# Toward Optimal Network Fault Correction via End-to-End Inference

Patrick P. C. Lee, Vishal Misra, and Dan Rubenstein

*Abstract*— **We consider an end-to-end approach of inferring network faults that manifest in multiple protocol layers, with an optimization goal of minimizing the expected cost of correcting all faulty nodes. Instead of first checking the most likely faulty nodes as in conventional fault localization problems, we prove that an optimal strategy should start with checking one of the *candidate nodes*, which are identified based on a potential function that we develop. We propose several efficient heuristics for inferring the best node to be checked in large-scale networks. By extensive simulation, we show that we can infer the best node in at least 95%, and that checking first the candidate nodes rather than the most likely faulty nodes can decrease the checking cost of correcting all faulty nodes by up to 25%.**

*Index Terms*— **network management, network diagnosis and correction, fault localization and repair, reliability engineering.**

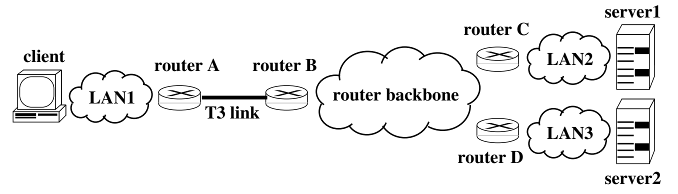


Fig. 1. An example network that is composed of components across different protocol layers.

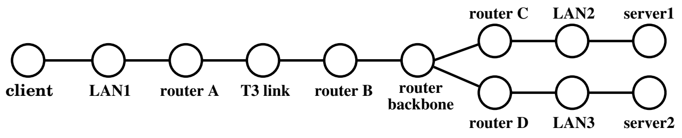


Fig. 2. The logical topology corresponding to Figure 1.

## I. Introduction

Network components are prone to a variety of faults such as packet loss, link cut, or node outage. To prevent the faulty components from hindering network applications, it is important to *diagnose* (i.e., *detect* and *localize*) the components that are the root cause of network faults, as in [12], [13], [23]. However, it is also desirable to *repair* the faulty components to enable them to return to their operational states. Therefore, we seek to devise algorithms that accomplish effective *network fault correction*, by which we mean not only to diagnose, but also to repair all faulty components within a network.

To diagnose (but not repair) network faults, recent approaches like [2], [19], [26] use all network nodes to collaboratively achieve this. For instance, in hop-by-hop authentication [2], each hop inspects packets received from its previous hop and reports errors when packets are found to be corrupted. While such a distributed infrastructure can accurately pinpoint network faults, deploying and maintaining numerous monitoring points in a large-scale network introduces heavy computational overhead in collecting network statistics [7] and involves complicated administrative management [6].

The diagnosis procedure becomes even more challenging when network faults can manifest in multiple protocol layers. Figure 1 depicts a network (derived from [25]) that contains components spanning different protocol layers. The failure of any of these components can disrupt the communication between the client and one (or both) of the servers, yet pinpointing the faults is difficult because low-layer components in general do not respond to high-layer probe messages. Therefore, it is essential that a fault correction scheme can diagnose faults across multiple protocol layers [23].

In order to simplify network management and support multi-layer fault correction, we consider an *end-to-end* approach which, using end-to-end measurements, *infers* components that are probably faulty and need to be further checked so that the actually faulty components can be repaired. Our solution is inspired by [6], [7], [10], [17], which also use end-to-end measurements to infer link statistics or the underlying network topology. We extend this end-to-end approach to the application of network fault management.

Our end-to-end inference approach can be characterized as follows. We start with a routing tree topology as in [6], [7], [10], [17], since a tree-based setting is justified in conventional shortest-path routing such as OSPF [20], where each router builds a shortest-path tree with itself as a root, as well as in multicast routing, where a routing tree is built to connect members in a multicast group. We first map each potentially faulty physical component to a *logical node*. For example, we can transform the physical topology in Figure 1 to a logical tree as shown in Figure 2. Given the logical tree, we monitor every (root-to-leaf) path that contains logical nodes. If a path exhibits any "anomalous behavior" in data delivery, then some "faulty" node on the path must be responsible. In practice, the definition of an "anomalous behavior" depends on specific applications (see Section II for details). Using the path information collected at the application endpoints, we can narrow down the space of possibly faulty components.

