution, interface and debugging functions. To demonstrate the underlining logic behind the CLIPS architecture, consider the following example as a typical Drive Module operation, a program containing the following rule only:

```
(defrule example
  (point ?x ?)
  (point ? ?x)
  =>)
```

Example 6.4: CLIPS drive module example

In this example, the term "defrule example" is the definition of the construction of a rule called "example". The left-hand side (LHS) contains two patterns with the first field "point" being identical, the second field of the first pattern being a variable called

"?x" (like a PROLOG variable) which is identical to the third field of the second pattern. The third field of the first pattern "?" (which is identical to the second field of the second pattern) can be matched with anything. There is no right-hand side to the rule. Figure 6.1 illustrates the join network for this example [Giarratano91b]. The top left box represents the join associated with the first pattern "(point ?x ?)". A join is where a pattern matching is performed for a specific pattern (rule pre-condition). The bottom right box illustrates the join associated with the second pattern "(point ? ?x)". Dark points to the left of the joins represent the beta memory. Dark points to the right represent alpha memory. Initially, all memories are empty. The circle represents the completion of the pattern node "(point ? ?)".

Figure 6.1: Initial Stage of the Join Network Example

The first join has one word "point", one variable "?x" and one field which can be matched with anything "?". If the fact "(point 3 4)" was asserted, it would be

automatically passed down to the join network where a match occurs with the pattern "(point ? ?)" (fact "f-1" in figure 6.2), stored in the pattern node alpha memory. This partial match is then passed down to the first join "(point ?x ?)" where the variable "?x" is bound to the number "3" (see figure 6.2).

Figure 6.2: Second Stage of the Join Network Example

Since the first join (top left box) has no expression, the partial match is then passed to the second pattern where a comparison is made between all the partial matches in the alpha memory and the beta memory. For the second join, the partial match in the alpha memory (there is only one fact in the alpha memory) is the partial match of the first join "(point 3 4)" and the partial match in the beta memory is the fact bound to the second pattern "(point 3 4)". Finally, the join network is evaluated and the rule does not have a complete match because the variable "?x" of the first pattern is bound to the

number "3" and the same variable "?x" of the second pattern is bound to the number "4" (see figure 6.3). Therefore, the rule is not activated.

Figure 6.3: Third Stage of the Join Network Example

Suppose that a second fact is asserted: "(point: 4 3)". The fact matches the partial match "(point ? ?)" and is passed to the first pattern (see figure 6.4). Since the first join (top left box) has no expression (dark point), the partial match is passed down to the second join. A comparison is then made between f-2 (point 4 3) in beta memory and f-1 (point 3 4) in alpha memory. Since the variable "?x" in f-1 (second field) and f-2 (third field) are both bound to the number "3", there is a partial match. Since the next join is a terminator join (no other join), there is a complete match and this rule would be activated and placed on the agenda.

Figure 6.4: Fourth Stage of the Join Network Example

The Drive Module does not stop here; there are still three other comparisons to be performed: two facts makes for four matching possibilities:

1. (point 3 4) (complete match with the example rule)
(point 4 3)
2. (point 3 4) (no match, ?x has two different values)
(point 3 4)
3. (point 4 3) (complete match with the example rule)
(point 3 4)

4. (point 4 3) (no match, ?x has two different values)

(point 4 3)

This process seems very laborious, but it is required for all possibilities to be covered for rule pattern matching.

6.3 Truth Maintenance

Truth maintenance is required to ascertain validity in an expert system. Truth maintenance is the alteration of information to conform to reality. The fact that certain facts are logically dependent on others needs to be captured within the expert system. CLIPS provides this functionality in support of truth maintenance with the reserved key word "logical". For example, suppose you have two meetings the same day, one at 10h00 and the other one at 11h00. Information required for the 11h00 meeting would be given at the 10h00 meeting. In addition, if the original database contained only these two facts (meetings at 10h00 and at 11h00), and the first meeting was cancelled, then it would be logical to reschedule the 11h00 meeting due to its dependence on the 10h00 meeting. It is only common sense, but the expert system needs to be aware of this logical dependence in the form of rules.

6.4 Integrating CLIPS with External Functions

An important feature of CLIPS is its relative ease of integration with C external functions (as compared with other expert system tools). A function call from CLIPS to an external function can be made at any point in the CLIPS program. CLIPS can also be embedded as an external function within a C program. In this case, the CLIPS program is called from the C program, performs its functions, and returns control back to the main C function. All external functions called from CLIPS need to be declared within the "main.c" file; this is done by modifying the function "Userfunctions" which is part of the "main.c" file. Within the "main.c" file, a call has to be made to the routine function "DefineFunction" for every external function that is to be integrated with CLIPS. Consider this example "UserFunctions" from the END system:

```
UserFunctions()
{
    extern VOID Maintenance(); (C function declarations)
    DefineFunction("maintenance",'b',Maintenance,"Maintenance"); }
```

Example 6.5: Integrating CLIPS with external functions

The call to the function "DefineFunction" is made to register user-defined

functions. The first argument to the function is the CLIPS function name. The second argument is the type of parameter(s) which are to be returned to CLIPS from the external C function. In this example, 'b' means that a boolean value (true or false) will be returned. The third argument is a pointer to the actual function and the fourth argument is a string representation of the third argument. The actual C function called "Maintenance" is stored in a separate file but within the same directory as the "main.c" file.

CLIPS is an expert system programming tool written in C which served as a versatile tool for the development of END. CLIPS allows a programmer to code with most of the functionality available in a procedural language like C, within a forward chaining environment.

Chapter 7. Expert Network Diagnostician (END) Prototype Evolution

7.1 Introduction

7.1.1 Objective

The primary objective of the Expert Network Diagnostician (END) system is to prove the effectiveness of using expert systems as a powerful tool in WAN network fault diagnosis. The second objective of END is to validate the output hypotheses of the BANES probabilistic paradigm. Thus, to fulfil these objectives, a prototype expert system (END) was developed. Although END's target is the BNR network, it can be viewed as a "proof of effectiveness" of the advanced capabilities that an expert system could bring to network management in general. Developing a full-scale "ready-to-use" application is not the objective of this research.

7.1.2 Scope

The scope of the END system prototype is limited to the diagnosis of the five classes of problems defined in Chapter 5. These five classes of faults make up the majority of faults occurring in the current BNR network.

7.2 END Development

This chapter describes the evolutionary approach used to develop the END system. The evolutionary approach simply means that the prototype was evolving step by step, with each version being a superior working prototype to the previous version. Expert system technology allows fast prototyping. Ideas can be implemented and tested quite rapidly. The development of the END system was divided into 2 phases. The first phase was to automate the current diagnostic procedures used by the BNR network operator using a similar environment and the same available diagnostic tools. The second phase in the evolution of the END system was to improve the automation of the network operator procedures with the incorporation of new tools into the network fault diagnostic procedures which are not currently available to the network operator. These new diagnosis tools (including END) are the other paradigms which form the Pegasus research project. The first Pegasus project paradigm used in conjunction with the END expert system is the BANES probabilistic approach.

7.2.1 Phase 1: Automation of Current Procedures

This first phase was sub-divided into three stages. Their description follows in the following paragraphs:

Stage 1: Knowledge acquisition,

Stage 2: Analysis, coding and validation,

Stage 3: Rule generalization and classification.

Knowledge acquisition was the first stage of phase 1. The first step in the knowledge acquisition stage was to become familiar with the application domain: network fault diagnosis. Once the application domain was generally understood, the next step was to interview the expert to extract his knowledge of the application, his job and most importantly, his problem-solving techniques. The Magnet operators are very busy and there are no available manual or training packages for Magnet operators. For these reasons, extracting the different problem-solving techniques from the expert necessitated following the procedural reasoning the Magnet operator uses to determine the diagnosis whenever a fault or multiple faults occur. The result of this initial knowledge acquisition stage was a set of scenarios which the BNR Magnet operators use during their tour of duty. The scenarios do not encompass all the operations of a Magnet operator but present most of the common procedures performed during specific monitoring and diagnostic sessions. The second stage was to analyze, code and validate the acquired knowledge.

The different scenarios were therefore coded into general rules and then translated into CLIPS when it became available. The different scenarios comprised many rules, some of which were very similar. The third stage was to generalize the different rules to produce a minimal set of manageable rules. This was possible because most scenarios had common variables that were manipulated and similar procedures were used to diagnose each and every problem.

The rules capturing the Magnet operator's procedures are best suited for a forward-chaining system like CLIPS because the problem or goal resolution is data driven. A good portion of the set of scenarios can be categorized in one of the following three diagnostic problem classes (each scenario being an instance of one of the problem classes):

1. Element (station or router) problem (as described in sections 5.3.1 and 5.3.2),
2. Routing problem (as described in section 5.3.4),
3. Monitoring problem (as described in section 5.3.5).

