Fault Detection and Identification in Computer Networks: A Soft Computing Approach

by

Abduljalil Mohamed

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Systems Design Engineering

Waterloo, Ontario, Canada, 2009

Abduljalil Mohamed

Abstract

Governmental and private institutions rely heavily on reliable computer networks for their everyday business transactions. The downtime of their infrastructure networks may result in millions of dollars in cost. Fault management systems are used to keep today's complex networks running without significant downtime cost, either by using active techniques or passive techniques. Active techniques impose excessive management traffic, whereas passive techniques often ignore uncertainty inherent in network alarms, leading to unreliable fault identification performance. In this research work, new algorithms are proposed for both types of techniques so as address these handicaps.

Active techniques use probing technology so that the managed network can be tested periodically and suspected malfunctioning nodes can be effectively identified and isolated. However, the diagnosing probes introduce extra management traffic and storage space. To address this issue, two new CSP (Constraint Satisfaction Problem)-based algorithms are proposed to minimize management traffic, while effectively maintain the same diagnostic power of the available probes. The first algorithm is based on the standard CSP formulation which aims at reducing the available dependency matrix significantly as means to reducing the number of probes. The obtained probe set is used for fault detection and fault identification. The second algorithm is a fuzzy CSP-based algorithm. This proposed algorithm is adaptive algorithm in the sense that an initial reduced fault detection probe set is utilized to determine the minimum set of probes used for fault identification. Based on the extensive experiments conducted in this research both algorithms have demonstrated advantages over existing methods in terms of the overall management traffic needed to successfully monitor the targeted network system.

Passive techniques employ alarms emitted by network entities. However, the fault evidence provided by these alarms can be ambiguous, inconsistent, incomplete, and random. To address these limitations, alarms are correlated using a distributed Dempster-Shafer Evidence Theory (DSET) framework, in which the managed network is divided into a cluster of disjoint management domains. Each domain is assigned an Intelligent Agent for collecting and analyzing the alarms generated within that domain. These agents are coordinated by a single higher level entity, i.e., an agent manager that combines the partial views of these agents into a global one. Each agent employs DSET-based algorithm that utilizes the probabilistic knowledge encoded in the available fault propagation model to construct a local composite alarm. The Dempster‘s rule of combination is then used by the agent manager to correlate these local composite alarms. Furthermore, an adaptive fuzzy DSET-based algorithm is proposed to utilize the fuzzy information provided by the observed cluster of alarms so as to accurately identify the malfunctioning network entities. In this way, inconsistency among the alarms is removed by weighing each received alarm against the others, while randomness and ambiguity of the fault evidence are addressed within soft computing framework. The effectiveness of this framework has been investigated based on extensive experiments.

The proposed fault management system is able to detect malfunctioning behavior in the managed network with considerably less management traffic. Moreover, it effectively manages the uncertainty property intrinsically contained in network alarms, thereby reducing its negative impact and significantly improving the overall performance of the fault management system.

Acknowledgements

First and foremost, I am deeply indebted to my supervisor Dr. Otman Basir for his continuous encouragement, helpful discussions, and insightful guidance throughout the research presented in this thesis. Without his continuous support and invaluable supervision this work would not have been possible.

I would like to extend my sincere gratitude to the members of my doctoral committee, Dr. Simon Yang, Dr. Keith Hipel, Dr. Ramadan El-Shatshat, and Dr. Eihab Abdel-Rahman for the time they spent to read this thesis and for their significant remarks and comments.

I also acknowledge Dr. William Philips of DalTech, Dalhousie University for his valuable help during my initial stage of graduate studies.

I would like to thank the faculty, staff, and students of the Pattern Analysis and Machine Intelligence (PAMI) research laboratory for the friendly atmosphere they have created. My special thanks go to Mr. Khaled Hammouda for his technical support with my various needs. Also I am very grateful to my friends Naser Younis, Ahmad Husein, Akrem Elgazel, Fathi Idris, Naji Al-Amrouni and the rest of the Libyan community in Waterloo for their support.

Last, but not least, I thank my family. I owe great deal of gratitude to my mother, the source of my strength. I express my great appreciations to all my brothers, Mohamed, Hassan, Ali, and Abdul-Salam for their continuous understanding and unlimited encouragement. I would not have come this far without their moral and physical support.

To my beloved brother
Dr.
Ali Abdul-Rahman Mohamed Netfa

Contents

List of Tables

List of Figures

List of Notations

A_{comp}	Local composite alarm
A_{Fcomp}	Local fuzzy composite alarm
A_k	The alarm space of intelligent agent *k*
$\overline{A}_k$	Set of observed alarms received by intelligent agent *k*
AC	Alarm Correlation
AM	Agent manager
ANN	Artificial neural network
AS	Autonomous system
Bayes	Bayesian approach
Bel(A)	Belief measure of A
BGP	Border gateway protocol
B^0	The support set of fuzzy set B
C	Cluster of alarms
C_{ave}	Average satisfaction of individual constraints
CSP	Constraint satisfaction problem
$dom_C()$	Domain of a fuzzy constraint
DAG	Directed acyclic graph
$D(a_i)$	Domain of alarm a_i
$D(f_j)$	Domain of network failure f_j
DSET	Dempster-Shafer evidence theory, or Dempster-Shafer evidence theory based reasoning

F_t	The set of the suspected nodes, at iteration t
F_X	The set of failures at node *X*
$\overline{F}_r$	Fault hypotheses of alarm cluster C_r
FCSP	Fuzzy constraint satisfaction problem
FPM	Fault propagation model
FSM	Finite state machine
G_t	The set of passed nodes, at iteration t
IA	Intelligent agent
ISO	International Standard Organization
k	Number of intelligent agents
MAC	Maintaining arc consistency
MAP	Maximum a posterior principal
MCD	Maximum commonality decision rule
MIB	Management Information Base
MODEL	Modeling Event Language
n	Number of network alarms
NDG	Network Dependency Graph
NE	Network entity
OSI	Open Systems Interconnection
$p(f_j)$	Prior probability of network failure f_j
$p(s \mid f_j)$	Conditional probability function of symptom *s* given network failure f_j
P_t	The set of the available probes
PDES	Proportional difference evidence structure scheme
PDU	Protocol data unit
Pl(A)	Plausibility measure of A
Ppig(A)	Pignistic probability of A
Q(A)	Commonality measure of A
SNMP	Simple Network Management Protocol
SOM	Kohonen self-organizing Map
TCP/IP	Transmission control/internet protocol

UDP	User data protocol
V	The set of random variables at Bayesian network
$\oplus$	Dempster’s combination operator
Σ_F	The domain of the variable of the suspected nodes
Σ_P	The domain of the variable of the available probes
α	Normalizing constant
μ_B	Membership function of the fuzzy set B
Ω	Frame of discernment
$\mid \cdot \mid$	Cardinality
$\delta(\cdot)$	Dirac’s delta function. $\delta(\cdot) = 1$ if the inside condition is satisfied; otherwise 0

Chapter 1

Introduction

1.1 Preface

As computer networks continue to grow in size and complexity, effective network management is expected to become even more crucially important and more challenging. Simply stated, the aim of a typical network management system is to monitor the managed system and to ensure that it is running as smoothly as possible. In order for the management systems to successfully manage the network a large amount of diagnostic information needs to be obtained and processed. This information can be either acquired using certain monitoring tools [1, 2, 3, 4, 5], or received from network entities in the form of network alarms [6, 7, 8, 9]. As such, fault management systems can be divided into two paradigms: (1) fault management systems that actively sample performance data from the managed network, commonly referred to as probe-based systems or active systems; and (2) fault management systems that utilize network alarms, commonly referred to as alarm correlation-based systems or passive systems. Both paradigms address certain challenges and offer alternative solutions to the fault network management problem and as such they may have their own merits and bear their own shortcomings. In this chapter, we present a brief review of the challenges and motivations of network fault management systems in the context of both paradigms.

1.2 Open Systems Interconnection Model

In order to appreciate the importance and complexity of fault management systems one needs to understand the more general topic of network management. According to the International Standard Organization (ISO), effective management of highly complex networks should be approached with respect to five main objectives, as described in the Open Systems Interconnection (OSI) model [10]: Configuration Management, Fault Management, Performance Management, Security Management, and Accounting Management. The objective of configuration management function is to initialize the network components and establish relationships among these components and to keep the manager informed about the status of these components. It is also responsible for reconfiguring the network resources (such as create new paths between network nodes) in case of changes in network topology. The performance management objective is dedicated for monitoring the network resources and controls their behavior (such as keeping track of current activities in the network and adjusting network component parameters to improve network performance). Furthermore, it collects and analyzes crucial information about network traffic to locate any bottleneck along network paths. The security management objective ensures restrict access to sensitive information and also generates and stores encryption keys. Users of the network can be charged for using the network resources and services through the accounting function which is also responsible for the user billings.

In this thesis, we are concerned with a central aspect of the OSI network management model, namely, fault management. Existing network fault management systems utilize widely available monitoring tools (such as ping, trace route, web page access, etc.) to examine the health status of network components. Furthermore, modern network devices are highly instrumented and configured to send diagnostic information to their manager once they encounter some network failures. This diagnostic information is analyzed to identify faults and their root causes, if possible. However, due to their dependency nature, a faulty component may cause a state of malfunctioning in one or more other system components [11]. As a result of this phenomenon, a huge amount of diagnostic information is generated by these dependent entities and delivered to the network management stations. As this information is overwhelming, it is not possible for a human expert to analyze and process this information in a reasonable time frame. Therefore, human experts tend to rely on their experience and intuition to examine the network behavior and in due course may overlook some valuable diagnostic information obtained from network entities.

1.3 Network Fault Management Terminologies

Any exceptional state that may take place in a given network layer is referred to as *network event* [12]. *Network faults* (a special class of network events) manifest themselves in the form of *alarms* (or *symptoms)*. Thus, dedicated software agents are installed in the network to monitor, collect and process traffic data. These agents send network alarms (such as SNMP traps [12, 13] or CMIP EVENT_REPORTs [10]) as notifications of possible malfunctioning if one (or more) of their monitored network parameters exceeds a predefined thresholds. Other alarms may be obtained by other means such as a trace rout utility, ping commands, system log files, etc. However, some network faults may not be directly observable due to the lack of a management function that can provide indications of their existence. Network fault management systems rely on network alarms to infer the main causes. In general, network faults are classified based on their time duration in the managed network [14]:

- Permanent faults,
- Intermittent faults,
- Transient faults.

Permanent faults are self-explanatory and exist in a network until they are repaired.

Examples of such faults include: a broken cable, malfunctioning interface card. Intermittent faults occur in a discontinuous and periodic manner and tend to cause a failure of current processes, and therefore result in a maximum degradation in the service level for a short period of time. Transient faults momentarily cause minor degradation in the service, and as they are often masked by management utilities are not observable to the user.

The main tasks of fault management systems can be divided into three stages:

- Stage 1: Fault detection – the process by which network fault indicators, in the form of alarms generated by faulty network components, are captured on-line [15].
- Stage 2: Fault identification – the process of identifying the most likely causes of the received alarms [8, 15].
- Stage 3: Testing operation – the process of determining the actual faults that caused the network to malfunction [8].

The main focus of this research work will be on *Stage 1* and *Stage 2*. The work is divided into two main parts. In the first part, we introduce a new approach based on Constraint Satisfaction Problem (CSP) to find an appropriate and optimal collection of probes for the purpose of fault detection. Furthermore, we introduce a fuzzy CSP-model that can adaptively select the most appropriate probes. In the second part, we concentrate on the second stage of fault management to exploit the availability of network alarms that are observed by the management system in order to identify the root causes that may be responsible of their occurrence.

1.4 Motivations

In this section, the motivations behind the proposed intelligent fault management system are described in light of critical examination of the current fault management research activities and directions. Important areas pertaining to network management systems are reviewed.

1.4.1 Intelligent probing

Dedicated software tools used for monitoring network components are commonly known as *probes* [1]. These probes (e.g., pinging, trace routing, etc.) are widely available. Network management systems that are based on such tools make them an appealing alternative to the alarm based (passive) systems due to their ability capture faults more effectively. In probes based fault management probing stations are first determined and located in different parts of the network. A set of probes emitted by these probing stations is then sent to network management on periodical basis. These probes are collectively analyzed to determine whether a failure has been detected. To effectively isolate the malfunctioning network component, the set of probes must cover all the nodes in the managed network [1, 2]. Depending on the number and locations of available probing stations this set has to include a large number of probes to be of practical use. Employing a large number of these probes for fault identification tasks certainly increases the accuracy of locating network malfunctioning components; nevertheless, the fault identification task will become more time-consuming. Furthermore, a large set of probes entails excessive management traffic injected into the managed network which may in fact exacerbate the situation. Hence it is desirable to minimize the negative impact of the extra management traffic induced by these probes.

One of the main motivations in this research work is to explore the use of intelligent probing techniques that can help in reducing the number of these probes while preserving the quality of the diagnostic power of the original set.

1.4.2 Uncertainty Management of Fault Evidence

Another motivation of this research work revolves around the management of uncertainty associated with_fault evidence, as it pertains to network alarms. One of the widely used approaches for network fault management systems is based on network alarms generated by network entities as a response to network failures. An alarm correlation mechanism is often implemented to infer a fault hypothesis that is considered as an explanation for the observed alarms. The alarm correlation mechanism views these alarms as fault evidence [16, 17, 18]. However, networks are such complex systems and their unreliability and non-determinism clearly affect the quality of the obtained fault evidence. This unavoidable distortion of information may lead to the wrong conclusion as a fault hypothesis. Network alarms that constitute fault evidence may contain high degree of uncertainty with respect to the following aspects:

- Fault evidence ambiguity

  Different network faults may cause a set of alarms to be generated. However, in many practical situations, the monitoring systems may generate the same alarm as an indication of many different faults [17]. That is, the domain of a single alarm may include a set of different fault hypotheses. Hence, it is up to the management system to determine which particular fault may have caused the alarm.
- Fault evidence inconsistency

  Monitoring software agents residing in the network devices (managed objects) have their own view regarding the operation state of the managed network. In the event of a fault occurrence, and based on their internal network parameters, a managed object may indicate a certain network entity as the source of network malfunctioning and thus generate a corresponding alarm accordingly. However, another managed object may have different opinion and determine that the same network entity is working properly [17]. These conflicting assessments by different network objects are not unusual and seem to be the result of the heterogeneity nature of computer networks.
- Fault evidence incompleteness

  The set of alarms triggered by a fault occurrence in the managed network is often transmitted from the managed objects to network managers over unreliable transport protocols. SNMP agents, for example, send their traps (alarms) to management stations using the user data protocol (UDP) as their transmission mechanism. UDP utilizes a best effort policy and does not guarantee that the SNMP message reach their destinations. As a result, the SNMP messages that were lost during transmission will not be recovered. Therefore, the fault management system should have the ability to conduct inference with incomplete information [16, 19].

- Fault evidence inaccuracy

  Alarms that are generated by transient faults are called *spurious* alarms. Management systems are often equipped with error recovery procedures to repair transient faults. However, network alarms triggered by such faults should be discarded and not be taken into account in the diagnosis process since their main cause has been removed by the network management system. Avoiding and reducing the effects of these spurious alarms in the alarm correlation process is required [18, 19].

- Fault and symptom non-deterministic relationship

  The majority of fault diagnosis techniques adopted by network management systems are expert-based. Their appeal stems from the fact that the reasoning process of a diagnostic system is intuitively similar to that of a human expert. Thus the realization of a fault management system is the transfer of the human expert knowledge to an automated system. Most of these systems are based on ad-hoc, unstructured deterministic network models; and the cause-and-effect relationship between network faults and their corresponding alarms is inherently non-deterministic [16, 18, 19, 20]. A probabilistic model is considered as a more accurate representation for the network fault models.

1.4.3 Distributed Fault Management System Architecture

Traditionally, network fault management systems consist of two main components: agents and managers. Agents are basically monitoring software components that are installed in every monitored entity in the computer network. They are responsible for collecting network traffic, storing it in ASN format, and monitoring some particular variables and sending traps to their managers when one or more of these variables exceed some predefined thresholds [12, 13].

The fault management activities are performed by the managers, upon retrieving some of these variables from their subordinate agents using the Simple Network Management Protocol (SNMP). This centralized framework of management system architecture can be tolerated in small size networks. As networks grow larger and become more heterogeneous, the centralized model creates a bottleneck at network management centers and introduces significant amounts of traffic, a considerable part of which is not important or necessary for the diagnosis process. Recently, more advanced fault management techniques have adopted a distributed approach by which the managed network is partitioned into distinct management domains, each managed by an independent management center [21, 22]. In this way, faults may be handled locally; thus reducing the amount of traffic that should be transmitted across the network. However, existing distributed techniques focus more on local view in the sense that the domain managers can only rely on its local information to identify the root cause of the network failure. Valuable global information is discarded in their fault analysis process. An effective fault management system should make this information available to all domain managers.

1.4.4 Automated Network Fault Management System

The complexity and heterogeneity of modern computer networks contribute, to a great extent, to the shear amount of information that floods the managed network when it experiences a malfunctioning behavior in one of its managed objects. A single fault may cause a large set of alarms to be generated and delivered to their corresponding domain manager. The occurrence of numerous alarms is highly attributed to, among others, the manager. The occurrence of numerous alarms is highly attributed to, among others, the following factors [6, 17]:

- Fault re-occurrence
- Multiple invocations of a service provided by a faulty component

- Repetitive alarm generation by the same device
- Fault propagation to other network components

Managing all this information by human experts is almost impossible. Therefore, an efficient fault management system that facilitates an effective and appropriate level of automation is greatly needed.

1.5 Contributions

Based on the above motivations, this dissertation is devoted to the design of an automated, intelligent, distributed fault management system for computer networks. The main objectives of the proposed network fault management system are:

- To reduce the overall management traffic (i.e., probes) required to periodically examine the components of the managed network without compromising diagnostic power,
- To alleviate the negative impact of the uncertainty problem that inherently exists in the fault evidence (i.e., network alarms) on the performance of the fault management system.

This research work introduces new methods for intelligent probing that can minimize the size of the probe set required for fault detection and identification. A distributed intelligent-agent-based fault management system is proposed. The proposed distributed system is based on the constraint satisfaction problem (CSP) formulation and the Dempster-Shafer Evidence Theory (DSET) [23]. The capabilities of DSET in dealing with imprecision and conflict that network alarms inherently possess make it suitable and appealing as a framework for knowledge representation and evidence propagation in computer networks. Fuzzy reasoning is employed to make the CSP more adaptive and to handle the positive alarms in the fault identification process.

This dissertation makes the following contributions to the field of network fault management:

- To minimize the number of alarms (symptoms) processed by network management systems an appropriate and effective method of selecting network probes should be developed. Probing technology is widely used as an end-to-end transaction that provides information about the availability of the nodes of the managed network. One such probing program used for determining network availability is the *ping* program. Other probing techniques include email messages, web-access requests, invoking a service from a database server, etc. Moreover, each probing transaction incurs cost to the network in terms of additional network management traffic. Considering the selection of an optimal number of available probes for diagnosis purposes as an optimization problem, we propose a novel constraint satisfaction problem (CSP) based model. The powerful search algorithms offered by the CSP techniques have been used to reduce the search space and produce an optimal set of probes in a very reasonable time.

- We propose a novel fuzzy CSP-based model to further reduce the number of probes required for the fault detection and identification tasks. In this new approach, instead of sending all the probes obtained by the dependency matrix, only a few informative probes are sent for fault detection purposes. If these probes returned successfully then the managed network is assumed to work properly and no further action is needed. The network manager waits for a fixed time interval and then sends these same probes again. Only if one or more of these probes fails to report back does the fault identification process proceeds. Using the information carried by the failed probe or probes, the fuzzy CSP model selects a new probe from the dependency matrix. The new probe is selected based on some criteria formulated as a set of fuzzy constraints. In contrast to the standard CSP, the fuzzy CSP (FCSP) implies that constraints need not be fully satisfied. Some instantiations of the problem variables may satisfy the problem constraints more than other instantiations. The FCSP model provides some flexibility in the probe selection process that is not achievable by the standard CSP. For instance, a candidate probe is selected only when the information inferred by its success or failure proves to be more valuable than other candidate probes. As such, the proposed scheme is adaptive in the sense that different outcomes of previous testing probes may yield different probes in the current probe selection process.

- Network elements often emit alarms in response to a fault. Each alarm represents the fault from the network element‘s point of view. We propose a DSET based approach for alarm correlation and fault identification. In the view of DSET, alarms emitted by a specific network element may only provide partial information about the fault. Our proposed technique collects partial observations of the network and infers the main cause of these alarms. It considers each received network alarm as a piece of evidence and a source of information. And as such, different alarms emitted by different network entities might have different assessment regarding the fault. The new technique constructs an evidence structure for each received alarm using the fault propagation model. DSET’s rule of combination is employed to fuse alarms represented as evidence structures.

- Treating each received network alarm as a source of information entails the recognition that different network alarms possess different diagnostic capabilities. To take these capabilities into considerations, we propose an adaptive fuzzy evidential reasoning based fault identification approach. Domain managers construct fuzzy evidence structures for each received network alarm using both fault propagation models and alarm domains (not to be confused with management domains). Hybrid entropy, as an information measure, is used for evaluating the overall uncertainty contained in the alarm fuzzy evidence structure. The new proposed algorithm utilizes two discounting schemes based on the obtained fuzzy evidence structures to achieve adaptive reasoning capabilities. For each network alarm, a local discounting scheme takes place during the decomposition of pieces of fuzzy evidence. A global discounting scheme is performed on the derived crisp evidence structures of the received alarms.

- Finally, to demonstrate the efficiency and effectiveness of the proposed network fault management system, extensive simulations are carried out for networks with simple network topologies and for networks with more complicated network topologies.

1.6 Thesis Outlines

The outline of the thesis is as follows: the existing state-of-the-art network fault management systems reported in the literature are discussed in Chapter 2, where we highlight their advantages and shortcomings. Since our proposed methods for the alarm-correlation based approaches demand the availability of fault propagation models which are used to build alarm evidence structures, we briefly discuss the different techniques used by network management systems to obtain these fault propagation models in Chapter 3. In Chapter 4, we introduce a novel approach for the selection of an optimal set of probes based on the technology of constrained satisfaction problem. In Chapter 5, an adaptive probe selection scheme based on a fuzzy CSP model is proposed by which outcomes of previous testing probes dynamically and adaptively influence the probe selection process. A new DSET based fault identification approach is proposed and discussed in Chapter 6. In Chapter 7, we present an adaptive fuzzy evidential reasoning based fault identification approach. In Chapter 8, we conduct extensive simulations to compare the proposed techniques with other popular approaches. We present our final thoughts regarding the research subject by including some suggestions for future work in Chapter 9.

Chapter 2

Fault Detection and Identification in Computer Networks

2.1 Introduction

A network fault management system may passively monitor the targeted network system by being on the look for indications of malfunctioning behavior; it can also proactively and periodically test network entities to determine the occurrence of a malfunctioning behavior. Modern networks are highly instrumented that when a failure occurs, several symptoms (alarms) are generated and sent to the network manager from different dependent network components. Fault management systems receive these alarms as input and produce an output in the form of a set of fault hypotheses (network failures) regarding these observed alarms. Furthermore, probes (such as, ping, trace route, etc.) can also be considered as symptoms. For example, a failed probe can be viewed as a negative observation of a symptom, which in this case may indicate that certain components in the failed probe path are not working properly. In this chapter, we provide a brief review of techniques reported in the literature with respect to both actual network alarms and monitoring probes.

2.2 Network Fault Management Schemes

As shown in Fig. 2.1, a fault management system analyzes symptoms such as network alarms or probe outcomes received during the period of a fault occurrence using different techniques and methods. In this section, we discuss well known existing approaches for the network fault detection and identification problem.

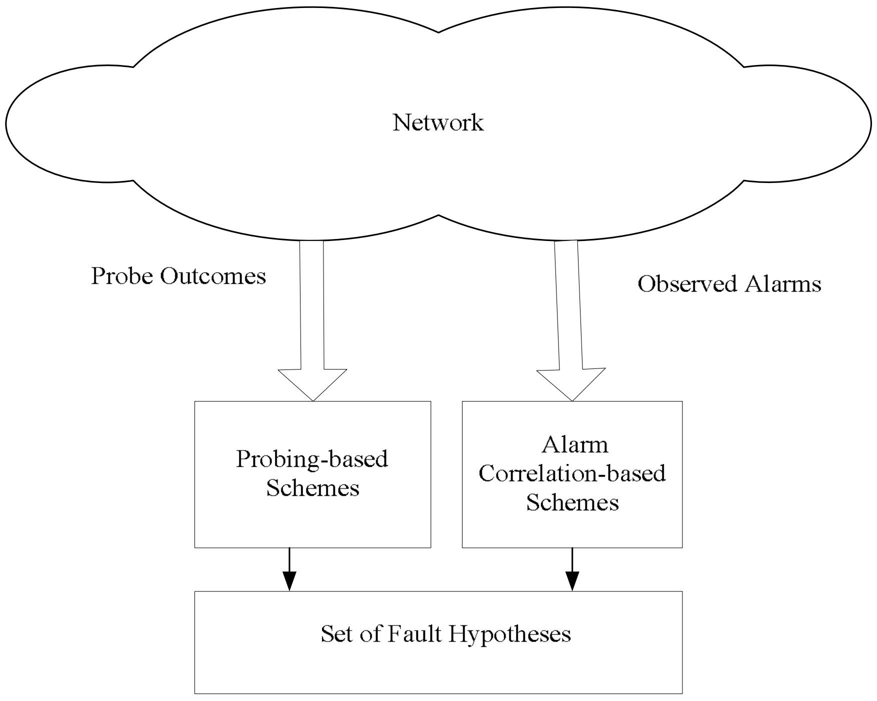


Figure 2.1: Proposed network fault management systems.

A network fault management system can be categorized based on the approach that it is built on, which is typically derived from one of the following paradigms:

- Control theory,
- Artificial Intelligence,

- Probability theory,
- Pattern Recognition theory.

Some fault management systems combine different approaches. For example, Hood and Ji present a hybrid approach in which they combine a probabilistic method in the form of Bayesian model of the managed network, with a proactive learning system to set auto regression parameters [24]. Broadly speaking, fault management system approaches can be divided into four categories, namely, model-based, AI, fault propagation model, and probing-based techniques.

2.2.1 Model-Based Techniques

In Model-Based Techniques an abstract model of the managed network that describes the functional and physical properties of the network components is first constructed using different techniques from logic to differential equations. This model represents failure inter-dependency among the network elements. Based on some input network parameters, the model predicts the network performance status. A network fault is detected once a discrepancy between the observations obtained from the managed network and the predictions produced by the model. In [8, 25, 26, 27, 28], finite state machine (FSM) models have been proposed. The managed network and its behavior in the existence of a fault is represented as a set of states, and the transition between these states is dictated by input events such as alarms coming from the network. The advantage of these algorithms is that they do not require learning and can cope with incomplete information. Nevertheless, it is well recognized that developing a fault model for a complex network is a daunting task.

2.2.2 AI Techniques

Among the commonly used techniques in network fault management systems are expert systems [29, 30, 31]. Expert systems use a rule-based representation to imitate the human knowledge of an expert. This knowledge can be either surface–resulting from experience, or deep-resulting from understanding of the system behavior from its principal. The purpose of these rules is to associate a network fault hypothesis with network alarms. The rules typically take the following form:

if alarm x **then** fault hypothesis is y

Sometimes several fault hypotheses are produced due to the received large number of alarms. To identify the most likely fault, a heuristic search is performed on the obtained fault hypothesis.

The expert system presented in [29] is comprised of four loosely coupled components: a monitor, a problem-clearing advisor, a trouble-ticket creation system, and a collection of network databases. The aim of the monitor sub-system is to respond to events and alarms as they occur on the network. It reads and formats alarms, filters out irrelevant information and redundant alarms, and then clusters together all alarms pertaining to a single network fault. In [30], a real-time interactive expert system is developed to simplify the task of fault identification based on a symptom description. The expert behavior is modeled by a knowledge base which is formulated using a knowledge engineering process. The targeted network components constitute the initial domain knowledge obtained from in-house data communication experts through a series of interviews. The resultant knowledge was coded in the form of complex condition-action production rules. In the case of fault occurrence, the system begins by attending to the symptom description, narrows the search to a relatively few suspicious components. Using its prior experience, the system determines the relative likelihood of each of the suspicious components and then focuses on the most likely fault locations. The expert system proposed in [31] implements all the network management functions. For the fault management part, it collects alarms generated by both predefined SNMP traps and user-defined internal threshold traps. Upon receiving a set of alarms, it first sorts and filters them, and then correlates the alarms to automatically diagnose the network fault. In [32], to identify problems at the TCP/IP level, an expert system, adopting the primary technique used by human experts, is designed to analyze static traces of TCP/IP packets and packets of some related protocols (e.g., ARP, ICMP). Heuristics used for reasoning about the cause of network failures are represented as production rules. Objects representing packets are manipulated by these rules. While the expert systems are effectively capable of modeling the human experience, the process required to form this experience into a set of production rules has proven to be difficult. Furthermore, this process tends to be sensitive to network topological changes which take place frequently and as a result the knowledge base has to be changed accordingly.

Artificial neural network (ANN)-based fault management systems have also been proposed in the literature. In [7], a kohonen self-organizing Map (SOM) neural network is trained for alarm clustering. Neural networks are considered black box systems that do not require the managed network to be explicitly modeled. However, the training process to tune its weights may take long sessions. Moreover, there are no particular rules to guide the selection of number of layers and the number of neurons in each layer. Therefore, a trial and error process is expected to be performed during the training period until the neural network finally stabilizes.

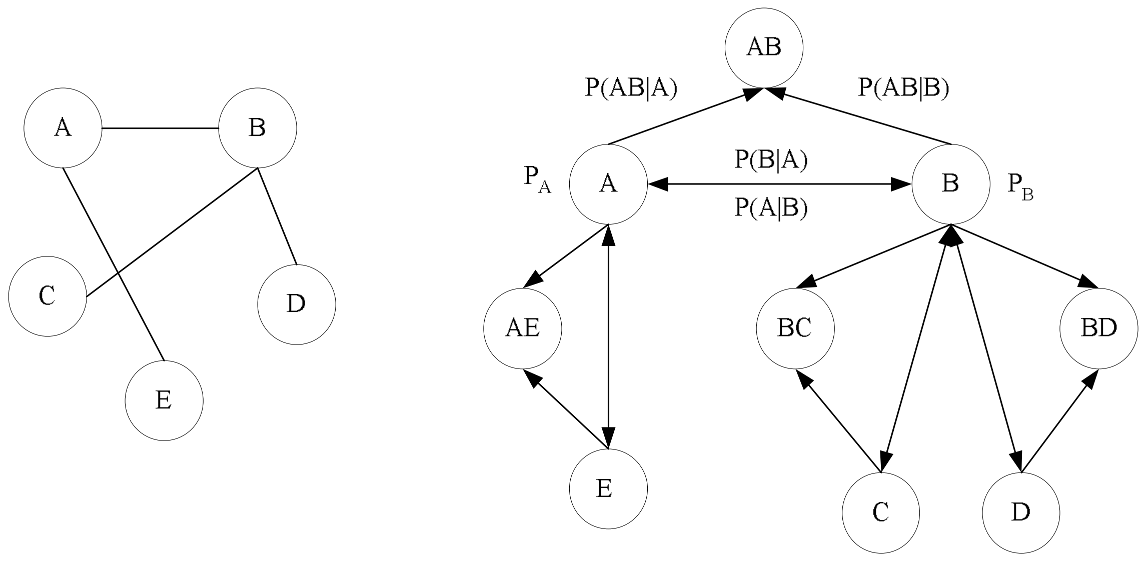
In [33], a mobile agent-based approach has been proposed. The hierarchical structure provided by the Internet model is exploited as a fault propagation model and used as an event correlation scheme.

2.2.3 Fault Propagation Techniques

Fault propagation (FP) refers to the fact that a fault in one network entity is able to affect the state of other network entities. Due to the hierarchal structure of computer networks, a malfunctioning component that affects the services and functions provided by lower layers may be observed in higher layers (vertical propagation) and it could also be observable in other hosts distant from the location where the fault originated (horizontal propagation). Based on this intrinsic property of computer networks, a graphical model of the managed computer network can be constructed in which the relationship between its entities is clearly specified. This graphical model is referred to in the literature as Fault Propagation Model (FPM). The FPM represents all faults and their respective symptoms that may occur in a managed network as nodes. Fault propagation models are often expressed using dependency or causality graphs.

A dependency graph is a directed graph *G = (O, D)*, where *O* is a finite, non-empty set of nodes representing network entities and *D* is a set of edges between these nodes. The directed edge (o_i, o_j) is an element of the set *D* denoting that a node o_i may depend on another node o_j. Every edge is assigned a certain weight. This weight indicates the strength of the relationship between the nodes in a given proposition. For example, given the network configuration of Fig. 2.2 (a), its dependency graph can be obtained as shown in Fig. 2.2 (b). The conditional probabilities assigned for each edge represent the relationship strength between the given nodes.

A causality graph can then be obtained from the dependency graph by simply reversing the directions of its edges. Most of the inference techniques reported in the literature utilize causality graphs using either OR model or AND model. An OR model combines possible causes (network failures) of a symptom using logical operator OR, meaning that at least one of the possible causes has to exist for the considered symptom to occur.

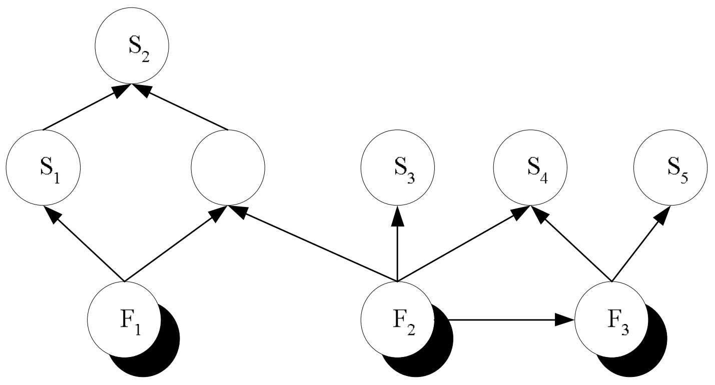


(a) Simple network configuration (b) Dependency graph

Figure 2.2: A simple network configuration and its corresponding dependency graph.

To simplify the fault identification problem, the reported techniques assume that only one fault exists in the network at a time or restrict the number of simultaneous faults to a certain number. Moreover, the symptom analysis process is often performed in a window-based fashion (i.e., a fault management system works with a group of symptoms observed only over a certain time-window).

The codebook, an FPM-based alarm correlation algorithm, utilizes a matrix of fault codes that represents a bipartite causality graph to distinguish faults from one another [6]. A fault code is a sequence of values from the set {0, 1}. The value of 1 at i_{th} position of a code, constructed for a fault f_j, indicates cause-effect implication between fault f_j and symptom s_i . As shown in Fig. 2.3, this technique allows for network faults to be uniquely coded by a given set of symptoms. The codebook approach uses minimum symbol distance as a decision making scheme.



Initial causality graph

	F_1	F_2	F_3
S_2	1	1	0
S_3	0	1	0
S_4	0	1	1
S_5	0	0	1

Correlation matrix

Figure 2.3: Correlation matrix derived from a causality graph [6].

Other popular FPM-based approaches are known as belief networks. A belief network is a Directed Acyclic Graph (DAG) whose nodes represent random variables, the edges denote the existence of direct causal influences between the linked variables, the strength of these influences are expressed by forward conditional probabilities. In the context of fault network management systems, these variables represent states of the network entities or the occurrence of symptoms. Hood et al. [24] reported an application of Bayesian network theory to proactive fault detection. The belief network is a tree-shaped based on the structure of the SNMP MIB's [13, 33]. In [20], this algorithm was adopted as an approximation scheme for performing fault identification in an obtained FPM. More fault identification techniques, based on the belief network theory, were reported in [9, 19, 34, 35].

2.2.4 Probing-Based Techniques

The main component of Probing-Based Techniques is a sample measurement called *probe*. A probe is basically a dedicated program (such as ping or trace route) or a network application (such as web access or email) installed in one of the network nodes, called a *probing station*. A probe is sent to examine a subset of nodes in the managed network on a periodic basis. Once a probe is sent to the network it either successfully returns to its probing station, signifying that all the network nodes in its path are in working order, or it fails to return to its probing station, indicating that one node or more in its path are in a failure state. In [1, 2, 3], a special matrix, called a *dependency matrix*, is first constructed based on the location and number of the available probing stations. One of the advantages of such techniques is that a subset of nodes of the managed network can be tested and examined at any time. Hence, fault localization can be restricted to a very limited number of nodes based on the results of the testing probes. The set of testing probes used for fault identification tasks may contain a large number of these probes. Since all the probes in the probe set have to be utilized this may entail an extra burden on the managed network in terms of bandwidth usages. Moreover, the extra management traffic may actually exacerbate the situation when a congestion problem detected in the managed network. To address these issues, several algorithms have been proposed in the literature. In [1], a new algorithm is proposed to minimize the number of these testing probes. In [2, 3], active approaches reduce these probes even further by selecting a small subset of probes based on the outcomes of previously used probes. In [4], a heuristic-based approach is introduced. The outcomes of previous probes are utilized to select the new probe. The probing-based techniques are still in their infancy and new rigorous techniques are definitely needed.

2.3 Summary

This chapter presents a brief review of some of the existing network fault management techniques. Each of these techniques has at least one of the following limitations:

1. Excessive management traffic is introduced to the managed network.
2. Uncertainty properties inherently contained in the fault evidence are often ignored during the alarm correlation process.

Inadequate work has been done in the past to effectively address these two important issues. The discussion has been focused on the reasoning algorithms not on the representation model of the managed system. The underlying knowledge representation of the managed system adopted by our alarm-based proposed algorithms utilizes fault propagation models. These models in turn are used to build evidence structures for the received network alarms. In the following chapter, some of the techniques used to obtain such models are reviewed. Furthermore, the Dempster-Shafer Evidence Theory as it constitutes the framework of the reasoning process of the distributed management system is introduced.

Chapter 3

FPM Techniques and Evidential Reasoning

3.1 Introduction

Alarm correlation schemes can be viewed as two-component systems. The first component is a model of the underlying managed system represented by a fault propagation model (FPM). The second component is a reasoning mechanism that actually performs the alarm correlation process. The purpose of a FPM is to clearly describe the causal relationship between network failures and their corresponding alarms. Upon receiving a set of alarms, the reasoning mechanism utilizes this relationship to identify their root causes. The alarm correlation techniques proposed in this thesis are based on the assumption that fault propagation models do exist and available to fault management systems. In this chapter, we provide a brief review of the systematic methods used to obtain such models. The Dempster-Shafer evidence theory is also introduced since it constitutes the framework of the proposed reasoning mechanism.

3.2 Alarm Correlation Architecture

The structure of alarm correlation systems is usually composed of two basic units (see Fig. 3.1): a *knowledge base* which includes a fault propagation model, and an *alarm correlation unit*. The reasoning mechanism performed by the alarm correlation unit utilizes the information presented by the knowledge base and provides a fault hypothesis set that may explain the observed alarms [36].

3.2.1 Knowledge Base

For network fault management systems to efficiently perform their tasks, they need to store knowledge about the managed network itself (such as network components and network topology). This knowledge can be acquired by interviewing human experts and saved in a typical relational database file or stored in a MIB (management information base). The knowledge base is also expected to contain important information about network events (network failures and their corresponding alarms). The most important component of the knowledge base is the fault propagation model which describes how these events propagate between objects in the supervised network [6, 9, 11, 15, 18, 36]. This propagation is due to the hierarchal nature of computer networks, and a fault propagation model is usually obtained from a dependency relationship model that reflects this hierarchy.

Fault propagation models show which alarms may be observed if a certain network failure occurs. For example, IP connection failure alarms over a specific router could be caused by a power failure in that router. Moreover, empirical information (obtained from a *log* that stores a history of activity of the network) is used to develop a probabilistic model of the managed network. This probabilistic model may also be included in the FPM [20].

3.2.2 Alarm Correlation Unit

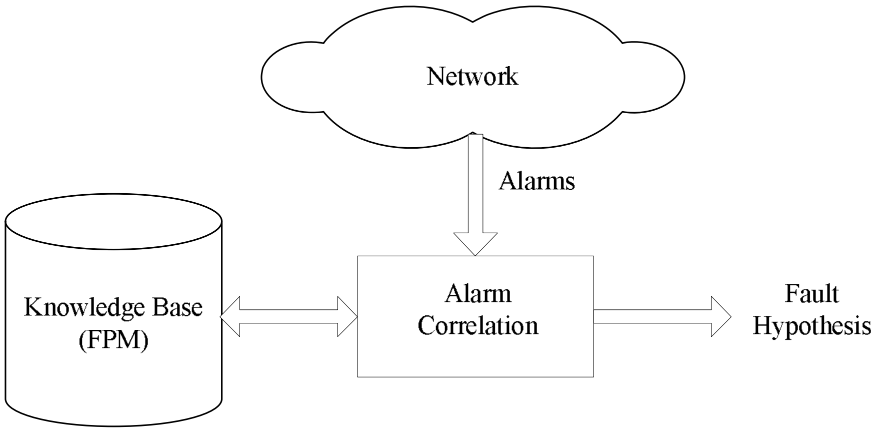


Figure 3.1: Alarm correlation system structure.

To correlate the observed alarms, alarm correlation units utilize different techniques. Some correlation techniques exploit the probabilistic knowledge provided by the fault propagation models to infer the most probable network failure. In these techniques, fault propagation models basically take the form of Bayesian networks [18]. Other correlation techniques rely on coding schemes by which a set of failure codes is constructed using the given fault propagation models. Upon receiving network alarms, they construct an alarm vector and compare it to a set of the predefined failure codes [6]. The closest failure code to the observed alarm vector is identified as the root cause of the problem. In Chapters 6 and 7, we propose new alarm correlation techniques. These algorithms utilize the probabilistic knowledge provided by the fault propagation model to create *local composite events*. A local composite event represents the alarm correlation performed locally by an intelligent agent in its domain. These local composite events are then combined by a higher level management entity into a global composite event.

3.3 Fault Propagation Models

A fault propagation model is essentially a representation of the cause-effect relationship that may exist among network events. Several methods have been proposed in the literature to capture this relationship. In what follows we review some of these methods. We then present a case study on obtaining a fault propagation model for the Campus network of the University of Waterloo in Chapter 8.

3.3.1 Object-Oriented Based Fault Propagation Model

The Object-Oriented Fault Propagation model has been introduced by Jackbson and Weisman. In this model the object-oriented paradigm is used to represent the network topology and the relationship among its entities [37]. The structural and behavioral abstract classes constitute the framework of the managed system. The network entities (NEs) and the network topology are described by the structural class. It contains two major subclasses namely, network *element class* and *element relationship class*. The *network element class* represents the network entity types such as switch, router, bridge, link, etc. The *network element relationship class* describes the dependency relationship between the managed objects as depicted in Fig 3.2. Objects created from the same class maintain similar types of relationship. The *behavior abstract class* as shown in Fig 3.3 should capture the mechanism by which network events are propagated. The major unit of this class is the *network message class* in which the alarm messages generated by different network entities (NEs) are hierarchically represented. Each dependency model may have two kinds of basic dependency:

- Direct dependency: one object is used/referenced directly by anther object.
- Indirect dependency: a chain of one or more direct dependencies joins two objects.

3.3.2 Fault Propagation Modeling Using the MODEL Language

One of the most appealing techniques used to model the cause-effect relationship among network events is a modeling language called MODEL [38]. It provides the following

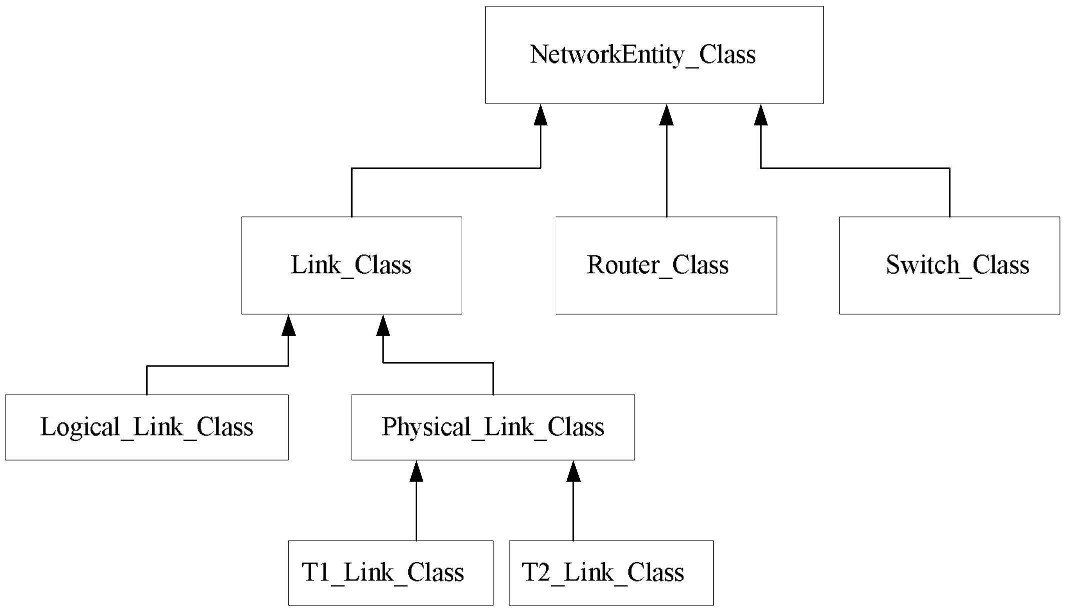


Figure 3.2: Network element class hierarchies.

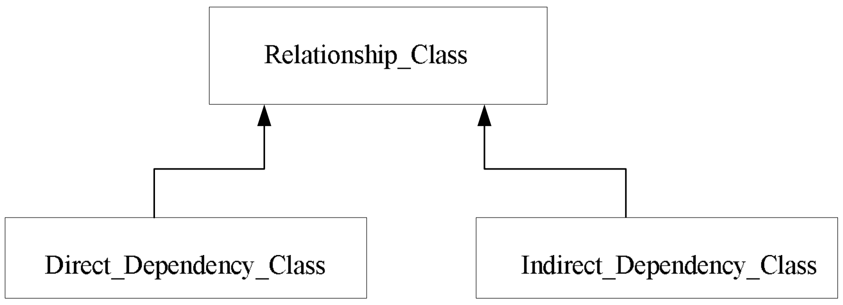


Figure 3.3: Direct and indirect dependency class.

advantages over other existing techniques:

- object-oriented data model capabilities,
- causality graph construction features,
- independent of reasoning algorithms.

Furthermore, it also extends the functionality of the underlying SNMP protocol. To better explain the basics of the MODEL language, let us examine the congestion fault that a router may experience [38]. To model the causal relationship between a congestion failure and its alarm indicating lost packets, the total number of discarded packets can be measured. Using SNMP commands [12], the corresponding MIB variables are retrieved:

```
interface IPRouter: IP
 {
   instrumented attribute long ipInDiscards;
   instrumented attribute long ipOutDiscards;
   attribute long discardsThreshold;

   event PacketDiscardsHigh "The level of discarded
                                         packets is high" =
          (delta ipInDiscards +  delta ipOutDiscards) /
                 delta _time > discardsThreshold;

   instrument SNMP;

}
```

The keyword *attribute* is used here to define measurable properties of the IP protocol. The keyword *event* defines the condition necessary for the packets loss alarm to occur. Therefore, it only occurs when the sum of the two measured IP MIB variables namely ipInDiscards and ipOutDiscards is over a predefined threshold (discardThreshold). The *delta* statement calculates the difference between the old and new values of the corresponding MIB variables. The keyword *_time* indicates the time at which the two MIB variables were pulled. Thus, this alarm is triggered when the discard rate reaches the threshold. The causal relationship between the congestion failure and the high packet discard alarm (with probability 1.0) can be expressed as:

```
Problem Congestion “High congestion“=
                PacketDiscardsHigh 1.0;
```

The above is a semantic declaration in the form of a rule. The rule that should be included in a single class to reflect the fault/alarm relationship is as follows:

```
Congestion (IPRouter(x)) ->
          PacketDiscardsHigh(IPRouter(x));
```

This way, a local alarm which indicates the network congestion failure is modeled. Network failures in one object propagate to related objects via relationships. In this example, the congestion failure would propagate to higher level connections which layered over the congested IP node. To express the fact that the congestion failure causes both local alarms PacketDiscardsHigh and ConnectionPacketLossHigh and propagates those alarms to higher level connections, we can write:

```
Problem Congestion “High congestion“=
               PacketDiscardsHigh 1.0;
               ConnectionPacketLossHigh 0.8;
Propagate symptom ConnectionPacketLossHigh =
               TransportConn, Underlying, PacketLossHigh;
```

Now the alarm ConnectionPacketLossHigh of the network congestion failure has been added with a causal probability of 0.8, where a value 1.0 indicates complete certainty. For further details about the MODEL language, you may refer to appendix A.

