Fault Isolation based on Decision-Theoretic Troubleshooting

Jean-François Huard and Aurel A. Lazar
Department of Electrical Engineering
and
Center for Telecommunications Research
Columbia University, New York, NY 10027-6699

ABSTRACT

A decision-theoretic approach for fault isolation in broadband networks is presented. Our approach considers faults due to software and hardware as well as performance degradation and configuration problems. Belief networks are used to represent the relationships among various network entities. During a troubleshooting session, the network manager iteratively derives a sequence of tests based on the conditional probabilities, computed from statistics gathered (via alarms and tests) about the state of the network, and the costs associated with testing entities. An online dynamic programming technique is used to get the optimal sequence of tests. A system prototype that was implemented based on data from the XUNET testbed is also described.

Keywords: Fault Isolation, Fault Management, Decision-Theoretic Troubleshooting, Broadband Networks

February 15, 1996.

Center for Telecommunications Research Tech. Rep. CU/CTR/TR 442-96-08

Contact author:
Jean-François Huard
CTR/Columbia University
Room 801 Schapiro Research Bldg.
530 West 120th Street
New York, NY 10027
Tel: +1 212 939-7155
Fax: +1 212 316-9068
email: jfhuard@ctr.columbia.edu

Fault Isolation based on Decision-Theoretic Troubleshooting

Jean-François Huard and Aurel A. Lazar
Department of Electrical Engineering
and
Center for Telecommunications Research
Columbia University, New York, NY 10027-6699

1. Introduction

The task of fault management is to detect, isolate, and repair malfunctions in the network and its control subsystem. The first step, detection, can be thought of as an online process that gives indication of malfunctioning. Real-time detection mechanisms are usually implemented within the network protocols and devices. For example, protocol errors can be detected by finite-state machines [1]–[3]. The second step, consists of fault localization and identification. Fault localization is typically achieved through algorithms that compute a possible set of faults while fault identification is done by testing the hypothetical faulty component(s). The last step, repair is achieved by taking corrective actions. This step may need equipment replacement, change of system configuration, or software removal of bugs.

In fault detection deterministic finite state machine models are used that take a detailed view to all network entities. By contrast, in fault localization probabilistic models are used that look at the system from a global perspective. A number of algorithms that use probabilistic reasoning for localizing failed network components appeared in the literature [4]–[9].

In this paper, we developed a framework for a class of fault localization and identification problems that can be modeled using belief networks [10, 11]. When a fault occurs and the problem is reported to the fault manager (either through the triggering of an alarm or a by a report from the customer service center,) an event chain is propagated through the belief network leading to a set of probable faults. Once possible faults are localized, operations are performed to identify the faulty components or to obtain additional information for further reducing the set of probable faults.

Our approach considers faults due to software and hardware as well as performance degradation and configuration problems. During a troubleshooting session, the network manager iteratively derives a sequence of tests based on the conditional probabilities, computed from

statistics gathered (via alarms and tests) about the state of the network, and the costs associated with testing entities. A key feature of our modeling methodology is the partition of the nodes of the belief network into repairable and observable nodes. The latter are not eligible for repair; however, they provide, at a low cost, information about the system state. An online dynamic programming technique was developed for obtaining the optimal sequence of tests that minimizes the overall expected cost of the troubleshooting task. A system prototype (some 10K lines of C++ code) was implemented for evaluating data derived from the XUNET testbed.

In Section 2 we informally introduce our approach to modeling the network for fault isolation and describe our troubleshooting procedure. In Section 3 we formally present the fault isolation problem. Section 4 presents a prototype implementation on XUNET, a Gigabit testbeds deployed by AT&T in collaboration with several universities and research laboratories [12].

2. Network problem modeling

In order to clarify the class of fault isolation problems considered in this paper, let us consider the following fault scenario of a typical network file system (NFS) that arises in practice.

2.1 A fault scenario

The network manager receives an e-mail pointing to an NFS fault. Typically such problems are caused by a file server going down due to an operating system bug. A quick test, however, shows that the server is alive. The problem is also caused by a disconnected transceiver, a bad cable or a loose coax connection somewhere in the network. Using the ping utility, the physical connectivity between the NFS server and client is found to be on. May be a disk crashed? No. May be the network is congested? No. May be the demand on the server is to high (which causes the connection establishement timeout of the automounter to expire and to conclude that there is no connectivity)? No. Could the cause be the new kernel?

2.2 Graphical representation of the fault scenario

In this section, we devise a model for representing the fault scenario above. Ideally, we would like to come up with a model that captures the network as a whole, provides the ability to include physical, logical as well as virtual objects and provides the basis for various computational studies. Furthermore, in the process of modeling, it should become apparent

whether some essential variables have been omitted or, whether redundant variables have been inadvertently included.

To this end, we use a graphical representation of the fault isolation model. In short, the graphical representation depicts dependency relationships between different network components or events. Each graph contains a special node that indicates the functional status of the subsystem. For example the node labeled NFS in Figure 1 represents the status of the NFS filesystem (namely, OK or Not OK) for the fault scenario described above. We can