Figures 7.1, 7.2 and 7.3 illustrate the control flow diagrams for the element (station and router) and routing problem classes. The control flow diagrams give a general orientation of how the rules were coded. These diagrams were derived from the rules themselves instead of the other way around. It was not required to write data flow

diagrams to create the system rules. The diagrams were written to facilitate the comprehension of the logical flow because it is nearly impossible to manually trace the execution of a rule-based program on a printout paper. The main reason for this difficulty is in the rule-based program execution, often some rules have the same pre-conditions and it is difficult to know which rule will be fired next by the inference engine. In fact, a rule-based program has to be running to be capable of accurately following its execution. In a conventional approach using a procedural language like Pascal or C, programming would be totally based on the control and/or data flow diagrams and it is relatively easy to understand the code on a printout paper. A major advantage of using a modular rule-based approach over a procedural language is that the code can be modified more easily because each rule is often independent of the others. Herein, the term "modular" means that the system is broken into multiple little expert systems. This advantage is not applicable in the case where the expert system is a single, huge program. A major disadvantage is that you cannot readily read the code and understand what it does.

Figure 7.1: Context and Station Problem Class Control Flow Diagram

Figure 7.2: Routing Problem Class Control Flow Diagram

Figure 7.3: Router Problem Class Control Flow Diagram

7.2.2 Phase 2: Augmenting END Diagnostic Capabilities

As mentioned in Chapter 2, most paradigms like the END system can be augmented by combining them with one or more different paradigms. In this fashion, one paradigm complements the work of another paradigm and superior performance is achieved. In this second phase, the END system is employed concurrently with the BANES paradigm. The BANES Pegasus paradigm can diagnose many faults based on knowledge of the network topology and status messages alone. BANES cannot diagnose faults where there are no status messages associated with the fault. Because BANES' simple belief algorithm is not 100% accurate, BANES will most likely make a few extra, incorrect diagnoses (false alarms). Its diagnostic hypotheses needed to be verified and validated. One of the complementary functions of the END system is to remove these extra diagnoses. In this second phase, the new Expert Network Diagnostician (END) system was developed in two stages. The objective of the first stage was to build an expert system which verifies the BANES output hypotheses without using status messages as triggers. The END system and BANES ran concurrently on the same simulation. The END system was triggered by a hypothesis posted on the blackboard by BANES. Its different diagnostic procedures allows it to validate or deny the hypotheses from BANES. The product is more accurate diagnostic results.

The objective of the second stage (of phase 2) was to add functionality to the END system so that it can handle hidden faults and problems reported by users by means of

telephone calls on different types of problems that BANES simply cannot handle. An example of a hidden fault is when a station in the network is continually sending a large amount of data to a router, congesting it. A status message would report a problem with the router when the actual fault is located at the station. More details on the subject are presented in Chapter 8.

The initial stage of phase 2 is divided into two modules to cover the following classes of problems:

- a. Veri-element (module 1): element (router and station) problem class,
- b. Veri-line (module 2): line problem class.

These two modules are separate, independent small expert systems waiting on different semaphores for specific pre-conditions to be met before they are triggered. The following is a pseudo-code example for module 2 (see example 7.1): line problem class. The BANES paradigm uses status messages to report hypothesized problems. END validates (improve the probability or deny) the hypothesis results from BANES. Experimental results are described in Chapter 8.

Module 2: Line problem class

Trigger: Module waits for a BANES hypothesis to be posted on the BB. The line source and destination node names are given.

Step 1: Initiate a Telnet session into the source node from the source node's closest router,

step 1.1: If the Telnet session is not successful then the problem is unknown.

step 1.2: If the Telnet session is successful, retrieve the line data count and the destination node's network monitoring protocol, then send a PING/SNMP query from the source node through the troubled line to the destination node. Read the new line data count.

step 1.2.1: If both line data counts are equal, then the line is down.

step 1.2.2: If the second line data count is higher than the first one, then the line is active.

Example 7.1: Line problem class module pseudo-code

The objective of the final stage (second stage of phase 2) was to advance beyond the current network diagnostic procedures and the validation of the BANES hypothesis with the expansion of the END expert system so that it can diagnose the following faults

or problems:

- a. hidden faults (faults which are not reported because there are no status messages reported by the data collectors),
- b. station, router, line and routing problems reported directly by the user via a telephone call.

The second stage of phase 2 is the final version of the END expert prototype. Here, END can diagnose and isolate faults, and suggest actions to the network operator to eliminate problems or faults. In some applicable cases, END can also repair faults autonomously. For example, if a station is forever sending packets to a router, it would appear to the data collector that the router is congested or down when the real source of the problem is the station. END will be triggered by the hypothesis that the router is down. In its diagnosis, END verifies the load on every port of the troubled router and it is able to correlate the problem with the station which is congesting the router. With the correct permission activated, END would reboot that station. In most similar cases, this action would resolve the problem. Chapter 8 provides a more detailed description of the final version of the Expert Network Diagnostician (END) prototype.

7.3 Interface to the Blackboard

The blackboard uses the shared memory segments and semaphores offered in the UNIX environment to serve as a global database. Each expert system module and all other KSs are assigned a private semaphore on which it can wait for changes in specific areas of the blackboard for which it is interested. The following example is a small C program called by the different END modules to perform a Telnet query on a given network element. There are many similar C programs which are used by END to interactively interface with the simulator via the blackboard.

```

11 VOID telnet(returnValuePtr)
12 DATA_OBJECT_PTR returnValuePtr;
13 {
14   char *arg1;
15   long index;
16   long flag2;
17   VOID *multifieldPtr;
18   multifieldPtr = CreateMultifield(1);
19   arg1 = RtnLexeme(1);

110 /* post request on blackboard to be answered by the simulator */

111 index = get_empty_request(&process_env);
112 request_array[index].function_no = 3;
113 request_array[index].request_para2 = 26;
114 strcpy(request_array[index].request_str0,arg1);
115 post_request(&process_env, index);

```

```

116 /* wait for reply */

117 flag2 = 1;
118 for (;;) {
119     templong = get_next_reply(&process_env, last_reply);
120     switch (templong) {
121         case 0:
122             if (flag2 == 1) {
123                 printf(" Waiting for a reply to a Telnet request\n ");
124                 flag2 = 2;    }
125             wait_change(&process_env);
126             break;
127         default:
128             printf("reply returned = %d\n", templong);
129             last_reply = templong;
130             if (reply_array[last_reply].reply_no == index) {
131                 if (reply_array[last_reply].reply_para0 == 1) {
132                     printf(" Telnet session successful\n");
133                     SetMFType(multifieldPtr, 1, WORD);
134                     SetMFValue(multifieldPtr, 1, AddSymbol("yes")); }
135                 if (reply_array[last_reply].reply_para0 == 0) {
136                     printf(" Telnet session Unsuccessful\n");
137                     SetMFType(multifieldPtr, 1, WORD);
138                     SetMFValue(multifieldPtr, 1, AddSymbol("no")); }
139                 SetpType(returnValuePtr, MULTIFIELD);
140                 SetpValue(returnValuePtr, multifieldPtr);
141                 SetpDOBegin(returnValuePtr, 1);
142                 SetpDOEnd(returnValuePtr, 1);
143                 return;
144             }
145     }
146 }

```

Example 7.2: Blackboard interface for a Telnet query

To activate the C program of example 7.2 from an END module, a call is made to "telnet(X)" where X is the name of the node to which a Telnet query is to be performed. At lines 12 to 19, program variables are initialized and a multi-field pointer

is defined for parameter passing between the C program and the END module. At lines 110 to 115, the information required for the simulator to perform a Telnet query is posted to the request array of the blackboard. At line 111, the C program gets an empty request array index number from the blackboard. The index is used to differentiate between the blackboard requests. At lines 112 and 113, two blackboard array fields are defined: "function_no = 3" for a request, "request_para2 = 26" for a Telnet query. At line 114, the node name is passed and line 115 performs the "post_request".

Lines 116 to 143 handle the reply to the Telnet query. At line 119, the variable "templong" is defined as a pointer to the blackboard reply array. Line 118 sets an indefinite loop where the "templong" pointer is incremented to look for the reply to the original request (with the same index number) through the reply array. If the reply has not yet been posted by the simulator to the blackboard reply array, the C program waits for the reply on a semaphore (line 125) and prints the appropriate message on the screen (lines 121 to 126). If the reply is located on the array (reply array field "reply_no" having the correct index number), then the Telnet reply is obtained from the "reply_para0" field of the reply (lines 127 to 138). The pointer "templong" points to the correct reply. At line 129, the pointer "last_rely" takes the "templong" pointer value so that, for the next blackboard request, the search pointer for the correct reply starts where the last reply occurred to avoid having to search for a reply from the start of the reply array. At lines 131 to 143, the CLIPS multi-field pointer is set to return a "yes" or a "no" in reply to the Telnet query.

7.4 Points of Interest in the Development of the END Prototype

Some points of interest and technical difficulties encountered in developing the END network diagnostic program using the CLIPS expert system shell will be discussed in this section.