In the above end-to-end solution, one can tell whether a path behaves anomalously, but cannot tell specifically which

P. Lee and V. Misra are with the Department of Computer Science, Columbia University, New York, NY, USA (emails: {pclee,misra}@cs.columbia.edu).

D. Rubenstein is with the Department of Electrical Engineering, Columbia University, New York, NY, USA (email: danr@ee.columbia.edu).

This work was supported in part by the National Science Foundation under Grants CNS-0238299, ITR-0325495, and CNS-0626795.

and how many nodes on the path are faulty. To determine the faulty nodes, we should check a sequence of nodes, and fix any nodes that are found to be faulty. Given the anomalous paths in a tree, our main goal is to infer the best node (or the best set of nodes) that should be first checked so as to minimize the expected cost of correcting all faulty nodes.

In this paper, we develop several surprising optimality results for inferring the best node that should be first checked by a network fault correction scheme, with an objective to minimize the expected cost of correcting all faulty nodes. Our contributions include the following:

- Unlike traditional network failure localization problems (e.g., [12], [13], [23]), whose objective is to identify the most likely faults, we show that checking first the node that is most likely faulty or has the least checking cost does not necessarily minimize the expected cost of correcting all faulty nodes.
- We formally identify a subset of nodes termed *candidate nodes*, one of which should be first checked in order to minimize the expected cost of correcting all faulty nodes. We develop a potential function that determines the candidate nodes.
- Based on the potential function and candidate nodes that we propose, we devise various heuristics for the best node inference in a single tree and multiple trees, where the latter forms a more general topology. We show via simulation that the candidate node with the highest potential value is in fact the best node that should be first checked by an optimal strategy in at least 95% of time under a special setting. In addition, we conduct simulation using large-scale network topologies. As compared to the strategies that first check the most likely faulty nodes, we show that by first checking the candidate nodes, the checking cost of correcting all faulty nodes can be decreased by up to 25%.

The remainder of the paper is organized as follows. In Section II, we formulate the network fault correction problem. Section III demonstrates that naive strategies that are intuitively optimal are in fact not optimal in general. In Section IV, we introduce a potential function for identifying the candidate nodes and show the optimality results. In Section V, we propose several heuristics for the best node inference. In Section VI, we evaluate the proposed heuristics in large-scale networks. We review related work in Section VII and conclude the paper in Section VIII.

## II. Problem Formulation

In this section, we formulate our end-to-end inference approach for a network fault correction scheme that diagnoses and repairs all faulty nodes in a network at minimum expected cost. Figure 3 summarizes the end-to-end inference approach.

We are interested in a routing tree that consists of potentially faulty physical components, each of which is transformed to a logical node (see Figure 2). To obtain the components of a routing tree, we can use the physical topology MIB [5], which captures the interconnections of the physical devices residing in the same administrative domain, or perform topology inference via standard tomography techniques (e.g., see [7], [10], [21]). Other techniques of identifying physical components in a given network have also been explored (e.g., see survey in [23]). For those devices that cannot be specified, we aggregate them into a single component, such as the router backbone in Figure 1. This enables us to model the components that are located in different administrative domains and cannot be readily identified.

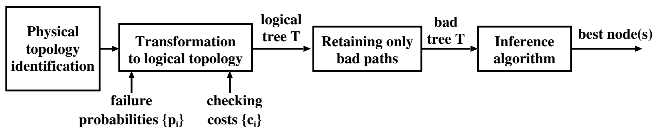


Fig. 3. End-to-end inference approach for a network fault correction scheme.