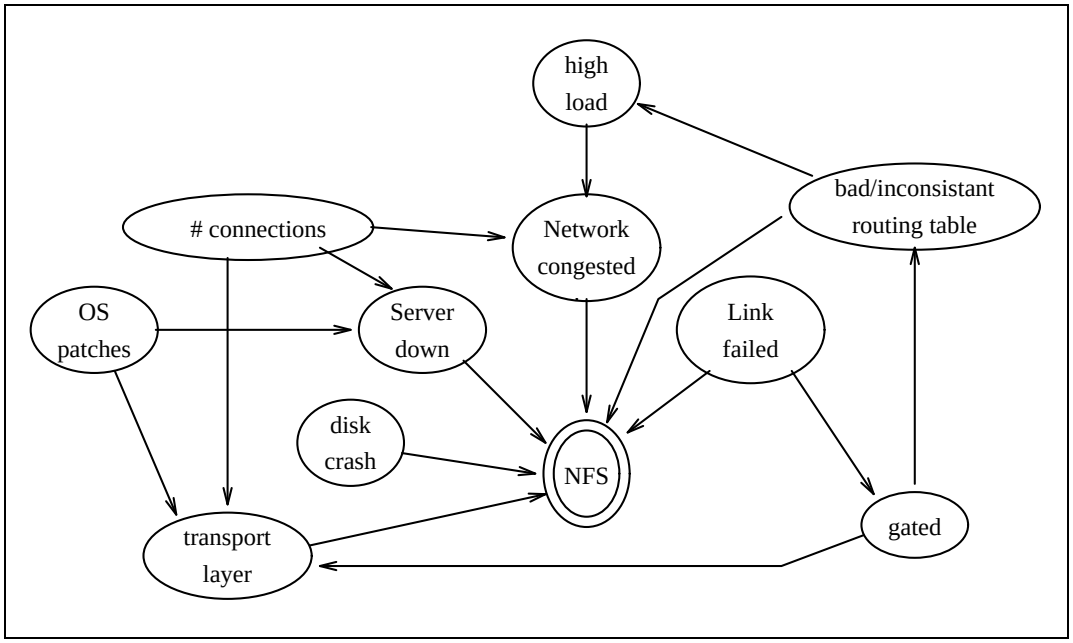

Figure 1: Influence diagram of the NFS problem.

see that the NFS fault problem might be caused by a bad disk, a server down, a congested network, non-connectivity at the physical layer or transport layer, etc.

In the diagram shown in Figure 1, a certain level of detail is maintained. The suppression of details in the model chosen is often supported by the computational efficiency gained by doing so. Moreover, omitting some details permits us to generalize some of our conclusions. A refinement of the diagram can be achieved by decomposing some possible causes of the NFS problem by adding finner levels of details such as the OS patches or the number of connections, etc.

As illustrated in the NFS problem scenario, our approach to fault management considers faults due to software and hardware as well as performance degradation and configuration problems. For example, problems where the degradation occurs gradually until an inefficiency state is reached, such as operating system memory leaks and packet losses that cause

congestion, can be incorporated. A similar approach was used in [8] to model the physical layer of a linear lightwave network. A simple example of a connection setup problem was presented in [7].

2.3 Quantitative representation of the fault scenario

In the process of modeling the NFS fault problem above, a number of details have been left out. These can be compensated for by associating weigths (probabilities) with the nodes in Figure 1. Thus, these probabilities represent a measure of uncertainty induced by the details left out by the fault model. Based on an independence assumption (d-separation principle [11],) and using Bayes' law, the set of a priori probabilities of the root nodes and the set of conditional probabilities relating neighboring nodes are sufficient to calculate the joint distribution of the variables of interest. This give rise to a model called belief (Bayesian, causal) network.

Belief networks are directed acyclic graphs on which a probabilistic model is superimposed. They model dependency (causal) relationships between different objects or events. The nodes of the diagram are propositional variables and the arcs indicate relationships between the propositions. Each node has associated with it, a set of conditional probabilities that relates it with its neighbors. For an intuitive introduction to belief networks the reader is referred to an excellent tutorial paper by Charniak [10]. (see Appendix A for a formal description of belief networks.)

2.4 Troubleshooting using belief networks

In the NFS scenario, some propositions are easily and cheaply tested. Examples are the number of connections active on the server, the number of mbufs available in the transport layer of the server, the physical connectivity, the network load, or the liveness of the server. Some other propositions require more time and might be more costly; for example which OS patch combination was used for the kernel or the inconsistency of the routing tables.

The diagnosis process is sequential by nature. At any stage, there are many observations, tests and repairs that can be applied. It is natural to take advantage of the information gathered on the way by testing propositions that might reduce diagnosis costs or give us high confidence in potential failures, etc. An example of such a proposition is the liveness of the server (tested by using the "ping" command) in the NFS fault scenario. The alternative of walking to a machine room or sending a technician in the field to verify if the machine is effectively alive is, of course, more costly. Our goal is to find a testing strategy that allows

us to identify a faulty network subsystem at minimal cost. When time is used as our cost function, then the objective is to minimize the expected time to identify a fault. If the costs are all unity, then the objective is to minimize the number of tests to find the fault.

3. Fault isolation problem

In this section, we formally present our mathematical model for solving the fault isolation problem introduced in Section 2. The problem is to find a plan of action that will minimize the expected cost of finding a faulty network component. The procedure of isolating a fault is iterative. In each iteration we compute a possible set of faults and test the hypothetical faulty component(s). We repeat this step until the faulty component is identified. The procedure leads naturally to a sequential optimization problem that exhibits a dynamic programing solution.

3.1 Representation of the problem

We begin by constructing a belief network as described in Section 2.3. Next, we partition the nodes of the belief network into three classes:

Status node: This node represent whether or not our system is working properly. In Figure 1, the status node is labeled NFS.

Repairable nodes: These nodes represent components of the system that can be repaired either through bug removal, software reinitialization or component change. Examples of such nodes in Figure 1 are "server down," "disk crash," "link failed," and "routing tables."

Observation nodes: These nodes carry information that can be obtained during the course of fault isolation. They are not eligible for repair. Although nodes in this class give us information about the system state, they do not confirm any system failure directly. For example, in Figure 1, the "number of connections" and "network congested" nodes are in this class.

For each node, we need to assess a testing cost that is assumed to be held fixed during the troubleshooting process. The testing cost of the repairable nodes will be taken to be the minimum of the labor cost for diagnosing a component to be faulty or of the cost of replacement of the component and the subsequent testing of the status node. The testing

costs of the observation nodes are assumed to be low when compared with the cost of repairables items. In particular, we expect some observation nodes to help us obtain low cost information that will lead to a high disbelief in faulty components with a high testing cost.

3.2 Diagnosis process

For simplicity, we develop the diagnosis process assuming that only a single fault can occur at any time. In such case, as soon as a repairable component fails its test, the fault isolation procedure terminates. We say that a component fails its test if it is diagnosed *not* OK (NOK) or if it is replaced and the state of the status node becomes OK. We assume throughout that any replaced component works properly.