7.4.1 Knowledge Acquisition

The END knowledge acquisition phase was limited in scope, i.e. it covered some of the most common faults as opposed to resolving all the BNR network faults. Therefore, the validation process for the END knowledge acquisition phase was limited to the scope of the END system.

7.4.2 Integration with External C Programs

One of the main reasons why CLIPS was selected as the expert system tool was the original belief that it was relatively easy to integrate CLIPS with C programs in comparison with other expert system tools. The emphasis here is on "relatively easy" because in fact, it is relatively complex to integrate CLIPS with external functions. All external functions must be described to CLIPS by modifying a special function. Within that special function, a call is required to a routine for every function which is to be integrated with CLIPS. Then, this source code has to be compiled and linked with

CLIPS. The complexity of integration of CLIPS with external functions is compounded by different pointer manipulations which are required whether arguments are transferred from CLIPS to external functions, returning values to CLIPS from external functions, or passing or returning symbols, strings, instance names and addresses, external addresses, unknown data types, boolean values or multi-field values. CLIPS direct parameter passing may be complex but it is better than most expert system shells which can only offer indirect parameter passing. In this latter case, communication with an external program is usually made via a file which serves as an interface between the programs. The file is continually opened to write to it to send some parameters from a source program or to read from it for the destination program to receive the passing parameters.

7.4.3 CLIPS Expert System Shell Limitations

The CLIPS expert system tool allows a programmer to accomplish most of what is available in a procedural language like C and has some additional features. Unfortunately, there were some tasks that had to be passed from CLIPS to C, processed and then passed back to CLIPS because CLIPS does not possess the capability; but at least, the task can be accomplished. For example, at one point in the program, a single word like "l0.bcarcl" which represents port 1 of node bcarcl had to be split into two words "l0" for port 1 and "bcarcl". The initial word "l0.bcarcl" had to be passed from CLIPS to C so it could be separated in two to then be transferred back to CLIPS.

7.4.4 Expert System Conversion to a Procedural Language

As mentioned several times in this thesis, one of the main advantages of programming using an expert system tool as opposed to using a procedural language like C is that it is easier to prototype, modify and expand the expert system code. Once the final version of the expert system code has been tested and no further modifications are required, the expert system rules can be converted or re-written in a procedural language like C. To translate the code, one has to consider that the firing of rules is controlled by the expert system inference engine. In a procedural language, control of the different rules or modules has to be explicitly stated within the program. The main reasons why conversion of the expert system code may be considered is that the C code usually runs faster and it is obviously easier to integrate with other C programs. However, the conversion is seldom performed because of the amount of work involved. In this research, the expert system modules were running on a SUN SPARC Station 2 and speed of execution was not a problem. Under normal simulation traffic load conditions, the most complex expert system module had a diagnostic execution time of less than 30 seconds. All other expert system modules had an execution time of less than two seconds. Another important conclusion made during the course of the development of END is that whenever fast prototyping is required and if the application domain is well bounded, using an expert system shell like CLIPS for the prototyping phase of the software development can significantly reduce the prototyping time (not considering the CLIPS learning curve) and therefore reduce the total software development time.

7.4.5 Path Generation Problem

Upon proceeding with the diagnosis of a routing problem where a user at station A cannot reach node B, the END system currently only considers the most probable path to verify the different elements in the path between node A and node B. With dynamic rerouting, there is most probably more than one possible path where the error might have occurred. Integrating a multiple paths generator within the project would be a natural progression towards a better, more accurate multi-paradigm diagnosis system.

The current method used to determine the most probable path where a routing fault has occurred is to send an explorer packet from the source node to the destination node, recording all node names on its path. The explorer packet function is a feature of the Dragon simulator which provides the Knowledge Source with the names of each device on the path from a given source node to a given destination node. This method was successful under low to medium network traffic load conditions. However, under heavy traffic conditions, dynamic rerouting occurred and this method would produce a probable path but it was not always the correct path taken when the fault originally occurred. For a large network like the BNR WAN, there can be thousands of possible paths between two distant nodes. A multiple path generator should use heuristics such as limiting the number of possible hops between two nodes to find the most probable path.

Chapter 8. Research Results

8.1 Introduction

The aim of this chapter is to provide a description of the experimental results of this thesis research. First, the results from the initial phase of automating the current BNR network diagnostic procedures are described, followed by a description and explanation of the logic behind the different modules of the final phase of the END prototype.

8.1.1 Initial Prototype

As originally discussed in Chapter 7, the first Expert Network Diagnostician (END) prototype was a single, large expert system providing network fault diagnosis for the current operation of the BNR WAN. This initial prototype was operating in a stand-alone mode, i.e. it interfaced with the blackboard and the simulator but did not take advantage of other paradigms simply because they were not fully developed thus far.

END was divided in three classes of problems: station, router and routing problem classes. Because it is difficult, if not impossible, to easily understand the expert system code, the control flow diagrams in figures 7.1, 7.2 and 7.3 were developed from the rules themselves. The control flow diagrams give a general orientation of the logical flow of the diagnostic procedures.

The END element problem class section is continually waiting on a semaphore, only to be triggered by an element status change to state 14 (inactive state), as posted on the status area of the blackboard and produced by the network simulator data collector (DCD). END can also be triggered by a problem which was reported by a user and posted in the problem area of the blackboard and an investigation into the matter is required. Once END has been triggered, it runs through a series of queries to more precisely determine the class of problem associated with the newly triggered problem. These queries range from determining if the element was initially scheduled for maintenance, determining its network monitoring protocol, PING or SNMP, responding to a network monitoring protocol query and/or a Telnet query from its closest router, determining if the element is a station, a bridge or a router, forcing packets on a specific line, rebooting a station, etc. The results from one set of rules would trigger another set of rules until a final diagnostic hypothesis on the element is posted to the hypothesis area of the blackboard. The END routing class of problems portion is triggered by a routing problem being posted on the blackboard where a user at station A cannot reach element B located anywhere else in the network. For this class of problem, another set of

interactive network queries are performed (see figure 7.2) to determine the routing problem. The routing source and destination nodes and the source station destination address are verified, the most probable routing path, its nodes and lines are determined and verified, etc. From the routing problem class, the element (station and router) problem classes are activated for these verifications. If the repair required to the network was simple and END had the proper authority, it would perform the remedial action (such as a station reboot). END was capable of providing the following correct hypotheses:

a. for the element - station problem class (described in section 5.3.1)

- (1) station is disconnected or undergoing maintenance,
- (2) station is down - unresponsive to a PING/SNMP query,
- (3) station is down - unresponsive to a Telnet query,
- (4) file server is down - attempting to reboot,
- (5) station performance degradation - attempting Remote Diagnostic Access (RDA) procedure,
- (6) problem with station has been resolved or has disappeared,

b. for the element - router problem class (described in section 5.3.2)

- (1) router is disconnected or undergoing maintenance,
- (2) router is down - unresponsive to a PING/SNMP query,
- (3) router is down - unresponsive to a Telnet query,

- (4) router error rate threshold exceeded, attempting RDA procedure,
- (5) station X is producing high traffic on router Y port Z, attempting RDA procedure,
- (6) problem with router has been resolved or has disappeared.

c. for the routing problem class (described in section 5.3.4)

- (1) routing source station is down,
- (2) routing destination station is down,
- (3) routing path router X is down,
- (4) routing path line X is down,
- (5) routing source station has the wrong IP address for the destination station,
- (6) link X CSU/DSU is down,
- (7) T1-link is broken,
- (8) routing problem has been solved or has disappeared.

Whenever END was unable to find the source of the problem, it would send a related message to the Network Operator. Another message would notify him or her if the reported problem was beyond the scope of the END prototype for END to continue its operation. The initial prototype detected the most common faults in the BNR network with an accuracy between 70% and 90% with a high degree of autonomy. However, because of its relatively large size, END became difficult to expand and debug, and was

quite slow. For this reason, the next generation of the END prototype consisted of a quantity of small expert systems with each one being an expert in a certain class of problems. Another problem encountered in the initial prototype was its lack of global vision of the network. END performed very well on individual problems but frequently failed to correlate multiple apparent faults to a single, real fault. This kind of problem arises from the method by which the network elements are polled by a few data collectors located across the network. The status of the different network elements is taken from the perception that the data collector has of its own polled elements. Elements closer to the data collector are used to carry monitoring (status) information of elements farther away from the data collector. If this closer element breaks down, monitoring information of other, farther elements would not be transmitted to the data collector. The data collector would therefore report that all elements underneath the broken element including itself are inoperable. END would be incapable of correlating the apparent multiple faults to the single source of the problem because END diagnoses each apparent fault individually.

This weakness in the END prototype is the strength of the BANES paradigm. The BANES paradigm tackles this shortcoming by providing a hierarchical nested evidence space representation of the network. Chapter 4 provides a summary description of the BANES paradigm. The final END prototype has been developed based on the lessons learned in the development of the initial prototype. The final prototype is composed of multiple little expert systems, each one being an expert in a specific problem class. To

2

of/de

2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

PIONEERS IN METHYLENE BLUE TESTING SINCE 1974

MICROD

1000 COUNTY ROAD 5, BURNING WOOD, OH 44027 USA
TEL: 614/426-7887 FAX: 614/426-7887 TLR: 6100000489

resolve the multiple faults correlation problem of the initial prototype, the final END prototype is used in conjunction with the BANES paradigm.