We define the logical tree as $T = (N, \{p_i\}, \{c_i\})$, where $N$ is the set of logical nodes, $p_i$ is the failure probability of node $i \in N$, and $c_i$ is the checking cost of deciding if node $i \in N$ is faulty. In practice, the failure probabilities $\{p_i\}$ are often characterized via statistical measurements of reliability indexes [11], vulnerability modeling [9], or the inference of link statistics [6], [17]. Here, we assume that node failures, and hence the failure probabilities $\{p_i\}$, are all independent. Likewise, we can characterize the checking costs $\{c_i\}$ based on the personnel hours and wages required for troubleshooting problems or the costs of the test equipment. Note that the checking costs can be highly varying. For instance, in Figure 1, the costs of checking the T3 link and the router backbone can be substantially higher than those of checking the client and server machines.

In our analysis, we assume that $p_i$ and $c_i$ can be any values in $[0, 1]$ and $[0, \infty)$, respectively. For instance, if $c_i = 1$ for all $i$, then the total checking cost denotes the number of nodes that have been checked. Also, depending on the fault definition, the failure probabilities $\{p_i\}$ can be significantly small for general failures (e.g., router malfunction) [12], but can also be non-negligible if we are concerned with how likely a node is compromised subject to worm or denial-of-service attacks.

Each node in a logical tree $T$ can be classified as faulty or non-faulty, depending on how we first define whether a (root-to-leaf) path exhibits any "anomalous behavior". In practice, such a definition varies across applications. For instance, in Figure 1, packets can be dropped or delayed when the T3 link loses synchronization and produces noise bursts [25]. We can then say that a path behaves anomalously if it fails to deliver a number of correct packets within a time window, and that some node on the path is faulty if it causes severe packet loss and delay. As stated in Section I, the inference approach only knows whether a path is anomalous, but does not know specifically which and how many nodes on the path are faulty. To help our discussion, each node in $T$ is referred to as *bad* if it is faulty, or as *good* otherwise. We say a path is *bad* if it contains at least one bad node, and is otherwise *good*. Also, we call $T$ a *bad tree* if every path in the tree is a bad path.

A node can exhibit the same or different behaviors on all paths upon which it lies. In the latter case, we can assume that all paths are independent and hence each path must

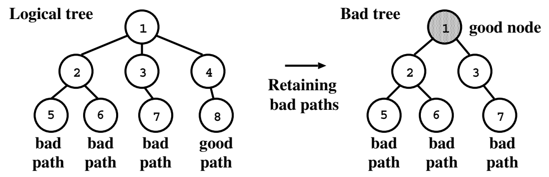


Fig. 4. Given a logical tree, we retain only the bad paths and indicate any good node. Since path $\langle 1, 4, 8\rangle$ is a good path, it is known that nodes 1, 4, and 8 are good. Nodes 4 and 8 can be pruned from the tree, and node 1 can be indicated as good. The resulting set of bad paths will lead to a bad tree.

be inspected individually[1]. However, it is more common to have hard failures (e.g., power outage or machine shutdown) reflected on all paths that contain faulty nodes. Thus, we assume that the behavior of each node is the same on all paths upon which the node lies. With this assumption, if a node lies on at least one good path, then it is a good node. Note that a good node may still lie on one or more bad paths, but this only means each such bad path contains some other bad node.

Given a logical tree $T$, we determine whether a path is good or bad via end-to-end measurements that are carried out between the root and leaf nodes of $T$. For example, the root can send probes to the leaf nodes, from which we collect the measurement results. Since a good path contains only good nodes that need not be checked, we only need to focus on the bad paths in $T$. Also, any node that lies on both good and bad paths is indicated to be good. To illustrate, Figure 4 shows how to retain only the bad paths in $T$ and indicate the good nodes. The resulting set of bad paths will then form a bad tree. With a slight abuse of notation, we denote this bad tree by $T$ as well.

We then pass the bad tree $T$ to the *inference algorithm*, which determines, from the set of nodes that are not indicated as good, the "best" node (or the "best" set of nodes) to be checked, and repaired if necessary.