The fault isolation procedure is iterative. We repeat the following iteration until a faulty repairable component is found: based on a decision rule, we select a node and test it; if the outcome is NOK and the node is repairable, we stop; otherwise, we update the joint distribution of the untested variables conditioned on the state of the tested variables. In each iteration, we gather one piece of information that constitutes the history of the process. Formally, we define the diagnosis process as follow:

(i). A finite collection of propositions, V. To each proposition, $v \in V$, we associate a random variable X_v that can take values in $\{T, F\}$. We denote by P the (joint) probability distribution of $X = \{X_v; v \in V\}$.

(ii). A time index set (or diagnosis horizon) $T = \{0, 1, \ldots, N\}$, with $N = |V|$.

(iii). A set of actions, A, taken to be the collection of propositions plus a stopping action, that is, $A = V \cup \{\texttt{stop}\}$. An action $a_k \in A, k \in T$, is performed by testing proposition a_k at the beginning of period k and by observing the outcome at the end of period k (which also corresponds to the beginning of period $k+1$). If the outcome of testing proposition a_k is OK, then $X_{a_k} = F$, otherwise $X_{a_k} = T$, which means that we just found the faulty component. It is assumed that the gathered information is perfect.

(iv). The history of the process at time k, I_k, is the information gathered by testing propositions $a_0, a_1, \ldots, a_{k-1}$. We denote by z_k the observation made at the imbedding time k (that is, the result of the test a_{k-1}). Therefore, we have

$$I_0 = z_0$$

$$I_k \;=\; (z_0,(a_0,z_1),(a_1,z_2),\ldots,(a_{k-1},z_k)) = (I_{k-1},(a_{k-1},z_k)).$$

For example, let's say that we started with no information and have tested propositions a, b and c, with respective outcome OK, OK, NOK. Before deciding which action to take at time $k=3$, we have the following information: $I_3 = (\phi,(a,F),(b,F),(c,T))$.

(v). A set of stopping propositions, $S \subset A$; that is, if proposition $a \in S$ is tested true, then the iterative process stops. The stopping set is the set of repairable nodes.

(vi). A decision process $\{D_k; k \in T\}$, where $D_k \in A$. This is what we are looking for.

(vii). A random process $\{S_k; k \in T\}$, where $S_k \in \mathcal{S} = \mathcal{P}(X) \cup \{\texttt{stop}\}$ is the state of the process at time k. The state space is assumed to be static during the entire diagnosis process. We will take the history I_k or `stop` as the state of our process, that is, $S_k \equiv I_k \vee \texttt{stop}$.

(viii). A collection of (time-homogeneous) one-step transitions probabilities $\{P_{x,y}(a); x,y \in \mathcal{S}, a \in A\}$ between the states. The transition probabilities represent the likelihood of going from state x to y given that we take action a.

$$P_{x,y}(a) = \begin{cases} \Pr\{X_a = F|x\} & \text{for } x \neq \texttt{stop},\ y = (x,(a,F)),\ a \neq \texttt{stop}, \\ \Pr\{X_a = T|x\} & \text{for } x \neq \texttt{stop},\ y = (x,(a,T)),\ a \notin S,\ a \neq \texttt{stop}, \\ \Pr\{X_a = T|x\} & \text{for } x \neq \texttt{stop},\ y = \texttt{stop},\ a \in S, \\ 1 & \text{for } x = \texttt{stop},\ y = \texttt{stop},\ a = \texttt{stop}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

(ix). A collection of cost functions $c(x,a), x \in \mathcal{S}, a \in A$ of the form

$$c(x,a) = \begin{cases} c(a) & \text{for } x \neq \texttt{stop}, \\ 0 & \text{for } x = \texttt{stop}. \end{cases} \tag{3.2}$$

The diagnosis process is a tuple made of the above components. The decision process, $\{D_k; k \in T\}$, represents our control, all the rest being fixed. Our goal is to find a decision rule and to select, during in each iteration of the troubleshooting phase, an action that minimizes the expected cost of isolating the fault.

3.3 Fault localization problem

The fault localization problem is to find a decision rule for selecting which network component to test in the next step of the isolation procedure so as to minimize the expected cost of finding a faulty network component. The iterative nature of the troubleshooting process leads to a sequential optimization problem that exhibits a dynamic programming solution. In this section we investigate various types of strategies and derive the recursive equation that is required for the dynamic program.

3.3.1 Static strategy

A *static strategy*, d, is a pre-determined combination (ordered sequence) of N different actions $d = \langle a_0, a_1, \ldots, a_{N-1} \rangle$, $a_k \neq \texttt{stop}$. The space of strategies contains all $N!$ permutations of tests. In this section, we show the form of the expected cost of a strategy, derive the equation in a recursive form and define what we call the best static strategy.

Expected cost of a static strategy

We denote by C^d the cost function of static strategy d. The function C^d is a random variable, that is, a function of the state of the network denoted by $X = \{X_v; v \in V\}$.

$$\begin{aligned} C^d \;=\; & c(a_0) + \\ & c(a_1)\,[1_{\{a_0 \in S\}} 1_{\{X_{a_0}=F\}} + 1_{\{a_0 \notin S\}}] + \\ & c(a_2)\,[1_{\{a_0 \in S, a_1 \in S\}} 1_{\{X_{a_0}=F, X_{a_1}=F\}} + \\ & \qquad 1_{\{a_0 \in S, a_1 \notin S\}} 1_{\{X_{a_0}=F\}} + \\ & \qquad 1_{\{a_0 \notin S, a_1 \in S\}} 1_{\{X_{a_1}=F\}} + \\ & \qquad 1_{\{a_0 \notin S, a_1 \notin S\}}] + \cdots + \\ & c(a_{N-1}) \left[\sum_{i=0}^{2^{N-1}-1} 1_{\{\langle a_0,\ldots,a_{N-2}\rangle = (i)_2\}} 1_{\{\langle X_{a_0},\ldots,X_{a_{N-2}}\rangle = \sim(i)_2\}} \right]. \end{aligned} \tag{3.3}$$

where $(i)_2$ is the binary representation of i, and $\sim(i)_2$ the bitwise complement of i. For $\langle a_0, \ldots, a_{N-2} \rangle = (i)_2$, if the jth bit corresponding to test j is one, j is in the set of stopping propositions. For $\langle X_{a_0}, \ldots, X_{a_{N-2}} \rangle = \sim(i)_2$, a 0 bit means a false outcome and a 1 bit means "do not consider the corresponding variable" because it is not in the stopping set.