8.1.2 Final Prototype

The final prototype (phase 2 of the END prototype development as referenced in Chapter 7) is composed of four independent modules or small expert systems:

- a. BANES output verification modules
 - (1) Veri-element,
 - (2) Veri-line,
- b. General network problem locator module,
- c. Network overload tracer module.

Each module contains a set of rules capturing the expertise to resolve a specific class of problems and waits on its own semaphore to be independently triggered on different problem situations. Input information is taken from the network simulator through the status and problem areas of the blackboard and also from the BANES diagnostic output via the blackboard hypotheses area. The final hypotheses of the different modules are posted to the hypotheses area of the blackboard (see figure 8.1).

Figure 8.1: END Interface Block Diagram

8.2 BANES Output Verification Modules

The final END prototype is used in conjunction with the BANES paradigm to resolve the multiple faults correlation problem of the initial prototype. The BANES paradigm has a high accuracy in providing correct diagnosis. For a 20-day simulation run (simulation time) with a very high network traffic load, the simulator randomly broke 1015 elements and lines and BANES correctly diagnosed 1009 element and line faults. However, BANES also produced 178 extra incorrect hypotheses. As described in Chapter 7, there are two BANES output verification modules: Veri-element for the element (station and router) problem class and Veri-line for the line problem class. These two independent modules wait for BANES to post its output in the hypotheses area of the blackboard which becomes the input of the END Veri-element and Veri-line modules. These modules verify and validate the BANES hypotheses with interactive queries unused by BANES such as Telnet queries. From all BANES broken element and line hypotheses, Veri-element and Veri-line validated all of the BANES correctly diagnosed hypotheses (1009). Furthermore, the modules also denied 174 of the 178 extra incorrect BANES diagnoses, amounting to only 4 wrong hypotheses. Therefore, the concurrent use of BANES and END results in a diagnostic accuracy of 99% (4 extra incorrect diagnoses and 6 undetected faults out of a total of 1015 faults). The 4 BANES extra hypotheses that END did not pick up are due to a simulator error. The 4 BANES extra hypotheses are for the same line between node bCARc2 and bMERc1, and since there are two lines between these two nodes (this is the only place in the network), one active line

can hide the fact that the other line is inactive.

8.3 END General Network Problem Locator Module

This module is a small stand-alone expert system which handles problems being reported by a user (problem being posted on the blackboard). This includes reported element, line and routing problems (all problem classes described in section 5.3 except for the data storm problems and the monitoring problem class). Reported element and line problems are automatically transferred to the Veri-element or Veri-line modules. The routing diagnostic procedure starts by verifying the status of the routing destination and source nodes. If one or both nodes do not react to a PING/SNMP query or a Telnet query, the problem with the node(s) is transferred to the Veri-element module. If both the source and destination nodes are functional, the General Network Problem Locator Module verifies all nodes and lines in the most probable routing path. The most probable routing path is determined by sending an explorer packet (in the simulator) from the same source node to the destination node, recording all nodes and lines in the path. If any node in the path does not respond to a PING/SNMP query or any line appears to be down, the node or line problem is transferred to the Veri-element or Veri-line modules. Finally, if all nodes and lines in the most probable routing path are functional, the source node IP address for the destination node is compared with the destination node's correct IP address. If the source of the routing problem is any of the following faults, END will find the problem (routing problem class is described in section 5.3.4):

- a. routing source station is down,
- b. routing destination station is down,
- c. routing path router X is down,
- d. ethernet X is down,
- e. line from node X to node Y is down,
- f. routing source station has the wrong IP address for the destination station,
- g. link X CSU/DSU is down,
- h. T1-link X is broken,
- j. routing problem has been solved or has disappeared.

8.4 END Network Overload Tracer Module

The END Network Overload Tracer Module has been developed to demonstrate the potential capability of using expert system technology in more complex network fault problems. This module will also be described in more detail to explain the type of logic used in the different modules. The END Network Overload Tracer Module detects a more complex routing problem. One example of this is caused by a network overload. A typical example is when a novice network user sends multiple huge files across the network to a distant destination node in the middle of the afternoon. Most nodes and lines in the path of the data storm would overload, causing all kinds of problems to other users wanting to use the same network resources. Figure 8.2 illustrates such a typical example: a novice user located on ethernet e0.bWDL is sending multiple huge files across

the network to a friend working on a workstation at ethernet e1.bSTM, causing high network traffic along its path. At the same time, a user at station A (e0.bGRG) reports that he cannot reach a station B located at ethernet e0.bATP.

Figure 8.2: Network Data Storm Example

The END Network Overload Tracer module determines the most probable path from the reported routing problem and verifies all nodes and lines on that path. All nodes and lines are functional. The next diagnostic procedure used is to obtain the load of every port for every node along the path. If a specific port has traffic exceeding a

certain threshold for that time of the day, the search for a data storm source and destination starts. In our example, the node bATPc1 has its traffic threshold exceeded on its port coming from bCARc4 and also from its port going to bSTMc1. It is important to set the traffic threshold to reflect abnormal traffic with a specific day time period.

From this point on, two separate searches are fired: one is to search the source of the data storm and the other is to search the destination of the data storm. At bCARc4, END obtains a traffic overload coming from e0.bCAR. The search never verifies the load on the port associated with the node where the search came from (at this point, it is the node bCARc4) to properly direct the search and avoid endless loops. The data storm source search continues to converge until the source (e0.bWDL) has been located. A similar parallel search is also performed to find the destination of the data storm (the destination search rule is put on the agenda for execution). The programming of the source and destination search rules is parallel but the actual rule execution is sequential, i.e. CLIPS only executes one rule at a time. In our example, the destination search ends at e1.bSTM. The search for the data source converges to the correct source. If there are multiple data storm sources, the END Network Overload Tracer module will always find all the data storm sources without giving extra wrongly diagnosed sources. In our example, the search for the destination of the data storm also converges to a single node but this is not always the case. In situations where there are multiple, similar short paths, a data storm may diverge to other routers not located on the data storm path. In

our example, the data storm passing from bCARc2 to e0.bCAR also overloads bCARc1, bCARc3, bCARcX, bCARcY and bCARcZ. Multiple data storm destination nodes are also correctly diagnosed but these nodes are not all the destination nodes originally targeted by the novice network user. The correct setting of the traffic overload thresholds is paramount so that traffic loads caused by a data storm can be distinguished from normal traffic loads.

For the development of the END Network Overload Tracer module, the threshold to distinguish abnormal traffic (data storms) was determined with a series of tests. Initially, the threshold was arbitrarily set and then modified with a series of different threshold settings until no false alarms (normal traffic diagnosed as data storms) were generated and no data storm were occurring without being detected.

Under low network traffic load conditions, the signal-to-noise ratio (the signal represents the data storm traffic load and the noise represents the normal traffic load) is high enough to produce 100% accuracy results in finding the source of the data storm. As the network traffic load increases significantly, the signal-to-noise ratio decreases and the diagnostic accuracy decreases. This translates into false alarms being generated and data storms occurring without being detected.

A portion of the Network Overload Trace module is included at appendix A. The

following is a step-by-step explanation of the expert system module logic (refer to appendix A). The C code has not been included. The first rule (11-17) is used to make the proper initializations. At line 15, a call is made to a C function to register the module with the blackboard. In rule two line 124, a C function call to the function "problem" is made. This function waits on a semaphore until a problem has been reported and posted onto the problems area of the blackboard. In the following program lines, information about the problem is transferred from the C program to the CLIPS module. Depending on the problem type, certain rules will be fired. If the problem is a routing problem (type 2), rule three (125-1117) is fired. At line 143, a call is made to a C program called "routers_of_path" which obtains all the nodes of the most probable routing path. A Telnet query is made to all the nodes on the path to determine if any node is down (145-157). From line 161-167, the traffic on all ports (up to 14) of all nodes in the path is obtained. The traffic load difference is calculated during a period of 10 seconds. At line 168, END obtains the initial time and the initial traffic load up to that point (170) for the traffic going out of that specific node and port. From the lines 174-180, END obtains the traffic of all neighbouring nodes sending traffic to that specific node. Lines 181-185 are used to wait for 10 seconds. At lines 186-195, a second reading of the traffic load entering and exiting the node is made. At lines 196-1102, if the load difference for the traffic of any port going out of the node exceeds the 6000 packets threshold (after the 10 seconds delay), then the search for the data storm destination(s) starts by the assertion at line 199. At line 103-109, if the difference in the load of any node port for which this specific node is a destination exceeds 6000 packets, then the

search for the data storm source(s) starts with the assertion at line 1106 which activates rule four (1118). If there is no overload on the most probable path, then a message stating this will be printed out to the screen (1113-1115). Lines 1116-1117 are used to reactivate rule two to take or wait on the next reported problem.