Before formalizing the notion of "best", we first state our optimization goal, namely, to minimize the expected cost of correcting all faulty nodes in a given bad tree $T$. Suppose that using end-to-end measurements to determine bad paths incurs negligible cost. Then the correction cost has two main components: the cost of checking all nodes and the cost of repairing all faulty nodes. However, we do not consider the repair cost since all faulty nodes have to be recovered eventually, and any successful repair strategy has the same cost of repairing all faulty nodes. As a result, by cost, we here refer to the checking cost only.

Because of our optimization goal, we can focus on the *sequential* case where we check one node at a time, since checking multiple nodes simultaneously does not improve the expected checking cost, even though it reduces the time required to repair all bad nodes on all bad paths. As a result, our theoretical analysis assumes that the inference algorithm returns only a *single* best node, while we evaluate via simulation the impact of inferring and checking multiple nodes in Section VI.

[1]If each path is treated independently, we can apply the optimal strategy as shown in Corollary 1 in Section IV-B.

With the optimization goal, we select the "best" node based on a *diagnosis sequence* $S = \langle l_1, l_2, \cdots, l_{|N|}\rangle$, defined as the order of nodes to be examined given a bad tree $T$. When node $l_i$ is examined, it is either *checked* or *skipped*. If node $l_i$ lies on bad paths only, we cannot tell whether it is good or bad. In this case, we have to check node $l_i$, and repair it if it is determined to be bad. On the other hand, if node $l_i$ lies on a good path, it is known to be a good node and does not need to be checked. In this case, we say we skip node $l_i$. After node $l_i$ has been checked or skipped, it is known to be good. Thus, given a bad tree $T$, the expected (checking) cost with respect to $S$ is:

$$\sum_{i=1}^{|N|} c_{l_i} \Pr(\text{node } l_i \text{ is checked} \mid \begin{array}{l}\text{bad tree } T\text{, and nodes } l_1, \cdots,\\ l_{i-1} \text{ known to be good}\end{array}).$$

Due to limited space, we detail in [15] the calculation of the expected cost of a diagnosis sequence. A diagnosis sequence $S$ is said to be *optimal* if its expected cost is minimum among all possible diagnosis sequences. Therefore, among all the nodes in a bad tree $T$, we formally define the *best node* as the first node in an optimal diagnosis sequence for $T$. In other words, the best node should be the node to be first checked in order to minimize the expected cost of correcting all faulty nodes.

We point out that the optimal decision of the inference algorithm is derived from the current topology. The decision will be revised for a new topology when the faults are actually checked and repaired. In spite of this, the topology may still change between the time of identifying the physical topology and performing the inference algorithm. However, as stated in [21], topology change occurs in a coarse time scale (on the order of minutes and hours). Thus, as long as the network fault correction scheme has its monitoring period bounded within a few minutes, the topology should remain fairly stable. In [15], we address how to incorporate into our inference algorithm the false alarms resulting from inaccurate modeling of physical components.

A straightforward way to implement the inference algorithm is based on the brute-force approach as shown in Algorithm 1, which enumerates all possible diagnosis sequences in order to determine the best node. However, the brute-force approach has factorial complexity $\Theta(|N|^3|N|!)$ (note that as shown in [15], the complexity of finding the expected cost of a diagnosis sequence is $\Theta(|N|^3)$). Therefore, in the following sections, we seek to answer the following question: *Given a bad tree T, how can the inference algorithm determine the best node in polynomial time?*

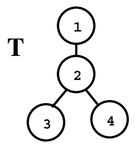
**Algorithm 1** Brute-force inference algorithm

```
Input: Bad tree $T = (N, \{p_i\}, \{c_i\})$
1: $S^* = \phi$, $c^* = \infty$
2: for all diagnosis sequence $S$ do
3:     compute $c$ = the expected cost of $S$
4:     if $c < c^*$ then
5:         $S^* = S$, $c^* = c$
6: return the first node in $S^*$
```

## III. Naive Heuristics for the Inference Algorithm

Intuitively, the best node returned by the inference algorithm could be either the node that has the highest conditional failure probability given a bad tree $T$ (see Appendix I for its calculation, whose complexity is $\Theta(|N|^2)$), or the node that has the least checking cost. In this section, we show that these naive choices do not necessarily minimize the expected cost of correcting all faulty nodes.