The expected cost of strategy d, $E[C^d]$, has the same form as Equation (3.3), with the indicator functions replaces with the probability of the corresponding events. The structure

of the expression for the expected cost enables us to write it recursively.

Recursive formulation of the expected cost

Let us write $V_n^d(x)$ for the expected cost of the n last tests in d, that is, $\langle a_{N-n}, a_{N-n+1}, \ldots, a_{N-1}\rangle$, given that we start the diagnosis process with initial information x:

$$\begin{aligned} V_1^d(x) &= c(a_{N-1}) \\ V_n^d(x) &= c(a_{N-n}) + 1_{\{a_{N-n}\notin S\}} V_{n-1}^d(x) + \\ &\qquad 1_{\{a_{N-n}\in S\}} \Pr\{X_{a_{N-n}} = F|x\} V_{n-1}^d((x,(a_{N-n},F))) \\ &= c(a_{N-n}) + \Pr\{X_{a_{N-n}} = T|x\} V_{n-1}^d((x,(a_{N-n},T)))1_{\{a_{N-n}\notin S\}} + \\ &\qquad \Pr\{X_{a_{N-n}} = F|x\} V_{n-1}^d((x,(a_{N-n},F))). \end{aligned} \tag{3.4}$$

For example, for $N = 3$, we have

$$\begin{aligned} V_3^d(x) &= c(a_{N-3}) + \\ &\quad c(a_{N-2}) \cdot [1_{\{a_{N-3}\in S\}} \Pr\{X_{a_{N-3}} = F|x\} + 1_{\{a_{N-3}\notin S\}}] + \\ &\quad c(a_{N-1}) \cdot [1_{\{a_{N-3}\in S, a_{N-2}\in S\}} \Pr\{X_{a_{N-3}} = F, X_{a_{N-2}} = F|x\} + \\ &\qquad 1_{\{a_{N-3}\in S, a_{N-2}\notin S\}} \Pr\{X_{a_{N-3}} = F|x\} + \\ &\qquad 1_{\{a_{N-3}\notin S, a_{N-2}\in S\}} \Pr\{X_{a_{N-2}} = F|x\} + \\ &\qquad 1_{\{a_{N-3}\notin S, a_{N-2}\notin S\}}]. \end{aligned}$$

Setting $x = \emptyset$, and taking the expectation of Equation (3.3), then we get the same first three terms; that is $E[C^d] = V_3^d(\emptyset)$.

Example: all variables are in the stopping set

For this example, the recursive relation of Equation (3.4) reduces to:

$$\begin{aligned} V_1^d(x) &= c(a_{N-1}) \\ V_n^d(x) &= c(a_{N-n}) + \Pr\{X_{a_{N-n}} = F|x\} V_{n-1}^d((x,(a_{N-n},F))). \end{aligned}$$

For $N = 4$, applying recursively the same equation, we obtain:

$$V_4^d(x) = c(a_{N-4}) +$$

$$c(a_{N-3}) \cdot \Pr\{X_{a_{N-4}} = F|x\} +$$
$$c(a_{N-2}) \cdot \Pr\{X_{a_{N-4}} = X_{a_{N-3}} = F|x\} +$$
$$c(a_{N-1}) \cdot \Pr\{X_{a_{N-4}} = X_{a_{N-3}} = X_{a_{N-2}} = F|x\}.$$

With $d = \langle a, b, c, d\rangle$ and $z_0 = \emptyset$, then

$$V_4^d(\emptyset) = c(a) + c(b)\Pr\{X_a = F\} + c(c)\Pr\{X_a = X_b = F\} + c(d)\Pr\{X_a = X_b = X_c = F\}.$$

Best static strategy

The best static strategy d^*, is the one that minimizes the expected cost; that is, $d^* = \arg\min_d E[C^d]$. Despite the fact that the formulation takes into account the possibility of testing non-repairable nodes, these actions should not be considered in d if the number of actions is less than the size of the stopping set. This is because no matter what the outcome of the observation nodes, these are not considered for the choice of the subsequent tests. In order to guarantee finding the fault, the set of actions is given by the stopping set.

The best strategy will be given by the order of actions defined by an increasing sequence $c(a_i)/\Pr\{X_{a_i} = T\}$, $a_i \in S$ [13]. For simplicity, let us write $p_i = \Pr\{X_{a_i} = T\}$ for the probability that test a_i has failed. From the mutually exclusive fault assumption, we have $\sum_{i=0}^{N-1} p_i = 1$. Furthermore, the probability that the kth variable needs to be tested is the probability that none of its predecessors have failed and is given by $\Pr\{X_{a_0} = \cdots = X_{a_{k-1}} = F\} = \sum_{i=k}^{N-1} p_i$. From this simple form of the joint probability distribution and from the interchange argument [14], the result follows trivially. The same result is obtained if the variables are independent.

3.3.2 Optimal solution by dynamic programming

The optimal solution to the fault isolation problem is an *optimal policy*, $\pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$ which specifies the optimal decision rules $\mu_k^* : \mathcal{S} \to A$ for each $k \in T$ that minimizes the expected cost achievable in each state.