Rule four is used for the data storm source search. This rule uses the same logic as rule three except that it is only searching for the neighbouring node which is sending a high level of traffic to the current node. The search never considers the traffic from the neighbouring node from which the search originated to correctly orient the search. Under normal traffic conditions and with the correct high traffic thresholds settings, the search always converges to the correct data storm source(s). As well, there is another rule to search for the data storm destination(s). In this case, only the traffic exiting the node is used to verify if there is an overload on any of its ports. The data storm destination search stops when there is no very high outgoing traffic on any port. Similarly, the data storm source search stops when it has found a node to which no neighbouring node is sending a high traffic load.

An on-screen result from a Network Overload Tracer module scenario is presented at appendix B. The scenario is as follows: a user on ethernet e0.bWDL is sending multiple large files (causing a data storm) to router bSTMcl (see figure 8.3). Not long after, a user on ethernet e0.bGRG reports that he cannot reach ethernet e0.bATP. The END Network Overload Tracer module is triggered on the reported problem.

Figure 8.3: Data Storm Scenario Example

At appendix B, reference point A, the module is loaded with a batch file command. At reference points B and C, the module waits and finds a reported routing problem. At reference points D and E, the module gets the list of all routers on the most probable routing path and verifies that they are all alive. From reference point F on, the

load on every port of every node on the path are examined to locate any data storm. On every port, END examines the load coming into the node (data storm source search) and the load going out of the node (data storm destination search). Although most nodes have less than 14 ports, the module always verifies 14 ports because the number of ports on each node is not available. At reference point F, END examines node bGRGc1; there is normal traffic on all its ports. Similarly with node bRCHc1 (ref point G) and node bRCHc2 (ref point H). At reference point I, END finds an overload on node bATPc1 port 1, coming from bRTPc1 and the data storm source search rule is activated. On node bATPc1 port 6, END finds an overload going to bSTMc1 and activates the data storm destination search rule (rule fired at ref point K) and finds that node bSTMc1 is the destination of the data storm (ref point L, data storm stops there). At reference point M, the data storm source search is fired to find the source. At reference N, the data storm source is found to be coming from bCARc4. The data storm search continues and initially suggests that it has found the data storm source (ref point V) and further corrects itself by finding the correct data storm source: e0.bWDLbr (ref point AA). Refer to figure 8.3 for the data storm source search route.

8.5 Experimental Results Summary

Table 8.1 provides an experimental results summary for the four END modules with respect to the level of network traffic. The tests were limited to the scope of the classes of problems as defined in Chapter 5.

MODULE	TRAFFIC	TESTS	DIAGNOSTIC ACCURACY
Veri-element	low	80	99%
Veri-element	medium/high	50	95%
Veri-line	low	80	99%
Veri-line	medium/high	50	95%
General Problem Locator	low	50	99%
General Problem Locator	medium/high	40	85%-95%
Network Overload Tracer	low	50	99%
Network Overload Tracer	medium/high	40	70%-90%

Table 8.1: END Modules Experimental Results Summary

Chapter 9. Conclusions and Future Research

9.1 Summary

The development of END was initially undertaken to investigate the potential effectiveness of using expert system technology to automate the procedures used in detecting, isolating, and correcting network problems from a Network Operating Centre (NOC). The first version of END demonstrated that an expert system could automate the most common detection and fault isolation procedural techniques used by the current BNR network operators.

Focus on prototyping resulted in rapid development of new procedures. As the evolution of END progressed, the research found other interesting orientations. One orientation is using multiple little expert systems, each one focusing on a specific problem class instead of having one large expert system. Another orientation was to use END (composed of multiple expert systems) as an integral part of a multi-paradigm approach

to network fault diagnosis. The final version of the END prototype demonstrated the capabilities of expert system technology as a crucial part in a multi-paradigm approach to network fault diagnosis. There currently exist many candidate paradigms for network fault diagnosis. Most current and past industry developments in the field have been based on single paradigm approaches. Each paradigm has its strengths and weaknesses. By combining END with the BANES probabilistic approach, one paradigm complemented the other and superior capabilities and accuracy were achieved. END represents a crucial part of the Pegasus multi-paradigm research project because it is the only interactive paradigm: END waits on semaphores for specific pre-conditions to be triggered, performs live tests on the network, assesses the results, provides a diagnosis and in some cases, even repairs the faults (e.g. with the appropriate permission, END can reboot a station causing a data storm).

The BNR WAN network was selected as the environment to prove the concept and effectiveness of different evolving END prototypes. The development of the END prototypes provided an initial study which proved the effectiveness of its implementation as opposed to being a full-scale application ready to be integrated into a real network. The results obtained in this thesis research proves it is not only feasible to use expert system technology for network fault diagnosis but it can also serve as an effective, enhanced tool as part of a virtually autonomous multi-paradigm approach to network fault detection, isolation and repair. Another important conclusion made during the course of the development of END is that whenever fast prototyping is required and if the

application domain is well bounded, using an expert system shell like CLIPS for the prototyping phase of the software development can significantly reduce the prototyping time (not considering the CLIPS learning curve) and therefore reduce the total software development time.

9.2 The COSMIC CLIPS Development Tool

USA NASA COSMIC CLIPS was used as the development tool for the END prototype. CLIPS is an expert system programming language tool written in C which is intended to serve as a versatile tool for the development and implementation of expert systems. CLIPS allows a programmer to code with most of the functionality available in a procedural language like C, within a forward-chaining environment. However, CLIPS has its own drawbacks. Some sub-routines had to be written and executed in C simply because CLIPS does not have the full functionality of a procedural language. The CLIPS syntax is uncommon. Also, it is relatively complex to integrate CLIPS with external functions. These drawbacks prevent a programmer from learning CLIPS rapidly. An expert system shell like CLIPS is a prime candidate for programming in the domain of network fault diagnosis because of the following reasons [Liebowitz88]:

1. The problem is bounded. The network fault diagnosis task is large enough to be of interest but not so large as to become unmanageable.

2. There is an expert available. BNR personnel were very cooperative in dedicating expertise for the knowledge acquisition phase.
3. There are almost as many exceptions as there are rules. Procedural programming is well suited for problems and resolution algorithms which can be clearly defined from the start. Network fault isolation and correction is a prime target for expert system development because the diagnostic procedures are not clearly defined and the tasks involved rely on incomplete or ambiguous information to make informed judgements.
4. Test data are readily available. Since expert system technology is usually based on rapid prototyping, test data must be available when the prototype is evolving. BNR management provided the Pegasus team with their network topology and periodic status messages files. These files gave status changes of the different elements and lines in the network. Testing was also made possible with the development of the Dragon simulator which replicates to a great extent the topology and behaviour of the real BNR WAN network.

9.3 Future Research

END demonstrated the effectiveness of using expert system technology as an

integral part of a multi-paradigm approach to network fault diagnosis. However, there are still areas which need further research. One area is the single path generation problem mentioned in Chapter 6. In a routing problem, END only considers the most probable routing path between node A and node B to verify all the elements and lines on that path. With dynamic rerouting, there is definitely more than one path where the fault may have occurred. Integrating a multiple path generator within the project would improve its diagnosis accuracy. Another interesting future area of research is the development of graphical interfaces to END and to other Pegasus sub-systems. Finally, END can be expanded to cover more classes of faults such as handling problems at all communication protocol layers. Alpha tests on the real BNR network would reinforce the fact that END and the Pegasus system present a very effective approach to network diagnosis. Alpha tests would also bring to light other types of problems to investigate.

9.4 Towards an Autonomous Network

The ultimate goal of this research was to develop a virtually autonomous system capable of diagnosing and recommending remedial action for the prevention, isolation and repair of most BNR network faults with minimum human intervention and specialized skills. Further developments and tests on END should be continued with this goal in mind. However, there is one important factor to consider, particularly during alpha tests: the knowledge engineer should have END request permission from the network operator or user before END is allowed to perform any alterations to the network, such as

rebooting a station which may be defective.

Although this research focused on network fault diagnosis, expert system technology is also a prime candidate for other telecommunication applications such as network performance, configuration, planning, design and training. Expert system development is a relatively new and untested technology where few support tools exist. By combining the qualities of humans and machines (favouring the latter), expert system technology represents a very powerful potential tool in tomorrow's complex, domain-restricted applications.

References

[BNR90]

BNR Information Technology, "BNR Local Area Network: An Introduction for the Network User," 1990, pp. 3-15.

[Brownston85]

L. Brownston, R. Farrell, F. Kant, N. Martin, "Programming Expert Systems in OPS5," Addison-Wesley, Boston, Massachusetts, 1985.

[Cebulka89]

Kathleen D. Cebulka, Michael J. Muller, Christine A. Riley, "Applications of Artificial Intelligence for Meeting Network Management Challenges in the 1990's," IEEE Global Telecommunications Conference & Exhibition, Dallas, Texas, 1989.