We first give a simple counter-example that disproves the above naive choices. Figure 5 illustrates a bad tree rooted at node 1 and the corresponding failure probabilities $\{p_i\}$ and checking costs $\{c_i\}$. As verified by the brute-force approach in Algorithm 1, the best node is node 2, where a possible optimal diagnosis sequence is $\langle 2, 1, 3, 4 \rangle$ and has expected cost 1.044. However, node 2 is neither the node with the highest conditional failure probability nor the node with the least checking cost.



| $i$ | $p_i$ | $c_i$ | Conditional $p_i$ |
|---|---|---|---|
| 1 | 0.1 | 1 | 0.6586 |
| 2 | 0.05 | 0.3 | 0.3293 |
| 3 | 0.08 | 0.2 | 0.1214 |
| 4 | 0.1 | 0.1 | 0.1405 |

Fig. 5. A counter-example of showing the best node (which is node 2) is not the one chosen by the naive choices. The conditional probability $p_i$ given a bad tree is computed as shown in Appendix I.

To further understand the performance of the naive choices, we evaluate two naive heuristics for the inference algorithm: (i) *Naive-Prob*, which returns the node with the highest conditional failure probability, and (ii) *Naive-Cost*, which returns the node with the least checking cost. We compare their performance to that of the brute-force inference algorithm in Algorithm 1 using a special small-scale setting where a bad tree comprises only two bad paths (e.g., see Figures 2 and 5). Such a setting is to describe a scenario where most routing paths are disjoint and at most two paths share the same physical components.

Our evaluation setup is described as follows. For a fixed number of nodes $|N|$, we randomly generate 200 bad trees, each of which comprises two bad paths such that the position of the only non-leaf node that has two child nodes is randomly chosen. Since, as stated in Section II, $p_i$ and $c_i$ can be any values in $[0, 1]$ and $[0, \infty)$, respectively, we arbitrarily choose three distributions for our evaluation: (a) $p_i \sim U(0, 1)$ and $c_i = 1$, (b) $p_i = 0.1$ and $c_i \sim U(0, 1)$, as well as (c) $p_i \sim U(0, 1)$ and $c_i \sim U(0, 1)$, where $U(u_1, u_2)$ denotes the uniform distribution between $u_1$ and $u_2$.

Figure 6 illustrates the proportions of instances (out of 200) in which Naive-Prob and Naive-Cost return correctly a best node (which may not be unique, if any one of the best nodes can be first checked to give the minimum cost). Depending on the distributions of $p_i$ and $c_i$, the proportion of instances where the best choice is made can be as low as 10% for both Naive-Prob and Naive-Cost (see Figures 6(a) and 6(b)). Also, when $p_i$ and $c_i$ are both randomly chosen, both naive approaches select the best node in only less than 75% of time (see Figure 6(c)).

## IV. Candidate Nodes

Instead of the naive choices described in the previous section, we show in this section that we should first check a *candidate node*, which is selected based on the maximization of a *potential function* as described below.

We first give the notation and definitions that we will use. Given a tree $T$, we define *ancestors* of node $i$ to be the nodes (not including node $i$) on the path from the root of $T$ to node $i$, and *descendants* of node $i$ to be the nodes that have node $i$ as one of their ancestors. Let $\mathcal{T}$ be the event that $T$ is a bad tree, and $\mathcal{X}_i$ be the event that node $i$ is a bad node. Let $\mathcal{A}_i$ be the event that the ancestors of node $i$ are all good. If node $r$ is the root node, then we let $\mathcal{A}_r$ be always true and $\Pr(\mathcal{A}_r) = 1$.