3.3.3 Layered Fault Propagation Model

In [11], a layered fault propagation model is presented. It captures the hierarchal characteristic of computer networks where the relationship among network layers on a single host and between network nodes within a single protocol layer are clearly defined. In this layered fashion, protocols provided by a specific layer are implemented by the network functions and services provided by the next lower layer. Even though the physical connectivity is established in the lowest layer, components of above layers have logical connectivity with their counter-part components in some other node within the same layer. This dependency among network components can be efficiently used for fault analysis. The Network Dependency Graph (NDG) may then be constructed as a representation of the recursive dependencies among the network components (protocols, services, functions). In this NDG, the services, protocols, and functions correspond to the graph nodes as depicted in Fig. 3.4 [11].

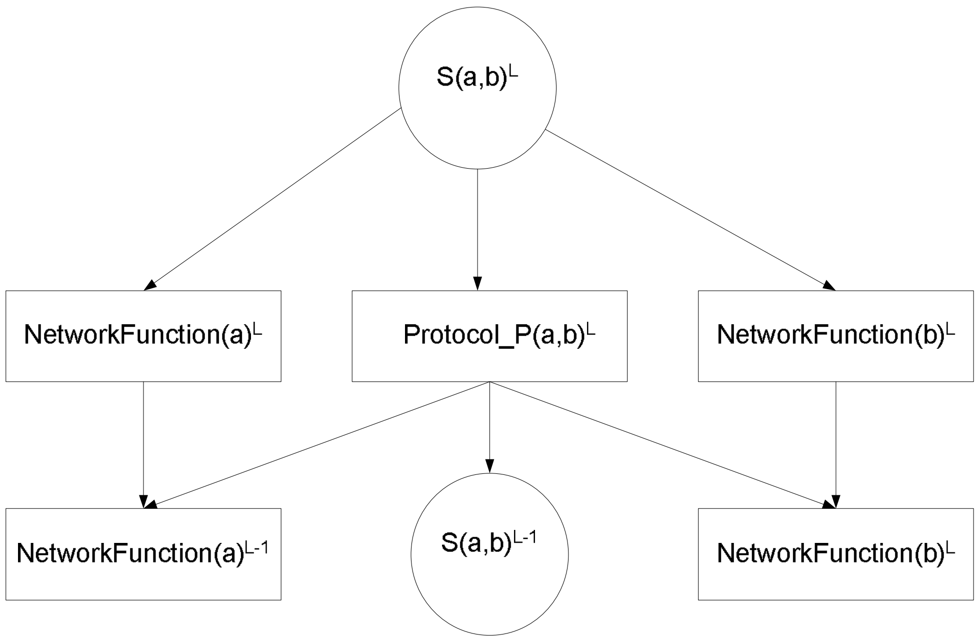


Figure 3.4: Dependency relationships between network components [11].

The edges among the graph nodes describe the dependency relationship among these nodes. The dependency relationship may represent the dependency between a service and network functions and protocols at a specific layer, between functions at adjacent layers, or between a protocol and services and network functions at the next lower layer. As shown in Fig. 3.4, service S at layer L, defined between any two network nodes a and b may be dependent on the set of all protocols between nodes a and b at layer L, the set of all network functions at layer L for node a, and on the set of all network functions at layer L for node b. From the perspective of Service S(a,b)^{L}, indications of malfunctioning behavior at Network Function(a)^{L}, Network Function (b)^{L}, and S(a,b)$^{L-1}$ are considered network failures. While indications of malfunctioning behavior at Service S(a,b)^{L} are considered alarms.

3.3.4 Fault Propagation Model based on Dependency Matrix

Probing-based schemes often represent the available probes in the form of dependency matrix as shown in Table 4.1. The dependency matrix describes a set of network entities and which probes that each entity may affect in case of their failure. Treating probes as network alarms, a fault propagation model can be obtained from the given dependency matrix. For example, a fault propagation model in the form of a bipartite graph is extracted from the dependency matrix of Table 4.1, as shown in Fig. 3.5. If we view the bipartite graph of Fig. 3.5 as a two-layer Bayesian network, then a probabilistic model can be built in terms of a joint distribution as follows [2]. Let us assume that the variable X represents the set of the network failures $X = \{f_{Router1}, f_{Router2}, f_{WebServer}, f_{DatabaseServer}\}$, P is the set of the available probes, and $Y(p_j)$ is the set of parents of the probe p_j. Then, the joint probability distribution can be stated as follows:

$$P_r(x,p) = \prod_{i=1}^{n} P_r(x_i) \prod_{j=1}^{m} P_r(p_j \,|\, Y(p_j))$$

Network failures are assumed to be independent, and the outcome of each probe depends only on the components examined by this probe. $P_r(x_i)$ represents network failure probabilities and conditional probability $P_r(p_j \mid Y(p_j)$ represents the dependency of probe outcomes on its components.

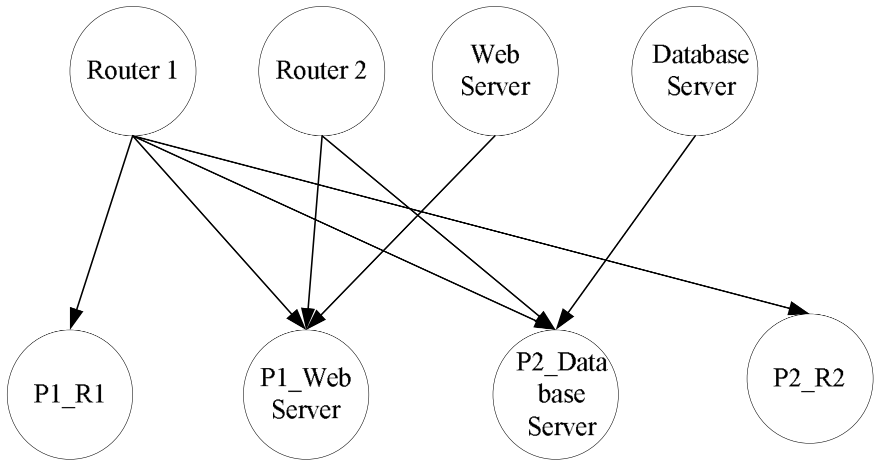


Figure 3.5: Bipartite graph.

3.4 Formulation of the Alarm Correlation Problem in the DSET Framework

Since we utilize the Dempster-Shafer evidence theory (DSET) as an inference engine for the alarm correlation unit, we introduce the necessary background of DSET in this section. Dempster-Shafer theory has continuously been gaining increasing attention among the researchers especially in expert systems and information fusion communities where reasoning under uncertainty is an active area of research. The evidence theory (developed by Glenn Shafer which is based on the earlier work of Arthur Dempster [23]) is considered a generalization of the Bayesian theory since it allows for manipulation of non-necessary exclusive events in a way that can explicitly represent computer networks status of uncertainty. From the evidence theory point of view, an observed alarm constitutes a piece of evidence regarding certain network failures. For example, the congestion failure at IP layer manifests itself as packet loss alarms in upper layers, as shown in section 2.2. However, the packet loss alarms may also be triggered by a whole set of failures other than the congestion failure. The non-deterministic nature of the relationship between the observed alarms and their potential network failures makes the application of the evidence theory as a correlation mechanism very appealing. For instance, let us say alarm a_1 is usually caused by network congestion (F_1) in a particular link, while another alarm, a_2, may be caused by either F_1 or link down (F_2). The evidence theory allows distributing support for a particular event (e.g., the network failure is F_1) as well as to the union of events (e.g., the network failure is either F_1 or F_2). The following sections discuss modeling notions of faults in the context of evidence theory.

3.4.1 Network Faults as Frame of Discernment

In the context of evidence theory, a *frame of discernment* Ω is a finite set and consists of the exhaustive and exclusive events [23]. 2^Ω is the power set composed of all possible subsets of Ω. To frame the alarm correlation problem in the context of evidence theory in the context, the frame of discernment represents singletons of all the possible network failures that may occur in the managed network. For example, if the abnormal behavior of a given managed network is caused by four different exclusive faults, F_1,F_2 (as described earlier), a server crash (F_3), and broadcast storm (F_4), then the network failures can be represented by the frame of discernment as $\Omega = \{F_1, F_2, F_3, F_4\}$. A belief measure over Ω, called a *mass* function, can assign a partial belief to every network failure in this frame of discernment. In this research work, an intelligent agent utilizes the message passing algorithm [39], also called Pearl's belief updating algorithm, to calculate a belief value for each network failure. The belief value is computed based on the observed alarms and the given fault propagation model of the managed network. After it assigns belief values for each network failure in its frame of discernment, the intelligent agent constructs an evidence structure and propagates the evidence structure to the network manger.

3.4.2 Mass Function and Focal Elements

The mass function (a basic probability assignment (bpa)) over a frame of discernment Ω is a mapping function $m: 2^{\Omega} \rightarrow [0,1]$, such that the following two conditions hold:

$$\sum_{A \subseteq \Omega} m(A) = 1, \tag{3.1}$$

$$m(\phi) = 0$$

A is the set of network failures and $A \subseteq \Omega$. The quantity $m(A)$ is a measure of belief that is assigned to exactly the set *A* (not to any proper set of *A*). It expresses a partial belief that a certain network failure of Ω belongs to the set *A*. For example, let us assume that *A* is given as $A = \{F_1, F_2\}$. Upon receiving a set of alarms, an intelligent agent may assign the belief value $m(A) = 0.4$ to the subset *A*. A value of 0.4 signifies that 40% of the intelligent agent's total belief that the current network failure is due to either F_1 or F_2. Different set of observed alarms may cause the intelligent agent to have different belief values regarding certain network failures. Subsets of network failures with belief values greater than zero are called *focal* elements.

3.4.3Evidence Structure

An evidence structure represents a collection of pieces of evidence and takes the following form:

$$(A, m(A)) \tag{3.2}$$

Where *A* is a focal element that represents a set of network failures and $m(A)$ is its mass quantity. The piece of evidence is distributed among all subsets of Ω rather than among elements of Ω as is the case in probability theory. The collection of all these pieces of evidence by an intelligent agent involved in the alarm correlation process constitutes the evidence structure of the intelligent agent. If the evidence structure contains only singleton focal elements, then, their mass values can be understood as their probabilities; and hence, the evidence structure of that alarm is called the Bayesian structure.

3.4.4 Evidential Measures and Belief Interval

Let A and B refer to certain sets of failure hypotheses, such that $A \subseteq \Omega$ and $B \subseteq \Omega$. An intelligent agent may use one of the following functions to calculate a belief value for each fault hypothesis set:

Belief function:

$$Bel(A) = \sum_{B \subseteq A} m(B) \tag{3.3}$$

Plausibility function:

$$Pl(A) = \sum_{B \cap A \neq \phi} m(B), Pl(\phi) = 0 \tag{3.4}$$

Commonality function:

$$Q(A) = \sum_{A \subseteq B} m(B) \tag{3.5}$$

Pignistic probability function:

$$Ppig(A) = \sum_{B \subseteq \Omega} \frac{|A \cap B| \times m(B)}{|B|} \tag{3.6}$$

The belief function is used to measure the total belief in a proposition (a failure hypothesis set) which takes into account the measures of belief assigned to the subsets of that proposition. It represents the total amount of probability that must be distributed among the network failures of that proposition. In a diagnostic context, if $A = \{F_1, F_2\}$, then *Bel (A)* is the sum of all the pieces of evidence (*$m(F_1)$*, *$m(F_2)$*, and *$m(F_1$*, $F_2)$) that support A (i.e., it supports the claim that, based on the observed alarms, the network failure is either F_1 or F_2). On the other hand, the plausibility function measures the maximal amount of belief that a proposition can take. It represents the maximal amount of probability that can be distributed among the network failures of that proposition. In a diagnostic interpretation, *Pl(A)* is the sum of all pieces of evidence that do not rule out that the network failure is either F_1 or F_2. Together the total belief and plausibility in a proposition constitute a confidence interval of that proposition. The probability that the network failure is either F_1 or F_2 is indicated by the confidence interval [*Bel(A)*, *Pl(A)*], where *Bel(A)* and *Pl(A)* are the lower and upper bounds of the confidence interval respectively and *Bel(A) - Pl(A)* expresses the ignorance regarding the proposition *A*.

3.4.5 Combining Agents Evidence Structures

After these evidence structures have been constructed by each intelligent agent involved in the alarm correlation process, the agent manager combines all of them into a single evidence structure using the combination rule, introduced by Dempster, as follows:

$$\oplus_{i=1}^{n} m_i(A) = \sum_{\bigcap_{i=1}^{n} A^i = A} \prod_{i=1}^{n} m_i\left(A^i\right) \tag{3.7}$$

where A is defined as before. This combination can be normalized as:

$$\oplus_{i=1}^{n} m_i(A) = \begin{cases} T \sum\limits_{\bigcap_{i=1}^{n} A^i = A} \prod\limits_{i=1}^{n} mi\left(A^i\right), \ldots if \ldots A \neq 0 \\ 0, \ldots if \ldots A = 0 \end{cases} \tag{3.8}$$

with $T = \dfrac{1}{1-t}$, and $t = \sum\limits_{\bigcap_{i=1}^{n} A^i = \phi} \prod\limits_{i=1}^{n} m_i\left(A^i\right),$

3.5 Summary

In this chapter, we present a brief review of techniques used by different management systems to obtain and model the underlying topology of the managed network. The basics of Dempster-Shafer Evidence Theory are also reviewed, along with the formulation of the alarm correlation problem within its framework. However, to employ the DSET, a mass function has first to be defined. The definition of mass functions has been considered a largely unsolved problem. In essence, a mass is referred to as a basic probability assignment [23], which is often associated with a probability distribution.

Chapter 4

A Novel CSP Approach for Probe Selection

4.1 Introduction

The number of generated alarms in a network of a moderate size can be overwhelming, while a considerable subset of these alarms can be redundant with little diagnostic value. As such, these alarms should not be considered in the fault analysis stage. Therefore, a sensible approach is needed to remove these alarms and consider only the most relevant alarms when performing the alarm correlation process. In this regard, we view probes (such as ping and trace route utilities) as active alarms. In contrast to regular alarms, active alarms are generated by pre-assigned nodes (called probing stations) and periodically sent to the network for node-availability testing tasks. In this chapter, we introduce a new approach, based on the constraint satisfaction problem (CSP) techniques, that is able to find an optimal number of active alarms. The powerful search techniques provided by the CSP technology make it an appealing alternative to the existing approaches. CSP-based approaches have been proposed and applied in other areas of network management, however, limited effort has been made to investigate their use for the purpose of fault detection and identification in computer networks.

4.2 Probing System Structure

In contrast to the *passive* (event-correlation based) approaches, the new CSP-based approach is an *active* approach in which certain and limited number of measurements are obtained and analyzed. These measurements are often referred to in the literature as *probes*. A probe is basically a test transaction (such as ping and trace-route) whose outcome depends on the health status of the network components that exist in its path. As shown in Fig. 4.1, a probing station is a dedicated system that sends these probes to test different network elements according to a predefined schedule. Depending on the size of the managed system, the number of the probing stations varies from one network to another. To use probes, probing stations must be first selected and installed at different locations in the managed network. The total number of probes issued by these probing

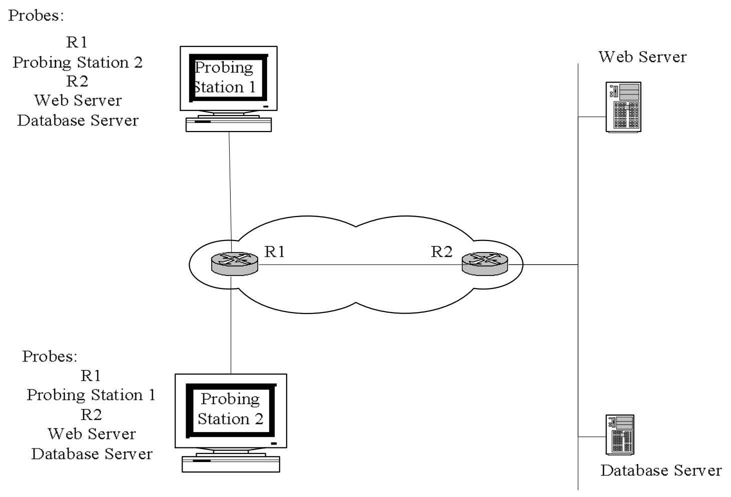


Figure 4.1: Configuration Example of two Probing Stations.

stations and used by the network management system is called the *probe set*. This probe set is often large. Since probes are basically embedded code and are installed, operated, and maintained in probing stations, they may impose a certain cost.

The objective is to obtain a subset of the probe set that is both small and exhaustive (i.e., it should cover all the network nodes). The new probe set is called the *solution probe set*. Moreover, the solution probe set must have the same *diagnostic power* as the original probe set. Figure 4.2 shows the necessary steps required to realize the active probing scheme.

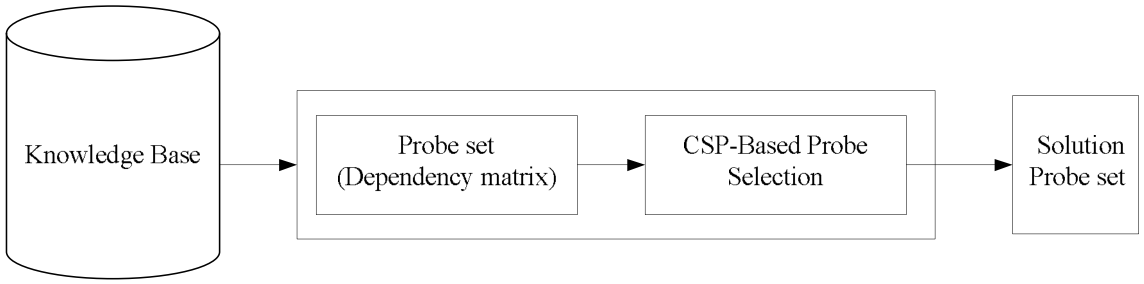


Figure 4.2: System Architecture.

Using the network topology information stored in the knowledge base and knowing the locations of the pre-assigned probing stations, we can identify all possible probes originated from the dedicated probing stations. The only pre-request here is that the obtained probes must be able to reach each network element at least once. The result of the first step is the probe set (also known as the *dependency matrix*). Using the dependency matrix, the objective of the proposed CSP-based approach is to find the solution probe set.

4.3 Problem Description and Notation

For the sake of simplicity, let us assume that the network to be managed is composed of six nodes and has the topology of Fig. 4.1, with the assigned probing stations. The possible probes that can be obtained form the depicted configuration are shown in Fig.

4.3. The probing stations and their respective probes are marked by the same color. A probe may effectively be represented using the coding approach to alarm correlation. According to this scheme, each probe is represented by a binary string of length that is equal to the network size. The value of one in position *j* in the binary string denotes that the given probe passes through the node Nj. If the probe does not test the considered node, then its position in the binary string is assigned the value of zero. Using this representation, the obtained dependency matrix for the network configuration shown in Fig. 4.1 is shown in Table 4.1.

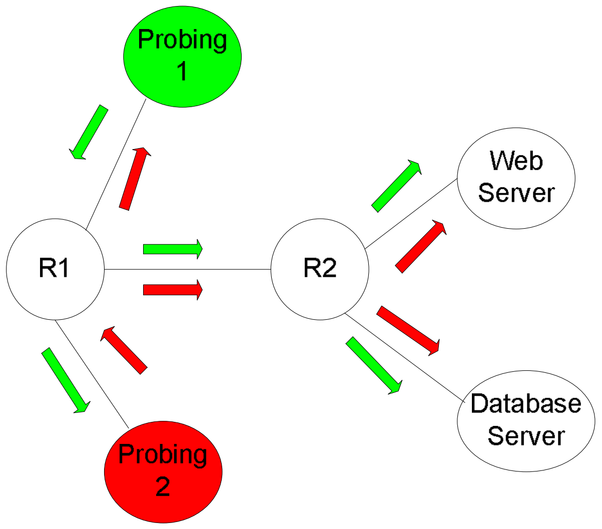


Figure 4.3: Simple Network Graph.

The superscript notation used to describe a probe indicates both its probing station from which it is issued and its final destination for which it is going. For example, the probe $P_{1\_WebServer}$, shown in the fourth row of the dependency matrix, means that this probe is originated at probing station 1 and terminating at the server (WebServer) hosting the web site. The path that this probe should go through to reach the WebServer node is $\Pr obing_1 \rightarrow R_1 \rightarrow R_2 \rightarrow WebServer$. Usually, the routes that these probes take during their transitions are determined by the routing information pertained in the routing tables.

Of course such routing configuration will be taken into consideration when designing and determining the required probes.

Table 4.1: The dependency matrix of the network of Fig. 4.3

Probes	Probing Station 1	Probing Station 2	Router 1	Router 2	Web Server	Database Server
P1_R1	1	0	1	0	0	0
P1_Prob2	1	1	1	0	0	0
P1_R2	1	0	1	1	0	0
P1_Web Server	1	0	1	1	1	0
P1_Database Server	1	0	1	1	0	1
P2_R1	0	1	1	0	0	0
P2_Prob1	1	1	1	0	0	0
P2_R2	0	1	1	1	0	0
P2_Web Server	0	1	1	1	1	0
P2_Database Server	0	1	1	1	0	1

Once a probe is generated by a probing station, it is expected to return two possible outcomes. The probe either succeeds at reaching its final destination or it fails. If the probe succeeds, then all the nodes and links that comprise its path are presumed up and functioning properly. If the probe fails, then at least one component of its path is assumed to be in a failure state. A single probe may only be used as a fault detection signal of the network entities in its path. However, to identify a malfunctioning entity, multiple probes are needed. For instance, if the initial solution set contains only the probes $P_{1\_WebServer}$ and $P_{1\_DatabaseServer}$, then the fault management system can only identify problems pertaining to the web and database servers. However, it may not be able to locate failures in the routers. The information contained in these two probes simply is not enough to be used for comprehensive diagnostic tasks. This diagnostic deficiency is evident and can be seen from their respective undistinguishable columns in the dependency matrix. Moreover, the node *Probing Station 2* can not be tested by the two probes as both probes have value of zero in their code at the node's position. The incompleteness characteristic exhibited by the initial solution set can be resolved by considering more probes from the dependency matrix. Hence, more probes are needed and must be added to the solution set. Before adding a new probe to the solution probe set, its diagnostic abilities should be first examined. For all the network nodes to be uniquely identified, the solution probe set must have non identical columns.

Let *N* refer to the network size, *D* to the dependency matrix, X_i to the i^{th} row in *D*, and *R* to the total number of rows in *D*. Each row in *D* represents a unique probe. Therefore, for each $X_i \in D$, the entry $X_i(j) = 1$ if the probe X_i passes through the node *j*. $X_i(j) = 0$ otherwise, where $i = 1,...,R$ and $j = 1,...,N$. The size of the dependency matrix *D* is determined by *R-by-N*.

4.4 CSP Model for Probe Selection

To model the probing problem in the framework of constraint satisfaction problems, three major components have to be identified namely the problem variables, their domains, and a set of constraints that govern the relationship among the problem variables. In order to facilitate the problem formalization, we will discuss and provide the definitions of various concepts relating to the constraint satisfaction problem paradigm.

4.4.1 Definitions

The constraint satisfaction research community introduces extremely diverse terminologies. In our study, we adopt Freuder's definitions of a standard CSP, variable instantiations (or value assignment), consistent instantiation, and solution to a CSP [40], as the basis for our formalization.

Definition 4.1 (constraint satisfaction problem) *A constraint satisfaction problem P is a tuple* (X, D, C)*, where*

- $X = \{x_1,...,x_n\}$ *is a finite set of variables,*

- $D = \{D_1, ..., D_n\}$ *is a set of domains. Each variable* $x_i \in X$ *is associated with a finite domain of possible values,* D_i.
- $C = \{c_1, ..., c_m\}$ *is a finite of m constraints or relations. Each constraint* $c_i \in C$ *is defined on a subset of k variables,* $\operatorname{var}(c_i) = \{x_{i_1}, ..., x_{i_k}\} \subseteq X$ *, and allows specified combinations of values that are subset of the Cartesian product of the domains of constraint variables* $\operatorname{var}(c_i)$*, that is* $c_i \subseteq D_{i_1} x ... x D_{i_k}$.

Definition 4.2 (Variable instantiation or value assignment) *An instantiation of a variable* x_i *is the assignment to* x_i *of a value d from the variable domain of values* D_i*, that is,* $x_i = d, d \in D_i$ *We denote a variable instantiation or value assignment by the assignment* $x_i = d$ *or the variable pair* (x_i, d). *An instantiation of a set of variables* $\chi = \{x_1, ..., x_k\}$ *is the simultaneous instantiation of all variables in the set* χ *with values from their associated domains, that is,*

$$\{x_1 = d_1, ..., x_k = d_k\}, \quad x_i \in X, \quad d_i \in D_i, \quad 1 \le i \le k$$

We denote an instantiation of a set of variables by the set of ordered pairs $\{(x_1, d_1), ..., (x_k, d_k)\}$*, or, simply, by the k-tuple of assigned values* $(d_1, ..., d_k)$.

Definition 4.3 (Consistent instantiation or satisfied constraint) *An instantiation* I_χ *of a set of variables* χ *is consistent with or satisfies a constraint c defined on the same set of variables* χ, $\operatorname{var}(c) = \chi$*, if and only if* $I_\chi \in c$.

Definition 4.4 (Solution to a CSP) *A solution to a constraint satisfaction problem P is a consistent instantiation of all variables in P.*

Based on the above definitions, we present in detail the formulation of the probing problem in the framework of the constraint satisfaction problem. Using definition 4.1, the CSP representation of the probing problem is composed of three main components as shown in Fig. 4.4 [40]:

- A set of CSP variables that represent the selected probes,
- Their associated domains in terms of the dependency matrix,

- A set of constraints that governs the mechanism of the probe selection process.

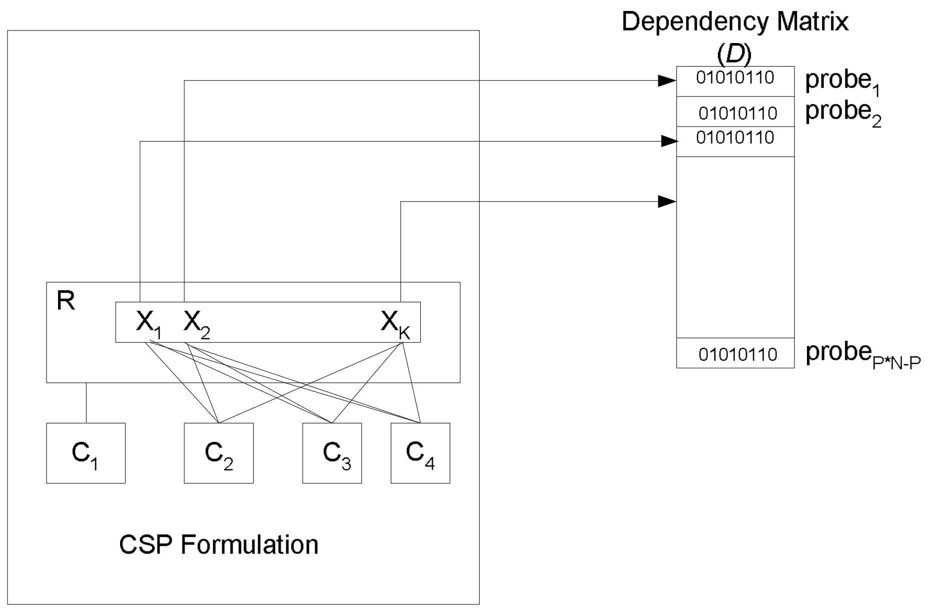


Figure 4.4: CSP model of the probe selection problem.

Clearly, the system probes provided by the dependency matrix represent the system to be modeled. Each CSP variable, x_i, will represent a selected probe from its domain D_i. In essence, the set of constraints should capture the dynamic behavior of the probe selection process. Therefore, the imposed constraints will guarantee that a selected probe does not introduce undesirable features (such as identical columns) to the solution probe set. This, of course, implies that the set of constraints must be well-defined and accurately reflect the strict conditions under which the selection process is to be performed.

4.4.2 Defining Testing Probes as CSP Variables

The probing problem can be modeled as a CSP problem by first defining an initial finite set of variables. The cardinality of the initial variable set depends on the size of the dependency matrix. Let *P* refer to the number of the probing stations and *N* to the number of the network nodes. Hence, the total number of probes, *R* can be determined by the following equation:

$$R = P * N - P \tag{4.1}$$

That means the obtained dependency matrix consists of *R* rows of probes. The initial set of the CSP variables may be determined using the lower bound restriction. To cover all the nodes of the managed network by the solution probe set, the number of the initial variables should not be less than the lower bound.

Let *K* refer to the number of probes to be selected from the dependency matrix. Since it is not known in advance how many of these probes are sufficient for the fault detection and identification task, *K* can not be determined beforehand. However, there is a lower bound, *L*, on the value that *K* may take at the start of the selection process. *L* can be computed using the following formula:

$$2^L \geq N \tag{4.2}$$

Hence, *L* can be calculated as:

$$L \geq \log(N)/\log(2); \qquad L \in Z \tag{4.3}$$

Therefore, *K* initially assumes the value of *L* as determined by (4.3). Logically, it should not, however, exceed the maximum number of the available probes. This can be stated as follows:

$$L \leq K \leq R \tag{4.4}$$

We will refer to the *K* CSP variables as the active variables. If there are no *K* probes in the dependency matrix that can satisfy the imposed constraints, then the algorithm will dynamically modify the initial variable set by activating another variable. This can be done simply by increasing *K* by one (i.e., $K = K + 1$). As soon as the search algorithm finds *K* probes that satisfy all the imposed constraints, the *K* probes will be considered as the solution probe set and the search process terminates. Hence, the set of the CSP variables can be generally defined as:

$$X = \{x_1, x_2, ..., x_K\}; \qquad K = L, ..., R \tag{4.5}$$

As shown in Fig. 4.4, each active variable may take any probe from the dependency matrix. Hence, the set of domains associated with the set of active variables will be a set of dependency matrices:

$$D = \{D_1, D_2, ..., D_k\} \tag{4.6}$$

The possible values that an active variable, x_i may take are defined in its domain, D_i such that $x_i = d$ where $d \in D_i$. In order to find an optimal set of K probes, the search algorithm should find an instantiation, I_X for the set of active variables, $X = \{x_1, x_2, ..., x_K\}$, from their domains that satisfy certain constraints, which will be discussed in detail next section.

4.4.3 Construction of the Problem Boolean Constraints

The objective of the proposed algorithm is to find a solution probe set that possesses some distinctive properties. These properties constitute the bases upon which the imposed constraints will be constructed. In what follows, we provide more elaboration on these properties and present the constraints that capture their inferred principals. Also, the advantages of adhering to these principals will be pointed out. During the discussion of the proposed constraints, the individual elements of the available probes are of binary form, and hence their possible values are restricted to the set {0, 1}. Moreover, the solution probe set will constitute a corresponding *solution matrix*, similar to the dependency matrix, except that it should contain smaller number of probes.

Property 1 *The number of selected probes contained in the solution probe set should be kept minimal.*

The purpose of the first property is to reduce the amount of management traffic imposed by the original probe set which is represented by the dependency matrix. Less management traffic means that more network bandwidth will be available for network users. Furthermore, in case of network difficulties (for example, network congestion), not eliminating this extra traffic may worsen these difficulties. To achieve this objective, we view the solution probe set as a cost function in terms of the number of its probes. Hence, the cost function may take the following form:

$$\min |K| \tag{4.7}$$

The realization of this crucial property lies in the use of the concept of the active variables. Since the number of active variables is paramount to an optimal number of selected probes, we should keep the value of K as low as possible. K is actually the cardinality of the solution probe set. Setting K initially to the lower bound and increasing it only when the chosen K probes are unable to successfully locate network problems is in fact guaranteeing that K will always maintain an optimal number of probes. Theoretically, of course, K can not have a value that is less than the lower bound L. This fact can be summarized by the following definition.

Definition 4.5 *The cardinality of the solution set, K, must be at least equal to L and less than P*N-P.*

The other properties will be ensured in the form of strict Boolean constraints and the cost function presented in (4.7) will be subject to these constraints. A violation of one of these constraints means that these properties are not being met by the current variable instantiations. During the search process, the probe committing these constraint violations will be replaced by a new one. Since the representation of the testing probes is in the form of binary string, Boolean manipulations of these probes make the realization of these properties simple. We will refer to a new probe being considered for the solution probe set as a *candidate probe*. Therefore, in order for a candidate probe to be added to the solution probe set, the following properties must hold.

Property 2 *Every node in the managed network should be covered by the solution set at least once.*

We should avoid selecting probes that may lead to creating a solution matrix in which one or more of its columns are zero columns. That is, for any column in the solution matrix, all its entries should not be equal to zero. To preserve this property in the solution probe set, the following definition can be used.

Definition 4.6 *The new probe set can be considered a solution probe set if and only if, each column of its corresponding solution matrix includes the value of 1 in their entries, at least once.*

Based on definition 4.6, property 2 can be captured by the following constraint:

$$C_1 : x_1(j) \vee \ldots \vee x_K(j) \qquad (4.8)$$
$$\forall j = 1, \ldots, N$$

The conjunctive normal form of this constraint ensures that C_1 is only true when an entry of value 1 exists in each column of the solution matrix. Each column in the solution probe set represents a distinctive network node. A column, that only containing zeros as its entries indicates that the node represented by that column is not covered by the current solution probe set. This constraint is extremely useful in pruning the search tree as will be explained in more detail later. An instantiation $I_X = \{(x_1, d_1), \ldots, (x_k, d_k)\}$ may satisfy this constraint, C_1 if and only if $I_X \in C_1$.

Property 3 *Each node covered by the solution probe set should be uniquely identified.*

In order for a fault management system to be able to identify any malfunctioning network node, the selected probes should have differentiating capabilities by which a failed node can be efficiently isolated and identified. Hence, candidate probes that introduce identical columns to the solution probe set should not be selected. Identical columns confuse the management system as which node (represented by an identical column) is actually responsible for the malfunctioning behavior. The essence of this property can be captured by the following definition.

Definition 4.7 *The found set can be considered a solution probe set if and only if, each column of its corresponding solution matrix has a different value from each other column in at least one of their entries.*

Using definition 4.7, property 3 can be realized by the following constraint:

$$C_2 : (x_i(j) \vee x_i(\bar{j})) \wedge (\neg x_i(j) \vee \neg x_i(\bar{j})) \qquad (4.9)$$
$$\forall i \in \{1,..,K\}$$
$$\forall j \in \{1,...,N-1\}$$
$$\forall \bar{j} \in \{(j+1,...,N\}$$

This constraint can only be violated by the solution probe set if two of its columns in its corresponding solution matrix are exactly the same. For this constraint to be satisfied, any two corresponding entries of two columns must have different values at the same row, regardless of what values the rest of their other entries may take.

Property 4 *Identical probes should not be included in the solution probe set.*

The purpose of this property is to avoid redundant probes. Since candidate probes are represented by active variables that have the same domain, it is possible that more than one of these variables may be instantiated by the same value. This can be summarized in the following definition.

Definition 4.8 *The found set can be considered a solution probe set if and only if, each row of its corresponding solution matrix has a different value from each other row in at least one of their entries.*

Using definition 4.8, the following constraint may capture property 4 as follows:

$$C_3 : \neg((x_i(1) \oplus x_K(1)) \vee .... \vee (x_i(N) \oplus x_K(N))) \qquad (4.10)$$
$$\text{for} \quad i = 1,...,K-1.$$

The purpose of the exclusive disjunction $\oplus$ is to examine the similarity of the individual elements of the candidate probe with their corresponding elements in the probes of the solution probe set. Two active variables are presumed to have different instantiations if any two of their corresponding elements have opposite values.

Any violation of one or more of these constraints prompts the search algorithm to discard the current instantiation of the last active variable x_k and selects a new one. If no instantiation is available for x_k from its domain D_k such that these properties hold, then a backtracking is performed in which a new instantiation of the previous active variable x_{k-1} will be chosen from its domain D_{k-1} .

4.5 *K*-Consistency and Constraint Propagation

In this section, we consider some consistency techniques that can be utilized to remove inconsistent values from the domains of the active variables without removing any solutions to the probing problem. That is, an instantiation of one of the active variables that may violate one or more of the constraints described in Equations (4.8), (4.9), and (4.10) should be removed from their domains. Since our constraints are expected to include *K* active variables, we adopt the following definition of *K*-arc consistency developed by Freuder [41].

Definition 4.9 (*K*-consistency) *A CSP is k-consistent if and only if consistent partial solution over k-1 distinct variables, there exist an instantiation of K^{th} variable such that the partial solution plus that instantiation is consistent.*

The *K*-consistency and propagation techniques are used to improve the efficiency of the backtracking search algorithm by detecting failures earlier in the search process. The constraint propagation is embedded in the backtracking algorithm and performed to reduce the domains of the unassigned active variables. If a domain of any variable becomes empty after propagation, the current CSP can not produce a valid solution and backtracking should be carried out. The domain values removed due to the propagation activity are returned when a backtracking is performed and another assignment for the variable is made by the search algorithm. The backtracking and constraint propagation continue until a solution is found or the backtracking process is out of active variables.

4.5.1 Domain Reduction Rules

The proposed CSP is considered as a Boolean CSP since its constraints are defined using Boolean expressions. In this framework, the Boolean constraints are divided into two classes, namely, *simple form* and *compound form*. In order to achieve *K*-arc consistency defined in 4.9, the compound constraints should first be transformed into simple forms which will be dealt with directly using domain reduction rules. Some auxiliary variables are introduced in the preprocessing stage before identifying the domain reduction rules for each constraint. A Boolean constraint is called a simple constraint if it is in one of the following forms:

- $x_i(j) = x_l(j)$; we call it the equality constraint,
- $\neg x_i(j) = x_l(j)$; we call it the NOT constraint,
- $x_i(j) \wedge x_l(j) = z$; we call it the AND constraint,
- $x_i(j) \vee x_l(j) = z$; we call it the OR constraint.

Where $x_i(j), x_l(j), z$ denote different Boolean variable, $i, l = 1,\ldots,K, i \neq l; j, \bar{j} = 1\ldots N$.

To apply the domain reduction rules on constraint C_1 defined in Equation (4.8), we first transform the constraint,

$$C_1 : x_1(j) \vee \ldots \vee x_K(j)$$
$$\forall j = 1,\ldots,N$$

into the following simple Boolean constraints:

$$x_1(j) \vee x_2(j) = z_1 \tag{4.11}$$

$$z_1 \vee x_3(j) = z_2 \tag{4.12}$$

$$\vdots$$

$$z_{K-2} \vee x_K(j) = z_{K-1} \tag{4.13}$$

For constraint C_1 to be satisfied, one of its simple Boolean expressions, defined in (4.11), (4.12), and (4.13), has to be true. The following domain reduction rules can be applied on the simple constraint defined in Equation (4.11) as follows:

$$\frac{\langle x_1(j) \vee x_2(j) = z_1 ; x_1(j) = 0, x_2(j) \in \{0,1\}, z_1 = 1 \rangle}{\langle ; x_1(j) = 0; x_2(j) \in \{0,1\} \cap \{1\}, z_1 = 1 \rangle} \tag{4.14}$$

or

$$\frac{\langle x_1(j) \vee x_2(j) = z_1 ; x_1(j) \in \{0,1\}, x_2(j) = 0, z_1 = 1 \rangle}{\langle ; x_1(j) \in \{0,1\} \cap \{1\} ; x_2(j) = 0, z_1 = 1 \rangle} \tag{4.15}$$

If z_1 is true, then the first constraint C_1 is satisfied. These domain reduction rules simply state that for z_1 to be true, the following rule should hold:

$$z_1 = 1 \rightarrow x_1(j) = 1 \text{ or } x_2(j) = 1 \tag{4.16}$$

That is for the simple constraint defined in Equation (4.11) to be true, then either $x_1(j)$ or $x_2(j)$ must be equal to 1. If z_1 is not true, then we apply the same reduction rules on the second simple Boolean constraint defined in Equation (4.12). In this, case z_1 will assume the value of 0. Applying the previous domain reduction rules on this constraint, the following rule can be deduced:

$$z_2 = 1 \rightarrow x_3(j) = 1 \tag{4.17}$$

And so on, until the last rule of last simple constraint is obtained. Thus,

$$z_{k-1} = 1 \rightarrow x_k(j) = 1 \tag{4.18}$$

The constraint C_1 may only be violated if all of its simple Boolean constraints are violated, that is $z_1 = z_2 = ... = z_{K-1} = 0$. If one of its simple Boolean constraints is true then C_1 is satisfied.

The second constraint C_2 defined in Equation (4.9),

$$C_2 : (x_i(j) \vee x_i(\bar{j})) \wedge (\neg x_i(j) \vee \neg x_i(\bar{j}))$$

$$\forall i \in \{1,..,K\}$$
$$\forall j \in \{1,...,N-1\}$$
$$\forall \bar{j} \in \{(j+1,...,N\}$$

can be transformed into these simple Boolean constraints:

$$x_i(j) \vee x_i(\bar{j}) = z_{i1} \tag{4.19}$$

$$\neg x_i(j) \vee \neg x_i(\bar{j}) = z_{i2} \tag{4.20}$$

$$z_{i1} \wedge z_{i2} = z_{i3} \tag{4.21}$$

Where $i, j, \bar{j}$ are as defined above. For the constraint C_2 to be satisfied, then the simple Boolean constraint defined in Equation (4.21) must be true. Since this constraint is an AND constraint, it means that the other two simple constraints defined in Equations (4.19) and (4.20) must also be true. The following domain reduction rules can be applied on the simple constraint defined in Equation (4.19) as follows:

$$\frac{\langle x_i(j) \vee x_i(\bar{j}) = z_{i1}; x_i(j) = 0, x_i(\bar{j}) \in \{0,1\}, z_{i1} = 1\rangle}{\langle ; x_i(j) = 0; x_i(\bar{j}) \in \{0,1\} \cap \{1\}, z_{i1} = 1\rangle} \tag{4.22}$$

or

$$\frac{\langle x_i(j) \vee x_i(\bar{j}) = z_{i1}; x_i(j) \in \{0,1\}, x_i(\bar{j}) = 0, z_{i1} = 1\rangle}{\langle ; x_i(j) \in \{0,1\} \cap \{1\}; x_i(\bar{j}) = 0, z_{i1} = 1\rangle} \tag{4.23}$$

These domain reduction rules simply state that for z_{i1} to be true, the following rule should hold:

$$z_{i1} = 1 \rightarrow x_{i1}(j) = 1 \text{ or } x_i(\bar{j}) = 1 \tag{4.24}$$

The following domain reduction rules may also be applied on the simple constraint defined in Equation (4.20) as follows:

$$\frac{\langle \neg x_i(j) \vee \neg x_i(\bar{j}) = z_{i2}; \neg x_i(j) = 0, \neg x_i(\bar{j}) \in \{0,1\}, z_{i2} = 1\rangle}{\langle ; \neg x_i(j) = 0; \neg x_i(\bar{j}) \in \{0,1\} \cap \{1\}, z_{i2} = 1\rangle} \tag{4.25}$$

$$\frac{\langle \neg x_i(j) \vee \neg x_i(\bar{j}) = z_{i2}; \neg x_i(j) \in \{0,1\}, \neg x_i(\bar{j}) = 0, z_{i2} = 1\rangle}{\langle ; \neg x_i(j) \in \{0,1\} \cap \{1\}; \neg x_i(\bar{j}) = 0, z_{i2} = 1\rangle} \tag{4.26}$$

These domain reduction rules simply state that for z_{i2} to be true, the following rule should hold:

$$z_{i2} = 1 \rightarrow \neg x_i(j) = 1 \text{ or } \neg x_i(\bar{j}) = 1 \tag{4.27}$$

Moreover, for the constraint z_{i3} to be true the following rule should hold:

$$z_{i3} = 1 \rightarrow z_{i1} = 1 \text{ and } z_{i2} = 1 \tag{4.28}$$

Hence, for C_2 to be violated only one of its simple constraints has to be violated.

The last constraint C_3 defined in Equation (4.10),

$$C_3 : \neg((x_i(1) \oplus x_K(1)) \vee .... \vee (x_i(N) \oplus x_K(N)))$$
$$\text{for} \qquad i = 1,...,K-1.$$

can be transformed into these simple Boolean constraints:

$$x_i(1) \oplus x_K(1) = z_1 \tag{4.29}$$

$$x_i(2) \oplus x_K(2) = z_2 \tag{4.30}$$

$$\vdots$$

$$x_i(N) \oplus x_K(N) = z_N \tag{4.31}$$

For constraint C_3 to be satisfied, one of its simple Boolean expressions, defined in (4.29), (4.30), and (4.31), has to be true. The following domain reduction rules can be applied on the simple constraint defined in Equation (4.29) as follows:

$$\frac{\langle x_i(1) \oplus x_K(1) = z_1; x_i(1) = 0, x_K(1) \in \{0,1\}, z_1 = 1\rangle}{\langle ; x_i(1) = 0; x_K(1) \in \{0,1\} \cap \{1\}, z_1 = 1\rangle} \tag{4.32}$$

$$\frac{\langle x_i(1) \oplus x_K(1) = z_1; x_i(1) \in \{0,1\}, x_K(1) = 0, z_1 = 1\rangle}{\langle ; x_i(1) \in \{0,1\} \cap \{1\}; x_K(1) = 0, z_i = 1\rangle} \tag{4.33}$$

These domain reduction rules simply state that for z_{i1} to be true, the following rule should hold:

$$z_1 = 1 \rightarrow x_i(1) = 1 \text{ and } x_K(1) = 0, \text{ or } x_i(1) = 0 \text{ and } x_K(1) = 1 \tag{4.34}$$

Similar reduction rules can be applied for the other simple constraints defined in (4.30) and (4.31) and the following rules can be obtained:

$$z_2 = 1 \rightarrow x_i(2) = 1 \text{ and } x_K(2) = 0, \text{ or } x_i(2) = 0 \text{ and } x_K(2) = 1 \tag{4.35}$$

$$z_N = 1 \rightarrow x_i(N) = 1 \text{ and } x_K(N) = 0, \text{ or } x_i(N) = 0 \text{ and } x_K(N) = 1 \tag{4.36}$$

Therefore, for the constraint C_3 to be satisfied, then any one of the simple constraints $z_1, z_2 \ldots$ or z_N must be true. Hence, for C_3 to be violated all the simple constraints must be violated, that is $z_1 = z_2 = \ldots. = z_N = 0$.

4.5.2 Backtracking Search Algorithm

The domain reduction rules developed in the last section will be embedded in the search algorithm as constraint propagation methods. Fig. 4.5 shows the main algorithm that calls the backtracking search with constraint propagation shown in Fig. 4.6. The main algorithm starts with constraint propagation for all the CSP problem variables. Then it calls the backtracking search if the constraint propagation function does result in an empty domain for one of the problem variables. The backtracking search is a recursive algorithm in which every time before it calls itself with a new variable, constraint propagation is performed once again for the new variable. It only calls itself if the constraint propagation was successful for the new variable.

The function *cost ()* is to check whether the new value assignment for the current variable is consistent. The *propagate ()* function is to apply the domain reduction rules developed in the previous section. In its first call, the *propagate ()* function will apply the domain reduction rules on all the problem variables. After an active variable is instantiated with a value from its domain, it applies them with the new assignment for the current variable and the rest of the problem variables. A simple checking forward algorithm can be utilized as constraint propagation algorithm as shown in Fig. 4.7.

4.6 Breaking Symmetries

Since the proposed CSP model is based on backtrack search algorithm, symmetrically equivalent states in the search tree may be explored more than once. Among these states,

```
/*ALGORITHM BACKTRACKING SEARCH WITH CONSTRAINT PROPAGATION
PROCEDURE MAIN
BEGIN
/*Initialization for global variables
        $D \leftarrow array[1..K]$;            /* D contains the domain of individual variables.
        $Solution \leftarrow array[1..K]$;   /* The solution set.
        $failure \leftarrow FALSE$ ;
        $Success \leftarrow FALSE$
/*End of initializations
        propagate (0, D, failure);
        IF NOT failure THEN
                BACKTARCK_PROP (1, D, Solution, Success)
        END
END
```

Figure 4.5: Backtracking main algorithm.

either all of them lead to a solution, or none of them lead to one. Exploring one of these equivalent states is enough to determine if a solution can be found for the CSP model. Hence, the backtrack search algorithm should limit itself to visit only one of these symmetrical states. The advantage of breaking these symmetries is that the search space can be reduced even further. The proposed CSP model introduces the following two forms of symmetry:

- Probes can be permuted among the *K!* combinations,
- Probes can be exchanged.

```
PROCEDURE BACKTRACK_PROP( i: INTEGER, D: DOMAINS, Success: BOOLEAN);
BEGIN
        WHILE D[j] <> {} AND NOT Success DO
        $x_i = d; d \in D[i]$;
        $D[i] = D[i] - \{d\}$;
        IF const (Solution, i, d) THEN
                Solution [i] = d;
                Success = (i==K);
                IF NOT Success THEN
                    propagate (i+1, D, failure);
                        IF NOT failure THEN
                                BACKTRACK_PROP ( i, D, Success)
                        END
                  END
           END
END PROCEDURE BACKTRACK_PROP
```

Figure 4.6: The backtrack search procedure.