[Clitherow89]

P. Clitherow, G. Fisher. "Knowledge Based Assistance of Genetic Search in Large Design Spaces," In 2nd International Conference on Industrial and Engineering Applications of AI and Expert Systems, Tullahoma, TN, June 1989.

[Dawes91a]

N.W. Dawes, "Dragon: A Very High Speed, Very Large Scale Network Simulator," TRIO SCE-91-31, Carleton University, Ottawa, 1991.

[Dawes91b]

N.W. Dawes, J. Altoft, B. Pagurek, "Network Diagnosis by Reasoning in Uncertain Nested Evidence Spaces," TRIO SCE 91-26, Carleton University, Ottawa, 1991.

[Dawes 91c]

N.W. Dawes, "Pegasus - General Architecture and Designs," TRIO SCE-91-04, Carleton University, Ottawa, 1991.

[Giarratano89]

Joseph Giarratano, Gary Riley, "Expert Systems Principles and Programming," PWS-Kent Publishing Co., Boston, Massachusetts, 1989, pp. 36-48.

[Giarratano91a]

Joseph Giarratano, "CLIPS User's Guide," Volume I: Rules, NASA Lyndon B. Johnson Space Center, 1991.

[Giarratano91b]

Joseph Giarratano, "CLIPS Reference Manual," Volume I: Basic Programming Guide, NASA Lyndon B. Johnson Space Center, 1991.

[Giarratano91c]

Joseph Giarratano, "CLIPS Reference Manual," Volume II: Advanced Programming Guide, NASA Lyndon B. Johnson Space Center, 1991.

[Goodman89]

R.M. Goodman, J. Miller, P. Smyth, "Real Time Autonomous Expert Systems in Network Management," First IFIP International Symposium on Integrated Network Management, Boston, May 1989.

[Guattery85]

S. Guattery, F.J. Villarreal, "NEMYSYS: An Expert System for Fighting Congestion in the Long Distance Network," Proceedings of Expert Systems in Government Symposium, K. Karna (ed.), McLean, Va., Oct. 1985.

[Kahn87]

N. Kahn, P. Callahan, R. Dube, J. Tsay, W. Van Dusen, "An Engineering Approach to Model Based Trouble Shooting in Communications Networks," Proceedings of IEEE GLOBECOM 1987, Tokyo, Japan, Nov. 1987.

[Kunz84]

J. Kunz, T. Kehler, M. Williams, "Applications Development Using a Hybrid AI Development System," AI Magazine, Vol. 5, No. 3, 1984.

[Liebowitz88]

Jay Liebowitz, "Expert System Applications to Telecommunications," John Wiley & Sons, Toronto, 1988, pp. 3-43.

[Lippmann87]

R.P. Lippmann, "An Introduction to Computing with Neural Networks," IEEE ASSP Magazine, April 1987, pp. 4-22.

[Marques88]

T.E. Marques, "A Symptom Driven Expert System for Isolating and Correcting Network Faults," IEEE Communications Magazine, Vol. 26, No.3, Mar. 1988.

[Miller85]

F.M. Miller, G.V.E. Otto, E.M. Siegfried, P.E. Zeldin, "ACE: Knowledge Based Maintenance Analyzer," Proceedings of the IEEE Automated Testing Conference, 1985.

[Milne87]

Robert Milne, "Strategies for Diagnosis," IEEE Transactions on Systems, MAN, and Cybernetics, Vol. SMC-17, No. 3, 1987.

[Rabie88]

S. Rabie, A. Rau-Chaplin, T. Shibahara, "DAD: A Real Time Expert System for Monitoring of Data Packet Networks," IEEE Network Magazine, Vol. 2, No. 5, Sept. 1988.

[Williams83]

T.L. Williams, P.J. Orgren, C.L. Smith, "Diagnosis of Multiple Faults in a Nationwide Communications Network," Proceedings of 8th IJCAI, Karlsruhe, West Germany, Aug. 1983.

Appendix A: END Network Overload

Tracer Module (in part)

; Rule one initiates access to the blackboard

```
11 (defrule one
12   (initial-fact)
13   =>
14   (printout t "rule one fired" crlf)
15   (initial)
16   (bind ?no 1)
17   (assert (start ?no)))
```

; Rule two verifies if the reported problem requires a data storm search

```
18 (defrule two
19   ?start <- (start ?no)
111  =>
112   (retract ?start)
113   (printout t "rule two fired" crlf)
114   (bind ?problem (problem))
115   (bind ?source (nth 1 ?problem))
116   (bind ?destination (nth 2 ?problem))
117   (printout t "problem destination node name: " ?destination crlf)
118   (bind ?prob_type (nth 3 ?problem))
119   (if (= ?prob_type 2)
120     then
121       (assert (routing_prob ?source ?destination ?no)))
122   (if (= ?prob_type 1)
123     then
124       (assert (node_prob ?source ?no))))
```

; Rule three finds all the routers on the most probable path, verifies if they
 ; are all alive, examines the traffic on every port of every router on the
 ; path and activates data storm source and destination searches if there is
 ; any traffic higher than a certain threshold

```

125 (defrule three
126   ?routing <- (routing_prob ?source ?destination ?no)
127 =>
128   (retract ?routing)
129   (printout t crlf "source node: " ?source crlf)
130   (printout t crlf "destination node name: " ?destination crlf)
140   (bind ?dest (node_id ?destination))
141   (bind ?destination_id (nth 1 ?dest))
142   (printout t crlf "destination node id: " ?destination_id crlf)
143   (bind ?routerslist (routers_of_path ?source ?destination_id))
144   (printout t ?routerslist crlf)
145   (bind ?i 1)
146   (bind ?router (nth 1 ?routerslist))
147   (while (neq ?router f)
148     (bind ?telnet_ok (telnet ?router))
149     (if (eq ?telnet_ok no)
150       then
151         (printout t crlf "ROUTING PROBLEM IS WITH " ?router
152           " WHICH IS DOWN" crlf)
153         (bind ?stop yes) ; flag
154       else
155         (printout t "path router " ?router " is active " crlf))
156     (bind ?i (+ ?i 1))
157     (bind ?router (nth ?i ?routerslist)))
158   (bind ?j 1)
159   (bind ?router (nth 1 ?routerslist))
160   (bind ?problem1 no)
161   (while (neq ?router f)
162     (bind ?port 1)
163     (while (< ?port 15)
164       (bind ?destnode (get_destination ?router ?port))
165       (bind ?destnode2 (nth 1 ?destnode))
166       (printout t crlf "CHECKING STORM node " ?router " port " ?port
167         " for high traffic" crlf)
168       (bind ?sim (simultime))
169       (bind ?simtime1 (nth 1 ?sim))
170       (bind ?loaddest (traffic ?router ?port))
171       (bind ?load_dest1 (nth 1 ?loaddest))
172       (bind ?sourcenode empty)

```

```

173 (bind ?port3 1)
174 (while (neq ?router ?sourcenode)
175   (bind ?sourcenode2 (get_destination ?destnode2 ?port3))
176   (bind ?sourcenode (nth 1 ?sourcenode2))
177   (bind ?port3 (+ ?port3 1)))
178 (bind ?port3 (- ?port3 1))
179 (bind ?loadsour (traffic ?destnode2 ?port3))
180 (bind ?load_sour1 (nth 1 ?loadsour))
181 (bind ?dif_time 0)
182 (while (< ?dif_time 10)
183   (bind ?sim2 (simultime))
184   (bind ?simtime2 (nth 1 ?sim2))
185   (bind ?dif_time (- ?simtime2 ?simtime1)))
186 (bind ?loaddest2 (traffic ?router ?port))
187 (bind ?load_dest2 (nth 1 ?loaddest2))
188 (bind ?loadsour2 (traffic ?destnode2 ?port3))
189 (bind ?load_sour2 (nth 1 ?loadsour2))
190 (bind ?dif_dest_load (- ?load_dest2 ?load_dest1))
191 (printout t "DESTINATION LOAD DIFFERENCE FOR 10 SECS 192
INTERVAL:" ?dif_dest_load crlf)
193 (bind ?dif_sour_load (- ?load_sour2 ?load_sour1))
194 (printout t "SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: "
195   ?dif_sour_load crlf)
196 (if (and (> ?dif_dest_load 6000) (neq ?destnode2 ?router))
197   then
198     (bind ?no_overload 1)
199     (assert (check_dest_router ?destnode2 ?router ?port ?no_overload 1100
?no))
1101     (bind ?no_overload (+ ?no_overload 1)) ; number for overload
1102     (bind ?problem1 yes))
1103 (if (and (> ?dif_sour_load 6000) (neq ?destnode2 ?router))
1104   then
1105     (bind ?no_overload 1)
1106     (assert (check_source_router ?destnode2 ?router ?port ?no_overload 1107
?no))
1108     (bind ?no_overload (+ ?no_o'verload 1)) ; number for overload
1109     (bind ?problem1 yes))
1110 (bind ?port (+ ?port 1)))
1111 (bind ?j (+ ?j 1))
1112 (bind ?router (nth ?j ?routerslist)))
1113 (if (eq ?problem1 no)
1114   then
1115     (printout t "No overload on path " ?source " to " ?destination crlf)
1116     (bind ?no (+ ?no 1))