### A. Definitions of a Candidate Node and Potential

*Definition 1:* The *potential* of node $i$ in a tree $T$ is defined as the value returned by the potential function:

$$\phi(i, T) = \frac{\Pr(\mathcal{T}|\mathcal{X}_i, \mathcal{A}_i) p_i}{c_i(1 - p_i)}.$$

Intuitively, the best node should be a node with a high potential, since such a node in general has a small checking cost, a large failure probability, and a large likelihood of leading to a bad tree. Note that the term $\frac{p_i}{c_i(1-p_i)}$ denotes the potential of a node for a single-bad-path case (see Corollary 1 and [11]). Therefore, the potential function $\phi(i, T)$ is to combine the potential of a single path (i.e., $\frac{p_i}{c_i(1-p_i)}$) and the likelihood of having a bad tree with respect to the tree topology (i.e., $\Pr(\mathcal{T}|\mathcal{X}_i, \mathcal{A}_i)$). The potential function is derived based on the optimality results presented in the following subsection.

The potential of node $i$ can be obtained by first computing $\Pr(\mathcal{T}|\mathcal{X}_i, \mathcal{A}_i) = \Pr(\mathcal{T} \mid p_i = 1, p_j = 0 \, \forall j \in A_i)$, where $A_i$ is the set of ancestors of node $i$. As shown in Appendix I, we can compute the potential of a node in $\Theta(|N|^2)$ time.

*Definition 2:* For each path $W$ in a tree $T$, we select a single node $i$ to be a *candidate node* if node $i$ lies on $W$ and its potential $\phi(i, T)$ is the highest among all the nodes on $W$. If more than one node on $W$ has the highest potential, then we choose the one that is closest to the root node of $T$ as the candidate node.

We can view the selection of candidate nodes as a mapping function that returns a single candidate node given $W$ and $T$. Note that it is possible for a path in $T$ to have multiple candidate nodes that correspond to different paths in $T$.

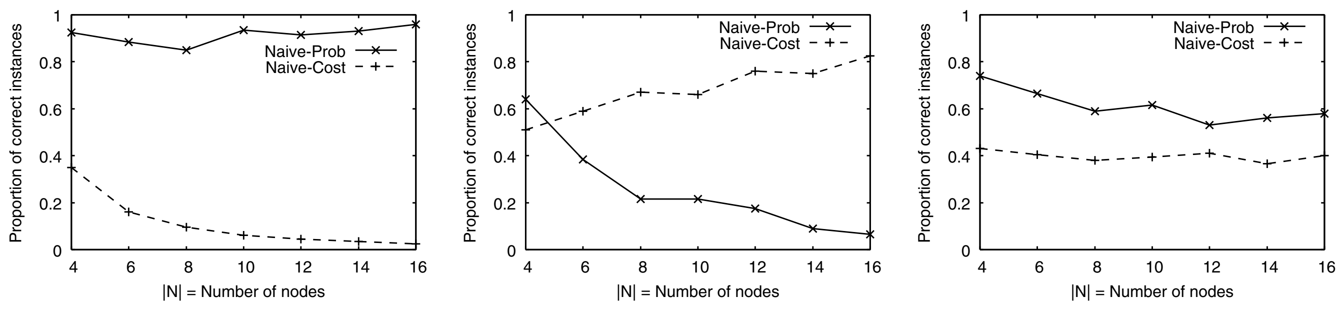
### B. Optimality Results

We now state the main optimality result as follows. In the interest of space, the formal proofs are detailed in [15].

*Theorem 1:* Given a bad tree $T$, there always exists an optimal diagnosis sequence that starts with a candidate node.

In other words, the best node returned by our inference approach should be one of the candidate nodes. We present a sketch of proof of Theorem 1 in Appendix II to illustrate the intuition of how the potential function is derived and why a candidate node should be first checked. The formal proof can be found in [15].

In some special cases, we can identify the best node from a set of candidate nodes, as shown in the following corollaries.



(a) $p_i \sim U(0,1), c_i = 1$ (b) $p_i = 0.1, c_i \sim U(0,1)$ (c) $p_i \sim U(0,1), c_i \sim U(0,1)$

Fig. 6. Proportions of instances in which Naive-Prob and Naive-Cost return a correct best node.

*Corollary 1:* Given a bad tree $T$ which is a single path, the best node is the one with the maximum $\frac{p_i}{c_i(1-p_i)}$.