The first symmetry may occur when a particular probe is being selected more than once. In fact, this symmetry will be avoided by the search algorithm because of the constraint C_3. The constraint C_3 will always make sure that identical probes will be discarded. However, we have not introduced a constraint that may break the second symmetry. The second symmetry can be avoided using *lexicographical* ordering [42]. A sequence $\vec{x} = \langle x_1, ..., x_n \rangle$ is *lexicographically smaller than or equal* to another sequence $\vec{y} = \langle y_1, ..., y_n \rangle$, written as $\vec{x} \le_{lex} \vec{y}$, if and only if:

```
PROCEDURE propagate (D: DOMAINS, failure: BOOLEAN)
BEGIN
        failure = FALSE;
        FOR i=1:K
                D[i] = $\{d \in D[i] \mid \{d \in d[i] is \quad a \quad constent\}$
                Failure = (D[i] == {};
        END
END propagate
```

Figure 4.7: Constraint propagation algorithm.

$$x_1 \le y_1 \text{ and } (\underset{1 \le \bar{i} < i}{\Lambda} x_{\bar{i}} = y_{\bar{i}}) \rightarrow x_i \le y_i$$

for $2 \le i \le n$.The sequence $\vec{x}$ is *lexicographically smaller than* $\vec{y}$, written as $\vec{x} <_{lex} \vec{y}$ if and only if $\vec{x} <_{lex} \vec{y}$ and $\vee_{1 \le i \le n} x_i \ne y_i$.

In general, a variable symmetry σ can be broken by the lexicographical ordering constraint [41]

$$\vec{x} \le_{lex} \sigma(\vec{x}),$$

where $\vec{x}$ is a sequence of the variables in the CSP. The following symmetry breaking constraints then can be used to break the variable symmetry as follows:

$$x_1 \le_{lex} \langle x_2, \ldots, x_K \rangle$$

$$< x_1, x_2 > \le_{lex} \langle x_3, \ldots, x_K \rangle$$

--

$$< x_1, x_2, \ldots, x_{K-1} > \le_{lex} \langle x_K \rangle$$

Using a constraint for each variable symmetry (σ), all symmetrical solutions in each symmetry class except the lexicographically smallest, with respect to the sequence $\vec{x}$, would be removed.

4.7 Summary

In this chapter, we present a novel approach for the probing problem. The CSP-based model can capture the dynamic nature of the selection process in the form of Boolean constraints. The objective is to choose a subset of the available testing probes in which the solution subset has to have three crucial features:

i. The cardinality of the solution set should be as minimum as possible,

ii. It has to cover all the nodes in the managed network,

iii. Every network node should be uniquely identified.

The proposed scheme dynamically changes the number of active variables that are involved in the selection process. Experiments have been conducted and the effectiveness of the proposed approach in terms of minimizing the number of the selected probes has been demonstrated and compared with other existing approaches.

Chapter 5

A New Fuzzy CSP Probing Algorithm

5.1 Introduction

Though the CSP-based model developed in the previous chapter tremendously reduces the total number of probes required for fault detection and identification tasks, there is still an imposed overhead. The fault identification process still can not be performed unless all the probes contained in the solution probe set are utilized. This unavoidable cost may be tolerated in small networks. However, as managed networks grow larger, this problem may become a serious issue and a more adaptive probe-selection mechanism is needed to alleviate its negative impact. In this chapter, we develop a novel Fuzzy CSP-based algorithm. Instead of selecting a group of probes at once, the new algorithm only considers a single probe at a time. To be eligible for the fault identification task, an informative probe should possess some diagnostic abilities. The most informative probe is recognized as a probe that may help reduce the set of the suspect nodes more efficiently than other competing probes. Since the competing probes may satisfy this essential requirement with different degrees, the standard CSP may not be suitable. Thus, a fuzzy CSP model is developed, in which a probe that satisfies the formulated fuzzy constraints the most is identified as the most informative probe.

5.2 System Architecture and Notations

The proposed approach is divided into two major components. As shown in Fig. 5.1, the first component performs the fault detection task. In this stage, a small number of probes are sent periodically to the managed network. If one or more of these probes fail, then the managed network is considered to be in an anomaly state.

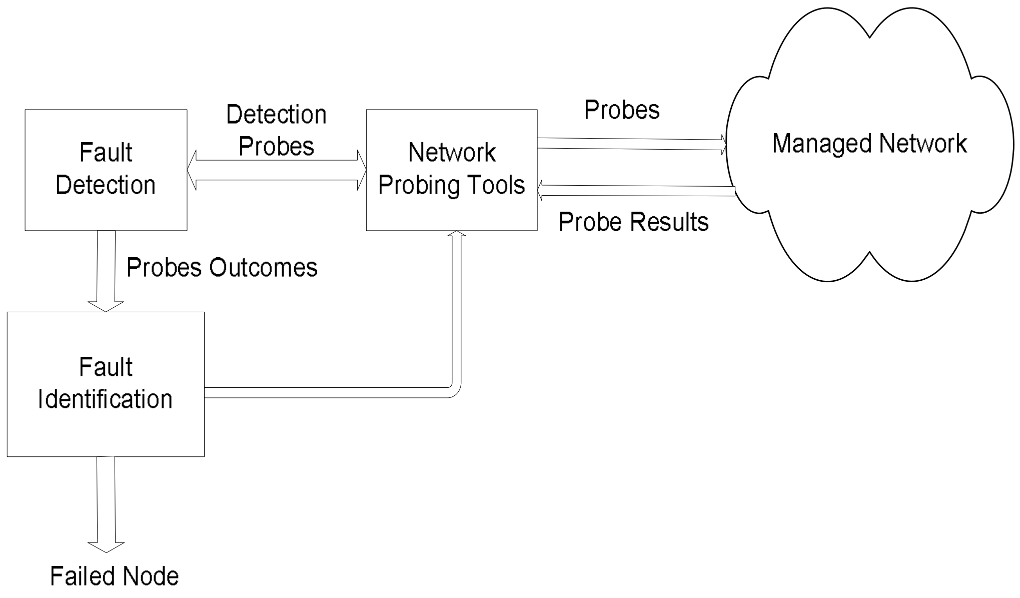


Figure 5.1: Probe-based system architecture.

Since, the fault detection probes may not be sufficient to identify the malfunctioning node, the task of the second component is to perform the fault identification process. During this stage, further analysis of the failed and successful probes is carried out. Taking advantage of the valuable diagnostic knowledge provided by the current status of the fault detection probes, the most appropriate probe is singled out for the fault identification task and sent to the managed network. Careful fault analysis is then conducted on the outcome of the probe. This step may result in either the failed node being identified or a new informative probe being selected.

5.2.1 Fault Detection Probe Set

The fault detection probe set may be obtained manually by the network administrator. A simple greedy approach applied on the dependency matrix (the probe set) can also be employed to attain this set. It has to cover all the nodes in the managed network with as minimum number of probes as possible. Naturally, the first selected probe should be the probe with the maximum number of network nodes in its path. This probe should be then eliminated from the probe set. A second probe is selected from the remaining probes such that it covers as many network nodes that have not been covered by the first probe. This probe selection process continues until all the network nodes are represented by the selected probes or no available probes are left.

Let us assume that the set *N* represents the set of nodes of the network under investigation, *P* represents the probe set, and *S* the set of the selected probes from *P*. The greedy approach can be implemented as shown in Figure 5.2. The greedy algorithm is simple enough to be self-explanatory. If sufficient testing probes are available, then a small set of fault detection probes can be found. However, if such set is not found, then more testing probes can be created and added to the probe set and the probe selection process starts over.

5.2.2 Dynamic CSP-Model for Fault Identification

The central premise of the proposed algorithm is that a failed probe gives a valuable clue on the set of nodes that may contain the malfunctioning node. Moreover, successful probes may indicate a set of nodes that should be ruled out as likely suspects of the malfunctioning behavior. The constituents of the failed probe, therefore, comprise a special node set called the suspect set (referred to by *F*), while the constituents of the

```
/*Greedy Algorithm for the selection of the fault detection
/*probe set
1. Inputs: N and P.
2. Outputs S.
3. Select a probe $\overline{P}$ from P such that it has the maximum number of non-
   covered nodes and the least number of covered nodes.
4. Add $\overline{P}$ to S.
5. Remove $\overline{P}$ from P.
6. Remove the nodes expected to be probed by $\overline{P}$ from N.
7.   if ( $N \neq NULL$ ) and ( $P \neq NULL$ ) then
            Repeat steps 3 to 6.
     end if
8.   if ( $N = NULL$ ) then
            Return S.
            Exit.
     end if
9.   if ( $N \neq NULL$ ) and ( $P = NULL$ ) then
            Report "insufficient probes in P."
            Exit.
     end if
```

Figure 5.2: Probe selection by the greedy algorithm.

successful probes constitute another special node set called the healthy set (referred to by *G*). Furthermore, probes whose constituents are subset of the set G are considered healthy probes and removed from *P*. Using *F* and *G* sets, the proposed fuzzy CSP-based algorithm selects a new probe from the modified probe set *P*. Based on the response of the new selected probe, it may either locate the malfunctioning node or dynamically adjusts the suspect, healthy, and probe sets. It should be noted that the fault identification task is an iteration process. In this process, probes will be temporally removed from the probe set *P* and nodes will be deleted or added from and to the suspect and healthy sets. This iteration stops only when the failure set *F* becomes a singleton set. A general schematic diagram of the iteration process is depicted in Fig. 5.3.

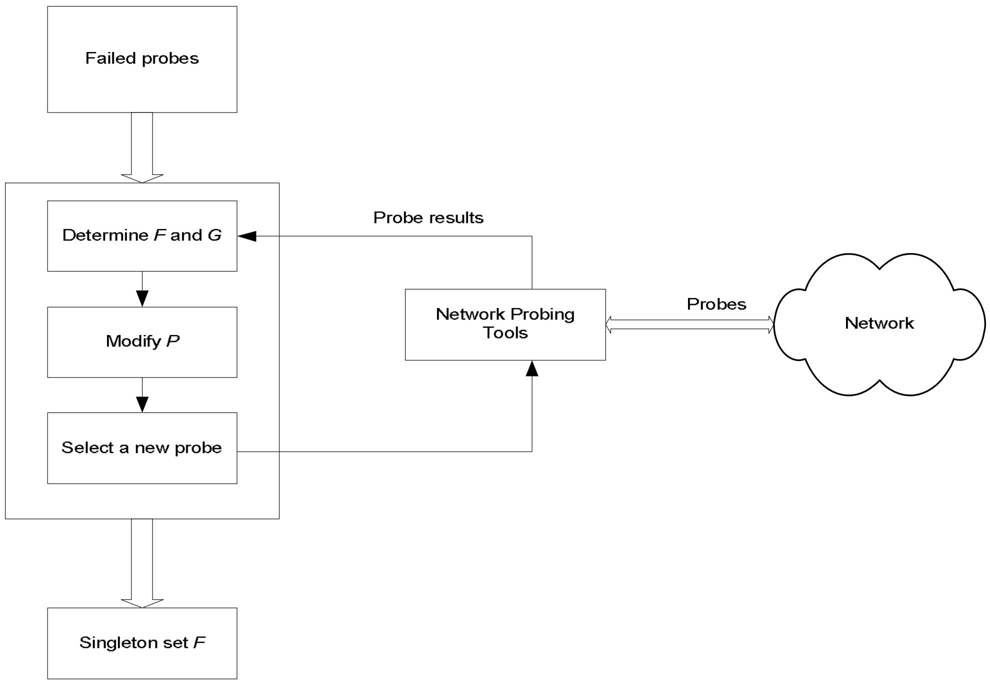


Figure 5.3: Schematic diagram of probing-based algorithm.

The objective of the new algorithm is to identify the malfunctioning node with as little management traffic (probes) as possible. To achieve the intended objective, the algorithm should meaningfully reduce the size of the suspect node set *F*. However, choosing less capable probes may result in an opposite effect and enlarge the suspect set. This could prolong the fault identification process causing more management traffic to be sent to the managed network. Therefore, the conditions with which testing probes are selected should be carefully examined and well represented in the problem formulation. The dynamic mechanism of the selection procedure should be captured by a set of well-defined constraints. These constraints must ensure that only the most informative probes are qualified for the fault identification task. Consequently, the domains of the problem variables will be effectively modified.

5.3 Problem Description

Assuming only a single node failure may occur in the managed network, let R_{p_i} refer to the response of the probe p_i, where i=1… *K*. *K* is the number of probes in the fault detection probe set. The probe p_i is defined by its constituent set (the network nodes in its path). R_{p_i} may take a value only from the set {0, 1}. If the probe p_i fails, then $R_{pi} = 0$; otherwise $R_{pi} = 1$.

- If a single probe fails ($R_{pi} = 0$) and the other probes succeed ($R_{pj} = 1$), then the initial suspect and healthy sets are constructed as follow:

$$F = \{n \mid n \in (p_i - (p_i \cap p_j))\} \tag{5.1}$$

$$G = \{n \mid n \in ((N - p_i) \cup (p_i \cap p_j))\} \tag{5.2}$$

- If multiple probes fail, that is $R_{p_i} = ... = R_{p_j} = 0$, then:

$$F = \{n \mid n \in (p_i \bigcap .... \bigcap p_j)\}, \tag{5.3}$$

$$G = \{n \mid n \in (N - (p_i \bigcap .... \bigcap p_j))\} \tag{5.4}$$

where $i, j = 1...k, i \neq j$.

If the result of Equation (5.1) or (5.3) is a singleton set *F*, then *F* identifies the malfunctioning node. However, if *F* contains more than one element, then further fault analysis will be carried out. The fuzzy CSP-based fault identification algorithm picks another probe according to some features described in detail next section.

Let p_m be the newly selected probe. The new probe is expected to contain a subset of suspect nodes or both suspect and healthy nodes in its path. The response of p_m will impact the elements in both *F* and *G* as follows:

- If $R_{p_m} = 0$, then

$$F = \{n \mid n \in (p_m \cap F)\} \tag{5.5}$$

$$G = \{n \mid n \in (N - (p_m \cap F))\} \tag{5.6}$$

- If $R_{p_m} = 1$, then

$$F = \{n \mid n \in (F - (p_m \cap F))\} \tag{5.7}$$

$$G = \{n \mid n \in (p_m \cup G)\} \tag{5.8}$$

If the probe p_m fails, then obviously the suspect nodes presented by the probe will become even more suspicious and should be singled out for more testing. Equation (5.5) suggests that the suspect set F will hold only the suspected nodes in the probe p_m. The other suspected nodes will be removed from *F* and added to the healthy set *G* as implied by Equation (5.6). However, if the probe p_m succeeds, its suspected nodes should be removed from the suspect set *F* as indicated by Equation (5.7) and added to the healthy set *G* as shown in Equation (5.8). If the suspect set *F* is not a singleton set then another probe is selected and the whole process will be conducted once again in the same manner. Depending on the status of the sets of *F* and G, the probe set *P* may also go through some refinement where probes of healthy nodes will be ignored from the probe selection process.

This problem can be best explained by the following example. The dependency matrix, *D*, obtained from the network configuration of Fig. 5.4, is presented in table 5.1. For the sake of convenient notations, it is summarized as follows:

$$D = \{P_{12}, P_{13}, P_{14}, P_{15}, P_{16}, P_{42}, P_{43}, P_{45}, P_{46}\}$$

The network nodes are represented by the set $N = \{N_1, N_2, N_3, N_4, N_5, N_6\}$. The set of probes are defined in terms of their constituent nodes as follows:

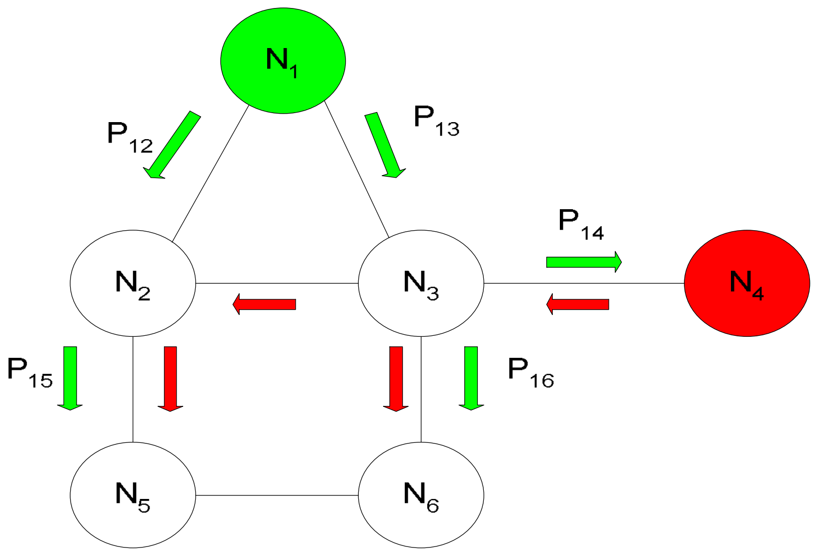


Figure 5.4: Simple Network Graph.

Table 5.1: The Dependency matrix of the network in Fig. 5.4.

Probes | Nodes	N_1	N_2	N_3	N_4	N_5	N_6
P_{12}	1	1	0	0	0	0
P_{13}	1	0	1	0	0	0
P_{14}	1	0	1	1	0	0
P_{15}	1	1	0	0	1	0
P_{16}	1	0	1	0	0	1
P_{42}	0	1	1	1	0	0
P_{43}	0	0	1	1	0	0
P_{45}	0	1	1	1	1	0
P_{46}	0	0	1	1	0	1

$$P = \{P_{12} = \{N_1, N_2\}, P_{13} = \{N_1, N_3\}, P_{14} = \{N_1, N_3, N_4\}, P_{15} = \{N_1, N_2, N_5\},$$
$$P_{16} = \{N_1, N_3, N_6\}, P_{42} = \{N_2, N_3, N_4\}, P_{43} = \{N_3, N_4\}, P_{45} = \{N_2, N_3, N_4, N_5\},$$
$$P_{46} = \{N_3, N_4, N_6\} \}$$

In this new representation, any node that is not covered by a given probe will not be reported by the node set of the probe. According to the new approach, the greedy scheme is first applied on the probe set. Consequently, the following two probes are selected:

$$P_{45} = \{N_2, N_3, N_4, N_5\}$$
$$P_{16} = \{N_1, N_3, N_6\}$$

It is apparent that P_{45} and P_{16} can completely cover the nodes in the managed network. Should one or both of these probes report the occurrence of a failure, the fault identification function will be invoked promptly. The information carried by the detection probes will be fully exploited as follows:

- If P_{16} has failed and P_{45} has succeeded, then, according to Equations (5.1) and (5.2), the node sets F and G will be constructed as follows:

  $$F = \{N_1, N_6\}$$
  $$G = \{N_2, N_3, N_4, N_5\}$$

  Naturally, nodes reported by the successful probe should be included in the healthy
  set G, as well as any node covered by both the failed probe and the successful probe simultaneously. Thus, faults pertained to node 3, N_3, is not included in the suspected set F.
- If P_{45} has failed and P_{16} has succeeded, following the same Equations, the sets F and G will formed as follows:

  $$F = \{N_2, N_4, N_5\}$$
  $$G = \{N_1, N_3, N_6\}$$
- If both probes have failed, then according to the Equations (5.3) and (5.4), the suspect and healthy sets will be formed as follows:

  $$F = P_{45} \cap P_5 = \{N_3\}$$
  $$G = N - (P_{45} \cap P_5) = \{N_1, N_2, N_4, N_5, N_6\}$$

The fundamental presumption behind this reasoning is that the outcomes of the probes force certain assignments to the problem variables, namely *F* and *G*. Note here, since the suspect set *F* has been adjusted and turned into a singleton set, no further fault analysis is needed.

5.4 A Fuzzy CSP Framework

Since the constraints imposed by the probing problem need not be fully satisfied, the crisp CSP formulation will not be flexible enough to accommodate them. The fuzzy CSP (FCSP) methodologies seem more appropriate to tackle this kind of problems. The main concept is to generalize the notion of crisp constraints [43], in which different tuples may satisfy a particular constraint with different degrees. Once the fuzzy constraints have been defined and constructed, standard search techniques to solve CSP problems can be adapted to solve fuzzy CSP problems. To formally define the probing problem in a fuzzy CSP framework, a basic definition of a fuzzy CSP is first presented.

5.4.1Fuzzy Constraint Satisfaction Problem (FCSP)

Definition 5.1 *A fuzzy constraint satisfaction problem (FCSP) is defined as a 3-tuple* (X, D, C^f), *where*:

- $X = \{X_i \mid i = 1,...,n\}$ *is a finite set of variables.*
- $D = \{D_i \mid i = 1,...,n\}$ *is the set of domains. Each domain* d_i *is a finite set containing the possible values for the corresponding variable* x_i *in X.*
- C^f *is a set of finite fuzzy constraints. That is,*

$$C^f = \left\{ R_i^f \mid \mu_{R_i^f} : \left(\prod_{x_j \in \text{var}(R_i^f)} d_j \right) \rightarrow [0,1], i = 1,...,m \right\}, \tag{5.9}$$

Where $\text{var}(R_i^f)$ denotes the set of variables of fuzzy constraint R_i^f. The main concept of a fuzzy CSP revolves around a *fuzzy constraint*. A fuzzy constraint is a mapping from the

Cartesian product of the domains $(D_1xD_2x....xD_k)$ of the constraint variables $x_1, x_2,..., x_k$, into the interval [0, 1]. The value assigned to *c* for the constraint $c(x_1, x_2,..., x_k)$ is the degree of satisfaction that the instantiations of the variables $X_1, X_2,..., X_k$ may present to the constraint. A fuzzy constraint corresponds to a membership function of a fuzzy set. If $c(x_1, x_2,..., x_k) = 1$, then we say that the constraint is *fully satisfied* by this instantiation. Conversely, If $c(x_1, x_2,..., x_k) = 0$, then the constraint is *fully violated* by this instantiation. The *support set* is formed by the instantiations of the variables for which c > 0.

5.4.2 Probing Fuzzy CSP Formulation

Formally, the fuzzy CSP-based probing problem can be defined as follows: Let F_t, G_t, and P_t refer to the sets of the suspect nodes, the healthy nodes, and the probes, respectively, at iteration *t*, where $t = 0,1,....$. Let $\{f, p\}$be the set of the probing problem variables. Let Σ_F and Σ_P refer to the domains of f and p, respectively. The domainΣ_{F_t} is basically a superset of F_t and$\Sigma_P = P_t$, and the empty set,Φ, and the whole set are excluded from the domain Σ_{F_t}. Therefore, to find a solution to the fuzzy CSP probing, we have to assign an instantiation for each variable$f_i \in \sum_F$and $p_l \in \sum_P$, where$i = (1,...,2^{|\Sigma_F|} - 1)$, and$l = (1,..., m_t)$, and subject to a set of binary constraints, *C*.

The binary constraints ensure that certain properties hold during the selection process. We will refer to a competing probe in the selection process as *candidate probe*, the nodes in its path as the *test set*, and the suspect and healthy sets as the *diagnostic sets*. In what follows we discuss these properties and their impact on the formulation of these binary constraints [44].

Property 1 *A candidate probe should be able to test at least one suspect node.*

Clearly that probes which contain only healthy nodes in their test set are not going to be useful in the fault analysis process. However, as network entities continuously are added or removed from both diagnostic sets, a probe may change its status in later stages of the fault analysis process. To preserve this property in the selected probe, the following definition can be used.

Definition 5.2 *The test set of a candidate probe* $p_l \in \sum_P$, *should contain at least one element from the current diagnostic set* F_t.

Based on definition 5.2, property 1 can be captured by the following constraint:

$$C_1: \ p_l \cap F_t \neq \Phi \tag{5.10}$$

The purpose of this constraint is to temporarily eliminate from the search space all the healthy probes since we are only interested in testing the suspect nodes represented by the diagnostic set F_t.

Property 2 *A candidate probe should be able to test a suspect node currently under investigation.*

Not all the suspect nodes are investigated at once. Only a selected subset of the diagnostic set F_t will be targeted for testing. A candidate probe should contain in its test set the targeted node. This property can be defined as follows.

Definition 5.3 *The test set of a candidate probe* $p_l \in \sum_P$, *should contain the targeted suspect node* $f_i \in \sum_F$ *from the current diagnostic set* F_t.

Based on definition 5.3, property 2 can be captured by the following constraint:

$$C_2: \ p_l \cap f_i = f_i \tag{5.11}$$

This constraint can only be consistent if the current instantiation of the problem variable f_i is clearly included in the test set of the candidate probe.

Property 3 *A candidate probe should not aim to test all the suspect nodes at once.*

Probes that contain all the suspect nodes in their test sets should also be avoided. Since one or more of the suspicious nodes in the diagnostic set F_t is the main cause of the malfunctioning behavior, a candidate probe that contains all the suspect nodes in its test set is doomed to fail, regardless of how many healthy nodes presented in its test set. We use the following definition to capture the essence of this property.

Definition 5.4 *The test set of a candidate probe* $p_l \in \sum_P$ *should not contain all the elements in the current diagnostic set* F_t.

Based on definition 5.4, property 3 can be captured by the following constraint:

$$C_3\colon\ p_l \cap F_t \neq F_t \tag{5.12}$$

The consequences of such failed probe is that the non-suspect nodes in its test set will be wrongly added to the current diagnostic set F_t and eliminated from the diagnostic set G_t. This of course will further complicate the diagnosis process and introduce unnecessary management traffic into the managed system. This may also result in removing more capable probes from the probe search space.

Property 4 *A candidate probe targeting a certain number of suspicious nodes is considered more desirable than other probes with more or less suspicious nodes.*

A high priority should be given to a candidate probe that is able test a certain number in the current suspect set. The following definition can be used to preserve this property.

Definition 5.5 *A candidate probe $p_l \in \sum_P$, with a certain number of suspect nodes from the diagnostic set F_t in its test set, which yields a higher value output of a continuous function in the range of (0, 1) is considered more eligible than other probes .*

Based on definition 5.5, property 4 can be captured by the following constraint:

$$C_4: (|p_l \cap F_t|) \Rightarrow (0, 1) \tag{5.13}$$

Hence, the candidate probe should be examined with the current diagnostic set F_t rather that a subset of it. The extreme function values of 0 and 1 are excluded from the function range. A value of 0 means that the candidate probe's test set may not include any of the suspect nodes in F which contradicts the constraint C_1. A value of 1 means that the candidate probe's test set could contain all the nodes of the suspect set F which contradicts the constraint C_3. Whether the preference will be given to candidate probes with more or less suspected nodes in their test set will be discussed in detail in the following section.

Property 5 *A candidate probe traversing a certain number of healthy nodes is considered more desirable than other probes with more or less of healthy nodes.*

Another factor that may play a major role in a candidate probe selection is the number of healthy nodes present in its test set. This property can be defined as follows.

Definition 5.6 *A candidate probe $p_l \in \sum_P$, with a certain number of healthy nodes from the diagnostic set G_t in its test set, which yields a higher value output of a continuous function in the range of [0, 1], is considered more desirable that other probes .*

A candidate probe with a certain number of healthy nodes in its test set is given high priority than other candidate probes. Based on definition 5.6, property 5 can be captured by the following constraint:

$$C_5: (|p_l \cap G_t|) \Rightarrow [0, 1] \tag{5.14}$$

The evaluation of this constraint can break a tie between competing probes and the preference will be considered based on the situation at hand. The value of zero implies that the candidate probe may not have a single healthy node in its test set. This probe should be given the highest priority since the probe test set is basically a subset of the suspect set. The success or failure of this probe can positively have a huge impact on the suspect set and radically modify its size. On other hand, a candidate probe whose test set include all the healthy nodes of the current *G* should be given the lowest priority.

Clearly the constraints C_1, C_2, and C_3 are crisp ones and they could be either fully satisfied or fully violated. However, the constraints C4 and C_5 are fuzzy in the sense that some probes will satisfy these two constraints with different degrees of satisfaction. The domains of the fuzzy constraints C_4 and C_5 are also fuzzy sets with membership functions dom_{C_4} and dom_{C_5}, respectively, where [21]:

$$dom_{C_4}(p_l) = \max\{C_4(p_l, F_t)\,|\}\forall p_l \in \Sigma_P \tag{5.15}$$

$$dom_{C_5}(p_l) = \max\{C_5(p_l, G_t)\}\forall p_l \in \Sigma_P \tag{5.16}$$

During the search process for a best solution, problem constraints will be evaluated individually. These membership functions can be used to find an instantiation for a certain constraint that can satisfy it the best.

5.4.3 The Satisfaction of the Fuzzy Constraints

The eligibility of a given probe for the fault identification task may depend on two crucial factors:

1. The ratio of the suspect nodes represented in its test set,
2. The ratio of healthy nodes represented in its test set.

Recognizing the importance of this knowledge is essential in order to make a better judgment pertaining to the probe selection. Therefore, the contributions of the fuzzy constraints to the selection process play a vital role in this regard. One way to extract this significant piece of information from a candidate probe is by obtaining the intersection set of the candidate probe test set with both diagnostic sets and then calculating their cardinalities. That is, the cardinality of the set { $p_l \cap F_t$ }, specified by the constraint C_4, determines the number of the suspected nodes that the candidate probe p_l may have. On the other hand, the cardinality of the set { $p_l \cap G_t$ }, specified by constraint C_5, gives the number of the healthy nodes it has. Based on this obtained information, the next step is to determine what probes are more appropriate for the fault identification task.

One approach is to iteratively select a probe that covers a large number of suspect nodes. The success of such a probe gives a large amount of information as its entire suspect nodes will be removed from the suspect set. However, if the probe fails then the its healthy nodes will be added to the suspect set. Furthermore, because the probe test set has a large number of suspect nodes, it will be more likely that this probe will fail, risking its perfectly good nodes to be added to the suspect set. This approach can be very effective if the candidate probe has no healthy nodes in its test set. That is, the probe test set is actually a subset of the suspect set. In this case, if the probe fails, then the suspect set will be reduced to the test set of this probe, which will be less than the original one. If the probe succeeds, then all the nodes of the probe test set will be removed from the suspect set. Thus, the new suspect set will be much less than the original one.

The other approach is to select a probe with the least number of the suspect nodes. Not much information may be gained if the probe succeeds, as the suspect set will be slightly reduced. However, if it fails, there are two possible scenarios:

- If the probe test set contains only a single suspect node, then the malfunctioning node has been identified. This is the best scenario as the fault analysis terminates.
- If the probe set contains more than a single suspect node and no healthy nodes, then the suspect set will be significantly reduced to that of the probe test set. Otherwise, the healthy nodes will also be added to the suspect set.

It is clear from the above discussion that a better probe has no or few healthy nodes in its test set. In case of failure, the candidate probe will not expand the suspect set dramatically. It may, however, have a large or small number of suspect nodes as each approach has their own merits and shortcomings. In this study, probes with the lowest number of both suspect and healthy nodes are given the highest priority.

Let $\overline{p}$ refer to the cardinality of the set $\{ p_l \cap F_t \}$, (i.e., $\overline{p} = | p_l \cap F_t |$), and $\overline{\overline{p}}$ refer to the cardinality of the set $\{ p_l \cap G_t \}$, (i.e., $\overline{\overline{p}} = | p_l \cap G_t |$). The constraints C_4 and C_5 are treated as fuzzy variables that are empirically associated with three linguistic terms (fuzzy sets): Low, Med, and High. Depending on the constraint at hand, each linguistic variable is semantically different from its counterpart. For example, the fuzzy set Low represents low number of suspect nodes in a given probe, according to constraint C_4. It also refers to the low number of healthy nodes in a given probe, according to constraint C_5. These fuzzy sets assume half-triangular, full-triangular, and S-shaped membership functions, respectively.

Given $\overline{p}$, the fuzzy set Low member function of C_4 is defined as follows:

$$\mu_{C4Low}(\overline{p}, b) = \begin{cases} 1; & \overline{p} = 1 \\ \dfrac{b - \overline{p}}{b - 1}; & 1 < \overline{p} \le b \\ 0; & \overline{p} > b \end{cases} \tag{5.17}$$

The fuzzy set Med membership function of C_4 is defined as follows:

$$\mu_{C4Med}(\overline{p}, a, b, c) = \begin{cases} 0; & \overline{p} \le a \\ 1 - \dfrac{b - \overline{p}}{b - a}; & a < \overline{p} \le b \\ \dfrac{c - \overline{p}}{c - b}; & b < \overline{p} \le c \\ 0; & \overline{p} > c \end{cases} \tag{5.18}$$

The fuzzy set High membership function of C_4 is defined as follows:

$$\mu_{C4High}(\overline{p}, b, c) = \begin{cases} 0; & \overline{p} \le b \\ 1 - \dfrac{c - \overline{p}}{c - b}; & b < \overline{p} \le c \\ 1; & \overline{p} > c \end{cases} \tag{5.19}$$

where the parameters $a = 0.25|F_t|$, $b = 0.50|F_t|$, and $c = 0.75|F_t|$. The shapes of the membership functions of the fuzzy constraint C_4 are shown in Fig. 5.5.

Given $\overset{\frown}{\bar{p}}$, the fuzzy set Low member function of C_5 is defined as follows:

$$\mu_{C5Low}(\overset{\frown}{\bar{p}}, f) = \begin{cases} 1; & \overset{\frown}{\bar{p}} \le K \\ \dfrac{f - \overset{\frown}{\bar{p}}}{f - 1}; & K < \overset{\frown}{\bar{p}} \le f \\ 0; & \overset{\frown}{\bar{p}} > f \end{cases} \tag{5.20}$$

The fuzzy set Med membership function of C_5 is defined as follows:

$$\mu_{C5Med}(\overset{\frown}{\bar{p}}, e, f, g) = \begin{cases} 0; & \overset{\frown}{\bar{p}} \le e \\ 1 - \dfrac{f - \overset{\frown}{\bar{p}}}{f - e}; & e < \overset{\frown}{\bar{p}} \le f \\ \dfrac{g - \overset{\frown}{\bar{p}}}{g - f}; & f < \overset{\frown}{\bar{p}} \le g \\ 0; & \bar{p} > g \end{cases} \tag{5.21}$$

The fuzzy set High membership function of C_5 is defined as follows:

$$\mu_{C5High}(\overset{\frown}{\bar{p}}, f, g) = \begin{cases} 0; & \overset{\frown}{\bar{p}} \le f \\ 1 - \dfrac{g - \overset{\frown}{\bar{p}}}{g - f}; & f < \overset{\frown}{\bar{p}} \le g \\ 1; & \overset{\frown}{\bar{p}} > g \end{cases} \tag{5.22}$$

where the parameters $e = 0.25|G_t|$, $f = 0.50|G_t|$, and $g = 0.75|G_t|$. The shapes of the membership functions of the fuzzy constraint C_5 are shown in Fig. 5.6.

Based on the previous discussion, simple fuzzy rules can be obtained as follows. For the constraint C_4, we have the following rules:

R1: if $\bar{p}$ is Low Then C_4 is A_1

R2: if $\bar{p}$ is Med Then C_4 is A_2

R3: if $\bar{p}$ is High Then C_4 is A_3

For the constraint C_5, we have the following rules:

R4: if $\overset{\frown}{\overline{p}}$ is Low Then C_5 is B_1

R5: if $\overset{\frown}{\overline{p}}$ is Med Then C_5 is B_2

R6: if $\overset{\frown}{\overline{p}}$ is High Then C_5 is B_3

The fuzzy consequents in the proposed rules are defined to reflect the desirable characteristics required in a given probe. For example, if probes with less number of both suspect and health nodes in their paths should be given high priorities over other probes, then the fuzzy consequents can be defined as follows: $A_1 = \{0.9/C_4\}$, $A_2 = \{0.5/C_4\}$,

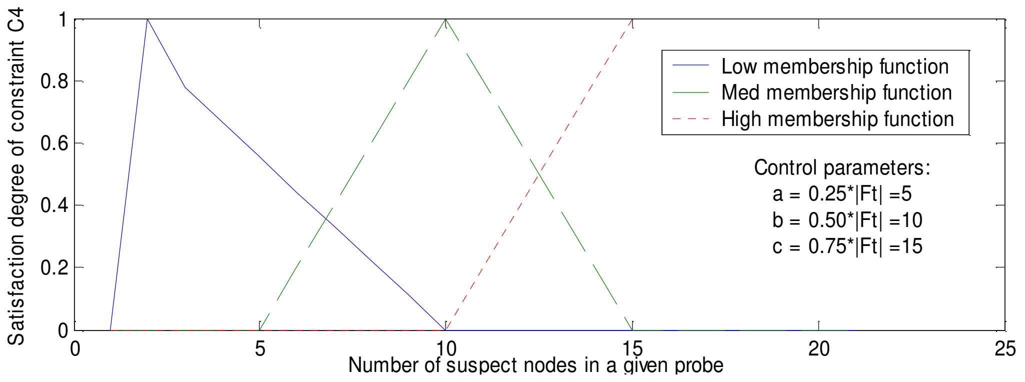


Figure 5.5: Low, Med, and High membership functions for the constraint C4 $(|F_t| = 20)$.

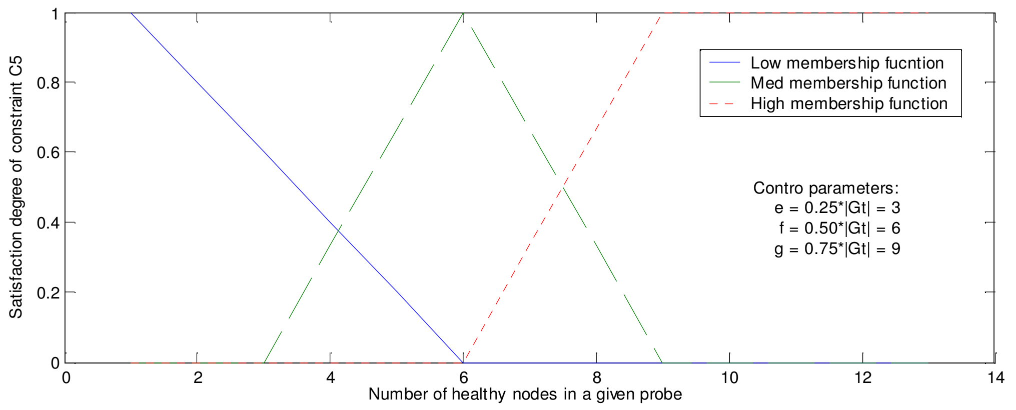


Figure 5.6: Low, Med, and High membership functions for the constraint C_5 $(|G_t| = 12)$. K=1.

$A_1 = \{0.2/C_4\}$, $B_1 = \{0.9/C_5\}$, $B_2 = \{0.5/C_5\}$, and $B_3 = \{0.2/C_5\}$. Different probe selection criteria require different fuzzy consequent definitions.

To determine the satisfaction degree of the fuzzy constraints C_4 and C_5, we group the relevant fuzzy rules for each constraint and apply the Mamdani inference mechanism on each group. By applying the Mamdani mechanism on the fuzzy rules (R1, R2, and R3), the satisfaction degree of the constraint C_4 is obtained as follows:

1. Calculate the firing level for each rule in the fuzzy rules group of C_4:

$$\tau_1 = \mu_{C4Low}(\overline{p})$$

$$\tau_2 = \mu_{C4Med}(\overline{p})$$

$$\tau_3 = \mu_{C4High}(\overline{p})$$

2. Calculate the output of each rule as follows:

$$\mu_1(C_4) = \min(\tau_1, \mu_{A_1}(C_4))$$

$$\mu_{j2}(C_4) = \min(\tau_2, \mu_{A_2}(C_4))$$

$$\mu_3(C_4) = \min(\tau_3, \mu_{A_3}(C_4))$$

3. Aggregate individual rule outputs to obtain overall fuzzy set C_4 with membership defined by:

$$\mu(C_4) = \max_{r=1}^{3}(\mu_r(C_4))$$

$\mu(C_4)$ represents the satisfaction degree of the constraint C_4 given the number of the suspect nodes, $\overline{p}$, in its path.

Similarly, we apply the same inference mechanism on the second group of the fuzzy rules, (R4, R5, and R6), to obtain the satisfaction degree of the constraint C_5, as follows:

1. Calculate the firing level for each rule in the fuzzy rules group of C_5:

$$\tau_4 = \mu_{C5Low}(\overline{\overline{p}})$$

$$\tau_5 = \mu_{C5Med}(\overline{\overline{p}})$$

$$\tau_6 = \mu_{C5High}(\bar{\bar{p}})$$

2. Calculate the output of each rule as follows:

$$\mu_1(C_5) = \min(\tau_4, \mu_{B_1}(C_5))$$

$$\mu_2(C_5) = \min(\tau_5, \mu_{B_2}(C_5))$$

$$\mu_3(C_5) = \min(\tau_6, \mu_{B_3}(C_5))$$

3. Aggregate individual rule outputs to obtain overall fuzzy set C_5 with membership defined by:

$$\mu(C_5) = \max_{r=1}^{3}(\mu_r(C_5))$$

$\mu(C_5)$ represents the satisfaction degree of the constraint C_5 given the number of the healthy nodes, $\bar{\bar{p}}$, in its path.

5.5 Constraint Propagation

The arc consistency takes the form of domain reduction rules. By removing the inconsistent values from the domains of the set variables $\{f, p\}$, the constraint propagation techniques narrow down the search space without ruling out any solution. If one of the domains is reduced to an empty set, then a solution to the probing problem can not be found from the given probes. The constraint propagation techniques are embedded within the branch and bound search algorithm.

5.5.1 Domain Reduction Rules

In order to perform the constraint propagation, certain domain reduction rules should be developed. Let *C* be the set of the problem constraints $C = \{C_1, C_2, C_3, C_4, C_5\}$. Since the problem constraints are binary, the following reduction rules can be applied to achieve the required arc consistency:

Arc Consistency rule 1

$$\frac{\langle c \in C; f \in \Sigma_f, p \in \Sigma_p \rangle}{\langle c; f \in \Sigma'_f, p \in \Sigma_p \rangle} \tag{5.23}$$

Where $\Sigma'_f := \{f \in \Sigma_f \mid \exists p \in \Sigma_p (f, p) \in c\}$

Arc Consistency rule 2

$$\frac{\langle c \in C; f \in \Sigma_f, p \in \Sigma_p \rangle}{\langle c; f \in \Sigma_f, p \in \Sigma'_p \rangle} \tag{5.24}$$

Where $\Sigma'_p := \{p \in \Sigma_p \mid \exists f \in \Sigma_f (f, p) \in c\}$

The first domain reduction rule simply modifies the domain Σ_f in a way that for each instantiation of the variable f with a subset of suspect nodes, there exist corresponding probes in the domain Σ_p. The notation Σ'_f refers to the new domain. The second reduction rule behaves in the same manner but operates on the probe variable of the constraint and results in the following new domain Σ'_p.

5.5.2 Maintaining Arc Consistency (MAC) Algorithm

The domain reduction rules described in the previous section are used by the arc consistency algorithm developed in this section. The following definition may help us formulate the arc consistency algorithm.

Definition 5.7

- *Consider a binary constraint C on the variables x and y with the domains D_x and D_y, that is $C \subseteq D_x X\ D_y$. We call C arc consistent if*
  - $\forall a \in D_x \exists b \in D_y (a,b) \in C,$
  - $\forall b \in D_x \exists a \in D_x (a,b) \in C.$
- *We call a CSP arc consistent if all its binary constraints are arc consistent*

According to this definition a binary constraint is arc consistent if every value in each domain has a *support* in the other domain. In the literature, MAC is alternatively called Arc Consistency Look Ahead or Full Look Ahead and summarized in Fig. 5.7.

To explain the algorithm of arc consistency, let us suppose that the following two probes were chosen for the fault detection as shown in section 5.2 and sent to test the healthy status of the running network:

$$P_{45} = \{N_2, N_3, N_4, N_5\}$$
$$P_{16} = \{N_1, N_3, N_6\}$$

```
S0: = $\{C_1, C_2, C_3, C_4, C_5\}$;
S: = S0;
WHILE $S \neq \phi$ DO
    choose $C \in S$;
    $\Sigma_f := \{f \in \Sigma_f \mid \exists p \in \Sigma_p (f, p) \in C\}$;
    $\Sigma_p := \{p \in \Sigma_p \mid \exists f \in \Sigma_f (p, f) \in C\}$;
    IF $\Sigma_f$ changed THEN
        $S := S \cup \{ C' \in S_0 \mid C'$ is on $f' \}$
    ELSE IF $\Sigma_p$ changed THEN
        $S := S \cup \{ C' \in S_0 \mid C'$ is on $p' \}$
    ELSE IF
        $S := S - \{C\}$
    END
END
```

Figure 5.7: Arc algorithm.

Let us further assume that the probe P_{16} fails and probe P_{45} succeeds. The outcomes of both probes will have the following effect:

$$F_t = \{N_1, N_6\}$$

$$G_t = \{N_2, N_3, N_4, N_5\}$$

As a result the domain $\sum_{F_t}$ of the variable f is constructed as follows:

$$\sum_{F_t} = \{\{N_1\}, \{N_6\}, \{N_1, N_6\}\}$$

The domain of variable $p \in \sum_P$ is the set of the available probes without the successful probe sent during the fault detection process:

$$\sum_P = \{\{N_1, N_2\}, \{N_1, N_3\}, \{N_1, N_3, N_4\}, \{N_1, N_2, N_5\}, \{N_1, N_3, N_6\}, \{N_2, N_3, N_4\}, \{N_3, N_4\}, \{N_3, N_4, N_6\}\}$$

During the constraint propagation, maintaining arc consistency can be achieved as follows:

1. The element $\{N_1, N_6\}$ will be removed from the variable domain $\sum_F$ as a direct result of applying the reduction rule 1 on the second constraint C_3. Hence, the new domain will be as follows:

   $$\sum_{F_t} = \{\{N_1\}, \{N_6\}\}$$

   Using the same reduction rule on the same constraint will also result in modifying the second domain $\sum_P$ in which the element $\{N_1, N_3, N_6\}$ will be removed and the new domain will be as follows:

   $$\sum_P = \{\{N_1, N_2\}, \{N_1, N_3\}, \{N_1, N_3, N_4\}, \{N_1, N_2, N_5\} \{N_2, N_3, N_4\}, \{N_3, N_4\}, \{N_3, N_4, N_6\}\}$$

2. Applying the domain reduction rule 2 on the first constraint C_1, the elements $\{\{N_2, N_3, N_4\}, \{N_3, N_4\}\}$ will be removed from the variable domain $\sum_P$. Hence, the resulted new domain will be as follows:

   $$\sum_P = \{\{N_1, N_2\}, \{N_1, N_3\}, \{N_1, N_3, N_4\}, \{N_1, N_2, N_5\}, \{N_1, N_3, N_6\}, \{N_3, N_4, N_6\}\}$$

For the fuzzy constraints C_4 and C_5, local consistency can be achieved by removing elements from the variable domains which can not satisfy the constraints better than a given threshold α. Estimating an appropriate value for α will depend on various factors which discussed on the previous section ($0 < \alpha \leq 1$).

5.6 Branch and Bound Search Algorithm

In the literature, there are several methods reported for determining the joint satisfaction of the problem fuzzy constraints. One of these methods is based on the average combination principal. It is more intuitive than the others since it looks at the joint satisfaction as the accumulative satisfaction of the individual constraints.

Definition 5.8 *The degree of joint satisfaction of the constraints* $C_i, i = 1,...,5$, *by the instantiation,* $d = (f_j, p_l)$, $f_j \in \Sigma_f$ *and* $p_l \in \Sigma_P$ *is the average of the satisfaction of the individual constraints. That is:*

$$C_{ave}((C_1,...,C_5),d) = \frac{1}{5}\sum_{i=1}^{5} C_i(d) \tag{5.25}$$

During the search process, a best solution is an instantiation d' if the degree of the satisfaction of all the constraints is maximal. That is:

$$C((C_1,C_2,C_3,C_4,C_5),d') = \max((C_1,C_2,C_3,C_4,C_5),d) \tag{5.26}$$

For a given instantiation, the degree of satisfaction is calculated using Equation (5.25) and will be compared with another degree of satisfaction yielded by another instantiation (saved in a special variable called *bound*). To choose the best instantiation, Equation (5.26) is employed.