Dynamic programming is a technique for finding the optimal policy of sequential decision processes like our diagnosis process. The basic underlying idea in dynamic programming is known as the principle of optimality. It states that *an optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.* [15]

Let us write $V_n(x)$ for the minimized expected cost when starting the diagnosis in state

x and n tests remain. V is also known as the value function and $V_n(x)$ represents the best *expected cost-to-go*. Our goal is to find V_n online, adjusting our policy dynamically, based on the information gathered by testing propositions. From Equation (3.4) and the principle of optimality, we have

$$V_1(x) = \min_{a \in A} [c(a)], \tag{3.5}$$

$$V_n(x) = \min_{a \in A} \left[c(a) + \sum_y P_{x,y}(a) V_{n-1}(y) \right]. \tag{3.6}$$

The major drawback of this approach is due to the discrete nature of the state space. To solve the problem, an explicit state space enumeration is necessary, which often makes the problem intractable. However, in practice, it is possible to implement efficient algorithms to search the state space rendering this approach feasible. Furthermore, in practice, several suboptimal searching methods, such as n-step lookahead, work well and give good results. Another issue which is hidden in the disscussion is the complexity for evaluating the conditional probabilities. In practice, using the method described in [16], the inference of probability is almost constant [17], even though the inference of an arbitrary belief network is NP-hard [18].

4. A prototype on XUNET

This section describes a prototype implementation of a fault management application on XUNET, one of the five Gigabit testbeds sponsored by the Corporation for National Research Initiatives. XUNET has been deployed by AT&T in collaboration with several universities and research laboratories [12]. The application is based upon the concepts of probabilistic reasoning and dynamic programming described in the previous sections. The prototype is one of the first fault management expert system implemented that uses statistical methods to isolate high-level faults in broadband networks. In Section 4.1, we give the overview of the system architecture. In Section 4.2 we describe the various steps taken towards the construction of the knowledge base. Finally, Section 4.3 presents some empirical results.

4.1 System architecture

The core of our fault isolation system is composed of two parts, an inference engine and a decision engine. The *inference engine* computes probabilities given the accumulated evidence (alarms and responses from queries in the network). It is based on Lauritzen and Spiegelhal-

ter's algorithm for computations of probabilities on a graphical structure [16]. The inference engine requires models that are contained in the knowledge base.

The *decision engine* is an online algorithm that decides which network object should be tested in the next phase of the diagnostic process. Its computation is performed using the probabilities provided by the inference engine. It is based on the concept of dynamic programming.

The *knowledge base* is an abstraction of the human understanding of network faults propagation and dynamics using causal relationships among various managed objects. This knowledge is modeled in the form of belief networks. The knowledge base contains multiple models, one for each typical problem that the manager may have to troubleshoot.

The *query interface* provides the facilities for the domain manager to acquire evidence about the network state and to communicate with other domain managers.

Finally, the *graphic user interface* presents a visual abstraction of the network problems, faulty components and diagnostic process for a human network operator in an informative way.

4.2 XUNET knowledge base engineering

In practice, the knowledge engineering is the most difficult part to realize in the development of an expert system because the knowledge engineer is usually not an expert in the field. When the knowledge base is not well designed, then the expert system does not work as expected. In particular, for this project, joint work was done with XUNET's designers and managers in order to construct the belief network. The steps taken toward the construction of the knowledge base were

(i). The *identification* of a set of faults that regularly occurs.

(ii). The *composition* of individual belief networks for the identified faults.

(iii). The *assignment* of probabilities for each belief network.

Each step is described in the following sections.

4.2.1 Identification of a set of faults and composition of belief networks

The starting point of the design of our belief networks is to identify a set of faults that occur (or has occurred) regularly, for which we understand the dynamics. Each belief network was composed at a time so as to focus our attention on a single problem at a time. In this way,

the belief networks became easier to build and to understand. In this project, we identified the following set of faults: *callsetup* between client and server; *no logfile mail* received at night; *queue module* in bad mode; *no board seen* in a switch slot; *configuration* out of date; and *Link down.* Once these problems were understood, a belief network was composed for each of them. In the following paragraphs we address only the *callsetup* and *no logfile mail* problems. The reader is referred to [19] for further details.

The *callsetup* problem (see figure 2) is similar to the NFS problem scenario in Section 2.1. On XUNET, this problem happens if the server is down, if the signalling code that is running is not compatible between some nodes, if too many channel connections are opened, if there is an intermediate hardware problem between the client and the server (because of the XUNET physical topology), and so on.

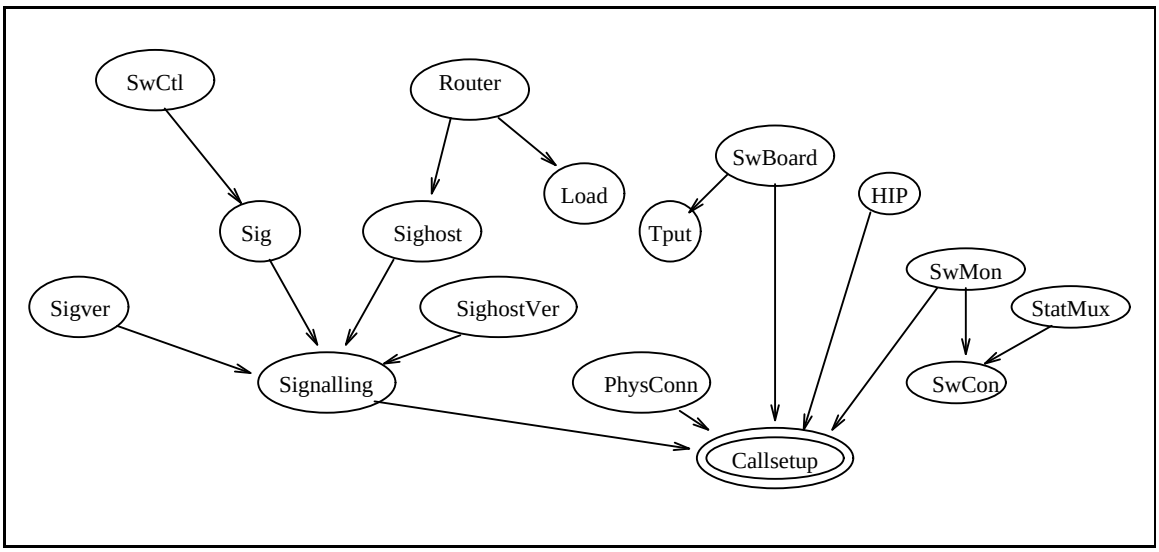

Figure 2: The *callsetup* belief network.