```

```
1117 (assert (start ?no))))
```

; Rule four is the data storm source search

```
1118 (defrule four
1119 ?check <- (check_source_router ?router ?previous_router ?port 1120
?no_overload ?no)
1121 =>
1122 (retract ?check)
1123 (bind ?port2 1)
1124 (bind ?problem2 no)
1125 (while (< ?port2 15)
1126 (bind ?destnode (get_destination ?router ?port2))
1127 (bind ?destnode2 (nth 1 ?destnode))
1128 (printout t "Destination node name: " ?destnode2 crlf)
1129 (if (neq ?previous_router ?destnode2)
1130 then
1131 (printout t "CHECKING SOURCE 2 node " ?router " port "
1132 ?port2 " for high traffic" crlf)
1133 (bind ?sim (simultime))
1134 (bind ?simtime1 (nth 1 ?sim))
1135 (bind ?sourcenode empty)
1136 (bind ?port3 1)
1137 (bind ?count 0)
1138 (while (and (neq ?router ?sourcenode) (< ?count 20))
1139 (bind ?sourcenode2 (get_destination ?destnode2 ?port3))
1140 (bind ?sourcenode (nth 1 ?sourcenode2))
1141 (bind ?count (+ ?count 1))
1142 (bind ?port3 (+ ?port3 1)))
1143 (bind ?port3 (- ?port3 1))
1144 (bind ?load11 (traffic ?destnode2 ?port3))
1145 (bind ?load1 (nth 1 ?load11))
1146 (bind ?dif_time 0)
1147 (while (< ?dif_time 10)
1148 (bind ?sim2 (simultime))
1149 (bind ?simtime2 (nth 1 ?sim2))
1150 (bind ?dif_time (- ?simtime2 ?simtime1)))
1151 (bind ?load22 (traffic ?destnode2 ?port3))
1152 (bind ?load2 (nth 1 ?load22))
1153 (bind ?dif_load (- ?load2 ?load1))
1154 (printout t "SOURCE NODE LOAD DIFFERENCE FOR 10 SECS 1155
INTERVAL:" ?dif_load crlf)
1156 (if (and (> ?dif_load 6000) (neq ?destnode2 ?router))
```

```
1157         then
1158             (assert (check_source_router ?destnode2 ?router
1159                 ?port ?no_overload ?no))
1160             (bind ?no_overload (+ ?no_overload 1))
1161             (bind ?problem2 yes)))
1162     (bind ?port2 (+ ?port2 1)))
1163 (if (eq ?problem2 no)
1164     then
1165     (printout t crlf "**** ROUTING PROBLEM IS CAUSED BY
1166         DATA STORM FROM " ?router " ****" crlf)))
```


Appendix B: END Network Overload Tracer On-screen Result Scenario

```

CLIPS (v5.0 3/05/91)
CLIPS> (batch "go4")
CLIPS> (clear)
CLIPS> (watch all)
CLIPS> (load "/home/antares/dussault/veridatastorm2")
Defining defrule: one +j
Defining defrule: two +j
Defining defrule: three +j
Defining defrule: four +j
Defining defrule: foura +j
Defining defrule: five +j
CLIPS> (reset)
==> f-0 (initial-fact)
==> Activation 0 one: f-0
CLIPS> (run)
FIRE 1 one: f-0
rule one fired
==> f-1 (start 1)
==> Activation 0 two: f-1
FIRE 2 two: f-1
<== f-1 (start 1)
rule two fired
Waiting for a problem to be reported
problem returned = 2
Found reported problem
problem source node name: bgrgc1
problem destination node name: e0.batp
problem type: 2
==> f-2 (routing_prob bgrgc1 e0.batp 1)
==> Activation 0 three: f-2
FIRE 3 three: f-2
<== f-2 (routing_prob bgrgc1 e0.batp 1)

source node: bgrgc1

destination node name: e0.batp

```

* ref point A *

* ref point B *

* ref point C *

Waiting to get a node id

reply returned = 2

Found node_id

destination node id: 3

Waiting to get the routers of the path

* ref point D *

event returned = 3

Found path router

event returned = 4

Found path router

event returned = 5

Found path router

event returned = 6

Found last path router

(bGRGc1 bRCHc1 bRCHc2 bATPc1 f)

Waiting for a reply to a telnet request

* ref point E *

reply returned = 7

Telnet session successful

path router bGRGc1 is active

Waiting for a reply to a telnet request

reply returned = 8

Telnet session successful

path router bRCHc1 is active

Waiting for a reply to a telnet request

reply returned = 9

Telnet session successful

path router bRCHc2 is active

Waiting for a reply to a telnet request

reply returned = 10

Telnet session successful

path router bATPc1 is active

CHECKING STORM node bGRGc1 port 1 for high traffic * ref point F *

Destination node name: e0.bGRG

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 2 for high traffic

Destination node name: e1.bGRG

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 3 for high traffic
Destination node name: bRCHc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 4 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 5 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 6 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 7 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 8 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 9 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 10 for high traffic
Destination node name: bGRGc1

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGFGc1 port 11 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 12 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 13 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bGRGc1 port 14 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 1 for high traffic * ref point G *
Destination node name: nNCXc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 144
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 2 for high traffic
Destination node name: bRTPc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 368
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 3 for high traffic
Destination node name: bGRGc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 72
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

CHECKING STORM node bRCHc1 port 4 for high traffic
Destination node name: e0.bRCH
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 5 for high traffic
Destination node name: e1.bRCH
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 6 for high traffic
Destination node name: e2.bRCH
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 7 for high traffic
Destination node name: e3.bRCH
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 8 for high traffic
Destination node name: bRCHc2
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 72
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

CHECKING STORM node bRCHc1 port 9 for high traffic
Destination node name: bRCHc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 10 for high traffic
Destination node name: bRCHc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 11 for high traffic

Destination node name: bRCHc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 12 for high traffic
Destination node name: bRCHc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 13 for high traffic
Destination node name: bRCHc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc1 port 14 for high traffic
Destination node name: bRCHc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 1 for high traffic * ref point H *
Destination node name: bATPc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 2 for high traffic
Destination node name: e0.bRCH
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 3 for high traffic
Destination node name: e1.bRCH
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 4 for high traffic
Destination node name: bRCHc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 5 for high traffic

Destination node name: bRCHc2

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 6 for high traffic

Destination node name: bRCHc2

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 7 for high traffic

Destination node name: bRCHc2

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 8 for high traffic

Destination node name: bRCHc2

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 9 for high traffic

Destination node name: bRCHc2

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 10 for high traffic

Destination node name: bRCHc2

DESTINATION

REFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 11 for high traffic
 Destination node name: bRCHc2
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 12 for high traffic
 Destination node name: bRCHc2
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 13 for high traffic
 Destination node name: bRCHc2
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bRCHc2 port 14 for high traffic
 Destination node name: bRCHc2
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 1 for high traffic * ref point 1 *
 Destination node name: bRTPc1
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 644
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 56256

==> f-3 (check_source_router bRTPc1 bATPc1 1 1 1)
 ==> Activation 0 foura: f-3

CHECKING STORM node bATPc1 port 2 for high traffic
 Destination node name: e0.bATP
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 3 for high traffic
 Destination node name: e1.bATP

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 4 for high traffic
Destination node name: e2.bATP
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 5 for high traffic
Destination node name: e3.bATP
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 6 for high traffic * ref point J *
Destination node name: bSTMcl
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 63828
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

==> f-4 (check_dest_router bSTMcl bATPc1 6 1 1)
==> Activation 0 four: f-4

CHECKING STORM node bATPc1 port 7 for high traffic
Destination node name: PnATLc2
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 144
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 184

CHECKING STORM node bATPc1 port 8 for high traffic
Destination node name: bRCHc2
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 9 for high traffic
Destination node name: bATPc1
DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 10 for high traffic
Destination node name: bATPc1

DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 11 for high traffic
 Destination node name: bATPc1
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 12 for high traffic
 Destination node name: bATPc1
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 13 for high traffic
 Destination node name: bATPc1
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

CHECKING STORM node bATPc1 port 14 for high traffic
 Destination node name: bATPc1
 DESTINATION LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0
 SOURCE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

FIRE 4 four: f-4
 < == f-4 (check_dest_router bSTMcl bATPc1 6 1 1) * ref point K *
 Destination node name: bATPc1
 Destination node name: e0.bSTM
 CHECKING DESTINATION 2 node bSTMcl port 2 for high traffic
 DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e1.bSTM
 CHECKING DESTINATION 2 node bSTMcl port 3 for high traffic
 DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl
 CHECKING DESTINATION 2 node bSTMcl port 4 for high traffic
 DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl
 CHECKING DESTINATION 2 node bSTMcl port 5 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 6 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 7 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 8 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 9 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 10 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 11 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 12 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 13 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bSTMcl