5.6.1 Constructing a Best Solution

The adopted measure of joint degree of satisfaction is characterized by two important aspects:

- The proportional contribution of each constraint to the joint satisfaction may be calculated independent of each other. The appropriateness of particular initiation for a given variable may be evaluated using the domains of the fuzzy constraints offered by Equations (5.15) and (5.16).
- The joint satisfaction is monotone with respect to the satisfaction to the individual constraints, that is, if d_1 and d_2 are such that

$$C_i(d_1) = C_i(d_2)$$
$$C_j(d_1) \geq C_j(d_2)$$

for $i = 1,\ldots,6$ and $i \neq j$. Then,

$$C((C_1, C_2, C_3, C_4, C_5), d_1) \geq C((C_1, C_2, C_3, C_4, C_5), d_2) \qquad (5.27)$$

In constructing the best solution, the monotonic aspect described by Equation (5.27) will be proven useful in the search process. When exploring the search space, we first compute the upper bound for the degree of satisfaction of the best possible extension of a given partial solution. The monotonic characteristic of the joint satisfaction permits a partial solution to provide an upper bound in terms of its degree of satisfaction. Any extension of the partial solution that may yield a degree of satisfaction lower than the upper bound will be excluded from further considerations.

5.6.2 Heuristic Search Mechanism

The main procedure of the search algorithm is shown in Fig. 5.8. The main algorithm is to initialize the domains of each variable, based on the responses of the chosen probes. It then performs simple constraint propagation on the formed domains to eliminate any inconsistencies that may be found in the variable domains. If the constraint propagation yields an empty set for any of the variable domains, then a solution can not be found. Otherwise, it invokes the bound and branch function. Since the variable p is the most constrained variable, the search for a best solution starts by initiating it from its domain first as shown in Fig. 5.9. After assigning a value for the variable f, the consistency

```
/* ALGORITHM BRANCH AND BOUND
PROCEDURE MAIN
BEGIN
/* Initializations of global variables*/
  $\Sigma_F$ = {superset of the suspect set $F$};
  $\Sigma_P$ ={superset of the probe set $P$};
  solution = array[1..3];
  bound = 0;
 failure = FALSE;
/*End of initializations
  propagate($\Sigma_F$ ,$\Sigma_P$ , failure);
    IF NOT failure THEN
          BOUND_AND_BRANCH ( $\Sigma_F$ ,$\Sigma_P$ , solution, bound);
    ELSE IF
       Report "solution can not be found"
    END
END
```

Figure 5.8: Bound and branch main algorithm.

is checked. If this instantiation is consistent, then the degree of satisfaction will be calculated using Equation (5.24) and examined with the bound variable. Initially, the bound variable is set to 0. This ensures that the first consistent instantiation will be saved by the search algorithm and examined along with other consistent instantiations during the search operation. Only instantiations with higher degree of satisfaction will be saved in the solution variable. If the current instantiation is inconsistent, the constraint propagation will be performed for the p domain with the value of the variable f being fixed. If the constraint propagation yields an empty set for one or both variables, then the

```
PROCEDURE BOUND_AND_BRANCH ($\Sigma_F, \Sigma_P$, solution, bound)
    BEGIN
        WHILLE $\Sigma_F <> \{\ \}$ DO
            choose f from $\Sigma_F$
            $\Sigma_F = \Sigma_F - f;$
            WHILE $\Sigma_P <> \{\ \}$ DO
                choose p from $\Sigma_P$
                $\Sigma_P = \Sigma_P - p;$
                IF $const(f, p)$ THEN
                    IF $satisfaction(f, p) > bound$
                        $bound = satisfaction(f, p);$
                        $solution = \{f, p\};$
                        $C = C \cup \{satisfaction(f, p) > bound\}$
                    END IF
                ELSE IF
                    $propagate(f, \Sigma_P, failure);$
                    IF failure THEN choose a new f
                END
            END
        END
    END BOUND_AND_ BRANCH
```

Figure 5.9: The search engine of the proposed algorithm.

current value of the variable f will be discarded and a new value will be assigned to f. The search for a best solution starts again in the same manner. The overloading propagate function is shown in Fig. 5.10.

```
PROCEDURE propagate ($\Sigma_F, \Sigma_P$, failure)
BEGIN
    failure = false;
    $\Sigma_F = \{ f \in \Sigma_F \mid \{ f \in \Sigma_F \}$; is a consistent instantiation};
    failure = ($\Sigma_F = \{\}$? );
    $\Sigma_P = \{ p \in \Sigma_p \mid \{ f, p \in \Sigma_p \}$ ; is a consistent instantiation};
    failure = ($\Sigma_P = \{\}$ ?);
END Propagate;
PROCEDURE propagate ($f, \Sigma_P$, failure)
BEGIN
    failure = false;
    $\Sigma_P = \{ p \in \Sigma_p \mid \{ f, p \in \Sigma_p \}$ ; is a consistent instantiation};
    failure = ($\Sigma_P = \{\}$? );
END Propagate;
```

Figure 5.10: Overloading constraint propagation function.

5.7 Summary

In this chapter, we present a new fuzzy CSP-based technique for probe selection. Instead of sending all probes, a small subset of these probes is utilized for the purpose of fault detection. If one or more of these probes fail, then a fault has been detected. Thus, the fault identification function is invoked. It starts by analyzing the responses of the fault detection probes, and based on the information gained by the fault analysis, it creates two special sets for the most suspect and healthy nodes. The new sets, along with well-defined crisp and fuzzy constraints, are used to choose an appropriate probe that satisfies the imposed constraints the most. The effectiveness of the proposed scheme is reported in Chapter 8.

Chapter 6

Distributed Alarm Correlation Algorithm

6.1 Introduction

Network components are often equipped with monitoring tools that are able to detect any network abnormalities. These tools are configured to send appropriate notifications (alarms) to their assigned managers, in case of network failure. In order to identify the source of the problem, the majority of fault managements systems analyze these alarms using deterministic approaches such as codebooks and expert systems. However, computer networks are such complex and noisy environment that the information carried by these alarms is often imperfect. Thus, taking into account this intrinsic property of network alarms has been considered crucial for achieving effective alarm correlation performance.

In this chapter, we propose a novel distributed, alarm-correlation based approach. The network is divided into disjoint management domains and each domain is assigned an intelligent agent. Network entities within each domain are configured to direct their alarms to the dedicated intelligent agent. Within the framework of the Dempster-Shafer evidence theory, the network alarms are viewed as pieces of evidence by the intelligent agent. Using a given fault propagation model, each intelligent agent correlates these alarms into a new alarm and sends the new alarm to its agent manager. The new alarm constitutes the view point of the intelligent agent regarding the current status of the managed network. To form a global and cohesive view point, these partial views are, therefore, fused by a higher management entity called the agent manager.

6.2 Distributed Fault Management Systems

One of the most important objectives of fault management systems is developing effective distributed fault management approaches [6, 21, 22]. The distributed approach is realized by dividing the network into management domains and each domain is assigned a dedicated manager. Generally speaking, distributed fault management approaches can be classified into two main categories, namely, the hierarchical localization and the decentralized localization. The first approach, the hierarchical localization, assumes the existence of a central manager that supervises all the domain managers and has a global view of the network. If a network failure affects only a single domain, then the alarm correlation process will be performed locally by the respective domain manager to identify the network failure. If the network failure spans over several domains, then the central manager will perform the alarm correlation as if there were no domain managers. On the other hand, the decentralized approach eliminates the role of the central manager altogether. Each domain manager separately identifies the network failure based on the alarms observed in their domain. The domain managers then cooperate with each other to form a global view of the network status. In the following section, we review some issues pertaining to each approach.

6.2.1Issues in Distributed Systems

Clearly, each distributed approach discussed above employs a different strategy to identify the source of the network problem. As such, each has its own merits and shortcomings. The hierarchical approach is considered technically simpler to implement. However, the fact that a central manager takes the responsibility of resolving a multi-domain network failure defeats the purpose of the distributed approach. Because the central manager exclusively performs the fault identification task, it has to collect and analyze the network alarms issued by all the management domains. This means that the workload is not being effectively distributed among the management domain managers. The second approach requires the collaboration among the domain managers to reach a global view. This requirement, however, is complicated by the fact that domain managers have a limited view and lack global information of the network as a whole. Without this knowledge, a domain manager may not be able to establish a causal relationship between alarms observed in its domain and their potential causes in other domains. Based on the above discussion, an effective distributed fault management approach has to address the following two fundamental issues:

- The availability of global information about the network topology and state. Due to the hierarchical nature of computer networks, alarms do propagate from one domain to another. Furthermore, the main cause of a certain alarm, observed in a particular domain, may not be visible in the same domain. Hence, while a domain manager can locate the source of alarms observed in its domain, it may not be able to identify potential sources that reside outside of its domain.
- Coordination among the domain managers. A failure in a network component in one domain can affect other components in other domains. Though the affected components will respond to the same failure by sending alarms to their respective managers, each domain manager can only correlate alarms observed in its domain to identify the same failure. Therefore, it is recommended that the managers of the affected domains cooperate with each other and collectively identify the failed entity and its domain.

The proposed distributed approach addresses these two issues as follows. First, a fault propagation model for the whole network is constructed using one of the methods described in chapter 3. Secondly, the central manager and each domain manager are required to have a copy of the same model. Consequently, not only will a domain manger be able to correlate its alarms to network failures in its domain, but it can also correlate them to failures in other domains. The domain manager need not know the exact location of the network failure responsible for its observed alarms. The network failure will be localized by the central manager. This way, while the alarm correlation is performed locally, it may indicate the source of the problem globally. The alarm correlation process results in a new alarm representing the view of the domain manager of the network status. The manager's view, however, is still incomplete as only local alarms are being considered in the alarm correlation.

Since the central manager has the same fault propagation model, it can globally correlate the new alarms constructed by the domain managers and more accurately identify the network failure. The global alarm correlation performed by the central manager eliminates the need for the domain managers to directly communicate with each other and exchange information about the status of their domains. They are simply required to communicate their new alarms to the central manager. Thus, excessive management traffic can also be avoided.

6.2.2 Assumptions and Notations

Alarms generated by network entities are called *primitive alarms*. The purpose of the local alarm correlation is to combine the primitive alarms into a new alarm. The new combined alarm is called a *composite alarm*. Each primitive alarm in the set $\{< a_i > | i = 1,...,n\}$ is expected to provide probabilistic evidence regarding each network failure f_j in the fault hypothesis space $\Omega = \{f_1,...,f_c\}$. The probabilistic evidence is represented by a set of belief assessments in the following form:

$$B^{ai} = \{bel(f_1^{ai}),...,bel(f_c^{ai})\} \tag{6.1}$$

B^{ai} represents the belief set given by the primitive alarm a_i and $bel(f_j^{ai})$ denotes the its belief assessment of the given fault hypothesis. The fault hypotheses, in the belief set, are arranged in a non-increasing order. $\Omega_{ai} = \{f_1^{ai},\ldots,f_c^{ai}\}$ is a permutation of Ω. To simplify the probabilistic model accompanied by the distributed fault propagation model, the following dependency assumptions are made:

- Failures in the given fault propagation model, which are perceived as symptoms and observed as network alarms, are not directly dependent, since these symptoms may occur as a result of the same underlying problem. Therefore, given known states of antecedent nodes in the model, the failure nodes (corresponding to symptoms) of dependent node are independent of each other [18, 20].
- Failures that may contribute to the occurrence of a particular symptom (alarm) and which perceived as faults are considered independent. This assumption is commonly reported in the literature [19, 20, 21, 22, 34, 45].

Based on the above assumptions, network alarms, representing the evidence set, are considered independent. Furthermore, hypotheses in the frame of discernment Ω, representing network failures, are assumed to be mutually exclusive and exhaustive.

6.3 Distributed Fault Propagation Model

Since computer networks are usually modeled in layered approach, the distributed fault propagation model can be constructed based on the dependency relationship among the network components. As shown in Fig. 3.4, a dependency relationship among the network components in different layers is usually modeled as a dependency graph. The dependency relationship may represent the dependency between network functions provided by the lower layers such the physical and data link layers and services provided by the higher layers such the transport and application layers. Due to this intrinsic property, failures can only propagate from components of lower to components of higher layers or between components on the same layer. Thus, a failure in a network function of the physical or data link layer is expected to manifest itself as a failure in the services provided by the application layers. In the dependency graph, failures in lower layers are considered faults; while failures in higher layers are considered symptoms or alarms. Since these dependencies among the network entities are non-deterministic in nature, the uncertainties about these dependencies are represented by assigning probabilities to the links and nodes in the given dependency model.

In a dependency graph, a weight assigned to a dependency link represents the probability that the node at the tail of the edge fails given the node at the head of the edge fails. The dependency graph, in turn, can then be easily mapped into a Bayesian (Belief) network. Therefore, the belief network theory can be utilized to calculate belief assessments for each fault hypothesis. Furthermore, the obtained Bayesian network of the whole managed network is then simplified into a *correlation graph* (bipartite graph). The resultant correlation graph is considered the distributed fault propagation model and stored in the knowledge base of each intelligent agent and the agent manager. The process of obtaining the distributed fault propagation model of the managed network is described in the subsequent sections. However, we next present a formal definition of the Belief network.

6.3.1 Belief Networks

Formally, a belief network is a pair $\langle \mathrm{G}, \mathrm{P} \rangle$, where G is a directed acyclic graph, in which each node represents a random variable over a multi-valued domain and is denoted by $\mathrm{V_i}$. The set of all nodes is denoted by V. Let $\mathrm{D_i}$ represent the domain of the variable $\mathrm{V_i}$. The set of directed edges represented by E indicates causal relationships between the variables. $P = \{P_i\}$, the conditional probability matrix associated with a random variable $\mathrm{V_i}$, reflects the strengths of casual influences among these variables. Let $Par(v_i) = \{\mathrm{V_{i1}}, \mathrm{V_{i2}}, \ldots\ldots \mathrm{V_{in}}\}$ be the set of parents of the variable V_i. P_i is a $(|Par(V_i)|+1)$-dimensional matrix of size $|D_i| \, x \, |D_{i1}| \, x \ldots\ldots x \, |D_{in}|$; where $P_i(v_i, v_{i1}, \ldots, v_{in}) = \mathrm{P}(\mathrm{V_i} = \mathrm{v_i} \mid \{\mathrm{V_{i1}} = \mathrm{v_{i1}}, \ldots\ldots \mathrm{V_{in}} = \mathrm{v_{in}}\}$. An assignment of variables in set V is denoted by $A = \{\mathrm{V_1} = \mathrm{v_1}, \mathrm{V_2} = \mathrm{v_2}, \ldots, \mathrm{V_n} = \mathrm{v_n}\}$ where each $v_j \in D_j$. Given a subset of random variables $U_k = \{V_{k1},....,V_{km}\} \subseteq V$, an assignment of U_K^A is consistent with assignment A. Evidence set e is an assignment U_o^A, where $U_o \subseteq V$ a set of variables whose values are known, and for each $V_{oj} \in U_o$, v_{oj}^A is its observed value.

Given evidence set *e*, belief networks can be used to calculate *belief assignment*. The belief assignment task is to calculate $bel(\mathrm{V_i} = \mathrm{v_i}) = \mathrm{P}(\mathrm{V_i} = \mathrm{v_i} \mid \mathrm{e})$ to variable $\mathrm{V_i}$. A belief updating algorithm, polynomial with respect to |V|, is available for poly-*trees*, i.e., directed graphs without undirected cycles [39].

6.3.2 Mapping Dependency Graph to a Belief Network

A dependency graph of a managed network is often obtained from the network topology. The dependency graph is then transformed into a Bayesian network simply by reversing its edges. For instance, the Bayesian network of the dependency graph shown in Fig. 3.4 is illustrated in Fig. 6.1. Each node in the belief network may represent a dependency graph node in one of its failure modes. The belief network corresponding to the layered dependency graph is constructed using the following steps [11].

- A random variable in the belief network will be created for each failure mode in the dependency graph. Let $\mathrm{V_i}$ be belief network node corresponding to a failure mode that represents a particular entity in the dependency graph (for example, $\mathrm{Service_L}$(a,b), $\mathrm{NetworkFunction_L}$, $\mathrm{Protocol_L}$, etc.) Its corresponding domain, $\mathrm{D_i}$, may take the values of {true, false} to indicate whether their corresponding failure has occurred or not.
- Let $\mathrm{V_i}$ and Vj represent two belief network nodes that correspond to node X in failure mode $\mathrm{F_i}$ and node Y in failure mode $\mathrm{F_j}$, in the dependency graph, respectively. If a dependency relationship exists in the dependency graph between X and Y in the form of $\mathrm{X} \rightarrow \mathrm{Y}$, and $\mathrm{V_i}$ is contributing to the occurrence of $\mathrm{V_j}$, then add an edge from $\mathrm{V_i}$ to $\mathrm{V_j}$ in the belief network.

- Using the independency assumptions made in the previous section, the probability matrix $\mathrm{P_j}$ associated with node $\mathrm{V_j}$ represents the following conditional probability distribution:

  $\mathrm{P}(\mathrm{V_j} = \mathit{false} \mid V_i = \mathit{false}) = 1$

  $\mathrm{P}(\mathrm{V_j} = \mathit{false} \mid V_i = \mathit{true}) = 1 - P(F_i, F_j)$

  $\mathrm{P}(\mathrm{V_j} = \mathit{true} \mid V_i = \mathit{false}) = 0$

  $\mathrm{P}(\mathrm{V_j} = \mathit{true} \mid V_i = \mathit{true}) = P(F_i, F_j)$

where $P(F_i, F_j) = P\{Y$ is in failure mode $F_j \mid X$ is in failure mode $F_i\}$.

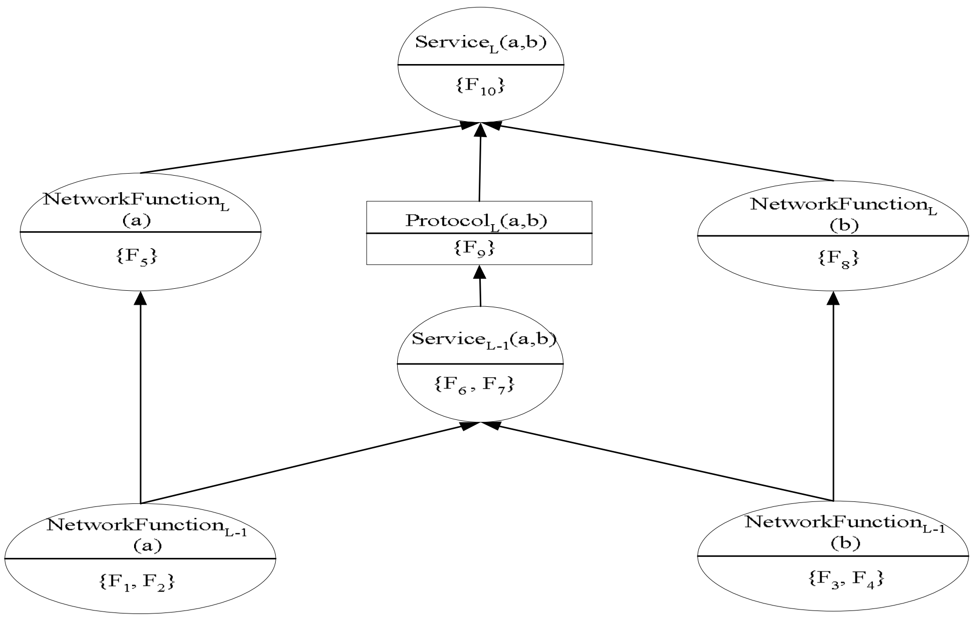


Figure 6.1: Belief network of the dependency graph in Figure 3.4.

6.3.3 Correlation Graph as a Distributed Fault Propagation Model

A causality graph may include information that does not contribute to the alarm correlation process. Certain alarms are not directly caused by any network failures. They are simply manifestations of other network alarms. These indirect manifestations may be eliminated without loss of information. Alarms may form many-to-one relation; e.g. $f_1, f_2, f_3 \rightarrow a_1$, or inference relation; e.g. $f_1 \rightarrow a_1 \rightarrow a_2$. All these relations represent causal equivalence. Consequently, all involved alarms can be aggregated into a single alarm. Therefore, the last step is to transform the obtained causality graph into a *correlation graph*. The detailed causality graph of the Belief network of Fig. 6.1 is shown in Fig. 6.2(a). Here, the failures in the lowest layer are considered faults; and the failures at higher layers are considered alarms. A causality graph can be converted into a correlation graph as follows [46]. If $f_j \rightarrow a_1 \ldots \rightarrow a_k$ in the causality graph, then in the correlation graph there is an edge from f_j to the first of $a_1 \ldots, a_k$ that is an alarm, i.e., there is an edge $f_j \rightarrow a_i$ where $a_1 \ldots, a_{i-1}$ are not alarms. Hence, the correlation graph corresponding to the causality graph given in Fig. 6.2(a) is shown in Fig. 6.2(b).

6.4 Distributed Alarm-Correlation Based Approach

The managed network supposedly has a tree-shaped topology and is logically partitioned into *K* disjoint management domains. *K* depends on the size and complexity of the given network. In this dissertation, we adopt a hierarchical structure for the fault management system. The lower-level domain managers referred to as *intelligent agents*, locally collect and analyze network alarms in their respective domains. They report to a higher-level manger, referred to as the *agent manager*, as shown in Fig. 6.3 [47]. The intelligent agents and the agent manger have the same distributed fault propagation model in the form of a bipartite causality graph as shown in Fig. 6.2(b).

6.4.1Intelligent Agent (IA)

Primitive alarms are generated by network components. The intelligent agents perceive these network components in their domains as sources of information. As such, the information carried by their primitive alarms may in fact be imperfect and exhibit a high degree of uncertainty. The intrinsic uncertainty properties are attributed to many factors including complexity, unreliability, and non-determinism in computer networks. As has

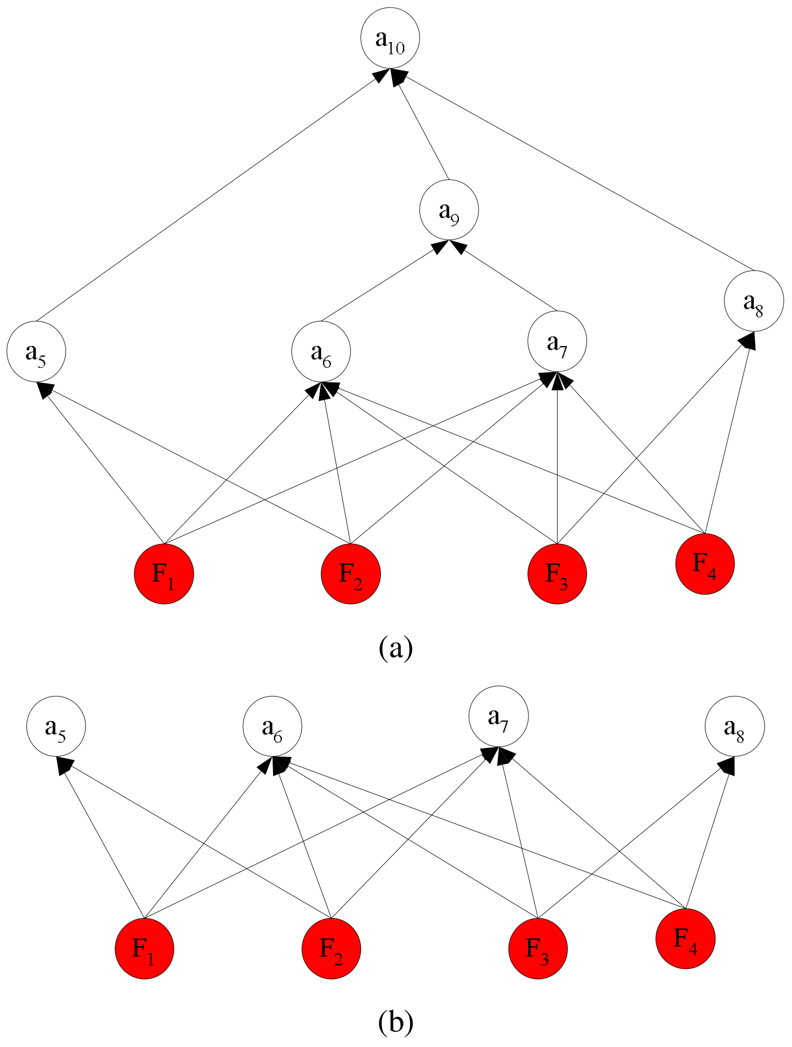


Figure 6.2: Alarm correlation causality graph.

been shown in [19, 47], the fault evidence (presented by the primitive alarms) may be ambiguous and inconsistent due to the following conditions:

- A primitive alarm may indicate a non-singleton set of fault hypotheses (*ambiguity*),
- Two or more of these primitive alarms may have disagreement regarding the main fault hypothesis (*inconsistency*).

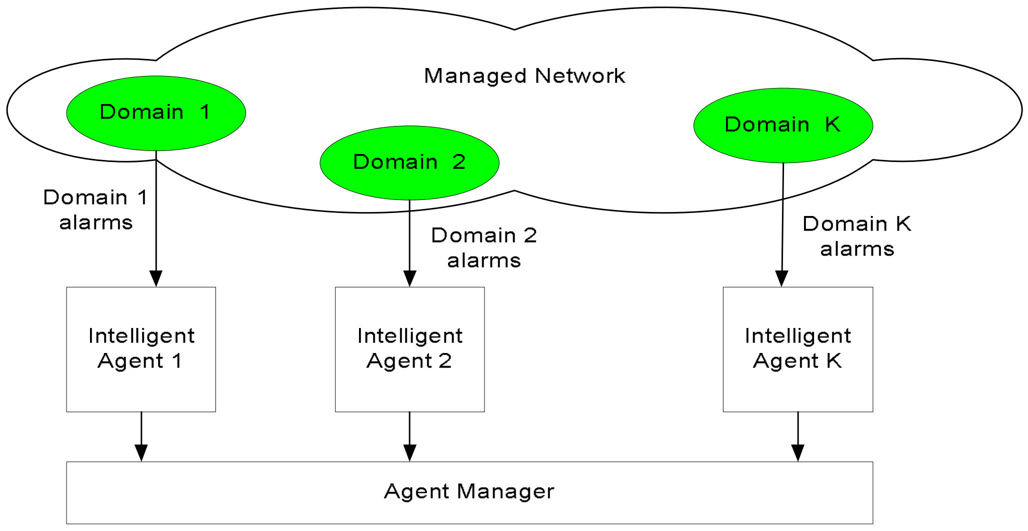


Figure 6.3: Distributed fault management system.

To reduce their negative impact on the local alarm correlation process, the intelligent agents carefully manage these uncertainty aspects. In this sense, the alarm correlation process is viewed as reasoning under uncertainty.

6.4.1.1 Intelligent Agent Structure

Given the uncertainty problem imposed by the set of the observed primitive alarms, the reasoning mechanism employed by an intelligent agent *k* should be capable of answering the following two questions:

1. What is the likely fault hypotheses that can explain the observed alarms?
2. If disagreement occurs among the primitive alarms, how can this conflict be resolved?

To deal with both aspects of the uncertainty issue, the intelligent agent performs the alarm correlation utilizing a hybrid combination of the probability theory and the Dempster-Shafer evidence theory. Using the given fault propagation model and the Pearl's belief propagation scheme, the intelligent agent constructs a belief assessment set B^{ai} for every primitive alarm a_i. Taking the form given in Equation (6.1), the belief assessment set identifies the most likely fault hypotheses from the view point of a_i. Within the framework of the evidence theory, the intelligent agent then constructs an evidence structure ES^{ai} for every belief set B^{ai}. To resolve any conflict among the primitive alarms, the intelligent agent fuses the obtained evidence structures into a single composite alarm A^{k}_{comp} using the Dempster's rule of combination. The structure of an intelligent agent *k* is shown in Fig. 6.4.

Thus, the intelligent agent's composite alarm is basically formed using the following three main steps:

- A belief assessment set is first obtained for each primitive alarm,
- An evidence structure is then constructed for each belief assessment set,
- Finally, the evidence structures are combined to form a new composite alarm.

The first step is discussed in detail next and the remaining two steps are discussed in the subsequent sections.

6.4.1.2 Alarm's Belief Assessment Set

To construct an evidence structure for each received network alarm, the intelligent agent needs first to determine the belief assessment for each primitive alarm. To accomplish this task, the intelligent agent uses its propagation fault model stored in its knowledge

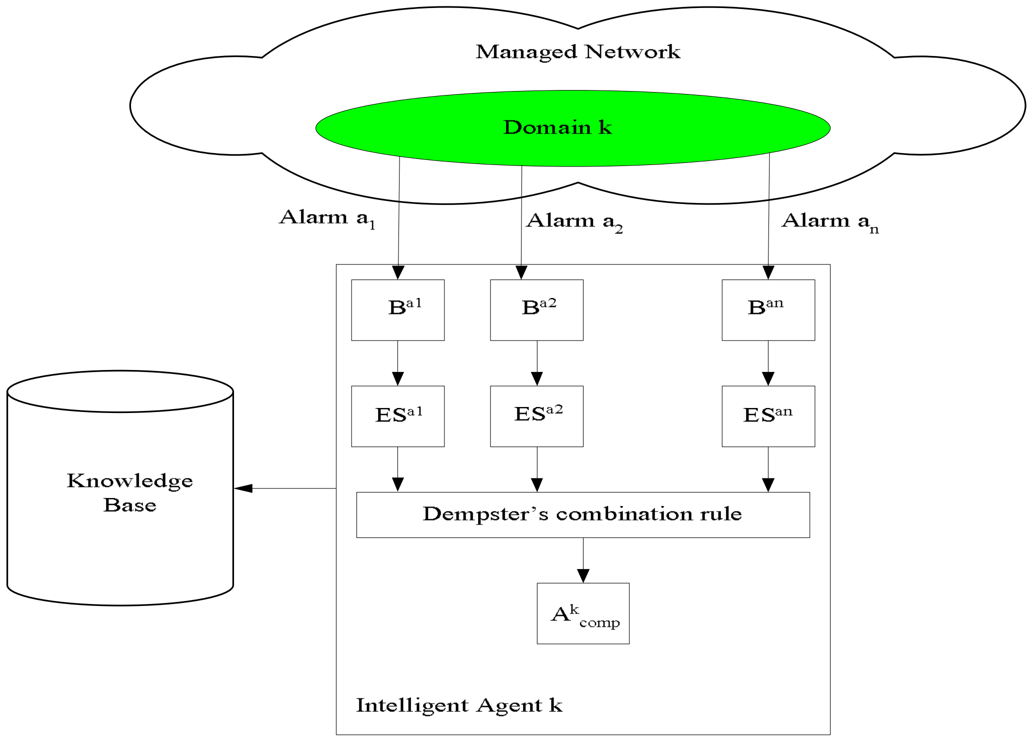


Figure 6.4: Structure of intelligent agent *k*.

database and the iterative belief updating proposed in [39]. The belief network (the fault propagation model) is viewed by the belief updating scheme as a noisy-OR model of probability distribution. In such networks, the belief assessment query calculated using the Pearl's message schema produces the posterior probability distribution.

According to Pearl's algorithm, the belief network nodes exchange λ and π messages (see Fig. 6.5). Message $\lambda_X(v_j)$ that node X sends to its parent V_j for every valid V_j's value v_j, denotes a posterior probability of the entire body of evidence in the sub-graph obtained by removing link $V_j \rightarrow X$ that contains X, given that $V_j = v_j$. Message $\pi_{U_i}(x)$, that node X sends to its child U_i, denotes a probability that $X = x$ (for every valid value of X) given the entire body of evidence in the sub-graph containing X created by removing edge $X \rightarrow U_i$. The complete description of the message passing created by removing edge $X \rightarrow U_i$. The complete description of the message passing algorithm is presented in Appendix B.

In a noisy-OR poly-tree, the term q_{XU_i} refers to the probability of activating the inhibitor controlling the link $X \rightarrow U_i$. The possible values that the random variables may have are {0, 1}, where 1 denotes the occurrence of the corresponding event and 0 means that the event did not occur. The probability that U_i occurs given that X occurs is $c_{XU_i} = 1 - q_{XU_i}$. Based on messages received from parents and children, node X computes $\lambda(x)$, $\pi(x)$ and $bel(x)$ as follows [39]:

$$\lambda(x) = \prod_{i=1}^{n} \lambda_{U_i}(x) \tag{6.2}$$

$$\pi(x) = \begin{cases} \alpha \prod_{j=1}^{m} (1 - cv_j x \pi_j x) & if \quad x = 0 \\ \alpha (1 - \prod_{j=1}^{m} (1 - cv_j x \pi_j x)) & if \quad x = 1 \end{cases} \tag{6.3}$$

$$bel(x) = \alpha \lambda(x) \pi(x) \tag{6.4}$$

The messages $\lambda(x)$ and $\pi(x)$ are computed using the following equations [39]:

$$\lambda_X(v_j) = \beta(\lambda(1) - q_{V_j x}^{v_j}(\lambda(1) - \lambda(0)) \prod_{k \neq j} (1 - c_{v_k x} \pi_{kx})) \tag{6.5}$$

$$\pi_{U_i}(x) = \alpha \prod_{k \neq i} \lambda_{U_k}(x) \pi(x) \tag{6.6}$$

where for $v_j = 1$, $\pi_j x = \pi(v_j)$; α is a normalizing constant, and β is any constant.

Initially $\lambda(x)$ will assume the value of 1 if *x* is indeed the observed value of *X*, for all observed nodes *X*. Otherwise, $\lambda(x)$ is set to 0. For all unobserved nodes $\lambda(x)$ is set to 1 for all values of *x*. For all parentless nodes $\pi(x)$ will be set to their corresponding prior probabilities. Once an intelligent agent receives one or more primitive alarms from its constituent entities, it consults its fault propagation model and assigns their corresponding belief network nodes with the value of 1. For those primitive alarms, which were not observed by the intelligent agent their corresponding belief nodes will be left unassigned (i.e., their $\lambda(0) = \lambda(1) = 1$). It then computes the belief assessments for all

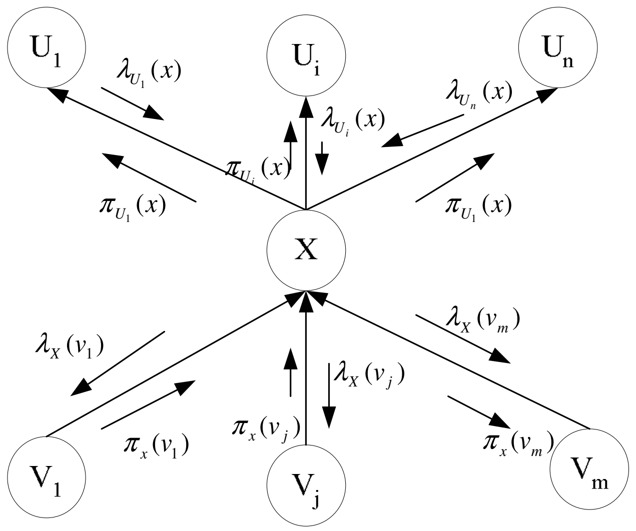


Figure 6.5: Message passing in Pearl's belief propagation.

the unobserved nodes (the fault hypotheses) for each primitive alarm using the belief updating algorithm. It starts from the evidence node (i.e., the belief network node representing the observed network alarm) and propagates the changed belief along the belief network edges by computing $bel(x)$, $\lambda_x(v_i)$, and $\pi_x(u_i)$ in every visited node. For every network alarm, a certain ordering is defined that is equivalent to the breadth-first order started from the evidence node. Thus, the belief propagation performed after receiving a network alarm a_i yields the following belief assessment set B^{ai}:

$$B^{ai} = \{bel(f_1^{ai}),...,bel(f_c^{ai})\} \tag{6.7}$$

6.4.1.3 Constructing Evidence Structures

The obtained belief assessment vector (Equation (6.7)) will be utilized by the intelligent agent to construct an evidence structure for each observed primitive alarm. The evidence structure takes the following form:

$$(A, m(A))$$

where A and $m(A)$ are the focal element and its mass, respectively as defined in section 3.5. We adopt the evidence model proposed by [48], which is called the Proportional

Difference Evidence Structure (PDES). One of the advantages of the PDES model is that it produces consonant focal elements; thus, the number of combined focals is reduced tremendously during the alarm correlation process. According to this model, the belief assessments obtained by the updating belief algorithm introduced in the previous section, are arranged in non-decreasing order such that the following relationship hold:

$$bel(f_1^{ai}) \geq bel(f_2^{ai}) \geq \ldots \geq bel(f_c^{ai})$$

where $f_j^{ai} \in \Omega_{ia}$ and Ω_{ai} is a permutation of Ω.

Thus, a primitive alarm can be assigned to a certain network failure as follows:

IF $bel(f_1^{ai}) \geq bel(f_2^{ai})$ then the network alarm, a_i, can be assigned to fault hypothesis class $\{f_1^{ai}\}$;

ELSE IF $bel(f_2^{ai}) \geq bel(f_3^{ai})$ then the network alarm, a_i, can be assigned to fault hypothesis class $\{f_1^{ai}, f_2^{ai}\}$;

ELSE IF $bel(f_3^{ai}) \geq bel(f_4^{ai})$ then the network alarm, a_i, can be assigned to fault hypothesis class $\{f_1^{ai}, f_2^{ai}, f_3^{ai}\}$;

$\vdots$

ELSE IF $bel(f_{c-1}^{ai}) \geq bel(f_c^{ai})$ then the network alarm, a_i, can be assigned to fault hypothesis class $\{f_1^{ai}, f_2^{ai}, \ldots, f_c^{ai}\}$;

If the primitive network alarm, a_i, yields an equal belief assessment to different fault hypotheses, then the above decision rule produces a compound set that contains all the fault hypotheses undistinguishable by a_i. Furthermore, this decision rule can be utilized to construct an evidence structure for each network alarm in four steps as follows:

1. The interval [0, $bel(f_1^{ai})$] is divided into c portions by means of the split points at $bel(f_j^{ai})$, $j = 2,\ldots,c$, which lead to c discrete values (let $bel(f_{c+1}^{ai}) = 0$),

$$m_j = bel(f_j^{ai}) - bel(f_{j+1}^{ai}), \qquad j = 1,2,\ldots,c.$$

2. The quantity a_j is associated with one of consonant class sets as follows:

$$m_1 \to \{f_1^{ai}\},$$
$$m_2 \to \{f_1^{ai}, f_2^{ai}\},$$
$$\vdots$$
$$m_{c-1} \to \{f_1^{ai}, f_2^{ai}, ..., f_{c-1}^{ai}\},$$
$$m_c \to \{f_1^{ai}, f_2^{ai}, ..., f_c^{ai}\};$$

where $\{f_1^{ai}, f_2^{ai}, ..., f_j^{ai}\}$ the is the union of f_1^{ai} through f_c^{ai} .

3. To obtain mass, m_j is normalized as follows:

$$m_j = m_j / bel(f_1^{ai}) = \frac{bel(f_j^{ai}) - bel(f_{j+1}^{ai})}{bel(f_1^{ai})}.$$

for all $j = 1,2,...,c$.

4. Given a network alarm a_i, its evidence structure can be formed as:

$$ES^{ai} = \left\{ \left\{ f_1^{ai}, ..., f_j^{ai} \right\}, \frac{bel(f_j^{ai}) - bel(f_{j+1}^{ai})}{bel(f_1^{ai})} \mid j = 1,2,...,c \right\} \tag{6.8}$$

The above process is intended to assign different weights to different fault hypothesis sets. The weight magnitude reflects the confidence about how much a particular hypothesis supports the received alarm. For example, the value $m_1 = bel(f_1^{ai}) - bel(f_2^{ai})$ signifies numerically the commitment that the received alarm a_i exactly belongs to the singleton fault hypothesis set $\{f_1^{ai}\}$. If the belief assessment is the same for two different hypotheses, then logically the weighted confidence should be assigned to a compound hypothesis set in which both fault hypotheses are included. For instance, if $bel(f_1^{ai}) = bel(f_2^{ai})$ then it is reasonable, since both hypotheses are not distinguishable given the received alarm a_i, to assign a_i to the union set of both fault hypotheses $\{f_1^{ai}, f_2^{ai}\}$. The union set indicates that both network faults are equally suspicious of the current network abnormality and a further analysis is needed to isolate a single cause.

6.4.1.4 Local Composite Alarm

The essence of the alarm correlation is to assign a new meaning to a set of observed alarms that can be explained by a certain set of fault hypotheses [49]. This is done so that the amount of information reaching the central manager is reduced and the network failure is more easily identified. Thus, the local alarm correlation is viewed as a fusion process by which the observed primitive alarms are combined into a new alarm. The new alarm resulted from the fusion process is called a composite alarm. The local composite alarm is obtained as follows. Let us assume that an intelligent agent k receives the following set of alarms:

$$A^k = \{a_1, ..., a_n\} \tag{6.9}$$

their associated belief assessment sets B^{a_i} for $i = 1, ..., n$, are presented as follows:

$$B^k = \{B^{a1}, ..., B^{an}\} \tag{6.10}$$

where each B^{a_i} has the form given in (6.7) and its elements are ranked in a non-increasing order. Their associated evidence structures are represented as follows:

$$ES^k = \{ES^{a1}, ..., ES^{an}\} \tag{6.11}$$

where each ES^{ai} has the form given in (6.8).

Considering the local alarm correlation as a fusion process, it can be defined as follows.

Definition 6.1 *Given a set of primitive alarms* $\{a_1, a_2, ..., a_n\}$, *a set of their associated belief assessments in the form of Equation (6.10) is obtained. Based on the obtained set of belief assessments, the set of evidence structures is then constructed, which has the form of Equation (6.11). A local composite alarm of the intelligent agent is formed by fusing (correlating) the set of the obtained evidence structures, without regard to the arrival order of the primitive alarms, into a single alarm* A^k_{comp}, *i.e.,* $A^k_{comp} \leftarrow ES^{a1} \otimes ES^{a2} ... \otimes ES^{an}$. *The symbol* $\otimes$ *refers to the Dempster's rule of* *combination.*

Therefore, based on definition 6.1, an intelligent agent can correlate the received set of alarms by simply applying the Dempster's rule of combination given in Equation (3.7) to combine all ES^{ai} for $i = 1,...,n.$ Due to the nature of the PDES scheme and the definition of commonality, the following formula can be used to calculate the commonality measure of the network failure f_j^{ai} for the primitive alarm a_i:

$$Q(\{f_j^{ai}\}) = \frac{bel(f_j^{ai})}{bel(f_1^{ai})} \tag{6.12}$$

To calculate the commonality measure of the network failure f_j^k for all the primitive alarms, the intelligent agent can use the following equation:

$$Q(f_j^k) = T\prod_{i=1}^{n} Q(f_j^{ai}) \tag{6.13}$$

where T is a normalization factor defined in Equation (3.8) and is independent of the network failure f_j^k of interest . Property 1, discussed in [50], leads to the following:

$$Q(\{f_j^k\}) \propto \prod_{i=1}^{n} bel(f_j^{ai}), \forall f_j \in \Omega \tag{6.14}$$

Thus, using the property of (6.14), the local composite alarm A_{comp}^k can be obtained as follows:

$$A_{comp}^k = \{Q(f_1^k),...,Q(f_c^k)\} \tag{6.15}$$

It should be noted that $Q(f_j^k)$ represents the combined belief assessment value of the network failure f_j^k. The elements in A_{comp}^k are arranged in a non-decreasing order and $\Omega_{A_{comp}^k}$ is a permutation of Ω.

A summary of the constructing local composite event algorithm employed by the intelligent is presented in Fig. 6.6.

6.4.2 Agent Manager

The task of the agent manager is to facilitate the cooperation of intelligent agents by correlating their local composite alarms. Each local composite alarm has the form of (6.15). If the agent manager receives only a single composite alarm, then it will identify the network failure as the one with the maximum belief assessment value given by the local composite alarm. However, if it receives multiple local composite alarms, then the global view is achieved by fusing all the received local composite alarms. The agent manger fuses the local composite alarms in the same manner as an intelligent agent fuses its primitive alarms. This composite alarm correlation process results in a global composite alarm. The network failure will be identified as the one with the maximum belief assessment value given by the global composite alarm.

6.4.2.1 The Agent Manager Structure

Assuming that the agent manger receives more than a single local composite alarm, its structure is shown in Fig. 6.7. Since, each composite alarm has the form of (6.15), the agent manager does not need to build a belief assessment vector for each composite alarm. However, an evidence structure will be constructed for each composite alarm. Utilizing the same PDES mechanism, the agent manger constructs a set of evidence structures for the composite alarms. Using the Dempster's rule of combination, the agent manger then correlates all the composite alarms, producing in the process a global composite event A_{glob}.

6.4.2.2 Global Composite Alarm

Let us assume that the agent manager receives the following set of composite alarms:

$$A = \{A_{comp}^{1}, \ldots, A_{comp}^{K}\} \tag{6.16}$$

where each A_{comp}^{k} has the form given in (6.15) and its elements are ranked in a non-increasing order. Their associated evidence structures are represented by the following set:

```
$S^k = \{a_i^k \mid a_i^k$ is a network alarm received by intelligent agent k, i=1,...,n$\}$;
$B^k = \{\phi\}$;      // belief assessment structure.
$ES^k = \{\phi\}$;    // evidence structure set.
BEGIN
   WHILE $S^i \neq \phi$
         BEGIN
                  $e_i \leftarrow a_i \in S^k$;
                  $S^k \leftarrow S^k - a_i$;
                  construct $B^{ai}$ for $e_i$;
                  $B^k \leftarrow B^k + B^{ai}$;
         END
    WHILE $B^k \neq \phi$
          BEGIN
                  $es_i \leftarrow B^{ai} \in B^k$;
                  $B^k \leftarrow B^k - B^{ai}$;
                 construct $ES^{ai}$ for $es_i$ ;
                 $ES^k \leftarrow ES^{ai}$;
           END
$A^k_{comp} \leftarrow ES^{a1} \in ES^k$;
    IF $\mid ES^k \mid = 1$,
                   send $A^k_{comp}$ to the agent manager AM;
   ELSE IF
                  $ES^k \leftarrow ES^k - ES^{a1}$;
                        WHILE $ES^k \neq \phi$
                               BEGIN
                                   $e^k \leftarrow ES^{ai} \in ES^k$;
                                   $A^k_{comp} \leftarrow A^k_{comp} \otimes e^k$;
                                   $ES^k \leftarrow ES^k - ES^{ai}$
                               END
     END
send $A^k_{comp}$ to the agent manager AM;
END
```

Figure 6.6: Constructing the local composite alarm A^k_{comp} by an intelligent agent.

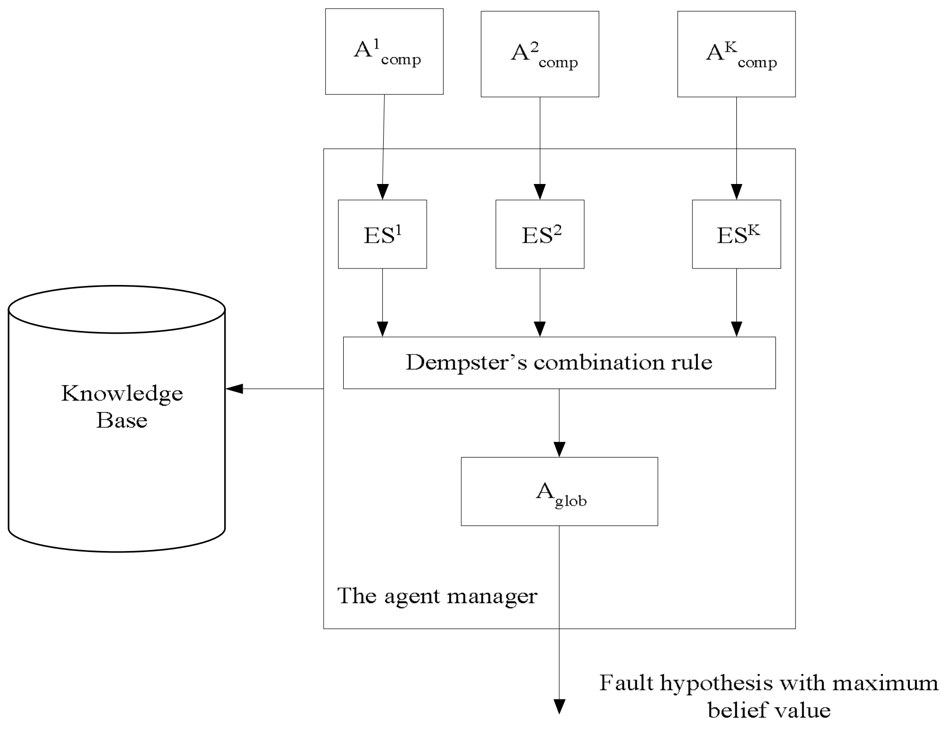


Figure 6.7: The agent manager structure.

$$ES = \{ES^1, ..., ES^K\} \tag{6.17}$$

where each ES^k is obtained by the same mechanism used by the intelligent agent, and $k = 1, ..., K$.