The *no logfile mail* problem (see Figure 3) has two forms. Every night, the daily logfile of the switch controllers are sent to the management center. In some case, no logfiles are received. This might be due to an Internet problem, to an XUNET link problem or it might be that the switch controller is not working properly and did not send the logfile. What might also happen, is that the daily logfiles are received, but the end of day status report is not reported in all of them. This would usually be caused by the absence of some switch monitoring processes that do not dump their report.

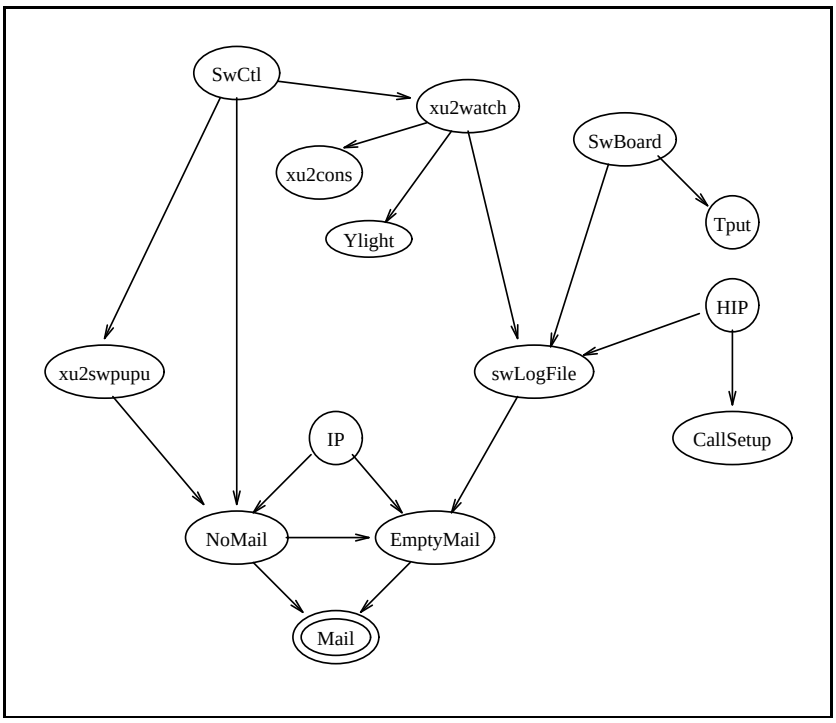

Figure 3: The *no logfile mail* belief network.

4.2.2 Probability assessment

The probability assessment was done for each belief network starting by assessing the *a priori* probabilities of all root nodes. Again, this work was done jointly with the system designers and managers. In order to facilitate the probabilities assessment, two approaches were used. The first one was to use a scale that assigns a probability interval to quantitative adjectives and adverbs [20] and the other one was simply to assign the probabilities directly. When the probabilities were unknown, they were assigned uniformly. The original diagrams were modified for consistency and complexity consideration by removing, clustering or adding some edges and nodes.

To validate our diagrams, some consistency tests were done. For example, if a router is down, then all the processes running on that router are dead with certainty. Therefore, the signalling should not be working. If the probabilities are assessed such that there is some probability that the signalling is working then the model is inconsistent. Moreover, if we confirm that the signalling is working, while it is not supposed to with probability one, then we have a consistency problem, and the wrong numerical results.

4.3 Empirical results

In this section, we present results obtained by applying our approach to troublehsooting problems occuring on XUNET. For a given problem, a large set of fault instances was generated using a "Monte-Carlo" method that we developped for this purpose. We compared the results of an omniscient, a static, a random and a decision-theoretic troubleshooter in the case of the daily switch log report. Other problems were tried out and results can be found in [21].

The main idea behind the simulation method was to use a belief network for generating instances by first assigning the status node the NOK outcome, and then selecting possibile faults at random. Then, we instantiated the nodes selected to their NOK outcome and all the other repairable nodes to their OK outcome and computed the joint distribution of the observation nodes given a repairable node instance. For all the observable nodes, we picked one at random, instantiated it based on its conditional marginal and recomputed the new marginal of the remaining nodes. We did the same until all nodes were instantiated.

Once we had a problem instance, we applied the troubleshooting method, recording the sum of the cost of each action. The method relies on an omniscient observer that reveals the outcome of an observation when a troubleshooter performs a test. In our approach, the belief network of the decision-theoretic troubleshooter is identical to the one used to generate instances of the problem. This assumption could be relaxed, however the models would required to be modified to accomodate a node characterized by an "unknown failure". The latter would have a testing cost equivalent to asking a human manager to perform the diagnosis for us. This node would typically have a high cost and low a priori probability.

In the following, we compare our dynamic decision-theoretic troubleshooter with a random, static and omniscient troubleshooters. The decision-theoretic troubleshooter can test both observable and repairable nodes while the others test only repairable nodes since they do not take advantage of the information gathering process along the sequence of tests. Two versions of the decision-theoretic troubleshooter is used. The first one computes the optimal decision in each step. The later is an heuristics that used a 4 step lookahead search. The static strategy is the best static strategy discussed in Section 3.3.1. The random troubleshooter test nodes at random (without repetition), based on the relative frequency of occurence of the failure of the repairable nodes. Finally, the omniscient troubleshooter tests right away the correct one, and is therefore the lower bound on the achievable by any strategy.