CHECKING DESTINATION 2 node bSTMcl port 14 for high traffic

DESTINATION NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

* ref point L *

**** ROUTING PROBLEM IS CAUSED BY DATA STORM TO bSTMcl ****

FIRE 5 foura: f-3

< == f-3 (check_source_router bRTPcl bATPcl 1 1 1) * ref point M *

Destination node name: bATPcl

Destination node name: bCARc4
 CHECKING SOURCE 2 node bRTPc1 port 2 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 59044

==> f-5 (check_source_router bCARc4 bRTPc1 1 1 1) * ref point N *

==> Activation 0 foura: f-5

Destination node name: e0.bRTP
 CHECKING SOURCE 2 node bRTPc1 port 3 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 368

Destination node name: e1.bRTP
 CHECKING SOURCE 2 node bRTPc1 port 4 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e2.bRTP
 CHECKING SOURCE 2 node bRTPc1 port 5 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bRCHc1
 CHECKING SOURCE 2 node bRTPc1 port 6 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 184

Destination node name: bRTPc2
 CHECKING SOURCE 2 node bRTPc1 port 7 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 920

Destination node name: bRTPc3
 CHECKING SOURCE 2 node bRTPc1 port 8 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bRTPcA
 CHECKING SOURCE 2 node bRTPc1 port 9 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 184

Destination node name: bRTPcB
 CHECKING SOURCE 2 node bRTPc1 port 10 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bRTPcZ
 CHECKING SOURCE 2 node bRTPc1 port 11 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bRTPc1
 CHECKING SOURCE 2 node bRTPc1 port 12 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bRTPc1

CHECKING SOURCE 2 node bRTPc1 port 13 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bRTPc1

CHECKING SOURCE 2 node bRTPc1 port 14 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

FIRE 6 foura: f-5

<== f-5 (check_source_router bCARc4 bRTPc1 1 1 1) * ref point O *

Destination node name: bRTPc1

Destination node name: nTORc1

CHECKING SOURCE 2 node bCARc4 port 2 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 1748

Destination node name: e0.bCAR

CHECKING SOURCE 2 node bCARc4 port 3 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 32956

==> f-6 (check_source_router e0.bCAR bCARc4 1 1 1) * ref point P *

==> Activation 0 foura: f-6

Destination node name: e1.bCAR

CHECKING SOURCE 2 node bCARc4 port 4 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bMTLc1

CHECKING SOURCE 2 node bCARc4 port 5 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

Destination node name: bMDHc1

CHECKING SOURCE 2 node bCARc4 port 6 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: nBRKc1

CHECKING SOURCE 2 node bCARc4 port 7 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bCARc2

CHECKING SOURCE 2 node bCARc4 port 8 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 13468

==> f-7 (check_source_router bCARc2 bCARc4 1 2 1) * ref point Q *

==> Activation 0 foura: f-7

Destination node name: bCARc3

CHECKING SOURCE 2 node bCARc4 port 9 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bCARc1

CHECKING SOURCE 2 node bCARc4 port 10 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 276

Destination node name: bCARcB

CHECKING SOURCE 2 node bCARc4 port 11 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bCARcW

CHECKING SOURCE 2 node bCARc4 port 12 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bCARcX

CHECKING SOURCE 2 node bCARc4 port 13 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 460

Destination node name: bCARcY

CHECKING SOURCE 2 node bCARc4 port 14 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

FIRE 7 foura: f-7

<== f-7 (check_source_router bCARc2 bCARc4 1 2 1) * ref point R *

Destination node name: bWDLc2

CHECKING SOURCE 2 node bCARc2 port 1 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 63036

==> f-8 (check_source_router bWDLc2 bCARc2 1 2 1) * ref point S *

==> Activation 0 foura: f-8

Destination node name: bKANc2

CHECKING SOURCE 2 node bCARc2 port 2 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 184

Destination node name: bFTZc1

CHECKING SOURCE 2 node bCARc2 port 3 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 184

Destination node name: e0.bCAR

CHECKING SOURCE 2 node bCARc2 port 4 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 6440

==> f-9 (check_source_router e0.bCAR bCARc2 1 3 1) * ref point T *

==> Activation 0 foura: f-9

Destination node name: e1.bCAR

CHECKING SOURCE 2 node bCARc2 port 5 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e2.bCAR

CHECKING SOURCE 2 node bCARc2 port 6 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e3.bCAR

CHECKING SOURCE 2 node bCARc2 port 7 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e4.bCAR

CHECKING SOURCE 2 node bCARc2 port 8 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e5.bCAR

CHECKING SOURCE 2 node bCARc2 port 9 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bQNNc1

CHECKING SOURCE 2 node bCARc2 port 10 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

Destination node name: bCARc3

CHECKING SOURCE 2 node bCARc2 port 11 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 552

Destination node name: bCARc1

CHECKING SOURCE 2 node bCARc2 port 12 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 184

Destination node name: bCARc4

Destination node name: bCARcB

CHECKING SOURCE 2 node bCARc2 port 14 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

FIRE 8 foura: f-9

<== f-9 (check_source_router e0.bCAR bCARc2 1 3 1) * ref point U *

Destination node name: e0.bCARbr

CHECKING SOURCE 2 node e0.bCAR port 1 for high traffic

SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 552

Destination node name: bCARc2
Destination node name: bCARc3
CHECKING SOURCE 2 node e0.bCAR port 3 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 828

Destination node name: bCARc1
CHECKING SOURCE 2 node e0.bCAR port 4 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

Destination node name: bCARc4
CHECKING SOURCE 2 node e0.bCAR port 5 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 4508

Destination node name: bCARcB
CHECKING SOURCE 2 node e0.bCAR port 6 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 92

Destination node name: bCARcW
CHECKING SOURCE 2 node e0.bCAR port 7 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bCARcX
CHECKING SOURCE 2 node e0.bCAR port 8 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bCARcY
CHECKING SOURCE 2 node e0.bCAR port 9 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bCARcZ
CHECKING SOURCE 2 node e0.bCAR port 10 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 276

Destination node name: e0.bCAR
CHECKING SOURCE 2 node e0.bCAR port 11 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bCAR
CHECKING SOURCE 2 node e0.bCAR port 12 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bCAR
CHECKING SOURCE 2 node e0.bCAR port 13 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bCAR
 CHECKING SOURCE 2 node e0.bCAR port 14 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

* ref point V *

**** ROUTING PROBLEM IS CAUSED BY DATA STORM FROM e0.bCAR ****

FIRE 9 foura: f-8
 <== f-8 (check_source_router bWDLc2 bCARc2 1 2 1) * ref point W *
 Destination node name: bCARc2
 Destination node name: e0.bWDL
 CHECKING SOURCE 2 node bWDLc2 port 2 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 542044

==> f-10 (check_source_router e0.bWDL bWDLc2 1 2 1) * ref point X *
 ==> Activation 0 foura: f-10
 Destination node name: e1.bWDL
 CHECKING SOURCE 2 node bWDLc2 port 3 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e2.bWDL
 CHECKING SOURCE 2 node bWDLc2 port 4 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc1
 CHECKING SOURCE 2 node bWDLc2 port 5 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 420

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 6 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 7 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 8 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 9 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 10 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 11 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 12 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 13 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: bWDLc2
 CHECKING SOURCE 2 node bWDLc2 port 14 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

FIRE 10 foura: f-10
 < == f-10 (check_source_router e0.bWDL bWDLc2 1 2 1) * ref point Y *
 Destination node name: e0.bWDLbr
 CHECKING SOURCE 2 node e0.bWDL port 1 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 602740

==> f-11 (check_source_router e0.bWDLbr e0.bWDL 1 2 1)
 ==> Activation 0 foura: f-11
 Destination node name: bWDLc2
 Destination node name: bWDLc1
 CHECKING SOURCE 2 node e0.bWDL port 3 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 1340

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 4 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 5 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 6 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 7 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 8 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 9 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 10 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 11 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 12 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 13 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 CHECKING SOURCE 2 node e0.bWDL port 14 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

FIRE 11 foura: f-11 * ref point Z *
 < == f-11 (check_source_router e0.bWDLbr e0.bWDL 1 2 1)
 Destination node name:
 CHECKING SOURCE 2 node e0.bWDLbr port 1 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDL
 Destination node name: e0.bWDLbr
 CHECKING SOURCE 2 node e0.bWDLbr port 3 for high traffic
 SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0

Destination node name: e0.bWDLbr

**CHECKING SOURCE 2 node e0.bWDLbr port 4 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 5 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 6 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 7 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 8 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 9 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 10 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 11 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 12 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 13 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

**Destination node name: e0.bWDLbr
CHECKING SOURCE 2 node e0.bWDLbr port 14 for high traffic
SOURCE NODE LOAD DIFFERENCE FOR 10 SECS INTERVAL: 0**

*** ref point AA ***

**** ROUTING PROBLEM IS CAUSED BY DATA STORM FROM e0.bWDLbr ****

END

2/3·12·92

FIN