Considering the global alarm correlation as a fusion process, it can be defined as follows.

Definition 6.2 *Given a set of local composite alarms* $\{A^1_{comp}, A^2_{comp}, ..., A^K_{comp}\}$, *each obtained by the Equation (6.15), a global composite alarm can be formed by (1) constructing a set of evidence structures for the given local composite alarms that has the form given in (6.17), (2)fusing (correlating) the set of the obtained evidence structures, without regard to the arrival order of the local composite alarms into a single alarm*

A_{glob} *, i.e.,* $A_{glob} \leftarrow A^1_{comp} \otimes A^2_{comp}, \ldots \otimes A^K_{comp}$. *The symbol* $\otimes$ *refers to the Dempster's rule of combination. The fusion process is performed by the agent manager.*

Therefore, based on definition 6.2, the agent manager can correlate the received set of composite alarms by simply applying the Dempster's rule of combination given in Equation (3.7) to combine all ES^k for $k = 1, \ldots, K$. Due to the nature of the PDES scheme and the definition of commonality, the following formula can be used to calculate the commonality measure of the network failure f^k_j for the composite alarm A^k_{comp}:

$$Q(\{f^k_j\}) = \frac{Q(f^k_j)}{Q(f^k_1)} \tag{6.18}$$

To calculate the commonality measure of the network failure f^k_j for all the composite alarms, the agent manager can use the following equation:

$$Q(f_j) = T \prod_{k=1}^{K} Q(f^k_j) \tag{6.19}$$

where T is a normalization factor defined in Equation (3.8) and is independent of the network failure f^k_j of interest . To determine the best fault hypothesis, the agent manager utilizes the maximum commonality decision rule $(MCD = \arg\max_{f_j} Q(\{f_j\}))$.

Remarks:

- Remark 1: It has been shown in [50] that the DSET based on the proportional difference evidence structure is equivalent to the Bayesian approach, in terms of decision making. However, a belief value of the fault-alarm probability causality graph is determined by many factors such as alarm loss, system bugs, alarm delay, and so on. As such, belief values are often difficult to calculate and perfect probabilistic evidence may not be available. The PDES, however, can be used to incorporate available fuzzy evidences, as will be shown in the next chapter, to increase its reasoning capabilities.

- Remark 2: The commutative property of the Dempster's rule of combination makes the alarm correlation process event-driven operation. Primitive alarms will be processed as soon as they arrive to the intelligent agents. Composite alarms will be processed as soon as they arrive to the agent manager. The exact order of their arrival is irrelevant in the correlation process.
- Remark 3: It is quiet possible that a single primitive alarm may be generated multiple times by the same network entity and therefore all of them are observed by the same intelligent agent. These alarms have the same diagnostic information and differ only in their timestamps. From the viewpoint of the respective intelligent agent, these alarms have the exact same effect on the creation of the local composite event. Hence, to reduce to the calculation cost entailed by these identical alarms, a compression mechanism by which the reduction of multiple occurrences of an alarm into a single representative one can be performed before the local correlation process takes place. Other methods can also be adopted to reduce the number of insignificant alarms participating in the correlation process. For example, a filtering mechanism can be utilized such that if some parameters of an alarm e.g., priority, type, timestamp, severity, etc, do not fall into some predefined legitimate values, then the candidate alarm is simply discarded or sent to a log file. The advantage of employing such mechanisms may be realized in obtaining a better quality of local composite alarm and more efficient performance by the alarm correlation algorithm.

6.5 Summary

This chapter proposes a distributed alarm correlation system based on the Dempster-Shafer evidence theory. The managed network is divided into several disjoint management domains. Each management domain is assigned an intelligent agent that keeps a global fault propagation model. Using the given fault propagation model and belief updating scheme, the intelligent constructs an evidence structure for each primitive network alarm received from its constituents. Using the Dempster's rule of combination, the intelligent agent then correlates the obtained evidence structures into a local composite alarm and sends it to the agent manager. The agent manager, in turn, correlates these local composite alarms into a global composite alarm. Based on the global composite alarm, the agent manager then identifies the network failure. The effectiveness of the proposed algorithm is examined by extensive experiments and its results are reported in Chapter 8.

Chapter 7

Adaptive Fuzzy Alarm Correlation Algorithm

7.1 Introduction

The primitive alarm space within a management domain is divided into several exclusive clusters. These clusters, in turn, divide the fault hypothesis space into disjoint and exhaustive fault hypothesis sets. Alarms that share the same cluster can be explained by the same set of fault hypotheses (i.e., the cluster domain). As such, a cluster is viewed as an association relation between a set of primitive alarms and a set of fault hypotheses. Moreover, an observed cluster may respond differently to each fault occurrence in its domain. In case of network failure, only relevant alarms within the cluster will be reported. The reported alarms, however, may wrongly indicate other network failures in the cluster domain. The fact that some alarms pertaining to the wrongly indicated failures have not been observed should decrease our confidence in the occurrence of those failures. Hence, an observed cluster can provide fuzzy cues about its domain of fault hypotheses. In this chapter, a novel adaptive fuzzy alarm correlation algorithm is introduced. Using the same distributed model presented last chapter, the algorithm takes into account the absence of relevant primitive alarms of each fault hypothesis when the intelligent agent correlates the cluster alarms into a local fuzzy composite alarm. To deal with conflict that may occur among the intelligent agents, the agent manager utilizes a discounting mechanism in which the quality of information of each local fuzzy composite alarm is weighted and fused accordingly.

7.2 Definitions and Notations

The distributed model is the same as the one proposed in the last chapter. To help explain the proposed algorithm, we will use the simple fault propagation model depicted in Fig. 7.1. The given FPM is a bipartite graph in which the parentless nodes represent network failures and the children nodes represent primitive alarms. The strength of the causal relationship between the network failures and their alarms are described by conditional probabilities. To facilitate the development of the adaptive fuzzy alarm correlation approach, three new concepts along with their notations are introduced and discussed in this section, namely *domain of alarms*, *cluster of alarms*, and *domain of faults*,

7.2.1 Domains of Alarms

Each primitive alarm a_i, emitted by the network entity *i*, is characterized by its domain, referred to by $D(a_i)$. $D(a_i)$ is defined as the set of network faults that may cause the network alarm a_i to be triggered. The alarm domain can be obtained by examining the available fault propagation model. One of the methods proposed in the literature is to associate an alarm a_i with all the faults f_j which have a dependency weights $p(a_i \mid f_j) \geq W$ in the FPM; where W is a parameter [45]. For example, the following set of domains, *S*, are extracted from the fault propagation model shown in Fig. 7.1 ($W = 0.5$):

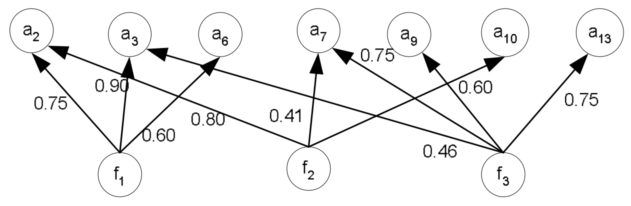


Figure 7.1: FPM of a simple network configuration.

$$S = \{\, D(a_2) = \{f_1, f_2\},\ D(a_3) = \{f_1\},\ D(a_6) = \{f_1\},\ D(a_7) = \{f_3\},$$
$$D(a_9) = \{f_3\},\ D(a_{10}) = \{f_2\},\ D(a_{13}) = \{f_3\} \,\}$$

Clearly, different values of W may create different associations of faults with a given alarm. Low value of W may lead to a higher association of an alarm with a set of fault hypotheses. In contrast, a higher value of W may lead to a lower association. This mechanism may be useful in the alarm correlation process. For instance, the correlation process can discard some primitive alarms as having low priority if their weights are below a predefined threshold. The domain of an alarm can be defined as follows.

Definition 7.1 *Let $G < V, P >$ be a distributed fault propagation model in the form of a bipartite graph, where V is a set of nodes. Let F, $F \subseteq V$ be the set of nodes at the tail of edges that represent network failures and A, $A \subseteq V$, be the set of nodes at the edge heads that represent primitive alarms. P is the associated conditional probability distribution that represents the influence of relationship between network failures and their corresponding alarms. Thus, the domain $D(a_i)$ for an alarm $a_i \in A$ can be defined as:*

$$D(a_i) = \{f_j \mid f_j \in F, \text{ there is an edge from } f_j \text{ to } a_i, \text{ and } p(a_i \mid f_j) \geq W\}, \qquad (7.1)$$

where W is a parameter.

Network failures that are members of a particular alarm's domain may also be members of other alarm domains. Alarm domains that share a singleton fault hypothesis set or more constitute a cluster of alarms as explained next.

7.2.2 Clusters of Alarms

A cluster of alarms is defined as the set of alarms whose domains intersect with each other. An alarm belongs to a cluster if its domain intersects with a domain of at least one alarm that belongs to this cluster. Different clusters share no intersection. For example, the alarm domains obtained by the FPM in Fig. 7.1 with $W = 0.5$, yield the following two clusters:

$$C_1 = \{a_2, a_3, a_6, a_{10}\}$$

$$C_2 = \{a_7, a_9, a_{13}\}$$

Again, the value of W can indirectly affect the cardinalities of each cluster. Based on the above discussion an alarm cluster can be defined as follows.

Definition 7.2 *Let S be the set of K alarm domains, $S = \{D(a_i) \mid a_i \in A\}$; and A and $D(a_i)$ are defined as in 7.1. Then, an alarm cluster, C_r, can be defined as:*

$$C_r = \{a_i \mid \bigcap_{i=1}^{K} D(a_i) \neq \phi, D(a_i) \in S\} \tag{7.2}$$

where r=1,2, ... The fault hypothesis set that can uniquely explain the cluster is called the cluster domain.

According to the definition 7.2, a cluster is just a collection of primitive alarms that share a set of fault hypotheses. This implies that the observed cluster can only be explained by this set. This observation leads to the following definition.

Definition 7.3 *Let C_r be a cluster obtained using definition 7.2. Let $\overline{F}_r$ be the set of fault hypotheses that can uniquely explain the observed cluster C_r and $\overline{F}_r \subseteq \Omega$. Then, the set $\overline{F}_r$ can be defined as:*

$$\overline{F}_r = \{\bigcup_{ai} D(a_i) \mid a_i \in C_r\} \qquad (7.3)$$

the set $\overline{F}_r$ *is called the domain of the cluster* C_r *.*

Since each cluster can only be explained by a distinct fault hypothesis set, the alarm correlation problem can be viewed as finding the best explanation (fault hypothesis) among the observed cluster domain. According to the definition 7.2, a cluster may be considered as an association relation among network alarms that share one or more elements of a certain set of fault hypotheses. Thus, an observed cluster can reduce the fault hypothesis space to that given by the cluster domain.

Definition 7.4 *Let* $C = \{C_1, ..., C_m\}$ *be a set of clusters obtained using definition 7.2 such that it divides the alarm space into disjunctive and exhaustive m clusters, i.e., :*

$$C_1 \cup ... \cup C_m = A$$
$$C_1 \cap ... \cap C_m = \phi$$

A is the alarm space. Then, the cluster domains divide the fault hypothesis space into disjunctive and exhaustive sets, i.e., :

$$\begin{aligned} \overline{F}_1 \cup ... \cup \overline{F}_m &= \Omega \\ \overline{F}_1 \cap ... \cap \overline{F}_m &= \phi \end{aligned} \qquad (7.4)$$

For instance, $\overline{F}_1 = \{f_1, f_2\}$ and $\overline{F}_2 = \{f_3\}$ explain the clusters C_1 and C_2, given above, respectively. A fault hypothesis $f_j \in \overline{F}_r$ can be considered as the best explanation of the observed cluster C_r if f_j is represented by most of the observed alarms in C_r. In the real world, most of the observed alarms are caused by the occurrence of a certain network failure that they directly or indirectly relate to.

Definition 7.5 *Network alarms that are members of the same cluster are called colleague alarms.*

Since colleague alarms are often caused by a common set of network failures, they should be correlated together to identify their most probable cause.

7.2.3 Domains of Network Faults

The set of alarms that may be observed as a result of the occurrence of a particular fault is referred to as the domain of that fault. A domain of a fault can also be obtained by investigating the available fault propagation model. For instance, given the FPM of Fig. 7.1, the domain of the fault f_1, referred to as $D(f_1)$, is the set of its children, $D(f_1) = \{a_2, a_3, a_6\}$. A particular alarm is a member of a fault's domain if the considered fault is itself a member of the alarm's domain. The FPM described in Fig. 7.1 clearly shows that an alarm can be a member of more that one fault domain. Formally, a fault domain is defined as follows.

Definition 7.6 *Let us assume that the set of domains of the available n observable alarms* $S = \{D(a_i) \mid a_i \in A\}$ *is obtained using definition 7.1. A given alarm* a_i *is considered as a member of the fault domain of* f_j *if and only if* f_j *is a member of the* a_i *domain. Hence, the domain* $D(f_j)$ *of the fault* f_j *can then be defined as:*

$$D(f_j) = \{a_i \mid D(a_i) \cap f_j \neq \phi, a_i \in A, f_j \in F, D(a_i) \in S\} \;, i = 1,\ldots,n. \tag{7.5}$$

An observed cluster, its domain, and fault domains will be utilized, as explained in later sections, in the new algorithm to determine a membership degree of a certain fault hypothesis in a given local fuzzy composite alarm.

7.2.4 Alarm Correlation Problem

The above definitions highlight very interesting properties regarding the alarm clusters and fault domains and their implications in the alarm correlation process. If the observed set of network alarms belong to the same cluster, then these alarms can be explained by the same set of fault hypotheses that their domains share. However, if the observed alarms belong to different clusters then a conflict may arise between clusters as which fault hypothesis set is the best explanation of the received alarms, since each cluster proposes its own exclusive explanation. Moreover, an observed alarm may in fact indicate a subset of fault hypotheses (i.e., its domain) as an explanation of its own observed cluster, which may be different from other colleagues. The diagnostic clue that can be gained from these facts is summarized as follows. While colleague alarms are highly expected to be observed together than with other non-colleague alarms, it is likely that some of these alarms may propose different fault hypotheses as an explanation of their cluster. In addition, if simultaneous occurrence of multiple network failures is permitted, then alarms belonging to different clusters may also be observed by the intelligent agents. Based on the above definitions, the alarm correlation problem can be formally defined as follows.

Definition 7.7 *Let C be a set of clusters, i.e.,* $C = \{C_i \mid C_i$ *is defined as in Def. (7.2), and* $i = 1,2,...\}$*,* $\overline{F}_i$ *be the domain of the cluster* C_i *and defined as in Def. (7.3), and the set* $\overline{A}$ *be an observed set of primitive alarms such that* $\overline{A} \subseteq A$*. Let* $\overline{C}_i$ *be a set of observed alarms such that* $\overline{C}_i \subseteq C_i$ *and* $\overline{C}_i \subseteq \overline{A}$ *.The fault hypothesis* f_{ji} *(i.e.,* $f_j \in \overline{F}_i$*) can be considered as the best explanation of* C_i *provided that:*

(i) $\mid D(f_{ji}) \cap \overline{C}_i \mid$ *is a maximum,*

(ii) its commonality value is a maximum (i.e., MCD= $Q(\{f_{j_i}\})$*.*

Based on the above definition, the issue is then of that the alarm correlation scheme needs to tackle are of twofold, namely:

- What impact may the absence of a colleague alarm a_i from the cluster observation set (i.e., $a_i \notin \overline{C}_i$) have on identifying f_{j_i} as an explanation of C_i? Where $a_i \in D(f_{ji})$,
- What impact may the set of the non-observed clusters C_K have on the fault

hypothesis space? (i.e., $C_K = \{C_k \mid C_k \cap \overline{A} = \phi\}$)

To address these two important issues, the adaptive alarm correlation scheme exploits the implied information of what is called positive alarms (non-observed alarms). Colleague alarms are often observed together (i.e., temporally related). Practically however, some of them may not be observed. As expressed in Def. (7.7), the total number of the observed alarms in the domain of a suspicious network failure may serve as a confidence measure that the failure has actually occurred. This valuable information can be incorporated in the local composite alarm built by the intelligent agent as described in 6.4.

A corresponding *local fuzzy composite alarm* can be constructed to account for the missing network alarms in a suspicious fault's domain. In this new composite alarm, the mass value assigned to every focal element can be modified according to the fuzziness provided by the observed cluster and the fault's domain. The observed cluster may regard its constituent alarms as a panel of experts. Regardless of their probabilistic information, the fact that a certain alarm in the given cluster, which is a member of a fault's domain, has been observed amounts to the corresponding expert decision of the positive occurrence of that fault. While the absence of the same alarm from the alarm observation set, amounts to the decision of the corresponding expert that the fault has not occurred. From this point of view, the more colleague alarms are being observed in a fault's domain, the more confident the relevant cluster should be in the occurrence of that fault. In contrast, the less colleague alarms are being observed in a fault's domain, the less confident the relevant cluster is in the occurrence of that fault. This concept can be very well represented by a local fuzzy composite alarm as will be explained next section.

7.3 Composite Alarm Representation

The alarm correlation process in a distributed system is performed at two levels, namely local and global levels. At the local level, a new fuzzy composite alarm is constructed from primitive alarms. At the global level, a new composite alarm is constructed from the locally-obtained fuzzy composite alarms. In the following sections, the representation of each new composite alarm is discussed.

7.3.1 Local Fuzzy Composite Alarm

A triggered primitive alarm is often an indication of multiple fault hypotheses as can be seen from its domain. If all the network failures in the alarm domain have the same conditional probability, then the alarm domain reflects a vagueness property of the associated alarm. It means that the triggered alarm is located in boundary areas between its domain members. Therefore, these hypotheses are not distinguishable clearly by the given alarm. Consider, for instance, the network alarm a_2 in Fig. 7.1. Its domain is given by the following compound set:

$$D(a_2) = \{f_1, f_2\}$$

The mere observation of the alarm a_2 only indicates the occurrence of the fault hypotheses f_1 or f_2. This vagueness property of a network alarm is transitive to its own cluster since the cluster is nothing but a collection of these alarms. Hence, an observed cluster in fact lies in the boundary of the union set of the domains of its alarms. Moreover, a certain fault hypothesis may not be uniquely explained by a single alarm as can also be seen from its domain. For instance, if we look at the domain of the fault hypothesis f_1, as one possible cause of the triggered alarm a_2, it yields the following compound set in the alarm space:

$$D(f_1) = \{a_2, a_3, a_6\}$$

Clearly, the alarm a_2 is not the only indication of the occurrence of the fault hypothesis f_1. Since the domain of the fault f_1 belongs to the cluster C_1, any missing alarm of $D(f_1)$ from the alarm observation set $\overline{A}$ should lower the confidence of the cluster C_1 in the occurrence of f_1.

One reasonable way to represent this information in the local composite alarm is to assign a certain weight for each fault hypothesis in the focal element, based on their observed alarms. However, the weights assigned to network failures can not be incorporated in the standard DSET. To accommodate for the vagueness property of an observed cluster, the crisp focal elements represented in the agent's local composite alarm are replaced with fuzzy sets. A corresponding fuzzy focal element can be expressed in a discrete form with finite elements as follows:

$$B = \left\{ \frac{\mu_B(f_1)}{f_1}, \ldots, \frac{\mu_B(f_i)}{f_i}, \ldots, \frac{\mu_B(f_{|B^0|})}{f_{|B^0|}} \right\}, \tag{7.6}$$

$\forall B, f_i$, $m(B) \neq 0$ and $\mu_B(f_i) \neq 0$. Where $B^0 = \{f_1, \ldots, f_{|B^0|}\}$, denoting the support set of fuzzy set B. B is associated with certain fault hypotheses in the frame of discernment Ω, i.e., $B \subseteq \Omega$. $B^0 \subseteq U$. U is the universe of discourse for the fuzzy focals, $f_i \in U$, and $\mu_B(f_i)$ denotes the membership grade of each element f_i in B; $|B^0|$ represent the cardinality of B^0. The information given by the observed cluster and the fault domains can be directly represented in fuzzy rules to determine an overall fuzzy set:

$$F = \left\{ \frac{\mu(f_1)}{f_1}, \ldots, \frac{\mu(f_c)}{f_c} \right\}. \tag{7.7}$$

Furthermore, the fuzzy focal corresponding to a crisp focal in the considered composite alarm is then determined by simply assigning membership values to the fault hypotheses of its crisp focal. For instance, the fuzzy evidence provided by Equation (7.7) is incorporated in the crisp evidence element $\{f_1, \ldots, f_j\}$ in the composite alarm as:

$$\left\{ \frac{\mu(f_1)}{f_1}, \ldots, \frac{\mu(f_j)}{f_j} \right\}$$

The intelligent agents keep the mass values originally assigned for crisp focal elements in their local composite alarms as mass values for their corresponding fuzzy focal in the new local fuzzy composite alarm. Thus, the corresponding fuzzy focal element of the crisp focal B is represented in the local fuzzy composite alarm as follows:

$$(B, m(B), \mu_B(f)) \tag{7.8}$$

To explain the roles played by both the alarm cluster and fault domains in determining the fuzzy focals, the scenario of the occurrence of f_3 in Fig. 7.1 is discussed. Let us assume that the network entities are assigned to report their primitive alarms to two intelligent agents as shown in Fig. 7.2.

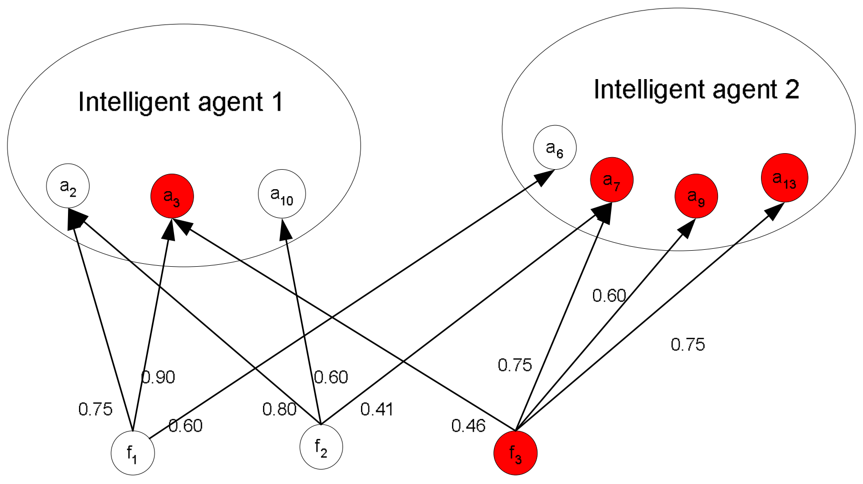


Figure 7.2: Simple network configuration divided into two domains

It can be noticed from Fig. 7.2 that the domain of the fault hypothesis f_3 has different constituents for each intelligent agent. The domain of the fault hypothesis f_3 is $\{a_3\}$ and $\{a_7, a_9, a_{13}\}$ to the intelligent agents 1 and 2, respectively. Obviously, in this case, more alarms are reported to the intelligent agent 2. This could be interpreted as that the network failure f_3 is probably caused by one of the network entities in domain 2. Furthermore, the intelligent agent 1 is clearly more sensitive to the fault hypothesis f_3. If the network alarm a_3 is not reported, it will not assume any role in the alarm correlation process and its local composite alarm will not be created. However, if a_3 is observed, it first checks the domain of a_3 and initially assumes that either the fault hypotheses f_1 or f_3 is responsible for triggering a_3. It then realizes that the colleague alarm a_2 (a member of the f_1 domain) has not been observed. This forces the agent to decrease its confidence in the occurrence of the fault hypothesis f_1. Thus, it will assign a full weight for f_3 and a 50% weight for f_1 in its local fuzzy composite alarm.

If the complete set $\{a_7, a_9, a_{13}\}$ is observed in domain 2, then the fault hypotheses f_2 and f_3 will be assigned full weight in the local fuzzy composite alarm. Though f_2 has not occurred, it is assigned full weight since a_7 is the only alarm in its domain. However, if none of the alarms in the set $\{a_{10}, a_{12}\}$ is observed in domain 1, the confidence in its occurrence will be greatly reduced by the agent 1. If a_7 has not been observed, then f_2 will be ruled out as an explanation of the current malfunctioning and f_3 will be assigned 66% weight in the local fuzzy composite alarm.

7.3.2 Global Composite Alarm

Once the local fuzzy composite alarms are obtained by the intelligent agents, they are combined into a global composite alarm by the agent manager. However, due to the fuzzy information embedded within the local composite alarms, they can not be directly correlated by the Dempster's rule of combination. Hence, the agent manager first transforms every fuzzy focal element into corresponding consonant crisp sets using the *resolution identity principal* [50]. During the decomposition process, each consonant crisp set is then assigned a certain amount of mass proportional to their element membership values. According to this scheme, the fuzzy focal B given in (7.8) can be represented with its $\alpha - cuts$. The mass $m(B)$ is then distributed among the produced crisp sets. The decomposition process is performed as follows [50]:

- Step 1: Decompose the fuzzy focal B into its $\alpha - cuts$ associated level-sets. The membership values of its elements are arranged in increasing order:

$$\mu_B(f_1) \geq \mu_B(f_2) \ldots \geq \mu_B(f_{|B^0|})$$

where $x_j \in B^0$ for $j = 1, \ldots, | B^0 |$.The $\alpha - cut$ of B at level $\mu_B(f_j)$ is represented by $B_{\alpha j} = \{f_1, \ldots, f_j\}$. Intuitively, each specifies a slice of the membership function. Therefore, it is conceivable that the original membership function can be reconstructed by adding these slices in order:

$$B = \alpha_1 x B_{\alpha_1} + \alpha_2 x B_{\alpha_2} + \ldots . + \alpha_j x B_{\alpha j}$$

where $\alpha_j x B_j$ represents a fuzzy set such as the one below:

$$\mu_{\alpha j} B_{\alpha j}(f) = \begin{cases} \mu_B(f_j), & if \quad f \in B_{\alpha j} \\ 0, & if \quad f \notin B_{\alpha j} \end{cases}$$

and + represents the disjunction operator (max operator).

- Step 2: Distribute the mass of fuzzy focal B into its $\alpha - cuts$:

$$m(B_{\alpha j}) = \frac{m(B)}{\max_x \mu_B(f)} \times (\mu_B(f_j) - \mu_B(f_{j+1})) \tag{7.9}$$

It can easily be seen from Equation (7.9) that a mass value for a derived crisp focal element is determined by two factors, namely, the original mass value and the varying degrees of memberships of its members to the fuzzy focal. By correlating all these crisp focals, which are equivalent to their corresponding fuzzy focals in local fuzzy composite alarms, the agent manager actually creates a *global fuzzy composite alarm*.

7.4 Adaptive Fuzzy Alarm Correlation Algorithm

The adaptive fuzzy alarm correlation algorithm is based on the distributed model presented last chapter. Though the distributed fault propagation model of the managed network is the same, each intelligent agent may actually have different alarm and fault domains as has been shown in section 7.3. This is so because each intelligent agent correlates the network failures only to the network alarms emitted from their constituent entities and remains unaware of any network alarms reported in other domains. As such, some intelligent agents are more sensitive to certain network failures than others. The cardinalities of the fault domains in a certain management domain reflect the degree of the sensitivity of its respective intelligent agent to these faults. Thus, to identify the network failure, the intelligent agents involving in the alarm correlation process should be given different weights.

7.4.1Intelligent Agent

The alarm space within each management domain is divided into exclusive and exhaustive clusters. An observed cluster is used by an intelligent agent to:

(1) reduce its fault hypothesis space,

(2) construct its local composite alarm,

(3) determine the fuzzy focals within its local composite alarm.

As shown in Fig. 7.3, based on the observed cluster $\overline{C}_{ik}$, the fault hypothesis space of the intelligent agent *k* is reduced to $\overline{F}_{ik}$. The same scheme presented in the last chapter is then utilized to construct the local composite alarm A^{k}_{comp}. The focal elements of the local composite alarm are then modified to accommodate for the fuzzy information provided by the observed cluster. These three steps are discussed in detail in the following sections.

7.4.1.1 Modifying the Fault Hypothesis Space

The partition of the alarm space based on their domains (as expressed in Def. (7.2)) results into exclusive alarm clusters. For an intelligent agent *k*, let A_k denote its alarm space, which is subset of the alarm set A provided by the FPM (i.e., $A_k \subseteq A$) and Ω_k be its fault hypothesis space (i.e., $\Omega_k \subseteq \Omega$). Based on the domains of alarms in A_k, the alarms are partitioned into *r* exhaustive and exclusive clusters such that:

$$\begin{aligned} C_{1k} \cup \ldots \cup C_{rk} &= A_k \\ C_{1k} \cap \ldots \cap C_{rk} &= \phi \end{aligned} \tag{7.10}$$

Based on Def (7.4), the fault hypothesis space of the intelligent agent *k* is also exhaustively and exclusively partitioned into:

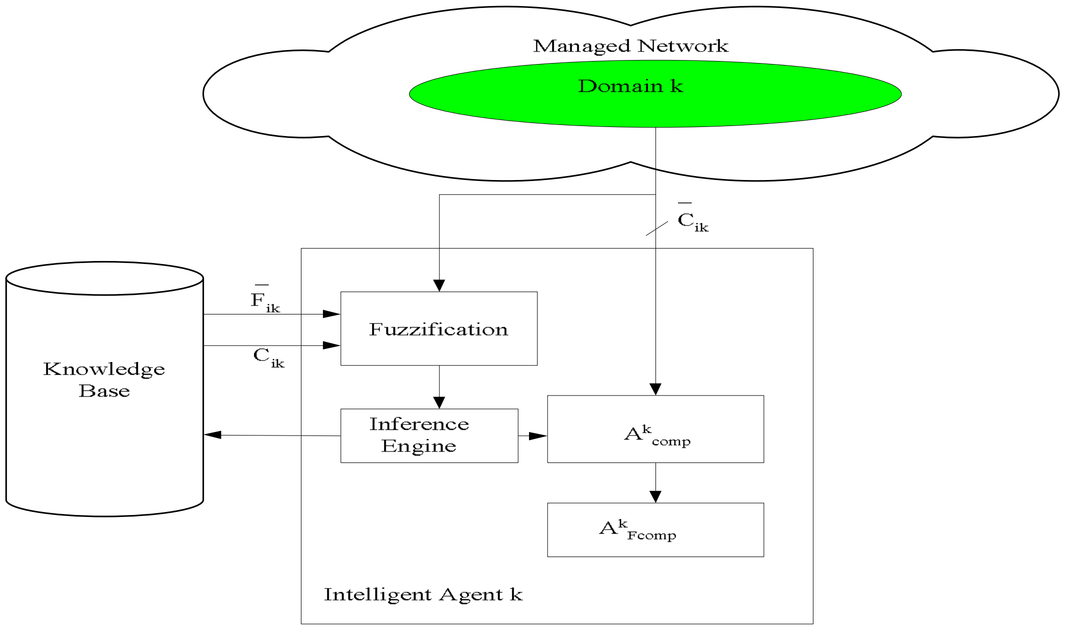


Figure 7.3: Intelligent agent structure.

$$\begin{aligned} \overline{F}_{1k} \cup \ldots \cup \overline{F}_{rk} &= \Omega_k \\ \overline{F}_{1k} \cap \ldots \cap \overline{F}_{rk} &= \phi \end{aligned} \tag{7.11}$$

An observed cluster C_{ik} can be explained by $\overline{F}_{ik}$, where $i = 1,\ldots,r$.

Let $\overline{A}_k$ be a set of alarms and $\overline{A}_k \subseteq A_k$, that has been observed by the intelligent agent *k*. If an alarm a_{lk} has been observed (i.e., $a_{lk} \in \overline{A}_k$) and a_{lk} is $a_{lk} \in C_{ik}$, then the cluster C_{ik} has been observed. Let $\overline{C}_{ik} = C_{ik} \cap \overline{A}_k$.

Alarms of the observed cluster that are members of $\overline{C}_{ik}$ will be called negative alarms. Alarms of the observed cluster that are members of the set $\{ C_{ik} - \overline{C}_{ik} \}$ are called positive alarms. Since the observed cluster C_{ik} can be explained by $\overline{F}_{ik}$, the fault hypothesis space of the intelligent agent *k* will be restricted to $\overline{F}_{ik}$ (i.e. $\Omega_k = \overline{F}_{ik}$). In other words, fault hypothesis sets of non-observed clusters are not considered in the alarm correlation process. The intelligent agent *k* analyzes both the negative and positive alarms of the observed cluster C_{ik} and assigns a weight for each fault hypothesis in $\overline{F}_{ik}$. Hence, a local fuzzy composite alarm can be defined as follows.

Definition 7.8 *Given a set of observed alarms* $\overline{C}_i$ of a given *cluster* C_{ik} *, an intelligent agent k may form a local fuzzy composite alarm* A^k_{Fcomp} *by (1) correlating* $\overline{C}_i$ *into a local composite alarm* A^k_{comp} *in the form given in Equation (6.15); (2) using* $\overline{C}_i$, C_{ik}*, and fuzzy rules to assign membership values for members of each crisp focal element in* A^k_{comp}.

Hence, an observed cluster C_{ik} provides probabilistic evidence, extracted from $\overline{C}_{ik}$, for each failure in $\overline{F}_{ik}$. Furthermore, fuzzy evidence can also be inferred from the observed cluster's negative and positive constituent alarms and the fault domains of its fault hypothesis set $\overline{F}_{ik}$.

7.4.1.2 Local Composite Alarm

Based on the observed alarms $\overline{C}_{ik}$ received from its constituents, the agent *k* takes the following three steps to form its local composite alarm [51]:

1. Calculating the belief assessments of each fault hypothesis in $\overline{F}_{ik}$ using the message updating algorithm (introduced in section 6.4). This step produces a belief vector for each network failure hypothesis in the form given in Equation (6.7).
2. Constructing an evidence structure set for $\overline{C}_{ik}$ in the form given in Equation (6.11).

3. Forming a local composite alarm as given in Equation (6.15), and is rewritten here for convenience:

$$A^{k}_{comp} = \{Q(f_{1ik}),\ldots,Q(f_{|\Omega k|ik})\} \tag{7.12}$$

7.4.1.3 Local Fuzzy Composite Alarm

The intelligent agent k may determine the degree for a particular fault hypothesis f^k_j ($f^k_j \in \overline{F}_{ik}$) to be a member of a given focal element based on how many of its constituent alarms in $D(f^k_j)$ have been observed (i.e., $|D(f^k_j) \cap \overline{C}_{ik}|$). However, the network failures of the same cluster share a considerable subset of their alarms among themselves. Upon careful examination of the fault domains, it can be easily noticed that a given failure with less than 25% observation ratio of its alarm domain should not be considered a strong explanation candidate. It is more likely that these alarms have been triggered by other network failures that happen to share these alarms with the given failure. A given failure is often considered a likely explanation for the observed cluster if at least 50% of its alarm domain has been received by its relevant intelligent agent. Moreover, a given failure with more than 75% observation ratio of its alarm domain is considered a more likely explanation of the observed cluster. Hence, considering each network failure as a fuzzy variable, three linguistic variables can be defined for each fuzzy variable to capture the previous empirical information as follows: "unlikely" (UL), "likely" (L), and "very likely" (VL). These fuzzy sets assume Z-curve, Pi-curve, and S-curve membership functions, respectively.

The membership function shapes of a given network failure are controlled by the observation ratio parameters. Let us assume that $\bar{f}^k_j$ represents the number of alarms in $D(f^k_j)$ that have been observed by its intelligent agent (i.e., $\bar{f}^k_j = |D(f^k_j) \cap \overline{C}_{ik}|$). The ratio parameters are determined as follows. $Low^k_{fj} = \frac{1}{4}|D(f^k_j)|$, $Mid^k_{fj} = \frac{1}{2}|D(f^k_j)|$, and $High_{fj}^{k} = \frac{3}{4} | D(f_j^k) |$. The VL membership function is defined as below:

$$\mu_{VL}(\bar{f}_j^k, a, b) = \begin{cases} 0; & \bar{f}_j^k < a-b \\ \dfrac{(\bar{f}_j^k - (a-b))^2}{2b^2}; & a-b \le \bar{f}_j^k \le a \\ 1 - \dfrac{((a+b) - \bar{f}_j^k)^2}{2b^2}; & a < \bar{f}_j^k \le a+b \\ 1; & \bar{f}_j^k > a+b \end{cases} \tag{7.13}$$

where $a = (High_{fj}^k + Mid_{fj}^k)/2$, $b = High_{fj}^k - a$.

The UL membership function is defined as follows:

$$\mu_{UL}(\bar{f}_j^k, a, b) = \begin{cases} 1; & \bar{f}_j^k < a-b \\ 1 - \dfrac{(\bar{f}_j^k - (a-b))^2}{2b^2}; & a-b < \bar{f}_j^k \le a \\ \dfrac{((a+b) - \bar{f}_j^k)^2}{2b^2}; & a < \bar{f}_j^k \le a+b \\ 0; & \bar{f}_j^k > a+b \end{cases} \tag{7.14}$$

where $a = (Low_{fj}^k + Mid_{fj}^k)/2$, $b = Med_{fj}^k - a$.

The L membership function is bell shaped and formed by placing both the VL and UL curves back-to-back. The expression is:

$$\mu_{L}(\bar{f}_j^k, a, b) = \begin{cases} \mu_{VL}\left(\bar{f}_j^k, a - \frac{b}{2}, \frac{b}{2}\right); & \bar{f}_j^k < a \\ \mu_{UL}\left(\bar{f}_j^k, a + \frac{b}{2}, \frac{b}{2}\right); & \bar{f}_j^k > a \\ 1; & \bar{f}_j^k = a \end{cases} \tag{7.15}$$

where $a = Mid_{fj}^k$, $b = High_{fj}^k - Med_{fj}^k$.

The overlap ratio (OR) between the UL and L membership functions is:

.

$$OR_{UL\_L} = \frac{Med_{fj}^{k} - Low_{fj}^{k}}{High_{fj}^{k}}$$

Let us assume that the cardinality of the fault domain of f_j^k is C (i.e., $|D(f_j^k)| = C$). Then, the overlap ratio (OR_{UL_L}) is:

$$OR_{UL\_L} = \frac{(1/2)C - (1/4)C}{(3/4)C} = \frac{1}{3}$$

Similarly, the overlap ratio between the L and VL membership functions is:

$$OR_{L\_VL} = \frac{High_{fj}^{k} - Med_{fj}^{k}}{|D(f_j^k)| - Low_{fj}^{k}}.$$

Hence, OR_{L_VL} is:

$$OR_{L\_VL} = \frac{(3/4)C - (1/2)C}{C - (1/4)C} = \frac{1}{3}$$

In both cases the overlap ratio is 33%. For example, if $|D(f_j^k)| = 20$, then the overlap between the membership functions of the linguistic variables for f_j^k is shown in Fig. 7.4.

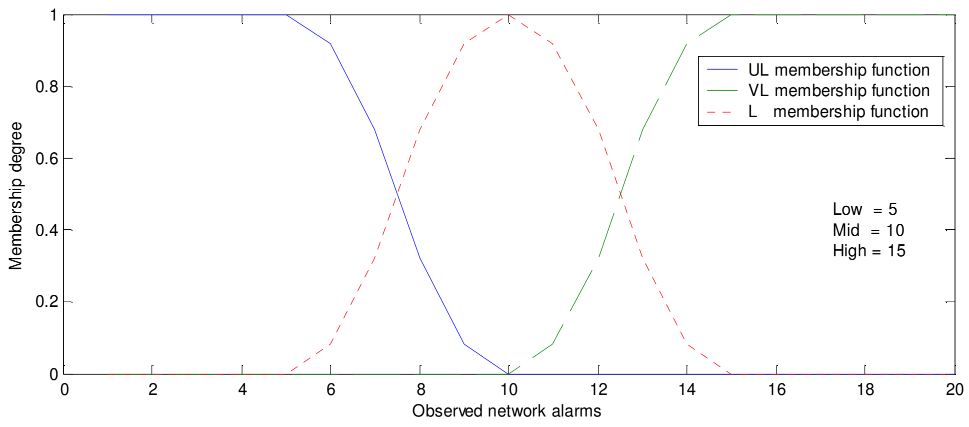


Figure 7.4: VL, L, UL Membership functions.

As shown if Fig. 7.4, if the number of alarms observed in the domain of f_j^k is less than five (Low_{fj}^k), then the fault hypothesis f_j^k is ruled out as an explanation for the observed alarm cluster. However, if 15 alarms or more of its domain are observed, then f_j^k is considered as a more likely explanation for the observed alarm cluster. Using our empirical knowledge, simple fuzzy rules can be obtained for each network failure as follows:

$$R_{j1}: \text{if } \bar{f}_j^k \text{ is VL THEN } F_j \text{ is } A_{j1}$$

$$R_{j2}: \text{if } \bar{f}_j^k \text{ is L THEN } F_j \text{ is } A_{j2}$$

$$R_{j3}: \text{if } \bar{f}_j^k \text{ is UL THEN } F_j \text{ is } A_{j3}$$

where j =1,…, $|\Omega_k|$.

The number of fuzzy rules for the whole managed network is linearly proportional to the cardinality of the frame of discernment. (i.e., $3\times|\Omega|$). The membership functions of the consequent fuzzy sets are defined for each network failure as follows:

$$\mu_{Aj_1} = \left\{ \frac{0.9}{F_1}, \ldots, \frac{0.9}{F_j}, \ldots, \frac{0.9}{F_{|\Omega|}} \right\}$$

$$\mu_{Aj_2} = \left\{ \frac{0.5}{F_1}, \ldots, \frac{0.5}{F_j}, \ldots, \frac{0.5}{F_{|\Omega|}} \right\}$$

$$\mu_{Aj_3} = \left\{ \frac{0.2}{F_1}, \ldots, \frac{0.2}{F_j}, \ldots, \frac{0.2}{F_{|\Omega|}} \right\}$$

A fuzzy inference process is then employed on all the obtained rules. The Mamdani inference scheme is implemented in the following manner:

1. Calculate the firing level for each rule:

$$\tau_{j1} = \mu_{VL}(\bar{f}_j^k)$$

$$\tau_{j2} = \mu_L(\bar{f}_j^k)$$

$$\tau_{j3} = \mu_{UL}(\bar{f}_j^k)$$

2. Calculate the output of each rule as follows:

$$\mu_{j1}(f_j) = \min(\tau_{j1}, \mu_{A_{j1}})$$

$$\mu_{j2}(f_j) = \min(\tau_{j2}, \mu_{A_{j2}})$$

$$\mu_{j3}(f_j) = \min(\tau_{j3}, \mu_{A_{j3}})$$

3. Aggregate individual rule outputs to obtain overall fuzzy set F_j with membership defined by:

$$\mu(f_j) = \max_{r=1}^{3}(\mu_{jr}(f_j))$$

4. Replace the given crisp focal with fuzzy set F_j.

The application of Mamdani inference scheme by the intelligent agent *k*, to every fault hypothesis contained in its fault hypothesis space Ω_k, results in one overall fuzzy set as follows:

$$\Omega_k = \left\{ \frac{\mu^k(f_1)}{f_1}, \ldots, \frac{\mu^k(f_{|\Omega k|})}{f_{|\Omega k|}} \right\} \tag{7.16}$$

The overall fuzzy set signifies the number of the constituent alarms in each fault domain that have been actually observed by the intelligent agent proportional to the total number of the alarms in their respective domains. Hence, membership degrees of different fault hypotheses in the fuzzy focal element vary according to the contributions of their constituents in the alarm correlation process. A fault hypothesis with higher number of its constituents participating in the alarm correlation process will be assigned a higher degree of membership compared with other fault hypothesis with less number of participating constituents. A full membership will only be given to those fault hypotheses whose all constituents are actually participating in the alarm correlation. On the other hand a fault hypothesis whose none of its constituents participate in the alarm correlation will be assigned a zero membership. This is consistent with Def. (7.7). It is very logical to assume that a certain fault hypothesis whose all or most of its symptoms being detected is more credible as a source of the malfunctioning than other fault hypotheses with less of their symptoms being observed. Hence, considering the proportionality a fault domain's constituents participating in the alarm correlation process as a measure of the fault belonging to a fuzzy focal is intuitive and conforms to the human thinking and reasoning convention. Thus, the local fuzzy composite alarm may then take the following form:

$$A^k_{Fcomp} = \left\{ B^k_j, \frac{bel(f^k_j) - bel(f^k_{j+1})}{bel(f^k_1)}, \mu_{B^k_j}(f_l) \mid j = 1,\ldots,|\Omega_k|, l = 1,\ldots, j \right\} \tag{7.17}$$

where B^k_j is a fuzzy set associated with an intelligent agent *k* and its membership is derived by Equation (7.16).

7.4.2 The Agent Manager

It is expected that local fuzzy composite alarms possess different discriminating capabilities in distinguishing distinct network failures. To effectively assess each local fuzzy composite alarm, the agent manager weights the probabilistic evidence against the fuzzy evidence provided by each composite alarm. This weighting process is embedded within the decomposition scheme described in section 7.3. As shown in Fig. 7.5, the decomposition scheme transforms each local fuzzy composite alarm into its equivalent crisp evidence structure. The resulting evidence structures are then combined using the Dempster's combination rule into a global composite alarm.

Let L be the set of local fuzzy composite alarms in the form of Equation (7.17).

$$\mathrm{L} = \{A^1_{Fcomp}, \ldots, A^K_{Fcomp}\}$$

The following definition is used by the agent manager to combine the received set of local fuzzy composite alarms.

Definition 7.9 *Given a set of local fuzzy composite alarms* $\{A^1_{comp},\ldots,A^K_{comp}\}$, *a global* *composite alarm is formed by (1) transforming each fuzzy focal element into a corresponding evidence structure set* $\{ES^1,...,ES^K\}$ *using the decomposition scheme of (7.9); (2) correlating the obtained evidence structures into a global composite alarm,* A_{glob} *without regard to their arrival order ,i.e.,* $A_{glob} \leftarrow ES^1 \otimes ... \otimes ES^K$*. The symbol* $\otimes$ *refers to the Dempster's rule of combination.*

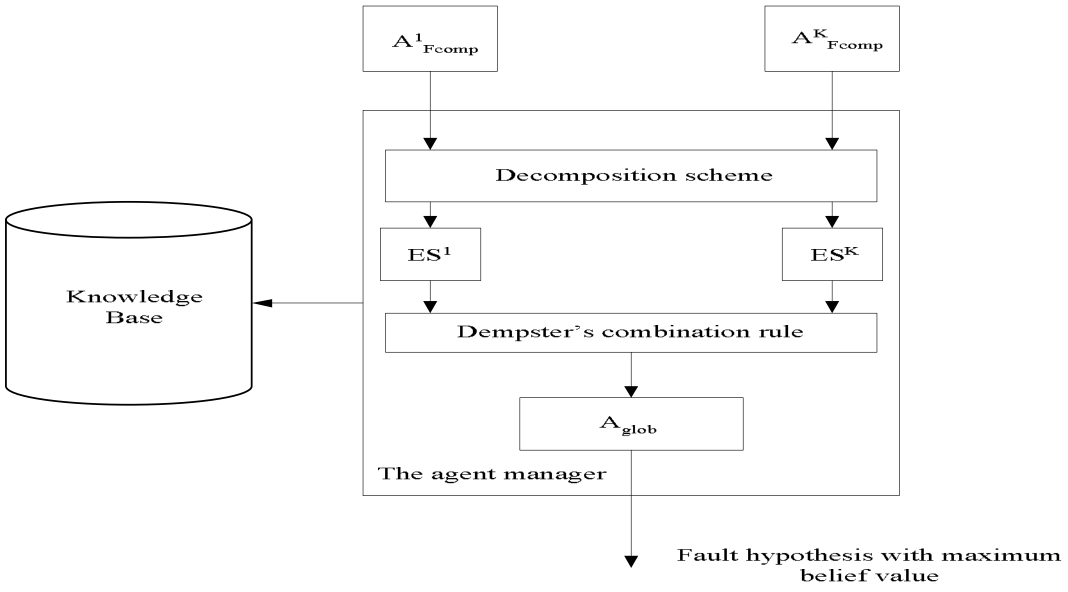


Fig. 7.5: The agent manager structure.