For the switch logfile mail problem, we generated 1000 instances of failure. Our decision-theoretic troubleshooter always perform better that the static and random troubleshooter

as can be seen in Figure 4. The distribution of cost for the random troubleshooter is almost uniform. For the static, the cost is distributed over 6, representing the cost of troubleshooting each failure. For the omniscient, the cost is either 10, 20 or 50, depending on the faulty component. Finally, for the optimal decision-theoretic troubleshooter, the cost is distributed on the between 0 and 130, while for the 4-step lookahead toubleshooter, the distribution is spreaded a little bit more. The average cost of troubleshooting was 29.9, 94.25, 93.23, 57.6 and 60.0 for the omniscient, static, random, decision-theoretic, 4-step lookahead decision-theoretic troubleshooters, respectively.

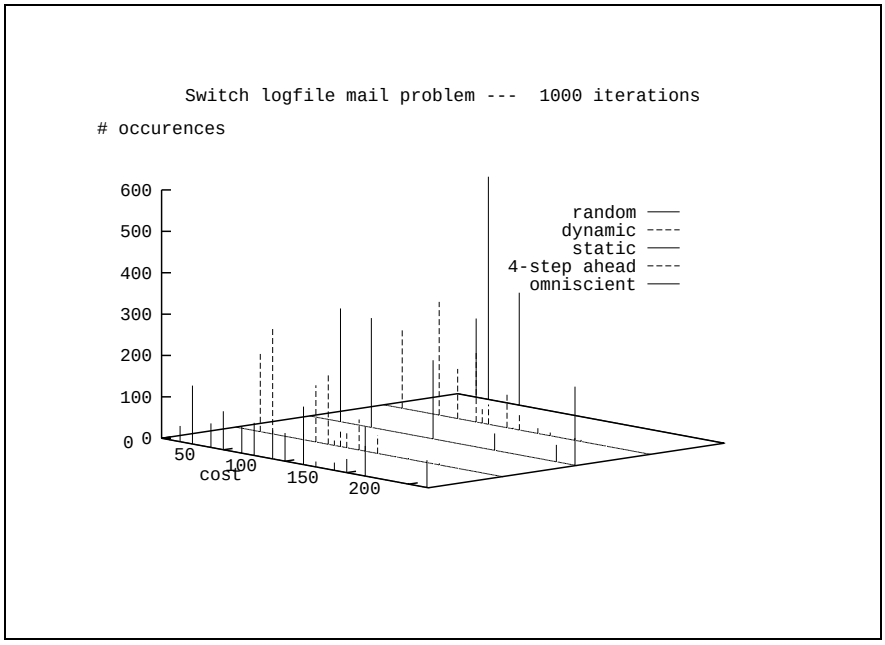

Figure 4: Histogram for the mail problem. From front to back, random, optimal dynamic, best static, 4-step lookahead, omniscient.

5. Conclusion

By identifying causal relationships among various components and events, the belief networks presented in this paper were used to model the human understanding of network fault propagation and dynamics. When an alarm is detected or a problem reported, the conditional distribution of the variables given the evidence and the conditional marginal probabilities are computed. These conditional probabilities are easily obtained using the Lauritzen and Spiegelhalter algorithm for computations of probabilities on graphical structures. Given these probabilities, a decision is made concerning which network object to test. The decision is designed to minimize the overall expected cost of the troubleshooting task.

The conditional independencies implied by the causal relationships with the restriction of no cycles in the diagram were our most restrictive assumptions. These, however, give rise to tractability and computational feasibility.

The results are satisfying along a number of dimensions. The numerical results obtained show that our approach permits us, on the average, substantial savings compared to the best static strategy that can be computed for a given problem. Furthermore, the plans of actions generated by both the static and dynamic troubleshooter usually follows our intuition. Interestingly, a heuristic troubleshooter can perform well provided that it looks far enough into the future.

Belief networks enabled us to model abstract views of the network and deal with the complex issue of hardware and software faults. More generally, the methodology is applicable to any set of *interrelated* network entities, as opposed to *interconnected* network components. Belief networks enabled us to avoid making independence assumptions about the network entities, and thus, permited us to introduce dependencies easily. A drawback, however, is that timing information is not imbedded in the model because of an implicit assumption of product form (whatever path we take from one state to another, the conditional distribution has to be independent of the path taken—similar to path independence in a potential field.)

Acknowledgements

The first author would like to thank the XUNET team and AT&T Bell Laboratories CSRC for the stimulating working environment during the summer of 1994.

A. Probabilistic reasoning using belief networks

An implicit assumption about belief networks is that, given all the direct parents of a node, the node is independent of all the other nodes of the network. This gives rise to an analytically tractable model whose equilibrium probabilities have product form, namely Markov fields. This assumption also reduces the storage requirement of the probabilities. The set of a priori probabilities of the root nodes and the set of conditional probabilities relating neighboring nodes are sufficient to calculate the joint distribution of the belief network. Formally, we define a belief network as follows [22]:

Definition 1 *Let V be a finite set of propositional variables defined on the same probability space, let $(\Omega, \mathcal{F}, P)$ be their joint probability distribution, and let $G = (V, E)$ be a directed acyclic graph. For each $v \in V$, let $c(v) \subseteq V$ be the set of all parents of v and $d(v) \subseteq V$ be the set of all descendents of v. Furthermore for $v \in V$, let $a(v) \subseteq V$ be $V - (d(v) \cup \{v\})$, that is, the set of propositional variables in V excluding v and v's descendents. Suppose for every subset $W \subseteq a(v)$, W and v are conditionally independent given $c(v)$; that is, if $P(c(v)) > 0$, then*

$$P(v|c(v)) = 0 \quad or \quad P(W|c(v)) = 0 \quad or \quad P(v|W \cup c(v)) = P(v|c(v)).$$

Then $C = (V, E, P)$ is called belief network, and the set $c(v)$ is called the set of the causes (parents) of v.

The next theorem gives us the equilibrium distribution associated with a belief network [22].

Theorem 2 *If $C = (V, E, P)$ is a belief network, then $P(V)$ is given by*

$$P(V) = \prod_{\substack{v \in V \\ P(c(v))>0}} P(v|c(v)),$$

where $\prod_{v \in V, P(c(v))>0}$ means we are taking the product of all propositional variables in V for which $P(c(v)) > 0$.