7.4.2.1 Intelligent Agent Discounting Factor

To account for their diagnostic capability, the agent manager calculates a discounting factor for each intelligent agent. Based on their factor, their composite alarm will be a given a certain weight during the global alarm correlation process. However, the probabilistic and fuzzy evidence is first evaluated within each local composite alarm as follows. The divergence between these two types of evidence can be evaluated using the

Kullback-Leibler (K-L) distance measure [51]. Given a belief assessment from an intelligent agent *k*,

$$B^k = \{bel(f_1^k),...,bel(f_{|\Omega k|}^k)\}$$

and its fuzzy set,

$$F^k = \left\{\frac{\mu^k(f_1)}{f_1},...,\frac{\mu^k(f_{|\Omega k|})}{f_{|\Omega k|}}\right\}$$

The terms $\mu^k(f_i)$ are first normalized, i.e., $\overline{\mu}^k(f_i) = \mu^k(f_i)/(\mu^k(f_1)+\mu^k(f_1)+\mu^k(f_1)$, $(i = 1,...,|\Omega_k|)$. The distance between B^k and F^k can be calculated using (K-L distance) as follows:

$$D(\overline{\mu}^k, B^k) = D(\overline{\mu}^k \| B^k) + (B^k \| \overline{\mu}^k)$$

the distance value can then be mapped into range [0, 1], using a unipolar sigmoidal function:

$$r(D(\overline{\mu}^k, B^k)) = \frac{1}{1+\exp(-P^k \times (D(\overline{\mu}^k, B^k) - D^{k0}))} \tag{7.18}$$

According to Def. (7.9), the agent manager should first transforms each fuzzy focal contained in A_{Fcomp}^k into a consonant crisp focal, where $A_{Fcomp}^k \in L$ and $k = 1,...,K$. P^k and D^{k0} are controlling parameters. The decomposition scheme as shown in (7.9) distributes the mass of each fuzzy focal into its $\alpha - cuts$. From the definition of commonality and the nature of the decomposition scheme, the commonality of a fault hypothesis within the corresponding crisp evidence resulted from the decomposition scheme can be defined as:

$$Q_j^k(\{f_l^k\}) = \frac{\mu_{F_j^l}(f_l^k)}{\max \mu_{F_j^l}} \times m(F_j^k)\delta(f_l^k \in F_j^{0k}), \tag{7.19}$$

According to (7.19), the commonality of each fault hypothesis is adjusted based on their memberships. The normalized commonalities in the established crisp evidence are defined by [51]:

$$P(\{f_j\}) = \frac{Q(\{f_j\})}{\sum_{f_j \in \Omega_k} Q(\{f_l\})}, \tag{7.20}$$

The normalized commonality is known as Bayesian probability. If all the network alarms of the fault hypothesis set have been observed (the corresponding elements in the local fuzzy composite event are crisp sets with full memberships), the normalized commonality of each fault hypothesis is identical to its belief value. Hence, the ratio $\mu_{F_j^l}(f_l^k)/\max \mu_{F_j^l}$ is used to adjust the resultant commonality. We adopt the discounting scheme proposed in [50], to guide the alarm correlation process. To determine the masses for the resultant crisp focal element, we substitute the membership function given in (7.19) with the following one:

$$\overline{\mu}_{F_j^l}(f_l^k) = r(D(\overline{\mu}^k, B_k) \times (\mu_{F_j^l}(f_l^k) - \min \mu_j^k) + \min \mu_j^k \tag{7.21}$$

The function $r(D(\overline{\mu}^k, B_k))$ measures the distance between the probabilistic evidence provided by an alarm cluster calculated as a belief assessment from the FPM and its fuzzy evidence which reflects the proportionality of each fault hypothesis in terms of their domain alarms that have been actually observed by the intelligent agent. The agent manager may discard the fuzzy evidence provided by an intelligent agent *k* if all memberships in the agent's local fuzzy composite alarm are equal (i.e., $r(D(\overline{\mu}^k, B_k)) = 0$. This indicates that all the alarms in the observed cluster have been actually observed by the intelligent agent in question. If $r(D(\overline{\mu}^k, B_k)) = 1$, network alarms in a certain cluster have not been reported and as such they are considered as positive alarms. Hence, the commonality of each fault hypothesis is adjusted according to what degree these fault hypotheses differ from each other in their membership to a fuzzy focal element. Therefore, the agent manager can calculate the commonality of a fault hypothesis associated with a local fuzzy composite alarm:

$$A_{Fcomp}^k = \left\{ B_j^k, \frac{bel(f_j^k) - bel(f_{j+1}^k)}{bel(f_1^k)}, \mu_{B_j^k}(f_l) \mid j = 1, \ldots, |\Omega_k|, l = 1, \ldots, j \right\}$$

provided by an intelligent agent k as follows:

$$Q_j^k(\{f_l^k\}) = \frac{\overline{\mu}_{F_j^l}(f_l^k)}{\max \overline{\mu}_{F_j^l}} \times m(B_j^k)\delta(f_l^k \in B_j^{0k}), \tag{7.22}$$

$\overline{\mu}_{F_j^l}(f_l^k)$ is defined in Equation (7.21).

As stated in Def. (7.9), the agent manager uses the Dempster's rule of combination to correlate the local fuzzy composite alarms presented in (7.17). However, due to their different discriminating capabilities, they should be treated according to their importance to the alarm correlation. To evaluate the overall uncertainty contained in a local fuzzy composite alarm, first the agent manager measures its fuzzy entropy.

Let $A_{Fcomp}^k = \left\{B_1^k, m_1, \mu_{B_1^k}(f), \ldots, B_{|\Omega_k|}^k, m_{|\Omega_k|}, \mu_{B_{|\Omega_k|}^k}(f)\right)\}$ be local fuzzy composite alarm and $\tilde{F}_{Fcomp}^k$ denote its fuzzy entropies $\tilde{F}_{Fcomp}^k = \{\tilde{F}_1^k, \ldots, \tilde{F}_{|\Omega_k|}^k\}$. The fuzzy entropy is defined as [51]:

$$\tilde{F}_j^k = \frac{1}{|B_j^0|}\sum_{l=1}^{|B_j^0|}\frac{\mu_{B_j} \cap \overline{\mu}_{B_j}(f_{jl})}{\mu_{B_j} \cup \overline{\mu}_{B_j}(f_{jl})}, \tag{7.23}$$

$\tilde{F}_j^k$ is *minimum* if and only if B_j^k is a crisp set (i.e., $\mu_{B_j}(f_{jl}) = 0$ or 1) and is $\tilde{F}_j^k$ the *maximum* if and only if B_j^k is the most fuzzy set (i.e., $\mu_{B_j}(f_{jl}) = 0.5$). Since $\tilde{F}_j^k$ can take values between 0 and 1 ($0 \le \tilde{F}_j^k \le 1$), the smaller is $\tilde{F}_j^k$, the less fuzzy is the fuzzy set B_j^k.

The overall uncertainty can then be measured using the following:

$$FH(A_{Fcomp}^k) = -\sum_{j=1}^{\Omega_k} m_j \log_2(m_j(1-\tilde{F}_j)). \tag{7.24}$$

$FH(A_{Fcomp}^k)$ is *minimum* if m_j =0 or 1, and B_j^k is a crisp set, (i.e., $\mu_{B_j}(f_{jl}) = 0$ or 1) and $FH(A_{Fcomp}^k)$ is *maximum* if $m_j = \frac{1}{|\Omega|}$ and B_j^k is the most fuzzy set (i.e., $\mu_{B_j}(f_{jl}) = 0.5$).

An intelligent agent that is less uncertain in both probabilistic and fuzzy evidence would have a smaller value of hybrid entropy. Based on the discounting scheme introduced by the DSET, the agent manager uses a discounting factor α_k for each intelligent agent k to capture the probabilistic and fuzzy entropies of all intelligent agents:

$$\alpha_k = \eta_k (H(A^1_{comp}),...,H(A^K_{comp}),(FH(A^1_{Fcomp}),...,FH(A^K_{Fcomp})) \tag{7.25}$$

The Shannon entropy, $H(A^K_{comp})$, is employed to quantify the uncertainty contained in the probabilistic evidence provided by the local composite alarm of intelligent agent *k*:

$$H(A^K_{comp}) = \sum_{f_j \in \Omega_k} -bel(f_j)\log_2 - bel(f_j), \tag{7.26}$$

$FH(A^k_{Fcomp})$ is the hybrid entropy defined in Equation (7.24). The smaller is $H(A^k_{comp})$, the_more certain the intelligent agent is in terms of its probabilistic evidence A^k_{comp}; and the less discounted is its evidence, the smaller is $FH(A^k_{Fcomp})$, and the more certain the intelligent agent is in terms of its fuzzy evidence contained in its local fuzzy composite alarm A^k_{Fcomp}.

7.4.2.2 Correlating Local Fuzzy Composite Alarms

Given the above discounting factor defined in Equation (7.25) and the new fuzzy membership defined in (7.21), the agent manager can now calculate the common commonalities for each intelligent agent using the final commonality defined as:

$$Q^k(\{f^k_l\}) = \alpha_k + \sum_{j=1}^{|\Omega|} \frac{\overline{\mu}_{F^l_j}(f^k_l)}{\max \overline{\mu}_{F^l_j}} \times (1-\alpha_k) m(B^k_j) \delta(f^k_l \in B^{0k}_j) \tag{7.27}$$

To combine all local fuzzy composite alarms, the agent manager applies the Dempster's rule which yields the final commonality [51]:

$$Q(\{F_l\}) = T \sum_{k=1}^{K} Q^k(\{F_l\}). \tag{7.28}$$

The agent manager decides the fault hypothesis with the maximum value as the root cause of the abnormality of the running network.

7.5. Summary

In this chapter, we developed an adaptive fuzzy evidential approach for the alarm correlation problem in computer networks. In the proposed approach, an intelligent agent takes advantage of the positive symptoms to construct a local fuzzy composite alarm. The commonality value of each fault hypothesis proposed by these composite alarms are adjusted according to the relationship between the probabilistic and fuzzy evidence within their relevant composite alarm. To deal with conflict among intelligent agents participating in the alarm correlation process, the agent manager adjusts these commonalities even further based on some discounting factor. Finally, the Shannon entropy and the hybrid entropy were used to calculate discounting factors on competing hypotheses.

Chapter 8

Simulation and Experiment Results

8.1 Introduction

In this dissertation we have proposed four schemes for fault probing and identification: CSP-based scheme for probe selection (introduced in Chapter 4); Fuzzy CSP-based scheme for fault identification (introduced in Chapter 5); Distributed Alarm Correlation scheme (introduced in Chapter 6); and Adaptive Fuzzy Alarm Correlation approach in scheme (introduced in Chapter 7). To demonstrate the effectiveness of these schemes, extensive experiments have been carried out and their results are reported in this chapter. For all experiments we have adopted and modified the simulation model proposed in [18] and implemented it using the C++ programming language under object-oriented environment.

8.2 CSP-Based Algorithm for Optimal Selection of Probes

In this section we investigate the effectiveness of the CSP-based algorithm as a viable approach for the preplanning of probing based schemes. A multi-scenario simulation model is used to conduct this investigation. The subtractive search and the greedy algorithms [2,5] are considered state-of-the-art and hence used for comparison. The investigation is performed in two parts: In the first part, we ran experiments for each of the three algorithms to find minimal solutions, i.e, determining an optimal number of testing probes for each algorithm. In the second part, we study the effects of having different number of probing stations deployed in a managed network for the solution set.

8.2.1 Simulation Model

The flow control of the simulation program is shown in Fig. 8.1. First we determine the size of the managed network. The network size is assumed, without loss of generality, to be in the range of 5-50 nodes. This could be extended to accommodate larger set of nodes. After determining the network size, the simulation program generates a random network topology. Both strong network and loose connectivity are contemplated. Strong connectivity implies that the network is fully connected, i.e, each node in the network is connected to every other node. The spanning tree routing algorithm is performed on the generated network to create a tree-shape topology model and to eliminate any cyclic paths. To obtain the dependency matrix of the network, a number of probing stations is determined. The probing stations are then randomly deployed in the network. The number of probes for each dependency matrix depends on the number of probing stations and network size. For example, given a network size of five nodes, the simulation model may produce a network configuration as shown in Fig. 8.2. To simplify the process of determining the testing probe paths, a spanning tree of the network is computed. This step may yield the routing configuration shown in Fig. 8.3. Using this routing information, a probing station can easily obtain all the paths required by its probes to examine the managed network.

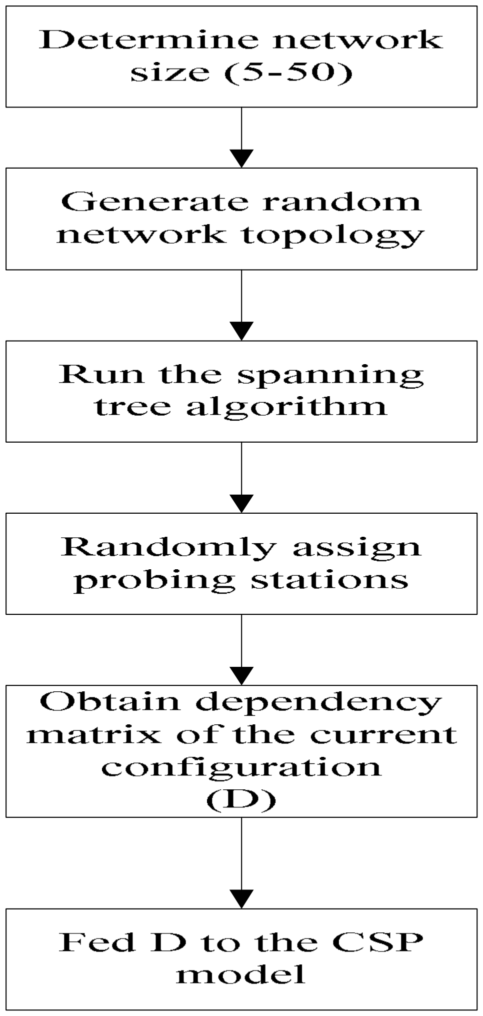

Determine network size (5-50)
Generate random network topology
Run the spanning tree algorithm
Randomly assign probing stations
Obtain dependency matrix of the current configuration (D)
Fed D to the CSP model

Figure 8.1: Obtaining a dependency matrix.

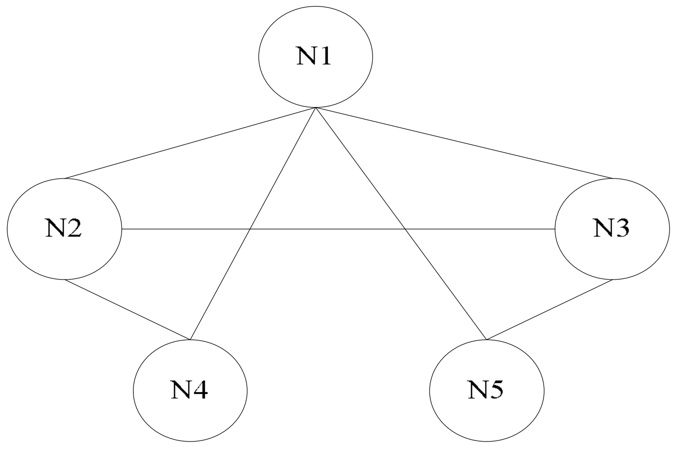


Figure 8.2: A simple network configuration of size 5.

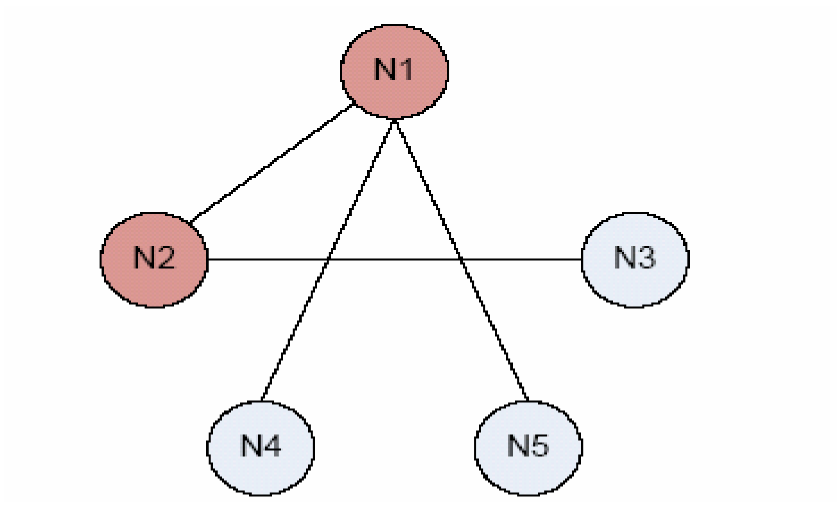


Figure 8.3: The resultant of spanning tree algorithm run on the network of Fig. 8.2.

Let us assume that the simulation model assigns nodes 1 and 2 as probing stations (both depicted in red color in Fig. 8.3). The dependency matrix *(D)* extracted from the network configuration of Fig. 8.3 is summarized as follows:

$$D = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The first four probes (rows) are issued from the first probing station; the remaining probes are issued from the second probing station. This dependency matrix is fed to the CSP-based algorithm as discussed in Chapter 4.

In the following section, we test the CSP scheme on different network configurations of different sizes. Since the objective is to reduce the size of the given dependency matrix while maintaining its diagnostic power, the performance measure used to compare the proposed CSP scheme with the subtractive search and greedy search schemes is chosen as finding an optimal probe set (called the solution set) that meets all the requirements discussed in Chapter 4.

8.2.2 Initial Variables

The proposed algorithm starts with an initial set of variables (called the active variables). The cardinality *(L)* of the initial set is determined by Equation (4.3), restated here for convenience:

$$L \geq \log(N)/\log(2); \qquad L \in Z$$

Where N is the network size and L is the number of initial active variables. The network size is varied between 5 and 50, which yields the initial active variables as shown in Table 8.1. At least three probes are needed for a network size of 5, and six probes or more are needed for a network size of 50, to successfully perform the fault detection and identification tasks. In order to evaluate the efficiency of the proposed model, we ran the simulation 10 times for each network size and obtained the average number of selected probes in each run.

Table 8.1: Initial set of active variables for different network sizes.

N	5	10	15	20	25	30	35	40	45	50
L	3	4	4	5	5	5	6	6	6	6

8.2.3 Optimal Probes for Different Network Sizes

In this section, the CSP-based model is tested for different network configurations. Again, the performance of the CSP-based model is compared with that of the greedy and subtractive search algorithms. The results are shown in Fig. 8.4. It is evident from the results that the proposed CSP algorithm outperforms both algorithms. It always produces the optimal number of probes. The subtractive search algorithm performs the worst. The subtractive and greedy approaches utilize the conditional entropy as a measure of diagnostic power of a given set of probes. The diagnostic ability *H (P)* of a set of probes *P* is defined as follows [5]:

$$H(P) = H(N \mid G)$$

Where $N = \{1,...,n\}$ denotes the node set, and $G = \{1,...,k\}$ denotes which group contains the node in the decomposition induced by *P*. Failures in nodes in the same group can not be distinguished by the probe *P*. If n_i is the number of nodes in group g_i, then:

$$H(P) = \sum_{i=1}^{k} p(G = g_i) H(N \mid G = g_i)$$

Assuming failures are equally likely in any node, the diagnostic ability is reduced to the following:

$$H(P) = \sum_{i=1}^{k} \frac{n_i}{n} \log(n_i)$$

For example:

$$H(P_1) = H(\{\{1,2\},\{3,4\}\})$$

That is adding P1 to the selected probes may have the following diagnostic power, based on the groups induced by P_1 :

$$H(P_1) = \frac{2}{4}\log 2 + \frac{2}{4}\log 2 = \log 2 = 1$$

whereas

$$H(P_2) = H(\{\{1\},\{2,3,4\}\})$$

yields the following diagnostic power:

$$H(P_2) = \frac{1}{4}\log 1 + \frac{3}{4}\log 3 = \frac{3}{4}\log 3 = 1.19$$

H (P) is simply the expected minimal number of probes needed to uniquely diagnose all nodes. A selected probe, *P*, that reduces the value of the *H (P)* of the current probe set is a better choice; in this example P_1 is better than P_2.

The subtractive approach starts with the given dependency matrix and considers each probe in turn. If the diagnostic ability remains the same after dropping the given probe from the dependency matrix, then the probe is discarded. The algorithm finds a subset of probes that has the same diagnostic power as that of the original dependency matrix, though the found subset is not minimal. The implication of this search paradigm is that its effectiveness depends highly on the order of the dependency matrix. In a rather different direction, the greedy approach starts with an empty set and adds a probe to this initial probe set and calculates the diagnostic power of the solution set. If the selected probe increases the diagnostic power of the initial probe set it keeps the selected probe in the solution set; otherwise the probe is discarded. Again the greedy approach is also very sensitive to the ordering of the probes in the dependency matrix. An early selected probe may initially seem a valuable contributor to the solution set, however, at later stages it

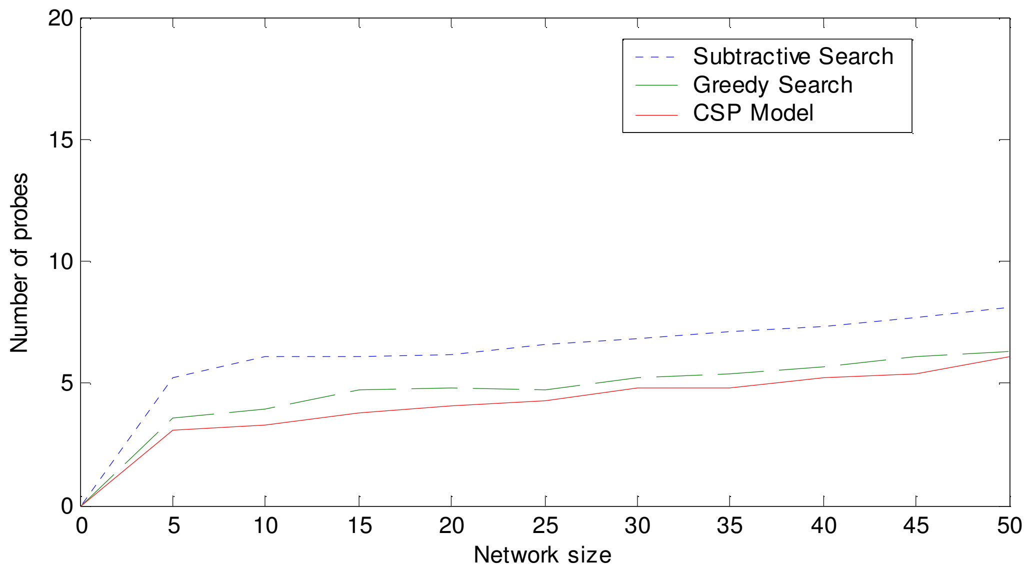


Figure 8.4: Comparisons of the CSP-based model with the greedy and subtractive algorithms.

can prevent more valuable probes from being selected. Thus both algorithms fail in finding an optimal probe set. In contrast, while it may prolong the search process, the ordering of the given dependency matrix has no effect on the final outcome of the proposed algorithm.

8.2.4 Varying Number of Probing Stations

The effects of having different number of probing stations on the number of the selected probes are investigated in this section. In real life networks, the process of determining a certain number of probing stations as well as their locations in the managed network is subject to administrative and economic considerations that are not part of our study. Thus, in this work we do not address the question of how to select the probing stations. Nonetheless, a preliminary study of their effects may help network administrators in that process. We, however, restrict our experiments on networks with number of probing stations range from 1 to 3. The results are shown in Fig. 8.5 and table 8.2.

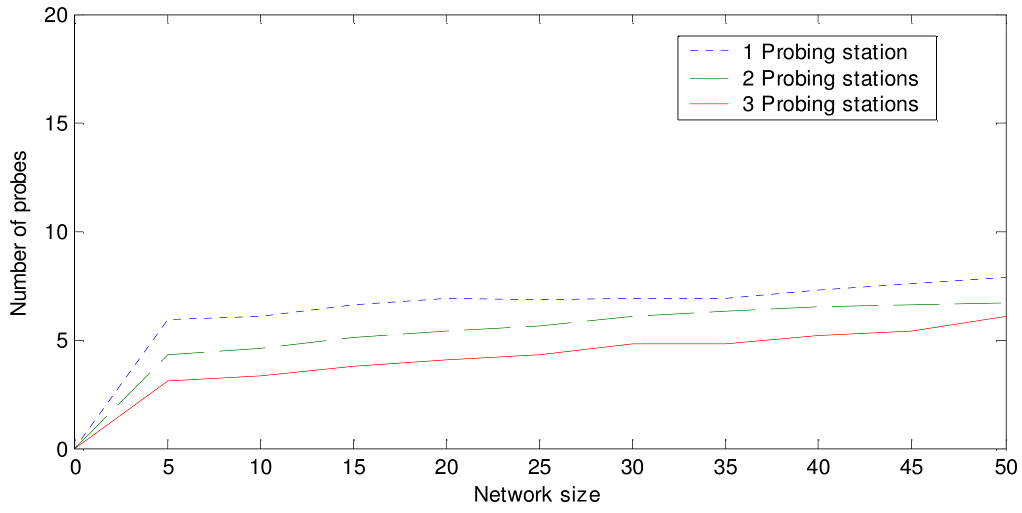


Figure 8.5: Number of probes obtained by the CSP model for different probing stations.

We can see from the results reported in Table 8.2, that as the number of probing stations increases an optimal probe set with even less probes can be found by the CSP model. Networks with higher number of probing stations provide more alternative choices and routes for testing a subset of network nodes. Hence, a small comprehensive set of probes can be easily found from a rich dependency matrix. This leads us to recognize that the locations and number of probing stations may yield different results for each configuration, however, given a certain configuration the proposed scheme will always produce better results if not comparable to those of the other two approaches.

Table 8.2: Average number of probes for different probing stations.

Network size	5	10	15	20	25	30	35	40	45	50
One Probing Stations	5.9	6.1	6.6	6.9	6.8	6.9	6.9	7.3	7.6	7.9
Two Probing Stations	4.3	4.6	5.1	5.4	5.6	6.1	6.3	6.5	6.6	6.7
Three Probing Station	3.1	3.3	3.8	4.1	4.3	4.8	4.8	5.2	5.4	6.1

8.3 FCSP-Based Probing Algorithm

In this section, the effectiveness of the adaptive fuzzy CSP-based algorithm is examined. As has been pointed out in Chapter 5, the main shortcoming of the preplanned probe schemes is that once an optimal set of probes is found the network management system should periodically send all the probes in the optimal set for both fault detection and identification tasks. The new algorithm avoids this overhead problem by considering only relevant probes. A simulation model is first developed and the proposed fuzzy CSP algorithm is tested on different scenarios.

8.3.1 Simulation Model

The simulation model presented in Fig. 8.1 is used to obtain the dependency matrix *(D)* which in turn is manipulated as shown in Fig. 8.6. The greedy algorithm is first performed on the obtained dependency matrix for each network configuration to extract a minimal subset of probes for the purpose of fault detection. We randomly then introduce a failure in the current network configuration and modify the probe, suspect, and healthy sets as illustrated in Chapter 5. We run the adaptive fuzzy CSP algorithm to select the most informative probe among the available probes of *D* based on the problem constraints highlighted in section 5.3. Every time a probe is selected a probe counter is incremented by 1. We repeat the same process until the root cause of the failure is isolated. The average number of the selected probes is then measured for each network configuration. We run the simulation 10 times for each network size. To compare the results obtained by the new algorithm, we have also implemented the Greedy Fault Localization (GFL) algorithm proposed in [3, 4]. The GFL uses two search methods to select a candidate probe from the dependency matrix, namely Max and Min search methods. The Max search approach selects a probe that covers maximum number of suspected nodes, while the Min search approach selects a probe with a minimum number of suspected nodes.

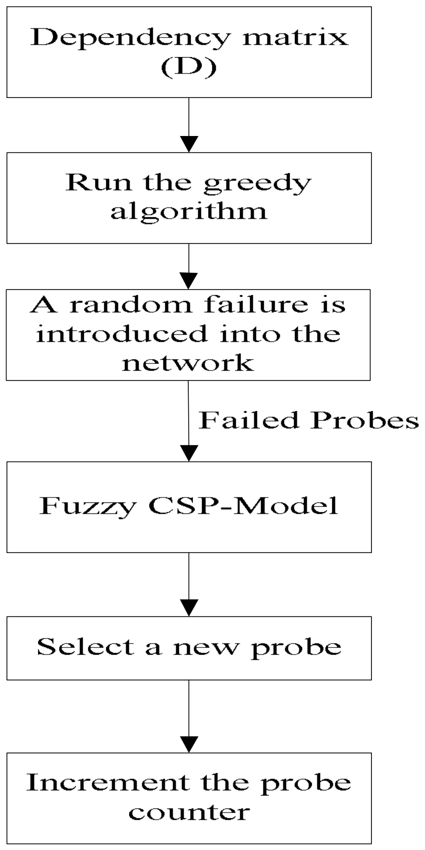


Figure 8.6: The simulation model.

8.3.2 Results by the Fuzzy CSP Algorithm

The results of the experimentations are shown in Fig. 8.7. The proposed algorithm always presents solutions with less number of probes than the other two algorithms. Though the Min search algorithm is a little closer to the fuzzy CSP algorithm, the Max search is obviously performing worse than both algorithms. As shown in Table 8.3, for networks of size 5 and 50, the average number of probes obtained by the new fuzzy CSP algorithm is around 3 and 10.2, respectively. On the other hand, the Max algorithm obtains on average 6.5 and 15.2 probes for the same network sizes. This deficiency can be traced back to the search mechanism adopted by the Max algorithm. Since the Max approach selects a probe with a maximum number of suspected nodes, a failed probe may result in enlarging the suspect node set significantly and thus prolongs the search process. Therefore, more probes are required to isolate the root cause of the problem.

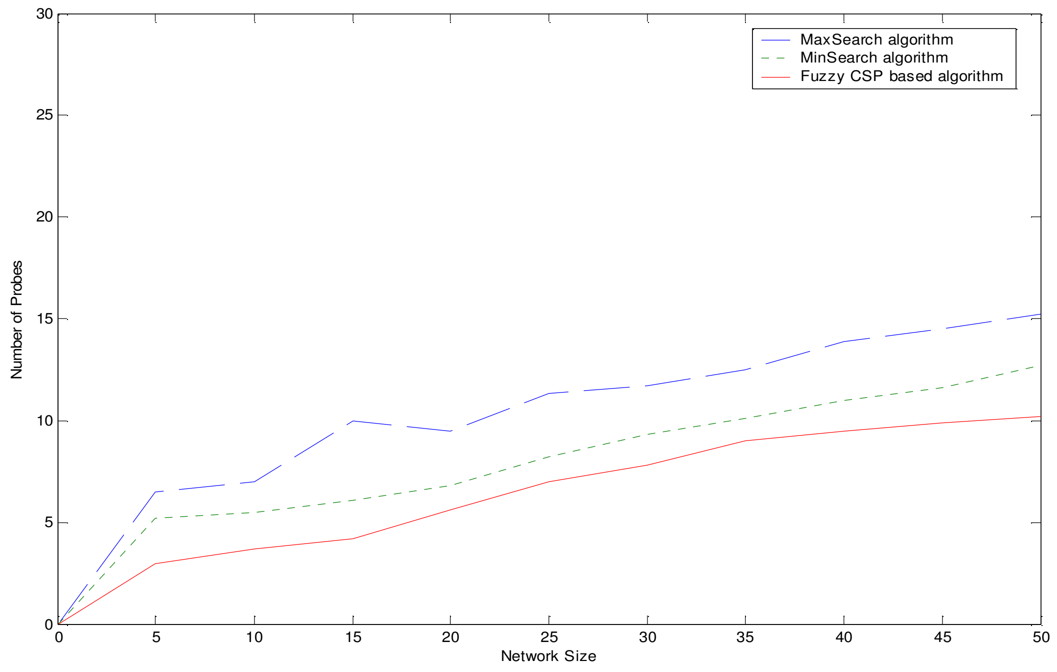


Figure 8.7: Comparisons of the fuzzy CSP-based, Min, and Max search algorithms.

Table 8.3: Average number of probes required by each scheme.

Network size	5	10	15	20	25	30	35	40	45	50
Min Search	5.2	5.5	6.1	6.8	8.2	9.3	10.1	11	11.6	12.7
Max Search	6.5	7	10	9.8	11.3	11.7	12.5	13.9	14.5	15.2
Fuzzy CSP-based	3	3.7	4.2	5.6	7	7.8	9	9.5	9.9	10.2

If we treat the received alarms as candidate probes, the performance of the proposed algorithm yields even lower number of testing probes than that of the Min search algorithm . As shown in Fig 8.8, the number of probes required for the fault identification task for networks of size 50 is reduced to approximately six probes, in contrast to 12 probes produced by the Min search algorithm. It cuts down the obtained probes by over 50%. This significant 50% improvement over the Min search algorithm can be achieved by slightly modifying the fuzzy CSP algorithm to accommodate for the already triggered alarms (no extra traffic is induced here). Since each received network alarm has to go through a set of nodes to reach its management node, these nodes can be considered by the fuzzy CSP algorithm as health nodes.

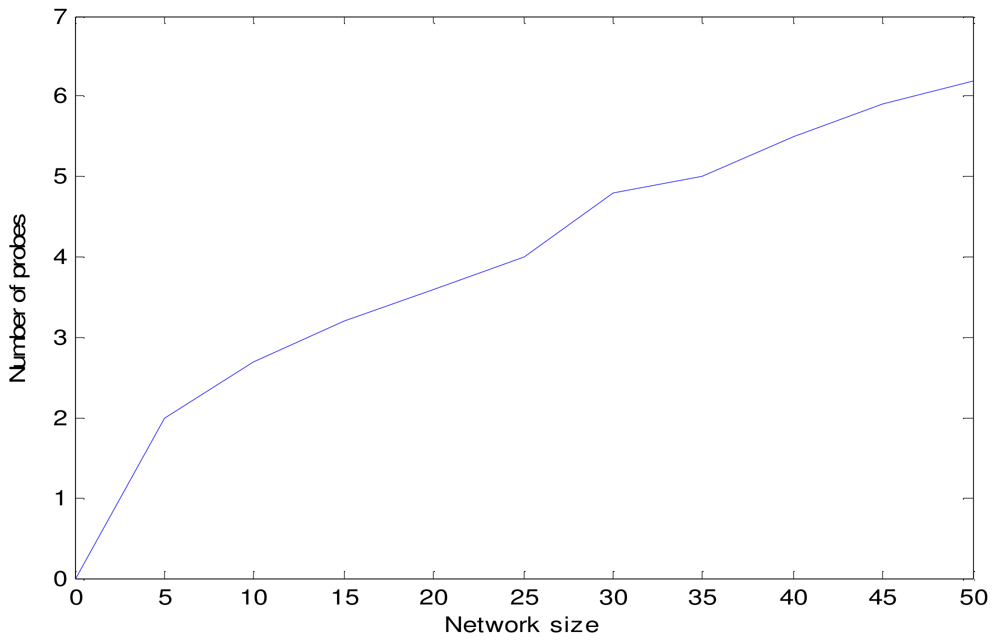


Figure 8.8: The average number of probes obtained by the modified fuzzy CSP algorithm

8.4 Distributed Alarm Correlation and Fault Identification Algorithms

In this section we will examine the efficiency of the proposed alarm-based algorithms. The proposed algorithms are based on the assumption that a single global fault propagation model of the managed network is available and distributed among the domain agents. While this assumption may not be realistic for huge networks that their network topologies span over several continents, it falls within the realm of possibility for small and large local networks (i.e., campus networks) or wide area networks that their entities are contained in a single province and owned by a single institution. Next we will present a case study of how to construct such a fault propagation model for a campus network which will be used as a prototype to create fault propagation models in the simulation process.

8.4.1Case Study

We present in this case study a bipartite dependency graph as a fault propagation model of the campus network of the University of Waterloo that a fault network management system can use. The structure of the campus network of University of Waterloo is shown in Fig. 8.9. It consists of 69 nodes including 48 building Ethernet switches, 12 core Ethernet switches, and 9 routers. The topology of the network is configured in a tree-shaped fashion. The advantage of the tree-shaped topology is that cyclic paths are eliminated. Though the size of the network is not so large the fault propagation model constructed from this topology may require a considerable effort to manually construct it. In the proposed dependency graph, we refer to the switch-to-switch and switch-to-router delivery service as *links*; while the *path* refers to the route that a packet may take from the first switch (the source) to the last one (the destination). The packet delivery service is actually built on the delivery services provided by links. Since we are interested in detecting and identifying performance failures in end-to-end service we want to identify all the paths that the current network configuration provides and all the links that

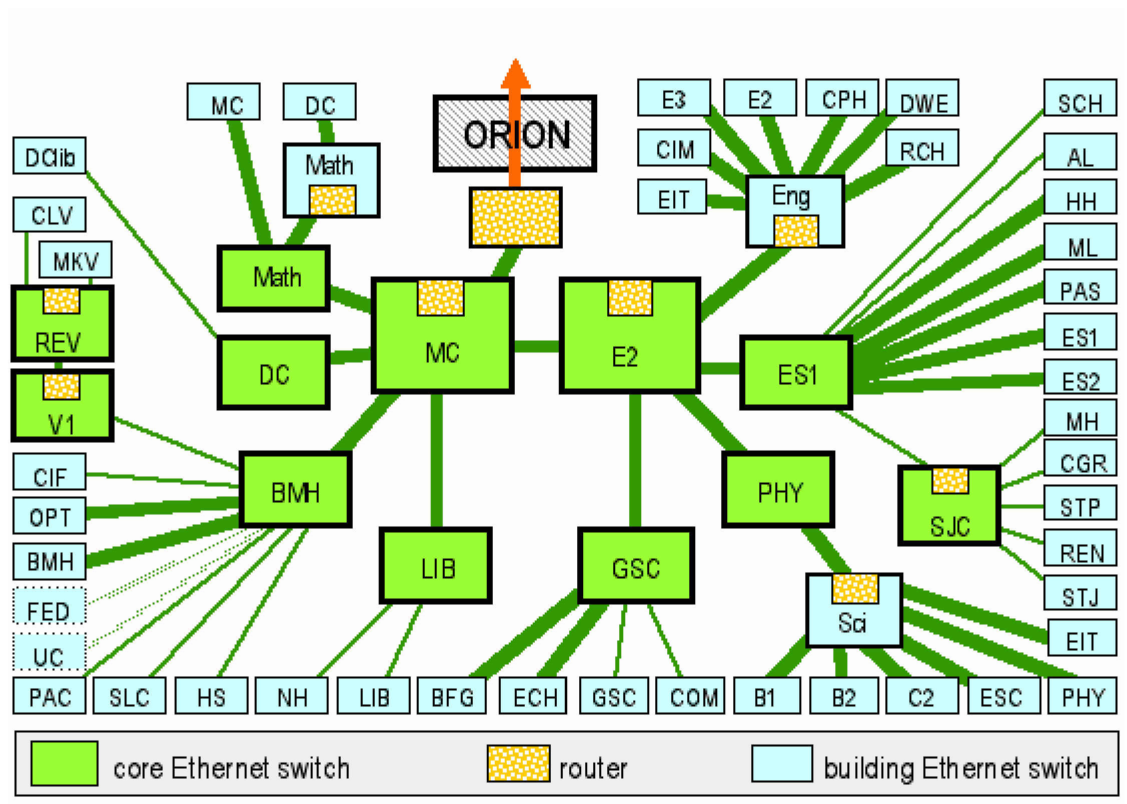


Figure 8.9: The topology of the campus network of University of Waterloo.

comprise each path. This process leads to the service-to-link relationship graph. In the service-to-link relationship graph each link between any two switches or a switch and a router is represented by a node. From this node we create a directed edge to any path that may rely on this link for packet delivery service. Due to the limited space, only a partial view of this graph is shown in Fig. 8.10. The ellipse shape indicates a link service and the circle shape indicates a path service. This graph is important because it shows each packet delivery service path from host to host which greatly narrows down the suspect network nodes in case a failure in packet delivery service arises. However, the path-to-link relationship graph does not explicitly show which network entities may generate network alarms in case such path failures are detected. Hence, a more elaborated graph is

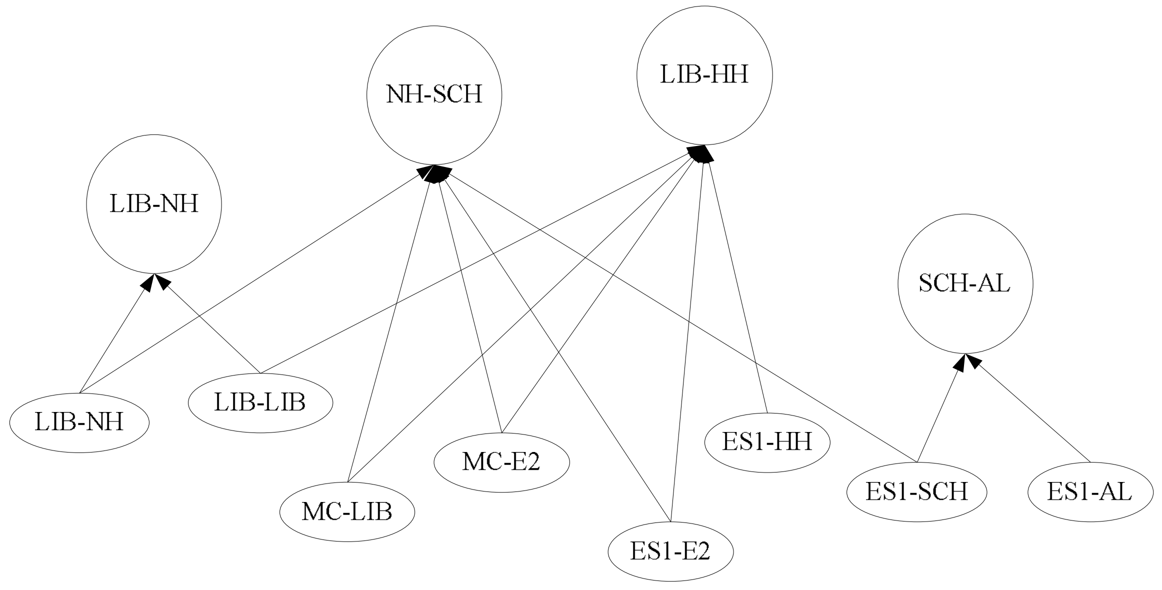


Figure 8.10: Path-to-link relationship resulted from the topology of the campus network of Fig. 8.9.

needed to identify these monitoring entities which will be used to correlate symptoms observed on the path level to identify and isolate the underneath link failures. The result of this process is the fault propagation model of the managed network as shown in Fig. 8.11 and in which we associate a set of performance failures (e.g., congestion, delay, broken link, etc) with each link node and associate a network alarm or a set of network alarms with each network entity that may report the packet delivery failure of that end-to-end service. For simplicity let us assume that only two intelligent agents are employed to monitor the campus network and are installed in the core Ethernet switches E2 and MC. The network entities connected to the core switch E2 through the core switch nodes ES1, PHY, GSC, and SJC will report their network alarms to the intelligent agent 1. While the network entities connected to the core switch MS through the core switches Math, DC, BMH, and LIB will report their network alarms to the intelligent agent 2. For example, symptoms related to the path service SCH-AL are expected to be reported to the intelligent agent 1 since the entities reporting these symptoms are the building switches SCH and AL and the core switch ES1 and all are residing in its domain. However, symptoms related to the path service NH-SCH will be reported to both agents since the entities comprising the packet route are residing in both domains. Symptoms generated from the building switch NH and the core switch LIB will be reported to the intelligent agent 2, while symptoms generated from building switch SCH and the core switch ES1 will be reported to the intelligent agent 1. Hence, both agents need to collaborate to identify the malfunctioning link that disrupted the path service. Though each agent has the same fault propagation model similar to the one shown in Fig. 8.11 each intelligent agent only relates the symptoms generated by its constituent entities. The rectangular shapes refer to the network entities. It should be noted, though, that in the fault propagation model the rectangular shapes will actually represent network alarms that are triggered by their corresponding network entities.

8.4.2 Simulation Model

The simulation model uses tree-shaped topologies similar to the network configuration shown in Fig. 8.9. Given a random network topology, smart bridges and switches may create a tree-shaped topology using the spanning tree routing algorithm. The proposed schemes will be evaluated for their fault detection and identification accuracies. To describe the simulation steps we will use the fault propagation model obtained for the campus network of University of Waterloo shown in Fig. 8.9 and Fig. 8.10.

8.4.2.1 Generating Random Network Topologies

Since the first step is to create a random and tree-shaped network topology, we will give a brief review of the simulation algorithm used to randomly create a network topology and the spanning routing algorithm used to create the tree-shaped topology. For further details, you may refer to Appendix C. Let us assume the nodes of the generated network

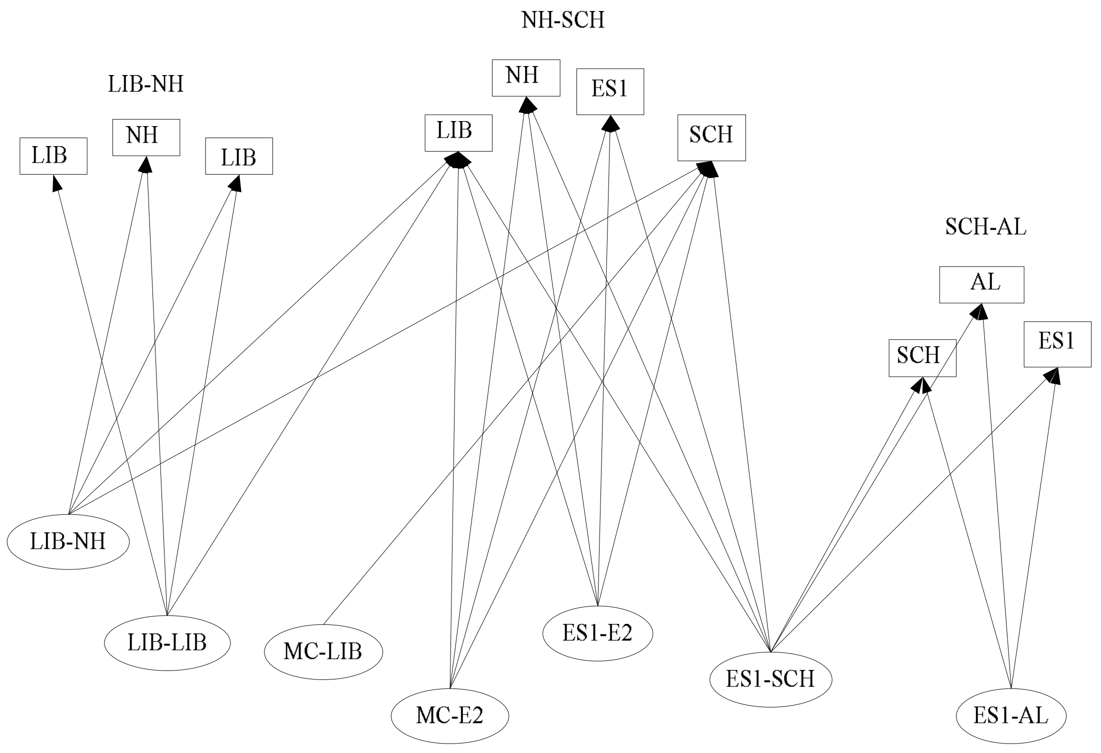


Figure 8.11: Partial view of the fault propagation model of the network of Fig. 8.9.

represent the data link layer in the protocol stack. That means a given network of size *n* has *n* bridges. The Network class encapsulates the dynamic behavior of the simulation algorithm. The topology of the generated network is stored in the private variable, *Top[MAX][MAX]*. The MAX constant refers to the network size, i.e., MAX=n. Each node in the generated network is assigned an ID which stands for its hard coded Ethernet address. The degree of connectivity of the generated network may be ranged from full connectivity (i.e.; each node in the network is connected to every other node) to a partial connectivity. Since the network nodes represent bridges, a tree shaped topology of the given network can be then obtained using the spanning tree algorithm [113]. Once a random tree-shaped network configuration is obtained, a network manager will obtain a its fault propagation model as described next.

8.4.2.2 Obtaining Fault Propagation Models

The next step in the simulation procedure is to obtain a valid fault propagation model for the obtained tree-shaped network. It can be accomplished as follows. Let us assume that the set of all possible alarms is referred to as A and the set of all faults is referred to as F . For a given tree-shaped n-network topology of size *n,* we create a fault propagation model as follows:

- We identify all the possible paths that the current network configuration may provide. For example, the path from building switch LIB to the building switch
- NH in Fig. 8.9 can be represented as a node LIB-NH in the path-to-link relationship graph as shown in Fig. 8.10.
- For each obtained path we identify the all the links that the path may depend on. For example, the path LIB-NH shown in Fig. 8.10, may depend on the link from
  the building switch NH to the core switch LIB and the link from the building switch LIB to the core switch LIB. Each link will be represented as a node as shown in Fig. 8.10.
- We identify network entities for each path that may report network performance failures for their related path in the form of network alarms. We will refer to these network entities as path entities.
- We create a directed edge between every identified link node and every network entity contained in a path which is using that link, as shown in Fig. 8.11.
- We associate with every link node a fault f_j. It is known that in an n-node tree-shaped network there are n-1 links which means that $|F| = n-1$. For example, based on the given PFM in Fig. 8.10, $F = \{f_1 = LIB - NH, f_2 = LIB - LIB, f_3 = MC - LIB, ...f_7 = ES1 - AL\}$.

- We create one network alarm for each path entity in the fault propagation model. For example, the network alarms expected according the FPM shown in Fig. 8.11 are represented by the set $A = \{a_{LIB}, a_{NH}, a_{ES1}, a_{LIB}, a_{SCH}, a_{AL}\}$.
- The priori fault probability distribution p_f is randomly generated and uniformly distributed over the interval [0.001, 0.01].
- To signify the causal relationship between networks failures (associated with links) and their corresponding alarms (associated with paths), we assign randomly a conditional probability for every edge shown in the fault propagation model. The conditional probability distribution is uniformly distributed over the range [0.5, 1).

Once the topology of the generated network is transformed into a tree-shaped one and the fault propagation model is obtained, we randomly divide the given network into separate management domains. In real networks, the number of number of management domains is determined based on geographical and security considerations. To simplify the simulation we will determine the number of intelligent agents (*K*) based on the network size using the simple following rule:

$$K = \frac{\log_2(n)}{2}$$

n is the number of network bridges. Applying this rule may yield management domains shown in Table 8.4. We assign intelligent agent for each management domain and the network entities are distributed among these domains based on their proximities of intelligent agents. Each intelligent agent pertain a copy of the obtained fault propagation model.

Table 8.4: Management domains.

Network size	5	10	15	20	25	30	35	40	45	50
Number of domains	1	2	2	2	2	2	3	3	3	3

8.4.3 Results by the DSET-Based Alarm Correlation Algorithm

For every simulation case *i*, we created 10 simulation scenarios as follows. Let k represent the k^{th} simulation scenario of the simulation case *i,* where $1 \le i \le 10$ and $1 \le k \le 10$. For every simulation scenario we proceed as follows:

- Based on the link priori probability distribution, we first randomly introduce a set of faults to network links. We will refer to this set of introduced faults as F_i^k.
- We create a probability distribution $P_a^k : A_i^k \rightarrow [0,1]$, where $P_a^k(a_j) = P\{a_j$ occurs | all faults in F_i^k occur$\}$.
- Using P_a^k we randomly introduce a set of network alarms $A_{iF^k}^k \subseteq A_i^k$ that may have been caused by the introduced faults F_i^k.
- Using the proposed DSET-based algorithm we calculate $F_{iD}^k \subseteq F_i^k$ the most likely explanation of the observe network alarms in $A_{iF^k}^k$. We calculate the detection rate (DR_i^k) using the following formula:

$$DR_i^k = \frac{| F_{iD}^k \cap F_i^k |}{| F_i^k |}$$

For every simulation case we calculate the average detection rate as follows:

$$DR_i = \frac{1}{10}\sum_{k=1}^{10} DR_i^k$$

We then calculate the values of detection rate for each simulation case denoted by $DR(n)$ for varied network sizes.

We implemented also the well known code-book-based algorithm for alarm correlation [6]. The correlation paradigm is based on coding schemes in which the information contained in the fault propagation model represented as a bipartite graph is converted into a set of codes. A code for each performance failure is extracted from the fault propagation model. For example, the fault propagation model of Fig. 8.11 may yield the codebook shown in Table 8.5. A performance failure code is represented by a series of bits. A certain bit takes the value of 1 if the corresponding network alarm can be triggered by the network failure; otherwise it takes the value of 0. To isolate the performance failure, the set observed alarms is matched to the codebook. Network failures that optimally match an observed alarm vector are identified as the root causes of the observed alarms. The identification mechanism is based on the Hamming distance between network failure codes. As can be seen form the table two failures, namely MC-E2 and ES1-SCH, can not be distinguished by the codebook since both failures have the same code. However, this codebook resulted from an incomplete fault propagation model and is only used here for explanatory reasons. A radius of a codebook is one half the minimal distance between failure codes.

Table 8.5: Codebook obtained from the fault propagation model of Fig. 8.11.

	LIB-NH	LIB-LIB	MC-LIB	MC-E2	ES1-E2	ES1-SCH	ES1-AL
LIB	0	1	0	0	0	0	0
NH	1	1	0	1	1	1	0
LIB	0	1	0	1	1	1	0
ES1	0	0	0	1	1	1	1
SCH	1	0	1	1	1	1	1
AL	0	0	0	0	0	1	1

In a noisy environment such as that of computer networks, it is highly expected that a percentage of network alarms will be lost before they reach their intended destinations. This can be caused by many factors such as, highly congested links on the alarms paths, using of unreliable transport mechanisms such as user datagram protocol UDP (used by the network management protocol SNMP), etc. Hence, we will test the reliability and effectiveness of the proposed algorithms in the presence of such alarm losses. We define the loss rate as number of network alarms actually received by intelligent agents to the total number of network alarms generated by the introduced network failures. We have set the loss rates for our experimentations to 10%, 20%, and 30%. The results of the simulation experiments are shown in Fig. 8.12, Fig. 8.13, and 8.14 respectively. The figures clearly illustrate that the proposed algorithm is always performs better than the codebook based algorithm in terms of the fault detection rate. We notice that the codebook algorithm performance deteriorates even further for small

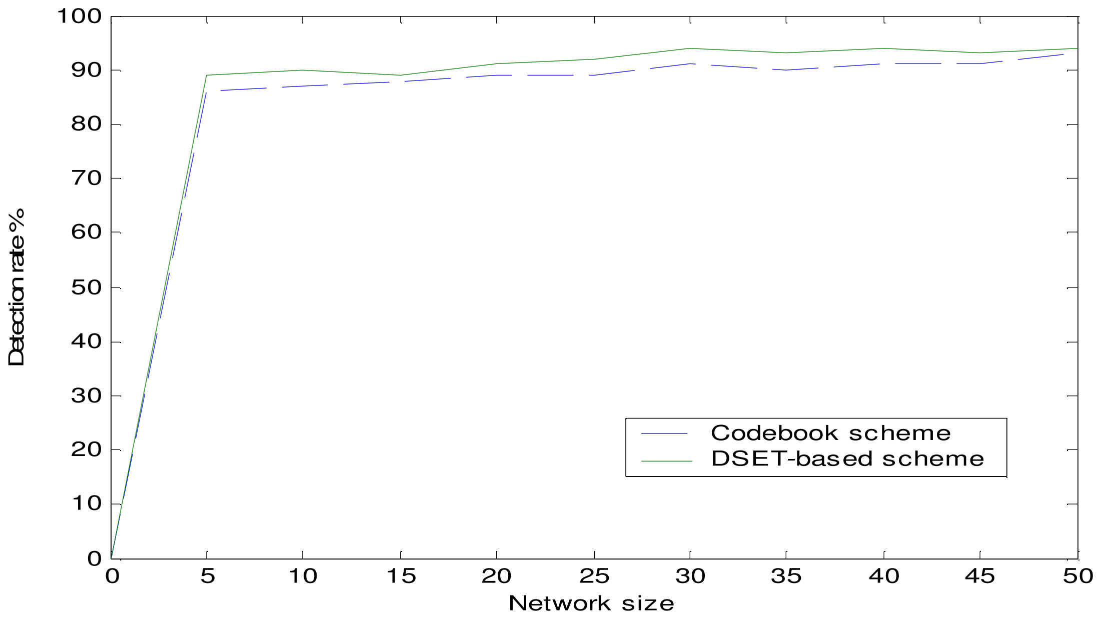


Figure 8.12: Failure detection rate of both algorithms with loss ratio 10%.

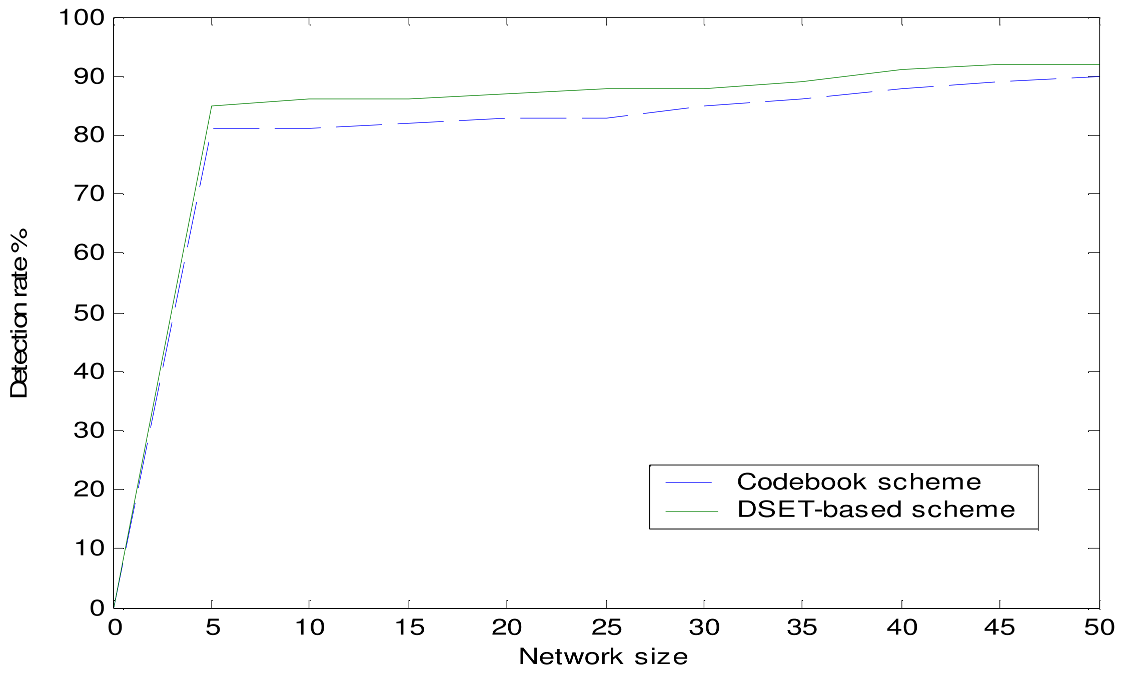


Figure 8.13: Failure detection rate of both algorithms with loss ratio 20%.

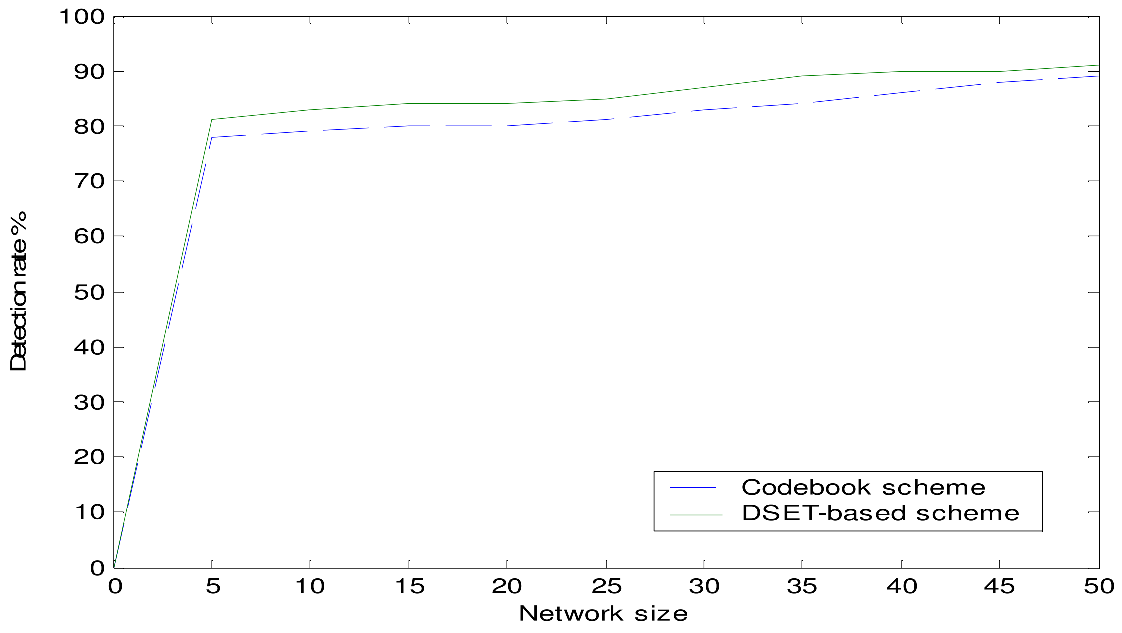


Figure 8.14: Failure detection rate of both algorithms with loss ratio 30%.

network sizes and high alarm losses. This is a major deficiency on the part of the codebook. We attribute this shortcoming to the sensitivity of the codes of network failures obtained from the available fault propagation model. This sensitivity is expressed in the form of radius as defined above. For example, the radius of the codebook shown in Table 8.4 is 0.5. When the radius is 0.5, the code is still able to provide distinction among network failures; however, it is not resilient to noise. For example, the two network failures ES1-E2 and ES1-SCH presented by the codebook of Table 8.4, are only distinguished by a single network alarm emitted from the building switch AL. This is also true in the case of network failures MC-E2 and ES1-SCH. This means that a loss of the network alarm generated by the switch AL will result in a potential decoding error. Of course to redeem this problem, the radius of the codebook has to be increased by increasing the number of network alarms. However, the proposed scheme does not suffer from this problem since it is non-deterministic in nature. Moreover, in the case of the codebook-based scheme, the agent manager, for the lack of a better combination rule, utilizes the majority vote mechanism to combine outcomes of its subordinate intelligent agents. This mechanism may have some negative impact on the overall decision when intelligent agents which have made correct judgments are outnumbered by intelligent agents which have made false identifications. This is true when some intelligent agents do not receive enough network alarms to make the right decision. For network of small sizes, the number of the generated alarms are smaller compared with networks of larger sizes. If some of these alarms are lost then the fault identification task becomes hard for the DSET-based algorithm and even harder for the codebook-based algorithm. The codebook algorithm, however, tends to enhance its performance as networks increase in size while the alarm loss ratios decrease.

We calculate the false positive rate (FPR_i^k) using the following formula:

$$FPR_i^k = \frac{| F_{iD}^k \setminus F_i^k |}{| F_{iD}^k |}$$

For every simulation case we calculate the average false positive rate as follows:

$$FPR_i = \frac{1}{10}\sum_{k=1}^{10} FPR_i^k$$

We then calculate the values of false positive rate for each simulation case denoted by $FPR(n)$ for varied network sizes. The false positive rates for the alarm loss of rates 10%, 20%, and 30% are shown in Fig. 8.15, Fig. 8.16, and Fig. 8.17, respectively.

On average the proposed scheme has 0.03%, 0.035%, and 0.05% false positive rates for the alarm losses rates of 10%, 20%, and 30%, respectively. It is noted that for alarm losses higher than 10%, the codebook scheme yields higher percentage of false positive rates as network sizes get larger (more than 0.07% false positive rate for networks of size 50). This is due to the fact that the number of potential network failures increases as the complexity of the network increases. It becomes difficult for the codebook scheme to differentiate between network failures that have close similarity of code signals. With high rate of missing crucial alarms, it may include non-existent network failures in their explanation hypotheses. Though the increase of the alarm loss ratio negatively affects the false positive rate of the proposed algorithm, it still yields better false positive rates than those of the codebook scheme.

8.4.4 Results by the Adaptive Alarm Correlation Algorithm

In this section we investigate the performance and the accuracy of the adaptive fuzzy DSET-based algorithm. As has been pointed out in Chapter 7 the DSET-based does account for the positive symptoms in the process of fault identification. The fact that fault hypothesis's relevant network alarms have not been observed should decrease our confidence in the occurrence of that hypothesis. We have utilized the simulation model introduced in Section 8.4.4 and used the same obtained fault propagation models and probability distributions. In this section, we have considered network alarms which have not been observed in the previous simulation experiment as positive alarms. However, we should not misinterpret lost negative alarms as positive alarms. The following additional

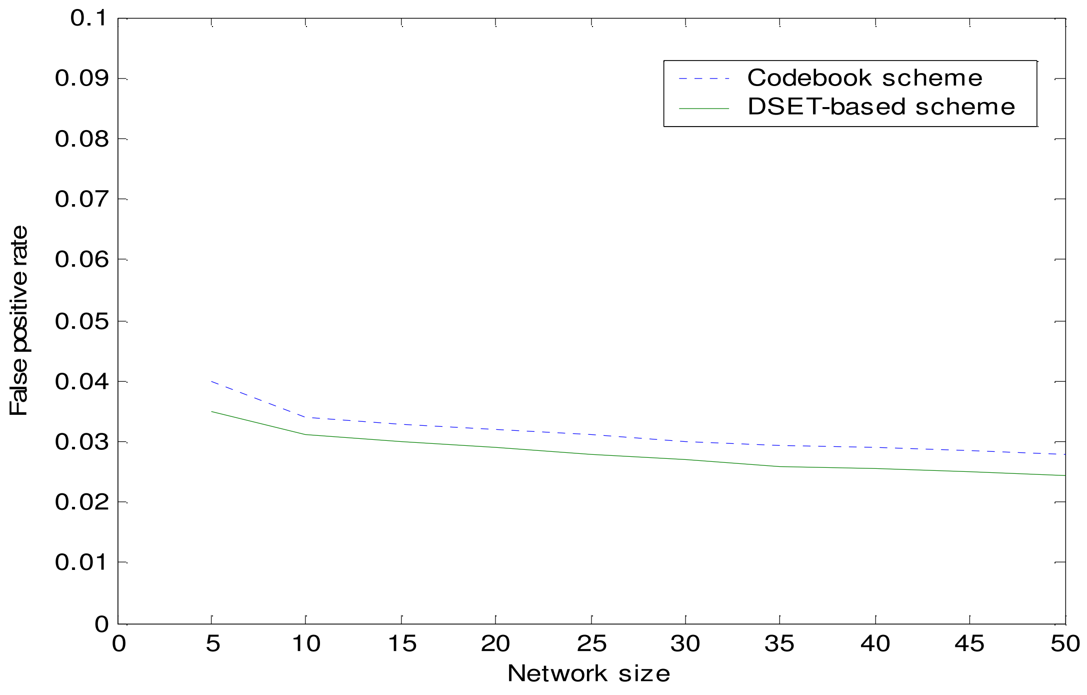


Figure 8.15: Failure positive rate of both algorithms with loss ratio 10%.

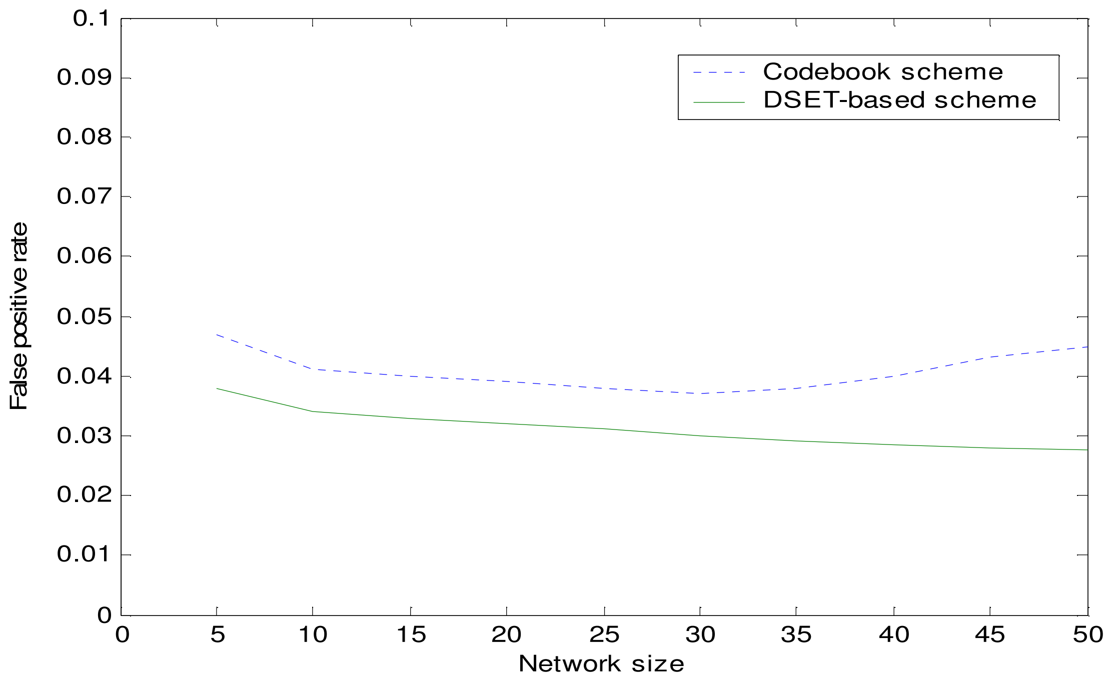


Figure 8.16: Failure positive rate of both algorithms with loss ratio 20%.

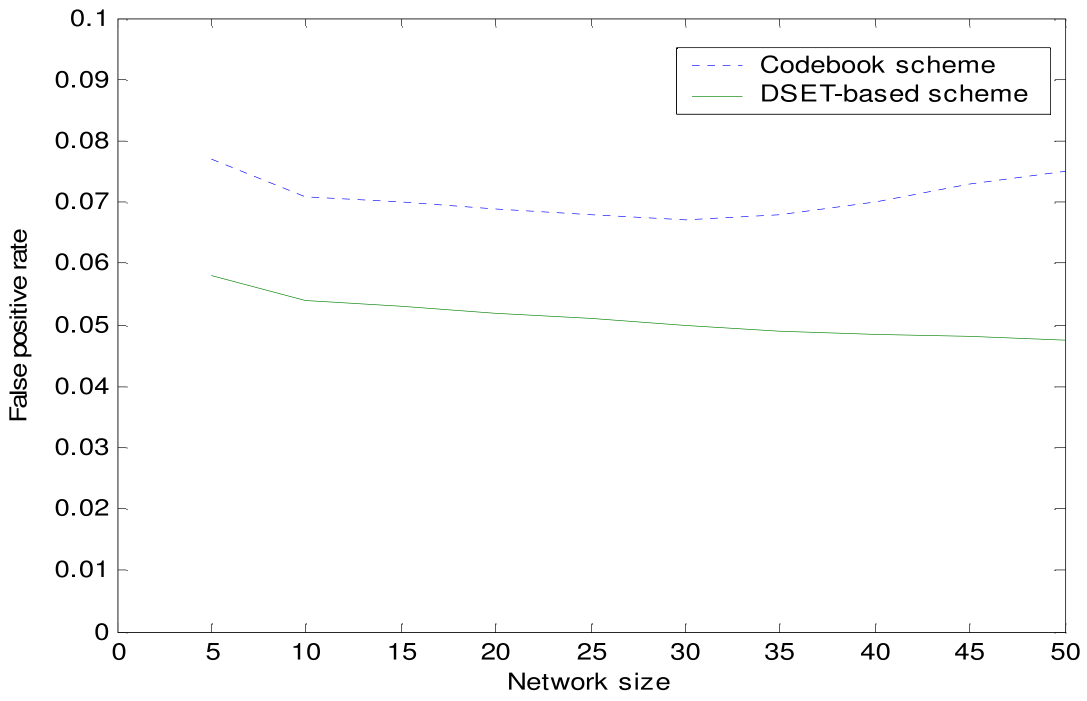


Figure 8.17: Failure positive rate of both algorithms with loss ratio 30%.

steps were added to the simulation process to account for the positive symptoms. For each simulation case:

1. Based on the current fault propagation model, each intelligent constructs a fault domain for each performance failure.
2. The set of symptoms that has not been observed by intelligent agents is referred to by the set $A_{iR}^k = A_i^k - A_{iF^k}^k$. Where A_i^k is the set of all possible observed alarms and $A_{iF^k}^k$ is the set of the actually observed symptoms resulted from the introduced performance failures.
3. Using the $1 - P_a^k$, we randomly generate a subset $A_{ipos}^k \subseteq A_{iR}^k$ of positive alarms from the set A_{iR}^k. Where P_a^k as defined above.

4. Using the fuzzy inference mechanism introduce in section 7.2, each intelligent agent calculates its local fuzzy composite alarm

We use the fault detection rate as an accuracy measure as described in the previous section. The results are shown in Fig. 8.18, Fig. 8.19, and Fig. 8.20, for 10%, 20%, and 30% alarm losses, respectively. The average fault detection accuracies of the adaptive scheme is compared with DSET-based and codebook-based algorithms and shown in Tables 8.6, 8.7, and 8.8. Simple investigation of the provided tables shows clearly that the fault detection rate has substantially improved. However, as the number of observed negative alarms increase the three algorithms tend to provide comparable fault detection accuracies. Intelligent agents in the adaptive-based scheme cooperate more effectively to reach a final decision. As different intelligent agents may have different weights their contributions to the fault identification task are based on how many negative and positive alarms have individually been received. The weight factor is based on the discounting factor as defined in Equation (7.25) and implemented as follows:

$$\alpha_k = \alpha_0 \times \frac{H(B^k_{comp})}{\log_2 |F|} \times \frac{FH(B^k_{comp}) - \min FH}{\max FH - \min FH}$$

Where $\max FH$ and $\min FH$ denote respectively the maximum and minimum of hybrid entropies among the intelligent agents and is the maximum Shannon entropy [51]. We empirically set the value of α_0 to 0.15 to stand as an upper bound for the discounting factor. Hence, the advantage of the adaptive scheme over both DSET and codebook based schemes is that the positive symptoms are incorporated in the fault identification process.

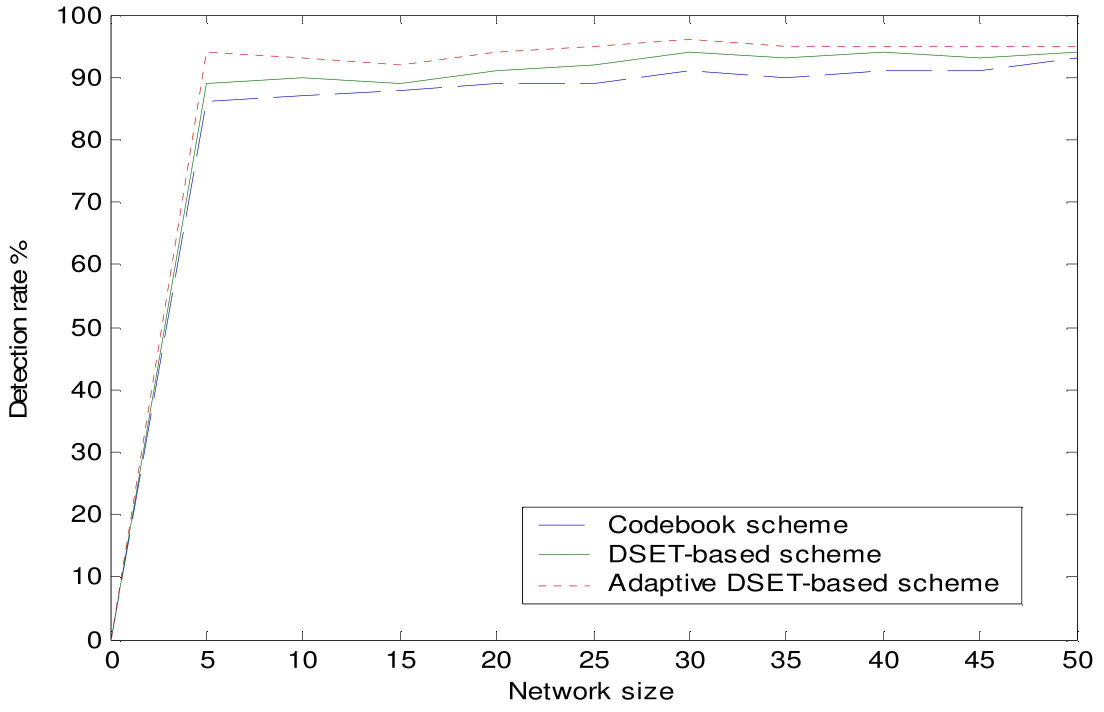


Figure 8.18: Failure detection rate of all algorithms with loss ratio 10%.

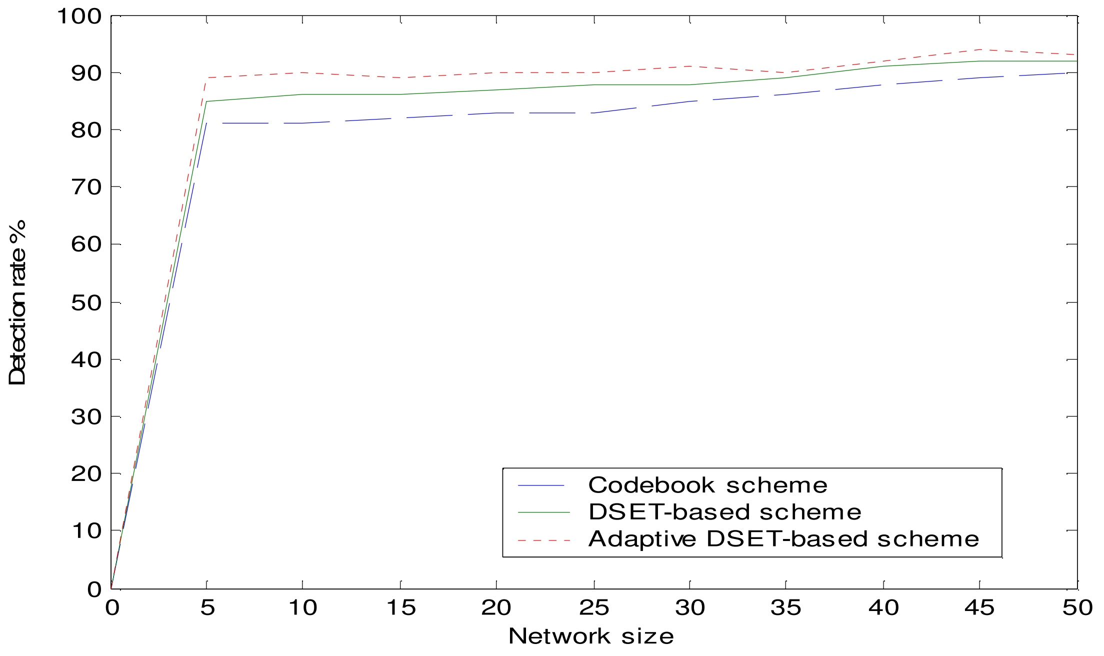


Figure 8.19: Failure detection rate of all algorithms with loss ratio 20%.

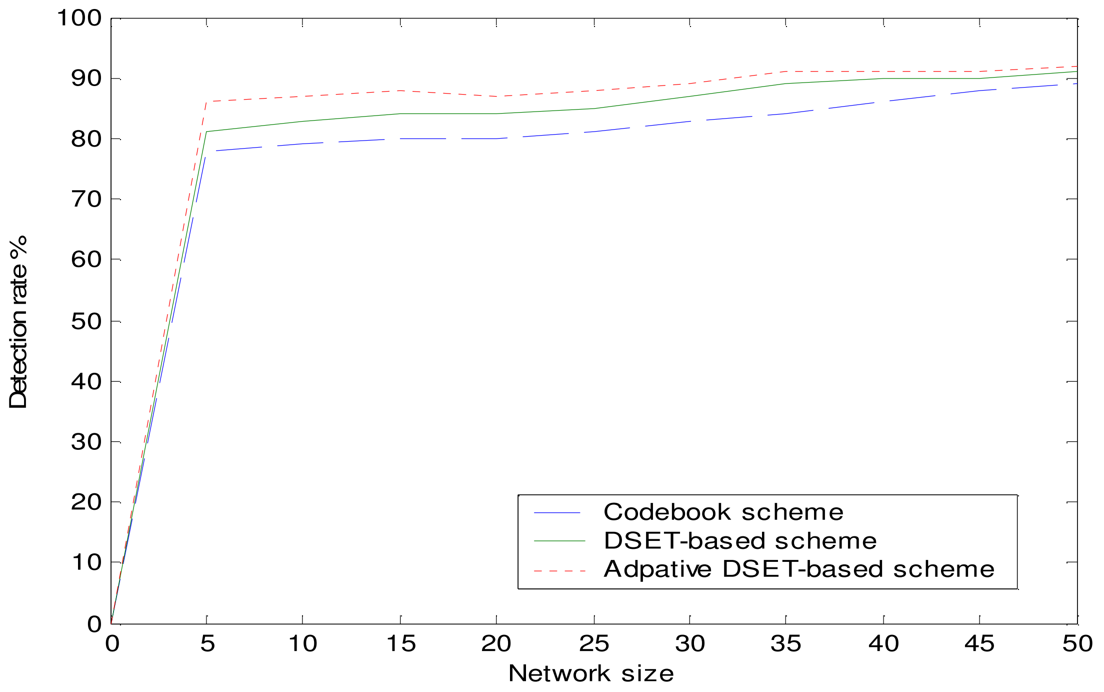


Figure 8.20: Failure detection rate of all algorithms with loss ratio 30%.

In some cases the detection accuracy of the adaptive scheme increases by 8% more than that of the codebook scheme and 5% of that DSET-based scheme for the 20% alarm loss. This is quite improvement since small networks with alarm loss of 20% can be considered as poorly instrumented networks (networks with fewer triggered symptoms). With this level of instrumentation, positive symptoms can play an important role in increasing the accuracy of fault identification tasks. Of course one can argue that this accuracy improvement may be achieved, however, the overall run-time of the fault identification is also increased. This tradeoff between accuracy and performance is an inherent characteristic of network management systems.

Table 8.6: Accuracy averages of the three algorithms for alarm loss 10%

Network size	5	10	15	20	25	30	35	40	45	50
Codebook Algorithm	0.86	0.87	0.88	0.89	0.89	0.91	0.90	0.91	0.91	0.93
DSET-based Algorithm	0.89	0.90	0.89	0.91	0.92	0.94	0.93	0.94	0.93	0.94
Adaptive DSET-based Algorithm	0.94	0.93	0.92	0.94	0.95	0.96	0.95	0.95	0.95	0.95

Table 8.7: Accuracy averages of the three algorithms for alarm loss 20%

Network size	5	10	15	20	25	30	35	40	45	50
Codebook Algorithm	0.81	0.81	0.82	0.83	0.83	0.85	0.86	0.88	0.89	0.90
DSET-based Algorithm	0.85	0.86	0.86	0.87	0.88	0.88	0.89	0.91	0.92	0.92
Adaptive DSET-based Algorithm	0.89	0.90	0.89	0.90	0.90	0.91	0.90	0.92	0.94	0.93

Table 8.8: Accuracy averages of the three algorithms for alarm loss 30%

Network size	5	10	15	20	25	30	35	40	45	50
Codebook Algorithm	0.78	0.79	0.80	0.80	0.81	0.83	0.84	0.86	0.88	0.89
DSET-based Algorithm	0.81	0.83	0.84	0.84	0.85	0.87	0.89	0.90	0.90	0.91
Adaptive DSET-based Algorithm	0.86	0.87	0.88	0.87	0.88	0.89	0.91	0.91	0.91	0.92

As shown in Fig. 8.21, Fig. 8.22, Fig. 8.23, the false positive rates are also improved by 0.01% for the alarm loss rates of 10% and 20% ratios and by 0.02% for the alarm loss ratio 30%. On average the adaptive alarm correlation scheme has 0.02%, 0.025%, and 0.03% false positive rates for the alarm loss rates of 10%, 20%, and 30%, respectively.

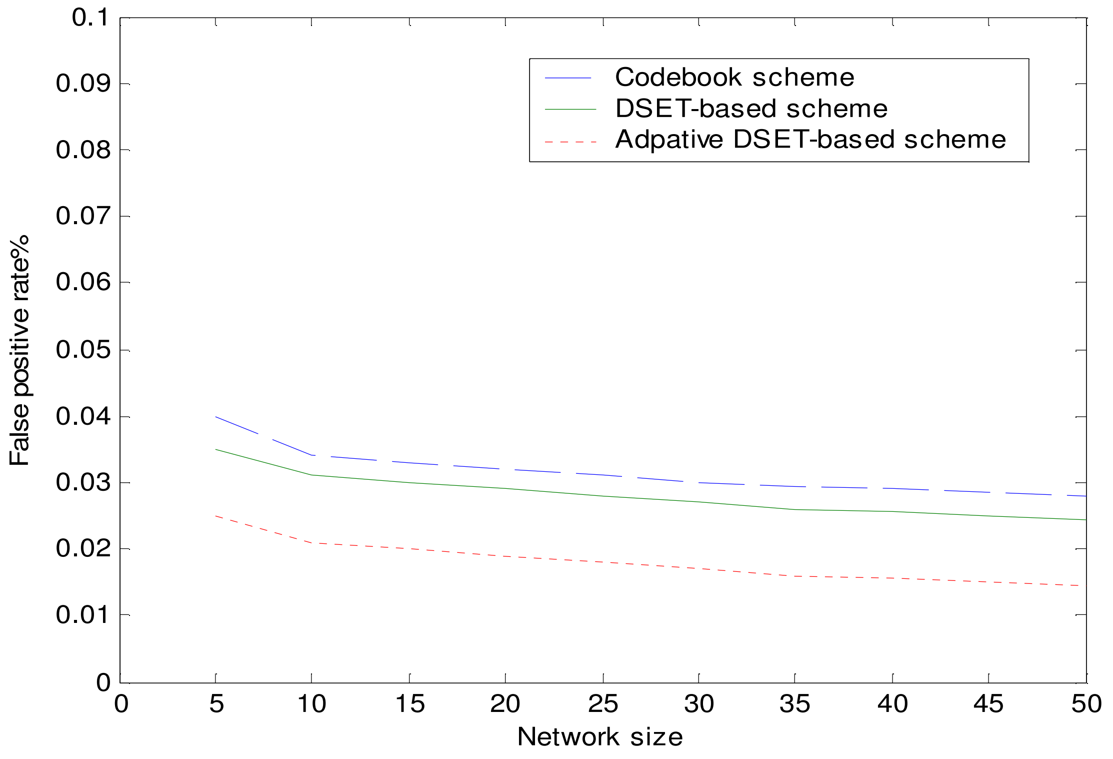


Figure 8.21: False positive rate of all algorithms with loss ratio 10%.

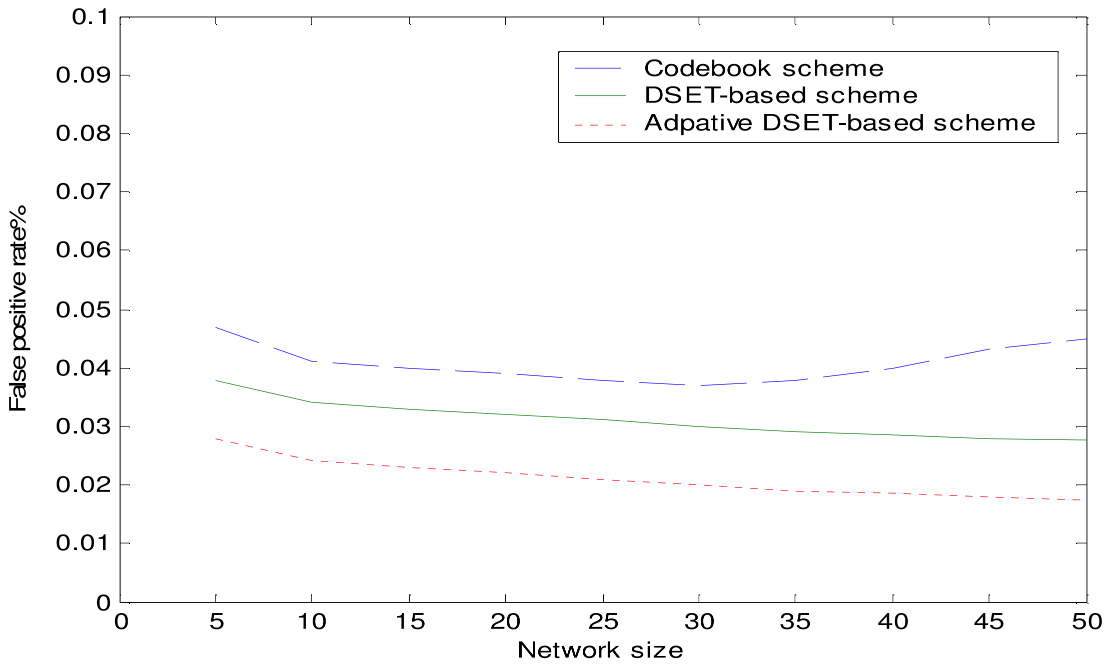


Figure 8.22: False positive rate of all algorithms with loss ratio 20%.

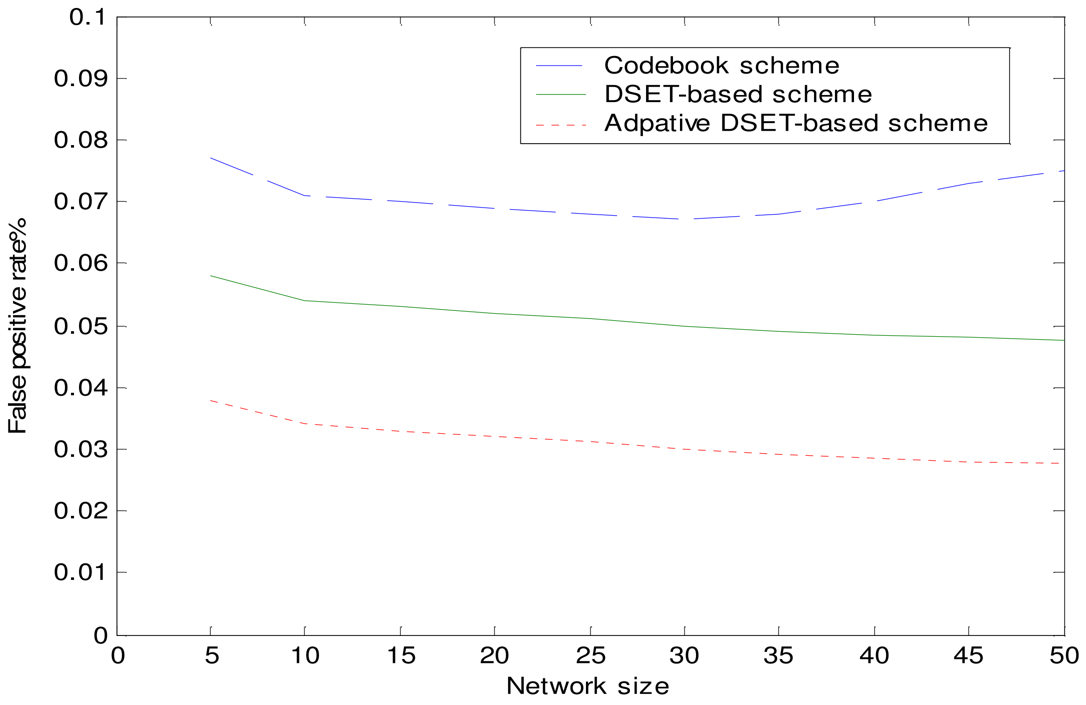


Figure 8.23: Failure positive rate of all algorithms with loss ratio 30%.

8.5 Summary

In this chapter, we have conducted extensive experimentations to examine the accuracy and efficiency of the proposed algorithms in the field of computer network fault management. For the active-based algorithms, the new CSP-based approaches always provide less testing probes that is capable of detecting and identifying observed network failures. The proposed DSET-based alarm correlation algorithms outperform the codebook-based approach (which considered as one of the most widely recognizable alarm correlation approaches) in terms of the fault identification accuracy as well as false positive rates. A cased study of building a fault propagation model (a pre-request for the new approaches) of the campus network of the University of Waterloo is also presented.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

The problem of fault detection and identification in computer networks is an interesting and a challenging one. The hierarchal nature of computer networks makes it possible that a failure in a single network entity may spread vertically and horizontally and affect multiple dependent entities that may otherwise just work perfectly. In a response to such failure, the affected entities send notification messages to their assigned manager in the form of alarms, trouble tickets, etc. In order to isolate the main cause of this failure, some fault management systems correlate these diagnostic messages. Other fault management systems use specific measurements, called probes, on a subset of network entities. The probe-based fault management systems send these probes to the managed network on a periodical basis and analyze their outcomes. Since each model has its own merits and shortcomings we have investigated new approaches for both paradigms in this dissertation. The main contributions of the work presented in this dissertation include the following:

- *A novel-model for the probe selection* is developed. The new constraint satisfaction problem (CSP)-based model presents well-developed domain reduction rules that are used by the search engine to reduce the search space and speed up the model convergence.
- *An adaptive novel-fuzzy CSP technique* is presented to refine the available testing probes and select the most informative among them. Instead of crisp constraints that may not be fully satisfied by certain probes, fuzzy constraints are introduced. Candidate probes that satisfy the fuzzy constraints the most are selected. In the beginning, only a few of such probes are sent to the network for fault detection tasks. The outcome of each fault detection probe is then utilized to either identify the network failure or select more probes.
- *A new distributed alarm correlation algorithm* is proposed. The managed network is divided logically into a cluster of disjoint management domains. Each management domain is assigned an intelligent agent that is responsible for collecting alarms emitted by entities in its domain. The reasoning engine of the intelligent agent is based on the Dempster-Shafer Evidence Theory (DSET). In the framework of the DSET, these alarms are considered as pieces of evidence. An evidence structure for each received alarm is constructed based on the posterior probability calculated for each fault hypothesis using the Pearl's updating belief algorithm. Intelligent agents then send their findings in the form of new local composite alarms to a higher level manager called the agent manager. The agent manager combines the new alarms using the Dempster's rule of combination.
- *An adaptive distributed alarm correlation algorithm* is presented. A fuzzy evidence structure is constructed for each alarm cluster observed by an intelligent agent. The intelligent agent utilizes its knowledge of network failure domains, presented by fault propagation models, and assigns weights for each fault hypothesis in the fuzzy focal. Alarms that were not observed by intelligent agent are called positive alarms. The membership degree for each fault hypothesis in a

given fuzzy focal is determined by both positive and negative network alarms in that fault domain. The fuzzy inference mechanism is based on the premise that the absence of observations of network alarms of a given network failure should decrease our confidence in the occurrence of that network failure. The agent manager that correlates the evidence structures provided by its subordinate agents assigns each intelligent agent a certain weigh and fuses its evidence piece accordingly.

Table 9.1 summarizes the main features of the proposed algorithms. In the table, DAC stands for distributed alarm correlation and ADAC stands for adaptive, distributed alarm correlation. The distributed alarm correlation algorithms are reactive in the sense that they do not start their fault analysis process until they receive malfunctioning indications from the managed network. Hence, they do not induce extra management traffic. This of course implies that they may not be able to anticipate network failures in advance. Being active, on the other hand, the CSP-based algorithms can predict potential network failures before their actual occurrence and quickly identify the root cause. However, this may entail some traffic overhead in terms of probes. Moreover, both CSP-based algorithms are deterministic in nature. That is, the outcome of a given probe is completely characterized by the health status of the network entities in its path. Thus, they are more sensitive to noise than the alarm correlation algorithms.

Table 9.1: Main features of the proposed algorithms.

Algorithm	CSP-Based	Fuzzy CSP-Based	DAC	ADAC
Fault analysis mechanism	Active	Active	Reactive	Reactive
Nature of the algorithm	deterministic	deterministic	Non-deterministic	Non-deterministic
Scalability	Yes	Yes	Yes	Yes
Multiple-fault scenario	No	No	Yes	Yes

In summary, form a theoretic development standpoint, to the best of our knowledge, this dissertation has presented the first effort in the following aspects: 1) propose the new CSP model for active probe selection and introduce new domain reduction rules for the search engine; 2) present new fuzzy CSP model for adaptive probe selection; 3) utilize fault propagation models to build evidence structures and exploit positive and negative symptoms observed by the management system; 4) introduce the Dempster-Shafer evidence theory as a framework for network alarm correlation mechanism; 5) develop the adaptive alarm correlation reasoning algorithm in the fuzzy evidential reasoning framework, with the capability of discounting less informed intelligent agents during the correlation process.

9.2 Future Work

One of the interesting suggestions is to investigate the use of a hybrid approach in which network alarms can be considered as probing tests. Since each received network alarm is an indication of the possible occurrence of certain network failures, we may conclude that the network entities in the received network alarm are working properly and should not be deemed suspected nodes. Furthermore, a *hyper*-arc consistency algorithm may be investigated for both the standard and the fuzzy CSP-models. As has been demonstrated throughout this work, fault propagation models play an important role in most of the network alarm correlations reported in the literature. However, an automatic mechanism by which such fault models can be extracted form current network configurations has not
by which such fault models can be extracted form current network configurations has not
been subjected to extensive study. Such tools can be very helpful in the dynamic environment of computer networks, where current configurations may change frequently.

Besides the issues studied in this dissertation, in the field of network fault management, the following two areas can be considered as open research problems:

- Temporal alarm correlation,
- FPM for wireless networks.

Temporal correlation is recognized as one of the important aspects of alarm correlation systems. In this research work, we have assumed that the received alarms are observed a short time after the fault occurrence, and as such, only those occurring within a time-window may be correlated. However, the proposed algorithms can be modified to incorporate temporal information such as the arrival, between arrival, and time duration in the alarm correlation process. Wireless networks introduce new challenges to the network fault management field. Obtaining an accurate and constantly updated fault propagation models prove to be a challenging task in a mobile wireless environment. Hence, advanced techniques should be investigated through which such models can be obtained.