For an illustration of the above theorem, consider the belief network of Figure 5 that illustrate a client/server communication problem where a client tries to set up a connection with a server and fails. The client then reports the repeated access-failure to the fault manager. Several causes are potentially responsible for this access-failure: network congestion, link fault or server fault. On the other hand, network congestion is often caused by high traffic load. Also, when a link fault occurs, a link alarm is normally triggered. Suppose that the propositional variables can take value either true (T) or false (F).

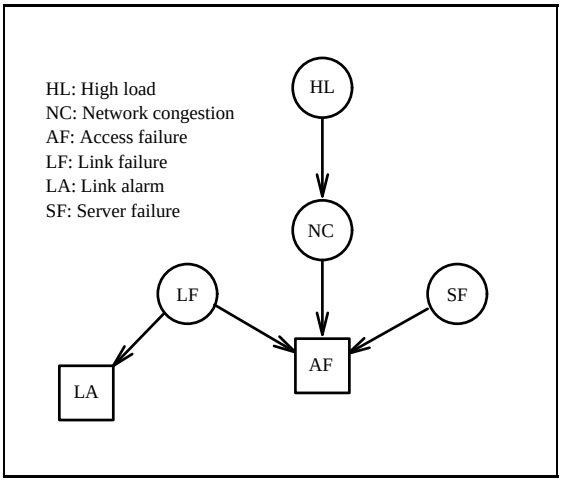

Figure 5: Belief network for the connection setup problem.

To compute the joint distribution, we need to asses the following a priori probabilities: $\Pr\{SF = T\}$, $\Pr\{LF = T\}$ and $\Pr\{HL = T\}$, and the following conditional probabilities: $\Pr\{NC = T|HL = T\}$, $\Pr\{NC = T|HL = F\}$, $\Pr\{LA = T|LF = T\}$, $\Pr\{LA = T|LF = F\}$, and the eight combinations of $\Pr\{AF = T|SF, NC, LF\}$. The joint distribution is obtained by applying Theorem 2 an given by

$$\Pr\{AF, \ldots, SF\} = \Pr\{AF|LF, SF, NC\} \cdot \Pr\{SF\} \cdot \Pr\{LA|LF\} \cdot \Pr\{LF\} \cdot \Pr\{NC|HL\} \cdot \Pr\{HL\}$$

References

[1] A. Bouloutas, G. W. Hart, and M. Schwartz, "Simple finite-state fault detectors for communication networks," *IEEE Trans. Commun.*, vol. 40, pp. 477–479, Mar. 1992.

[2] C. Wang and M. Schwartz, "Fault detection with multiple observers," in *Proc. IEEE Infocom.*, (Florence, Italy), May 1992.

[3] G. Varghese, *Self-Stabilization by Local Checking and Correction.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Mass., Oct. 1992.

[4] P. Hong and P. Sen, "Incorporating non-deterministic reasoning in managing heterogeneous network faults," in *Integrated Network Management, II*, (Amsterdam, The Netherlands), pp. 481–492, Elsevier Science Publishers B.V., 1991.

[5] N. Dawes, J. Altoft, and B. Pagurek, "Network diagnosis by reasoning in uncertain nested evidence spaces," *IEEE Trans. Commun.*, vol. 43, no. 2–4, pp. 466–476, 1995.

[6] C. Wang and M. Schwartz, "Identification of faulty links in dynamic-routed networks," *IEEE J. Select. Areas Commun.*, vol. 11, pp. 1449–1460, Dec. 1993.

[7] A. A. Lazar, W. Wang, and R. H. Deng, "Models and algorithms for network fault detection and identification: A review," in *Proc. IEEE Int'l Conf. Commun.*, (Singapore), pp. 999–1003, November 16–20 1992.

[8] R. H. Deng, A. A. Lazar, and W. Wang, "A probabilistic approach to fault diagnosis in linear lightwave networks," *IEEE J. Select. Areas Commun.*, vol. 11, pp. 1438–1448, Dec. 1993.

[9] I. Katzela and M. Schwartz, "Schemes for fault identification in communication networks," *IEEE/ACM Trans. Networking*, Dec. 1995.

[10] E. Charniak, "Bayesian networks without tears," *AI Magazine*, pp. 50–63, Winter 1991.

[11] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, California: Morgan Kaufmann Publishers, 1988.

[12] A. G. Fraser, C. R. Kalmanek, A. E. Kaplan, and R. C. Restrick, "XUNET 2: a nationwide testbed in high-speed networking," in *Proc. IEEE Infocom.*, (Florence, Italy), May 1992.

[13] J. Kalagnanam and M. Henrion, "A comparison of decision analysis and expert rules for sequential diagnosis," in *Uncertainty in Artificial Intelligence 4*, (Amsterdam, The Netherlands), pp. 271–281, Elsevier Science Publishers B.V., 1990.

[14] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[15] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[16] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *J. R. Statist. Soc. B*, vol. 50, no. 2, pp. 157–224, 1988.

[17] D. Heckerman, J. S. Breese, and K. Rommelse, "Decision-theoretic troubleshooting," *Comm. ACM*, vol. 38, pp. 49–57, Mar. 1995.

[18] G. F. Cooper, "Probabilistic inference using belief networks is NP-hard," Tech. Rep. KSL-87-27, Stanford University, Stanford, CA, 1988.

[19] J.-F. Huard, "Probabilistic reasoning for fault management on XUNET," Sept. 1994. summer report AT&T Bell Laboratories.

[20] M. Henrion and M. J. Druzdzel, "Qualitative propagation and scenario-based schemes for explaning probabilistic reasoning," in *Uncertainty in Artificial Intelligence 6*, (Amsterdam, The Netherlands), pp. 17–32, Elsevier Science Publishers B.V., 1991.

[21] J.-F. Huard and A. A. Lazar, "Fault isolation based on decision-theoretic troubleshooting," Tech. Rep. CU/CTR TR 442–96–08, Center for Telecommunications Research, Columbia University, New York, NY, 1996.

[22] R. E. Neapolitan, *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. New York: John Wiley & Sons, 1990.