APPENDICES

Appendix A

MODEL: Event Modeling Language

This summary review has been extracted from [38].

A.1 Overview

Event modeling is an essential component of event correlation systems. Event correlation is the process of automatically grouping related events based on their underlying common cause. An event correlation system consists of two basic components: an event definition and propagation model (i.e., event model), and a reasoning algorithm. The event model describes the underlying system. The reasoning algorithm correlate events based on the knowledge contained in the event propagation model. The MODEL language is basically the event modeling component of SMARTS InCharge. The features of the language will be demonstrated through examples from the multimedia Quality of Service (QoS) domain. MODEL language supplies an object-oriented data model complete with inheritance and overloading. It also provides instrumentation capabilities to automatically tie attributes in the model to SNMP MIB variables. Boolean expressions are used for a declarative specification of events. The user can specify local event propagation rules in which the causality graph can be constructed from the combination of the class-level event propagation model and the current object topology. Event propagation patterns depend heavily on the way in which objects are currently interconnected. Changing the topology of the modeled objects will drastically alter of the observed symptoms of a problem. One of the most desirable features of the MODEL language is that it is correlation algorithm independent.

A.2 Multimedia Quality of Service (QoS) Domian

A scenario from the Multimedia Quality of Service (QoS) domain is illustrated in Fig. A.1. On the local area network 2 (LAN 2) a video sender wishes to send some live video to a receiver located on LAN 1 using a specific video tool that utilizes the UDP transport protocol. The UDP connection transports IP packets through routers D, C, B, and A

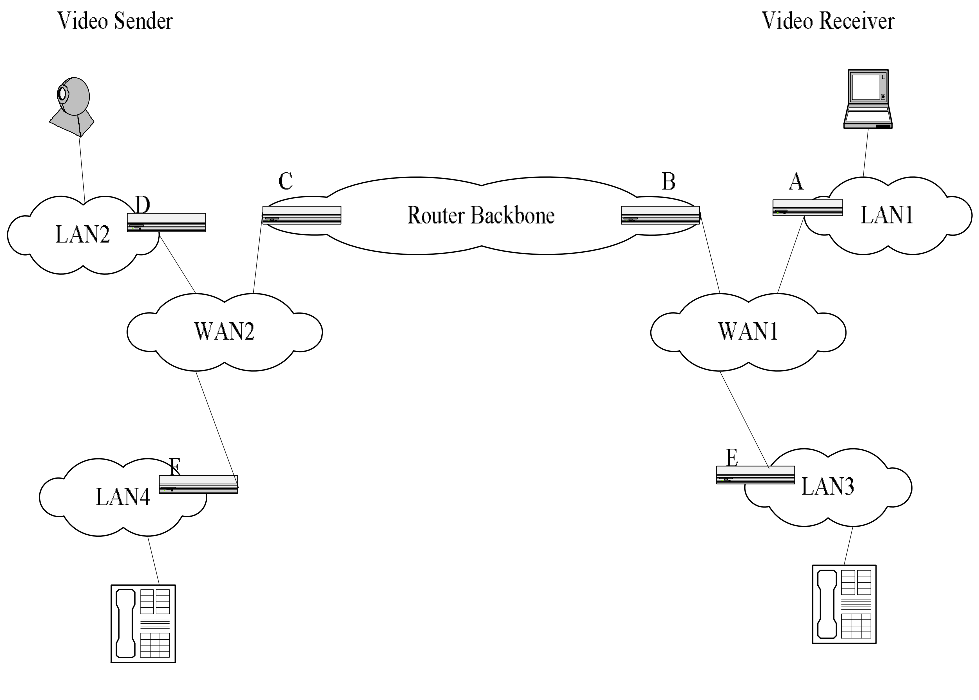


Figure A.1: Multimedia over a multi-domain network.

which connect the LAN domains through a router backbone. The router backbone domain uses physical-layer wide-area network (WAN) domains. Similarly, an audio sender, Internet phone, on LAN4 wishes communicate with a receiver on LAN3, using a specific audio-tool. Its IP packets are routed via F, C, B, and E. These transmissions cause the rate of packets arriving at C to be high. Consequently, the buffer at C overflows, causing the multimedia transmission to lose packets. The packet losses at router C will propagate to all UDP connections which router C is a part of. Since UDP does not retransmit lost packets, these losses will in turn propagate to the multimedia transmissions and hence the quality ate the receiver may become unacceptable.

The event model consists of a class-level model and a run-time object topology. The class-level model describes the general rules for propagating events from objects of one class to another. For the scenario described above, the class-level event model consists of the following: a definition of the "poor video quality" event, and a rule describing the propagation of router congestion to packet loss and then to poor video quality, and optionally a "high packet loss" event at the router level. While the object topology describes a particular instantiation of the run time model which reflects the current state of the actual system. For the above scenario, the object topology consists of the individual routers and multimedia applications and their relationship in the underlying network. A reasoning algorithm would infer the presence of the congestion problem based on the poor video and audio quality and the event model illustrated.

A.2 QoS Management

Consider the following scenario. Due to high traffic volume, router C experiences congestion. As a consequence, its buffers overflow and incoming IP packets get lost. The audio and video receivers experience QoS violation (an average transmission rate that is drastically below tolerance). Using the knowledge provided by the corresponding model, a correlation reasoning algorithm should report a high probability that the problem causing these violations is located in the domain of the router backbone. To implement the causal relationship (congestion causing lost packets), we assume the router implements the IP protocol and is instrumented via SNMP. We can then measure the total number of discarded packets by querying the SNMP MIB-II variables ipOutDiscards and

```
ipInDiscards:
  interface IPRouter: IP
{
        instrumented attribute long ipInDiscards;
        instrumented attribute long ipOutDiscards;
        instrumented long discardsThreshold;

        event PacketDiscardsHigh "The level of discarded packets is high" =
        (delta ipInDiscards + delta ipOutDiscards) / delta _time >
        discardsThreshold;

        instrurment SNMP;
 }
```

The *attribute* statements define measurable properties of the IP protocol entity. The *event* statement defines the circumstance under which the event can be said to have occurred. In this case, the event PacketDiscardHigh will be deemed to have occurred whenever the sum of the changes ipInDiscards and ipOutDiscards per time exceeds a threshold. The *delta* keyword indicates that the difference between the new and old values of the attribute is desired. The *_time* keyword refers to the time at which samples are taken. Thus this event is triggered when the discard rate reaches the threshold.

Now we want to express the fact that there is a causal relationship between the congestion problem and the high packet discard event (with probability 1.0):

```
        problem Congestion "High congestion' = PacketDiscardsHigh 1.0;
```

This line will be added to the MODEL class definition above. Note that this is a semantic declaration in the form of a rule; however, it does not have any specific algorithmic or operational meaning. It simply expresses the fact that there is a causal relationship between these two events. To include the problem and the symptom in the scope of a single class, we should write the following rule:

```
Congestion (IPRouter(x))  ->  PacketDsicardsHigh (IPRouter(x));
```

We have modeled the local symptom which indicates the problem of congestion. We also like to relate the problem to the other observed symptoms at the multimedia application level. In this way, anomalies observed at the multimedia level can be correlated with the problem detected at lower level.

Problems in one object propagate to related objects via *relationships*. In this example, the congestion problem would propagate to higher level connections which are layered over the congested IP node. To indicate this relationship between IP nodes and connections, we would the following statement:

```
relationshipset Underlying, TransportConn, LayerdOever;
```

The keyword *relationshipset* indicates that many connections may be layered over a single IP node. Now we want to express the fact that the congestion problem causes both the local symptom PcketDiscardHigh, and propagates those discards as losses in the higher level connection:

```
problem Congestion “High congestion” =
                PacktDiscardsHigh 1.0,   connectionPacketLossHigh 0.8;

propagate symptom ConnectionPacketLossHigh =
                TransportConn, Uncerlying, PacketLossHigh;
```

We have added the symptom ConnectionPacketLossHigh to Congestion problem with a causal probability of 0.8, where a value of 1.0 indicates complete certainty. This indicates that the congestion at the IP node may not cause packet losses on all connections above it, depending on the circumstance surrounding the congestion. We would not want to rule out congestion simply because a single connection which is layered over the node is not experiencing problems.

The *propagate symptom* statement says that the symptom ConnectionPacketLossHigh refers to an event in a ralted obect, namely the event PacketLossHigh in any TransportConn which layered over this IP node. The MODEL code that propagates the problem to its observable symptom in the multimedia layer can be presented as follows:

```
interface TransportConn
{
        propagate symptom PacketLossHigh =
                        Port, ConnectedTo, PacketLossHigh;
}
interface UDPPort: Port
{
        propagate symptom PacketLossHigh =
                        Appl, Underlying, PacketLossHigh;
}
{
interface MM_InPort: Appl
{
        instrumented attribute long MinRate;
        instrumented attribute long MaxRate;
        instrumented attribute long MsgCounter;
        instrumented attribute long ActTime;

        computed attribute ActualRate = (MsgCounter)/(_time – ActTime);
        event BadRate = (MinRate > ActualRate) || (_time – ActTime);
        problem PacketLossHigh = BadRate 1.0;
}
```

Note that a Transportconn simply propagates the packet loss to the ports to which it is connected; a UDP port (which, being a subclass of Port, inherits from Ports) in turn propagates the packet losses to Applications which are LayeredOver the port. For simplicity, the relationships which are utilized for this propagation, ConnectedTo and Underlying, are not defined here. Typically they would be inherited from generic link and node classes in the Netmate hierarchy, which is described later.

The multimedia receive port, MM_InPort, is a subclass of Appl. Therefore, it receives, via inheritance, the PacketLoss symptom form the UDP_Port which it is layeredOver. The PacketLossHigh event in the MM_InPort has a single locally defined symptom, thus we again utilize the *problem* statement to define its symptom. In this case, PacketLossHigh causes the observable symptom BadRate, which indicates the reception rate is out of tolerance. Since this symptom is observable, it is defined using the *event* statement and an expression to detect the symptom.

The MODEL language can also express the one-to-many relationships. For example, suppose that there were many multimedia connections over the same congested router (possible causing the congestion). In this case, there will be many UDP connections (subclass of TransportConn) layered over the single IP object. The congestion problem may cause symptoms in any or all of the connections which are layered over the IP object.

A.3 Class Libraries in MODEL

In MODEL development, a three stage modeling process works best. In the first stage, a generic library of networking classes is used to define the basic relationship between objects in any modeled system. This set of classes is called the *Netmate hierarchy* is depicted in Fig. A.2.

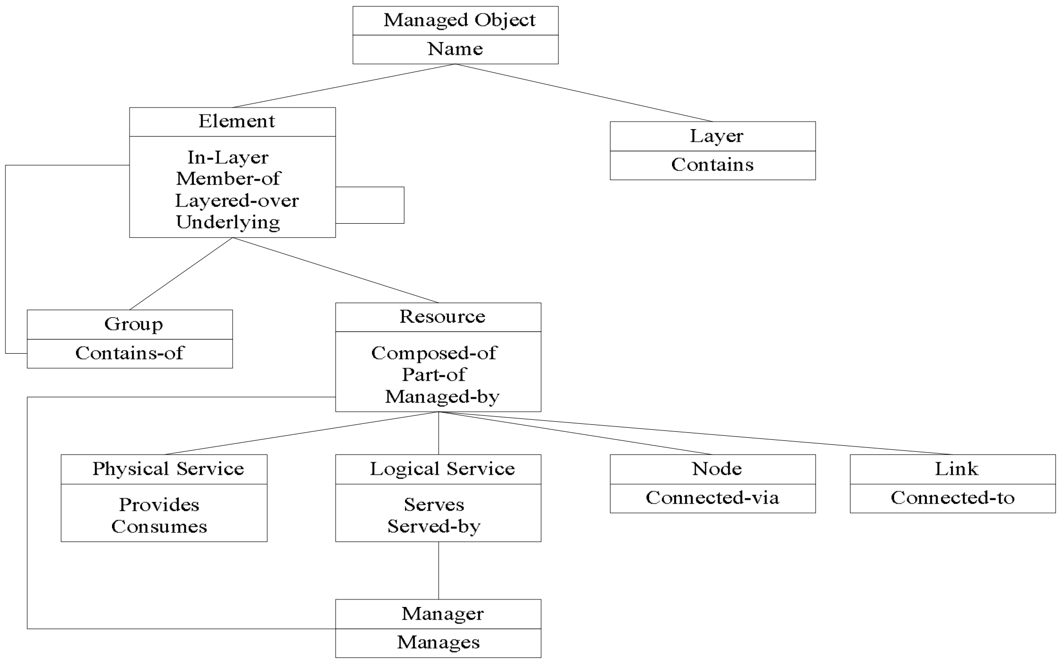


Figure A.2: Netmate calss hierarch.

The next stage consists of data modeling. Data modeling involves deriving domain specific classes from the Netmate classes and adding the appropriate attribute and instrumentation statements to produce an accurate data model of the domain. The third stage involves adding the actual event propagation information to the model, either directly into the second stage data model, or into subclasses of this model. At this stage, it may be necessary to add additional relationship and attributes to the data model, if it is seen that event propagation occurs over relationship that were not contemplated in the Netmate model., or that important events can not monitored in the original data model.

Using this methodology, a Multimedia QoS management library can be developed. Fig. A.3 illustrates the class hierarch of the Multimedia library. The “root” node is actually the resource class of the Netmate class library. The attributes of classes in the library are instrumented via the OoSMIB, which provides quality of service metrics that are important to diagnosing problems in the multimedia domain.

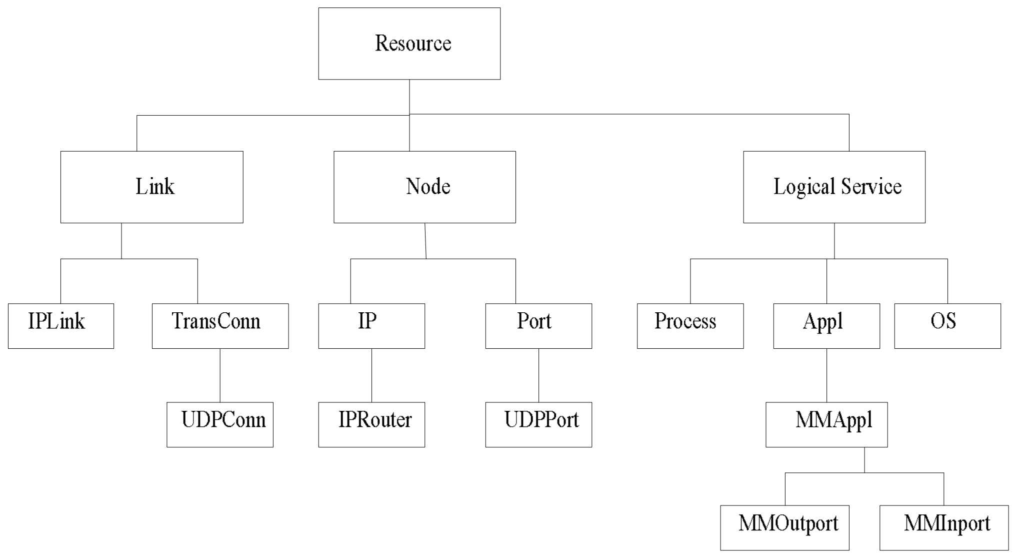


Figure A.3: Multimedia Class Hierarchy.

Appendix B

Belief Updating Algorithm

This appendix presents Pearl's belief updating algorithm. Pearl's algorithm is distributed in nature. Each node (RV) in the Bayesian network is considered as an individual processor. The network nodes are expected to perform local computations and communicate their results only to their neighboring nodes. A typical of a singly connected Bayesian network is shown in Fig. B.1. The messages to be passed between neighboring nodes are also illustrated. As shown in this figure, node X has n parents $U_1,...,U_n$, m children, $Y_1,...,Y_m$. The conditional probability $P(x \mid u_1,...,u_n)$ quantitatively relates the node X to its parents.

Let W_{XYj}^{-} denote the evidence contained in the sub-network on the head side of the arc $X \rightarrow Y_j$, and W_{UiX}^{+} denote the evidence on the tail side of the arc $U_i \rightarrow X$. The total evidence is given by $W = \{W_X^-, W_X^+\}$, where $W_X^- = \{W_{XY1}^-,...,W_{XYm}^-\}$ and $W_X^+ = \{W_{U1X}^+,...,W_{UmX}^+\}$. Note that for singly connected networks, all W_{XYi}^- and W_{UiX}^+ are disjoint.

In Fig. B.1, the π message

$$\pi x(u_i) = P(u_i \mid W_{UiX}^+) \tag{B.1}$$

is the current strength of the causal support contributed by incoming arc $U_i \to X$, and the λ message

$$\lambda_{Yj}(x) = P(W^{-}_{XYj} \mid x) \tag{B.2}$$

is the current strength of the diagnostic support contributed by each outgoing arc $X \to Y_j$.

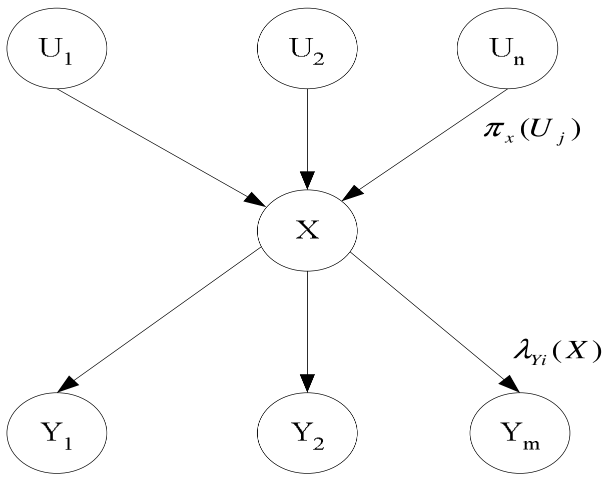


Figure B.1: A typical node X in a Bayesian network.

Pearl's belief updating algorithm [39]

A node *X* is activated when it receives the π messages from its parents, λ the messages from its children, or node itself is instantiated for a specific value *x*. Upon activation, *X* performs the following steps in any order.

Step 1: Belief updating. The node *X* updates its belief measure to

$$b(x) = \alpha\lambda(x)\pi(x) \tag{B.3}$$

If *X* is not instantiated, the $\lambda(x)$ and $\pi(x)$ messages can be calculated by:

$$\lambda(x) = \prod_j \lambda_{Yj}(x) \tag{B.4}$$

$$\pi(x) = \sum_{u_1,\ldots,u_n} P(x \mid u_1,\ldots,u_n) \prod_i \pi_x(u_i) \tag{B.5}$$

Otherwise, if *X* is instantiated for *x,*

$$\lambda(x) = \pi(x) = 1 \tag{B.6}$$

or if *X* is instantiated but not for *x*,

$$\lambda(x) = \pi(x) = 0 \tag{B.7}$$

α is a normalizing constant rendering $\sum_X b(x) = 1$.

Note that (B.3)-(B.7) implies that $b(x) = 1$ if *X* is instantiated with value *x* and 0 if *X* is instantiated with values other than *x*.

Step 2: Bottom-up propagation. The node *X* computes new λ messages and posts them to its parents:

$$\lambda_X(u_i) = \sum_x \lambda(x) \sum_{u_k:k\neq i} p(x \mid u_1,\ldots,u_n) \prod_{k\neq i} \pi(u_k) \tag{B.8}$$

Step 3: Top-down propagation. The node *X* computes new π messages and posts them to its children. If *X* is not instantiated, then

$$\pi_{Yj}(x) = b(x) / \lambda_{Yj}(x) \tag{B.9}$$

Otherwise, if *X* is instantiated for *x*,

$$\pi_{Yj}(x) = 1 \tag{B.10}$$

or if *X* is instantiated but not for *x*,

$$\pi_{Yj}(x) = 0 \tag{B.11}$$

This algorithm needs to be initialized by the following procedures:

1. Set all λ values, λ messages, and π messages to 1.
2. For all roots *U*, set $\pi(u) = P(u)$.
3. For all roots *U* and all children *X* of *U*, the node *U* posts new messages to *X*. If U is not instantiated:

$$\pi_x(u) = P(u) \tag{B.12}$$

Otherwise, if *U* is instantiated for *u*, then

$$\pi_x(u) = 1 \tag{B.13}$$

or if *U* is instantiated but not for *u*, then

$$\pi_x(u) = 0 \tag{B.14}$$

Initially, when no evidence is available, the probability distribution embedded in the Bayesian network is in equilibrium. Upon the instantiation of a node (i.e., the arrival of a new piece of evidence), the equilibrium state is broken. In order, for the network to enter a new equilibrium state (i.e., the belief functions converge to their true values), the number of belief updates to be performed by each node is proportional to the diameter of the Bayesian network [39].

Appendix C

Simulation Software

This summary review outlines the simulation software which has been used to generate random network topologies of different sizes and create probabilistic fault propagation models for each obtained network topology.

C.1 Class Libraries

The main classes of the simulation model are shown in Fig. C1. It basically consists of three classes namely, Components, Network, and Manager. The Components class defines the network physical entities (such as routers and switches) and logical entities (such as TCP connections) that comprise the managed network. The Network class defines the link connectivity among the network physical entities and each path two network nodes may use to exchange data utilizing certain routing algorithms. The Manager class defines the responsibilities of the network manager including obtaining fault propagation model for the current network configuration, assigning prior probabilities, analyzing received alarms, etc. The relationship between the Network class and the Components class is one-to-many as a single network topology may contain random number of network components. However, a network may have only one manager; thus, the relationship between the Network class and the Manager class is one-to-one. In the following, a summary of the main functions of each class is presented.

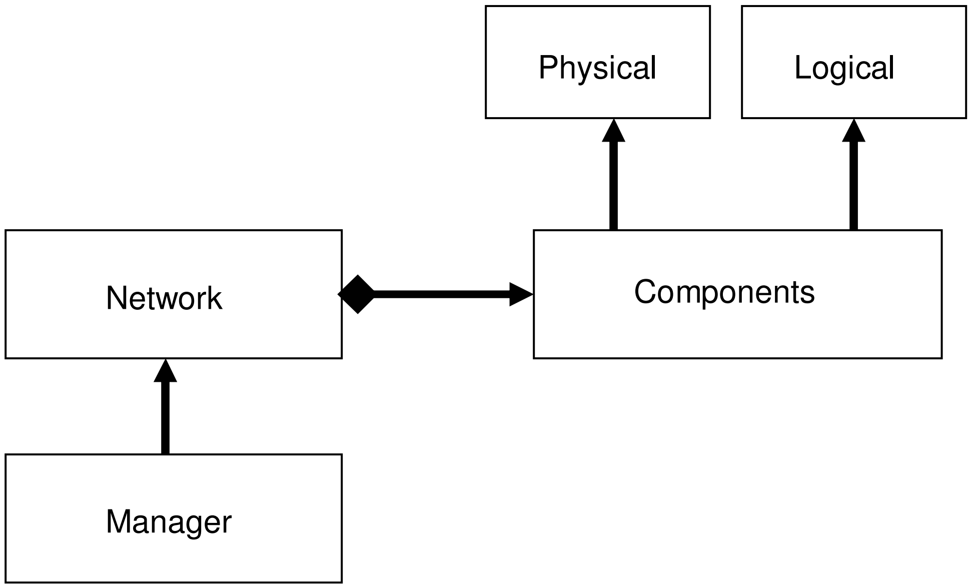


Figure C.1: The main classes of the simulation software.

C.2 The Components Class

Since we only consider problems in lower layers and the network topology is tree-shaped, the instances of the network Components class are restricted to the physical entities in the second protocol stack, namely bridges. Therefore, we use the following Bridge class to implement the Components class. The bridges will exchange configuration messages which have the following form:

```
struct Message
{
 int root;
 int cost;
 int transmitter;
 int port;
 Message (int id = 0) :root(id), cost(0), transmitter(id), port(0)
 { }
 };
```

The bridge class presents the following basic public functions which its instants need to communicate with each other:

```
class Bridge
{
 private:
    Message config;
 public:
     Bridge ( int id = 0, int co = 0, int trn = 0, int po = 0)
     {
     config.root        = id;
     config.cost        = co;
     config.transmitter = id;
     config.port        = po;
      }
      void set (Message msg)
        {
          config.root        = msg.root;
          config.cost        = msg.cost;
          config.transmitter = msg.transmitter;
          config.port        = msg.port;
        }
      void send(Message);
      Message getConfigM()
       { return config;  }
      void showConfig() const
       {
        cout<<"\nRoot ID     Cost        Transmitter           Port\n";
        cout<<"---------------------------------------------------\n";
cout<<setw(5)<<config.root<<setw(10)<<config.cost<<setw(15)<<config.transmitter
<<setw(20)<< config.port;
        cout<<endl;
       }
};
```

The bridges exchange these configuration messages among themselves and modify their parameters based on the spanning routing protocol until they reach a stabilized state. Hence, each bridge is expected to send and receive these messages to and from its neighbors using the *send()* function. The *send()* function is implemented as follows:

```
void Bridge::send(Message msg)
  {
        if (config.root > msg.root)
         {
          config.root = msg.root;
          config.cost = msg.cost+1;
          config.port = msg.transmitter;
          }
         else if ( config.root == msg.root && config.cost > msg.cost)
         {
          config.root = msg.root;
          config.cost = msg.cost+1;
          config.port = msg.transmitter;
          }
 }
```

C.2 The Network Class

The following Network class encapsulates the dynamic behavior of the simulation algorithm:

```
class Network
{
protected:
    Bridge Bridges[MAX];
    Message msg[MAX];
    Host   Hosts[MAX][MAX];
    int CONN;
    int Top[MAX][MAX];
    int nodeID[MAX];
    void initializeTop();
    void generateTop();
    void initializeBridges();

    bool randPerm(int start, int end, int range, int* result);
    void printToplogy1(int [][MAX]);
    void run();
public:
    Network()
    { run(); }
    void printToplogy();
    void printNodeID() const;
    void printRoutingTable() const;
};
```

Due to the limited space, we will only focus on the most important functions provided by the Network class and ignore the less important ones. The topology of the generated network is stored in the private variable, *Top[MAX][MAX].* The MAX constant refers to the network size, i.e., MAX=n. The network class constructor contains the function *run()*, which is basically an initialization function and is described as follows:

```
void Network:: run()
{
 randPerm(10, 1000, MAX, nodeID);
 generateTop();
 initializeBridges();
 initializeHosts();
 }
```

Each node in the generated network is assigned an ID which stands for its hard coded Ethernet address. One of the most important tasks of the class constructor is to run the *generatTop()* function which actually creates a random network topology, given the network size *n (MAX)* as follows:

```
void Network::generateTop()
{
   initializeTop();
   int connect[MAX];
   // CONN is the degree of Network Connectivity; CONN = N network is fully
   //connected. As CONN decreases, the network becomes less conncetd, however,
   //CONN should never be equal to zero (i.e., the network is completely not
   //  connected).
   CONN = MAX % 2;
   if (CONN == 0) CONN = MAX/2;
   else CONN = (MAX+1)/2;
   //----------------------------------------------------------------------
   //              Generate random network topology
```

```
  for (int j=0; j<MAX; j++)
   {
    for (int i=j+1; j<MAX; j++)
     {
       randPerm(0, MAX, MAX, connect);
         for (int k=0; k<CONN; k++)
          {
             if (connect[k] >=i)
              {
             Top[j][connect[k]]=1;
             Top[connect[k]][j]=1;
              }
          }
       }
     }
}
```

The degree of connectivity of the generated network may be ranged from full connectivity (i.e.; each node in the network is connected to every other node) to a partial connectivity.

C.3 The Manager Class

The main functions of the Manager class are to set the hosts and run the spanning routing algorithm if any change occurs in the network configuration.

```
class Manager
{
protected:
    int Manager_ID;
   int spannigTree[MAX][MAX];
```

```
    void setSpanningTree();
    void initSpanningTree();
   void printSpanningTree();
    void runBridges();
    void setHosts();
   void showHosts();
   void run();
public:
    Manager()
    { run(); }
   };
```

The constructor function of the Manager class is defined as follows:

```
void Manager:: run()
{
runBridges();
 setSpanningTree();
 setHosts();
 initializeHosts();
 }
```

The dynamic of the spanning tree routing protocol, shown below, is based on the algorithm proposed in [47], which is widely regarded as the industry standard of the data link layer routing protocol. The bridges of a given network are first initialized with random Ethernet addresses; however, they are assigned an identical rout cost. The network manager may activate the spanning tree algorithm on the network bridges at any time by simply invoking the function *runBridges()* :

```
void Manager::runBridges()
{
  for (int k =0; k<MAX; k++)
  {
   for (int i = 0; i<MAX; i++)
        {
        for (int j = 0; j<MAX; j++)
          {
            if (i != j && Top[i][j] != 0)
              {
                 Bridges[i].send(Bridges[j].getConfigM());
                 Bridges[j].send(Bridges[i].getConfigM());
              }
          }
        }
   }
 }
```

The outcome of the previous exchange of messages among network bridges determines the tree-shaped topology of the generated network. Based on the information stored in network bridges after running the spanning tree algorithm the new topology is formed as follows:

```
void Manager::setSpanningTree()
{
      initSpanningTree();
      Message msg1;
      for (int i = 0; i<MAX; i++)
      {
           int nodeIndex =MAX ;
```

```
            msg1 = Bridges[i].getConfigM();
            for (int k = 0; k<MAX; k++)
               {
                if (nodeID[k] == msg1.port) nodeIndex = k;
               }
            if (nodeIndex != MAX)
               spannigTree[i][nodeIndex] = 1;
            else  spannigTree[i][i] = 0;
        }
        for (int i=0; i<MAX; i++)
         for (int j=0; j<MAX; j++)
           if ( spannigTree[i][j] == 1) spannigTree[j][i] = 1;
}
```

The generated tree-shaped network topology is stored in the SapanningTree private variable.

Bibliography

[1] M. Brodie, I. Rish, S. Ma, *Optimizing probe selection for fault localization*, In the 12th International Workshop on Distributed Systems Operations Management, 2001.

[2] M. Brodie, I. Rish, S. Ma, G, Grabarnik, N. Odintsova, *Active probing*, Technical Report IBM, 2002.

[3] M. Natu, A. S. Sethi, *Active probing approach for fault localization in computer network*", In E2EMON'06, Vancouver, Canada, 2006.

[4] M. Natu, A. S. Sethi, *Efficient probing techniques for fault diagnosis*, Second International Conference on Internet Monitoring and Protection, IEEE, 2007.

[5] M. Brodie, I. Rish, S. Ma, N. Odintsova, A. Beygelzimer, G. Grabarnik, K. Hernandez, *Adaptive diagnosis in distributed systems*, Technical Report IBM, 2002.

[6] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, D. Ohsie, *High speed and robust event correlation*, IEEE communications Magazine 34 (5) (1996) 82-90.

[7] R. Gardner, D. Harle, *Alarm correlation and network fault resolution using Kohonen Self-Organizing map*, Globecom 97 proceedings, pp. 1398-1402, 1997.

[8] A. T. Bouloutas, G. W. Hart, M. Shwartz, *Fault identification using a FSM model with unreliable partially observed data sequences*, IEEE Transactions on Communications, 41(7):pp. 1074-1083, 1993.

[9] C. Wang, M. Schwartz, *Identification of faulty links in dynamic-routed networks*, IEEE Journal on Selected Areas in Communications,11 (3) 1449-1460, 1993.

[10] ISO, Information Processing Systems-OSI, ISO Standard 9596-1: Common management information protocol, part 1: Specification.

[11] Rajeev Gopal, *Layered model for supporting fault isolation and recovery*, in Proc. Of Network Operation and Management Symposium (72), pp. 729-742, 2000.

[12] W. Stallings, SNMP, SNMPv2 and CMIP: The practical Guide to Network Management Standards, Addison Wesley, Reading, MA, 1993.

[13] J. D. Case, K. McCloghrie, M. T. Rose, S. Waldbusser, Protocol operations for version 2 of the simple network management protocol (SNMPv2), IETF Network Working Group, RFC 1905, 1996.

[14] Z. Wang, *Model of network faults*, Integrated Network Management I, North-Holland, Amsterdam, 1989, pp. 345-352.

[15] A. T. Bouloutas, S. Calo, A. Finkel, *Alarm correlation and fault identification in communication networks*, IEEE Transaction on Communications, V. 42, pp. 523-533, 1994.

[16] R. Deng, A. Lazar, *A probabilistic approach to fault diagnosis in linear lighwave networks*, IEEE Journal on Selected Areas in Communications, V. 11, N. 9, 1438-1448, 1993.

[17] M. Steinder, A. Sethi, *A survey of fault localization techniques in computer networks*, Elsevier, Science of Computer Programming 53 (2004) 165-194.

[18] M. Steinder, A. Sethi, *Probabilistic fault diagnosis in communication systems through incremental hypothesis updating*, Elsevier, Computer Networks 45 (2004) 537-562.

[19] P. Hong, P. Sen, *Incorporating non-deterministic reasoning in managing heterogeneous network faults*, Integrated Network Management II, 1991, pp. 481-492.

[20] M. Steinder, A. S. Sethi, *Non-deterministic fault localization in communication systems using belief networks*, IEEE/ACM Transactions on Networking, 2004.

[21] A. T. Bouloutas, S. Calo, A. Finkel, I. Katzela, *Distributed fault identification in communication networks*, Journal of Network and System Management, vol. 3, no. 3 , 1995.

[22] I. Katzela , A. T. Bouloutas, S. Calo, A. Finkel, *Centralized vs distributed fault localization*, Inetegrated Network Management IV, 1995, pp. 250-261.

[23] G. Shafer, A mathematical theory of evidence, Princeton University Press, Princeton, NJ, 1976.

[24] C. S. Hood, C. Ji, *Proactive network fault detection*, IEEE Transaction on reliability, 46(3), 1997.

[25] C. Chao, D. Yang, A. Liu, *A LAN fault diagnosis system*, Computer Communications 24 (14), pp. 1439-1451, Elsevier Science B. V., 2001.

[26] I. Rouvellou, G. Hart, *Automatic alarm correlation for fault identification*, Proc. IEEE INFOCOM 95, the Con. On Computer Communications, 1995, 553-561.

[27] C. Wang, M. Schwartz, *Fault detection with multiple observers*, IEEE INFOCOM proceedings, pp.2187-2196, 1992.

[28] C. S. Li, R. Ramaswami, *Fault detection and isolation in transparent all-optical networks*, IBM research report, RC-220028, 1995.

[29] S. Rabie, D. Rau-Chaplin, T. Shibahara, *DAD: a real-time expert system for monitoring of data packet networks*, IEEE Network, pp. 29-34, 1988.

[30] T. E. Marques, *A symptom-driven expert system for isolating and Correcting network faults*, in Expert System Applications in Integrated Network Management (E. C. Ericson, L.T. Ericson, and Minoli, eds.), pp. 251-258, Artech Mouse, 1989.

[31] W. Fuller, *Network management using expert diagnostics*, International Journal of Network Management, 199-208, 1999.

[32] LI Jing-hua, XU Guang-hui , *A new network Management framework design and application realization,* Proceedings of the Sixth International Conference on Parallel and Distributed Computing, Applications, and Technologies, IEEE Computer Society, 2005.

[33] K. McCloghrie, M. Rose, Management information base for network management of TCP/IP-based internets: MIB-II, IETF Network Working Group, RFC 1213,1991.

[34] R. H. Deng, A. A. Lazar, W. Wang, *A probabilistic approach to fault diagnosis in linear lightwave networks*, Integrated Network Management III, Northern-Holland, Amsterdam, pp. 697-708 [36], 1993.

[35] P. Smyth, *Markov monitoring with unknown states*, IEEE Journal on Selected Areas in Communications 12 (9) 1600-1612, 1994.

[36] D. A. Ohsie, Modeled abductive inference for event management and correlation,

Ph.D. dissertation, Columbia University, 1996.

[37] W. Tracxyk, *Probes for fault localization in computer networks*, Journal of Telecommunications and Information Technology, 2004.

[38] D. Ohsie, A. Mayer, S. Kliger, S. Yemini, *Event modeling with the MODEL language, Integrated Network Management*, Northern-Holland, Amsterdam [99]. pp. 625-637.

[39] J. Pearl, Probabilistic reasoning in intelligent systems: networks of plausible inference, Morgan Kaufmann Publisher, 1988.

[40] A. Mohamed, O. Basir, *A new probing scheme for fault detection and identification*, IEEE International Conference on Electro Technology, 2009.

[41] E. Freuder, *Synthesizing constraint expressions,* Communications of ACM 21, 958-966, 1978.

[42] Y. C. Law, Using constraints to break value symmetries in constraint satisfaction problems, PhD Dissertation, The Chinese university of Hong Kong, 2005.

[43] Z. Ruttaky, *Fuzzy constraint satisfaction*, IEEE Conference on fuzzy systems, 1994.

[44] A. Mohamed, O. Basir, *A new fuzzy adaptive approach for fault identification in computer networks*, accepted in the 10th IASTED International Conference on Artificial Intelligence and Applications (AIA 2010), 2010.

[45] I. Katzela, M. Schwartz, *Schemes for fault identification in communication networks*, IEEE Transactions on Networking, 3, number 6, pages 733-764, 1995.

[46] M. Hassan, B. Sugla, R. Viswanathan, *A conceptual framework for network*

management event correlation and filtering systems, IFIP/IEEE, pp. 233-246, 1999.

[47] A. Mohamed, O. Basir, *Fusion based approach for distributed alarm correlation in computer networks*, accepted in the International Conference on Communication Software and Networks (ICCSN 2010), 2010.

[48] H. Zhu, O. Basir, *A scheme for constructing evidence structures in Dempster-Shafer evidence theory for data fusion*, Proc. 5th IEEE int. Symp. Computational Intelligence in Robotics and Automation, 2003.

[49] G. Jakobson, M. Weissman, *Alarm correlation*, IEEE Network, 52-59, 1993.

[50] H. Zhu, Data fusion using neuro-fuzzy embedded evidential reasoning, Ph.D. dissertation, University of Waterloo, 2004.

[51] A. Mohamed, O. Basir, *An adaptive multi-agent approach for distributed alarm correlation and fault identification*, accepted in the 9th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2010), 2010.

[52] E. Ekaette, B. Far, *A framework for distributed fault management using intelligent software agents*, Proc. IEEE Canadian Conf. on Elect. and Comp. Engineering, V. 2, pp. 797-800, 2003.

[53] L. Ho, S. Cavuto, S. Papayassiliou, A. G. Zawadzki, *Adaptive anomaly detection in transaction-oriented networks*, Journal of Network and Systems Management, V. 9, N. 2, pp. 139-160, 2001.

[54] B. L. Hitson, *Knowledge-based monitoring and control: an approach to understanding the behavior of TCP/IP network protocols*, ACM 0-89791-279-

9/88/008/0210, 1988.

[55] D. Wu, *Information exchange protocol: a new approach for future network management*, Proc. IEEE Infocom Conf.: Network Management , pp. 546-552, 1995.

[56] K. McCloghrie, M. Rose, Management information base for network management of TCP/IP-based internets: MIB-II, IETF Network Working Group, RFC 1213, 1991.

[57] H. Tran, J. Shonwalder, *Distributed cased-based reasoning for fault management*, Springer-Verlag Berlin Heiderlberg, 2007.

[58] Marina Thottan, C. Ji, *Properties of network faults*, in IEEE/IFIP Networks Operations and Management Symposium, pp. 941-942, 2000.

[59] S. Kandula, D. Databi, J. Vasseur, *Shrink: A tool for failure diagnosis in IP networks*, Proc. MineNet Workshop at SIGCOMM ACM, 2005.

[60] T. Zhang, S. Covaci, *The semantic of network management information*, Proc. IEEE INFOCOM Conf. , 1996.

[61] T. Zhang, P. G. Tsigaridas, *A Knowledge-based model for network service management*, Proc. of the first IEEE Symposium on Global Data Networking, 1993.

[62] J. Choi, M Choi, S. H. Lee, *An alarm correlation and fault identification scheme based on OSI managed object classes*, 1999 IEEE International Conference on Communications, pp. 1547-51, 1999.

[63] C. C. Lo, S. H. Chen, *Robust event correlation scheme for fault identification in communication networks*, International Journal of Communication Systems, V. 12, N. 3, 1998.

[64] N. Dawes, J. Altoft, B. Pagurek, *Network diagnosis by reasoning in uncertain nested evidence spaces*, IEEE Trans. Comm., V. 43, pp. 466-476, 1995.

[65] M. Albghdadi, B. Briley, M. Evens, R. Sukkar, M. Petiwala, M. Hamlen, *A framework for event correlation in communication systems*, Proc. of the IFIP/IEEE International Conference on Management of Multimedia Networks and Services, pp. 271–284, 2001.

[66] R. D. Gardner, D. A. Harle, *Methods and systems for alarm correlation*, Proc. of Globecom'96, pp. 136-140, 1996.

[67] T. Ndousse, T. Okuda, *Computation intelligence for distributed fault management in networks using fuzzy cognitive maps*, in Proc. IEEE ICC, pp. 1558-1562, 1996.

[68] J. L. Chen, P. Huang, *A fuzzy expert system for network fault management*, Proc. of IEEE International Conf. on Systems, Maintenance, and Cybernetics, V. 1, pp. 328-331, 1996.

[69] M. Thottan, C Ji, *Anomaly detection in IP networks*, IEEE Trans. On Signal Processing, V. 51, pp. 2191-2204, 2003.

[70] E. Aboelela, C. Douligeris, *Switching theory approach to alarm correlation in network management*, 25th IEEE International Conf. on Local Computer Networks, 2000.

[71] M. Yu, W. Li, L. Chung, *A practical scheme for MPLS fault monitoring and alarm correlation in backbone networks*, Computer Networks, Elsevier, pp. 3024-3042, 2005.

[72] L. Lewis, *A case-based reasoning approach to the management of faults in communications networks*, in Proc. IEEEINFOCOM, V. 3, pp. 1422-1429, 1993.

[73] L. Lewis, G. Dreo, *Extending trouble ticket systems to fault diagnostics*, IEEE Network, V. 7, pp. 44-51, 1993.

[74] L. Hua, X. Guang-hui, *A new network management framework design and application realization*, Sixth International Conference on Parallel and Distributed Computing, Applications, and Technologies, 2005.

[75] R. Stephan, P. Ray, N. Paramesh, *Network management platform based on mobile agents*, International Journal of Network Management, 14(1), pp. 59-73, 2004.

[76] C. Perng, D. Thoenen, G. Grabarnik, S. Ma, J. Hellerstein, *Data-driven validation, completion, and construction of event relationship networks*, Proc. of the Ninth ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining, pp. 729-734, 2003.

[77] C. Chao, D. Yang, A. Liu, *An automated fault diagnosis system using hierarchical reasoning and alarm correlation*, Journal of Network and Systems Management, V. 9, N. 2, pp. 183-202, 2001.

[78] Y. Nygate, *Event correlation using rule and object based techniques*, in Integrated Network Management IV, A. S. Sethi, F. Faure-Vincent, Y. Raynaud, pp. 278-289, 1995.

[79] R. Neapolitan, Probabilistic reasoning in expert systems: theory and algorithms, John Wiley and sons, Inc., New York, NY, 1990.

[80] Y. Breibart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, A. Silberschatz,

Toplogy discovery in heterogeneous IP networks, In Proc. Of IEEE INFOCOM, pages 265-274, 2000.

[81] B. Gruschke, *Integrated event management: Event correlation using dependency graphs*, in Integrated Network Management IV, A. S. Sethi, F. Faure-Vincent, Y. Raynaud,, pp. 130-141, 1995.

[82] J. F. Jordan, M. E. Paterok, *Event correlation in heterogeneous networks using the OSI management framework*, in Proc. IFIP/IEEE Communications Society, pp. 683-695, 1993.

[83] S. Katker, *A modeling framework for integrated distributed systems fault management*, Proc. IFIP/IEEE Internet. Conf. on Distributed Platforms, pp. 187-198, 1996.

[84] S. Katker, K. Geihs, *A generic model for fault isolation in integrated management systems*, Journal of Network and Systems Management V. 5, N. 2, pp. 109-130 1997.

[85] S. Katker, M. Paterok, *Fault isolation and event correlation for integrated fault management*, In: A. Lazar, R. Sarauo and R. Stadler, Editors, Integrated Network Management V, pp. 583-596, 1997.

[86] G. Liu, A. K. Mok, E. Yang, *Composite events for network event correlation*, in :M. Sloman, S. Mazumdar and E. Lupu, Editors, Integrated Network Management VI, IEEE, pp. 247-260, 1999.

[87] K. Appleby, G. Goldszmidt, M. Steinder, *Yamanja-a layered event correlation engine for multi-domain server farms*, IFIP/IEEE International Symposium on Integrated Network Management, 2001.

[88] A. Lazar, W. Wang, R. Deng, *Models and algorithms for network fault detection an identification: a review*, in Proc. IEEE International Contr. Conf., 1992.

[89] C. Chao, Y.Chen, A. Liu, *Abnormal event detection for network flooding attacks*, Journal of Information Science and Engineering, V. 20, pp. 1079-1091, 2004.

[90] M. Thottan, C. Ji, *Proactive anomaly detection using distributed intelligent agents*, IEEE Network, V. 12, pp. 21-27, 1998.

[91] Y. Yu, Q. Liu, L. Tan, *A graph-based proactive fault identification in computer networks*, Elsevier, Computer Communication, pages 366-378, 2005.

[92] Q. He, M. Shayman, *Using reinforcement learning for proactive network fault management*, Proc. of the International Conf. on Communication Technologies, 2000.

[93] R. Maxion, *Anomaly detection for diagnosis*, in 20th International Symposium on Fault-Tolerant Computing IEEE, 1990.

[94] A. Clemn, Network Management Fundamentals, Cisco Press, 2007.

[95] D. Comer, Internetworking with TCP/IP vol I: Principals, protocols, and architecture, Fourth Edition, Prentice Hall, 2000.

[96] G. Jackobson, M. D. Weissman, *Real-time telecommunications network management: Extending event correlation with temporal constraints*, IFIP/IEEE, 1995.

[97] S. Chutant, H. Nussbaumer, *On the distributed fault diagnosis of computer networks*, Technical Report 94/56, Ecole Polytechnique Federale de Lausanne, Lausanne, CH, 1994.

[98] A. Hanemann, D. Schmitz, M. Sailer, *A framework for failure impact analysis and*

recovery with respect to service and level agreements, Proc. of the IEEE International Conf. on Services Computing (SCC05), 2005.

[99] M. Mountzia, G. Rodosek, *Using the concept of intelligent agents in fault Management of distributed services*, Journal of Network and Systems Management, V. 7, N. 4, 1999.

[100] G. Goldszmidt, Y. Yemini, *Delegated agents for network management*, IEEE Communications Magazine, V. 36, N. 3, pp. 66-70, 1998.

[101] M. Albaghdadi, B. Briley, M. Evans, *Event storm detection and identification in communication systems*, Elsevier, Reliability Engineering and System Safety, 2006.

[102] A. Osmani, F. Krief, *Model-based diagnosis for fault management in ATM networks*, Proc. of International Conf. on ATM ICATM, pp. 91-99, 1999.

[103] N. Jailani, A. Patel, *FMS: a computer network fault management system based on the OSI standards*, Malaysian Journal of Computer Science, 1998.

[104] R. Cronk, P. Callahan, L. Bernstein, *Rule-based expert systems for network management and operations: an introduction*, IEEE Network Magazine, V. 5, N. 4, pp. 7-21, 1988.

[105] L. Crutcher, A. Lazar, *Management and control for giant gigabit networks*, IEEE Network, V. 7, N. 6, pp. 67-71, 1993.

[106] H. Lutifyya, M. Bauer, A. Marshall, D. Stokes, *Fault management in distributed systems: a policy-driven approach*, Journal of Network and Systems Management, 8 4 , pp. 499-525, 2000.

[107] A. Dupuy, S. Sengupta, O. Walfson, Y. Yemini, *Netmate: a network management environment*, IEEE Network magazine, 1991.

[108] S. Bapat, *Towards richer relationship modeling semantics*, IEEE Journal on Selected Areas in Communications, V. 11, N. 9, pp. 1373-1384, 1993.

[109] B. Pagurek, A. Kaye, D. Helmy, *Knowledge based fault location in a data communication network*, IEEE, 1988.

[110] L. Olson, A. Blackwell, *Understanding network management with OOA*, IEEE Network magazine, 1990.

[111] G. Tjaden, M. Wall, J. Goldman, C. Jeromnimon, *Integrated network management for real-time operations*, IEEE Network magazine, 1991.

[112] L. Barford, *Diagnosis and design for diagnosability for Internet routers*, Proceedings of the 7th International Symposium on Quality Electronic Design, IEEE, 2006.

[113] R. Perlman, Interconnections, Second Edition: Bridges, Switches, and Internetworking Protocols, Addison Wesley, 1999.