*Remark:* Corollary 1 implies that the optimal diagnosis sequence is in non-increasing order of $\frac{p_i}{c_i(1-p_i)}$. This conforms to the result in [11].

In general, whether a node is the best depends on the failure probabilities $\{p_i\}$, the checking costs $\{c_i\}$, as well as the tree topology $T$. In spite of this, there are cases where the most likely faulty node is in fact the best node.

*Corollary 2:* Given a bad tree $T$, if every node $i$ has $p_i = p$ and $c_i = c$, where $p$ and $c$ are some fixed constants, then the best node is the root node of $T$.

*Remark:* If $T$ is a bad tree, then the root node $r$ is also the node that has the maximum conditional failure probability since $\Pr(\mathcal{X}_r|\mathcal{T}) = \frac{p}{\Pr(\mathcal{T})} \geq \frac{\Pr(\mathcal{T}|\mathcal{X}_i)p}{\Pr(\mathcal{T})} = \Pr(\mathcal{X}_i|\mathcal{T})$ for every node $i$.

*Corollary 3:* Consider a two-level tree $T$ whose root node is attached to $|N|-1$ leaf nodes. If $T$ is a bad tree and every node $i$ has $c_i = c$ for some constant $c$, the best node is the one with the maximum conditional failure probability.

## C. Why Selecting the Best Node from Candidate Nodes is Difficult?

While we have proved that one of the candidate nodes is the best node, we have not yet identified which candidate node should be selected as the best node in the general case. We show that deciding the best node is non-trivial using several counter-examples that violate our intuition. We summarize our major findings in the following discussion, and provide the counter-examples in [15].

- Given a bad tree, the best node is not the one with the highest potential.
- Checking simultaneously all candidate nodes in a bad tree does not minimize the expected cost of correcting all faulty nodes.
- The best node for a bad tree is not necessarily the best node for a subtree. Thus, we cannot determine the optimal solution to a problem by first solving for the optimal solutions to the subproblems, and this makes the best node inference difficult.

## D. Evaluation of Candidate-based Heuristics

Given the difficulty of finding the best node among a set of candidate nodes, we evaluate the performance of three candidate-based heuristics that approximate the best node selection of the inference algorithm. These heuristics are: (1) *Cand-Prob*, which selects the candidate node with the highest conditional failure probability given a bad tree, (2) *Cand-Cost*, which selects the candidate node with the least checking cost, and (3) *Cand-Pot*, which selects the candidate node with the highest potential. Our evaluation setting is the same as that in Section III, i.e., we determine the proportion of instances (out of 200) in which a candidate-based heuristic selects a best node for a given two-path bad tree of size $|N|$ under different distributions of $p_i$ and $c_i$.

Figure 7 plots the results. Comparing the results to those in Figure 6, we find that all candidate-based heuristics are more likely to make the best choice than the naive ones. In particular, Cand-Pot outperforms all naive and candidate-based heuristics. In most cases, the candidate node with the highest potential is actually the best node in at least 95% of time.

## E. Complexity of Finding the Node with Maximum Potential

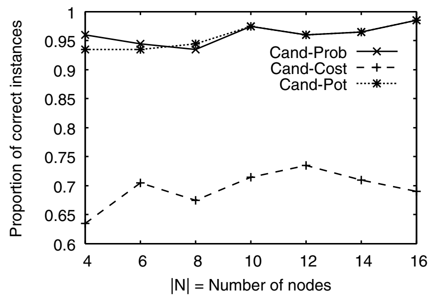
Since the computation of the potential of a node needs $\Theta(|N|^2)$ time (see Section IV-A), the complexity of Cand-Pot, which searches among all nodes for the one with the highest potential, is $\Theta(|N|^3)$. This may lead to the scalability issue when the network grows. However, in most cases, the bad tree is only a subset of the entire routing tree. Hence, the bad tree that is parsed by Cand-Pot is likely to be within a manageable size in general.

## V. Heuristics for the Inference Algorithm

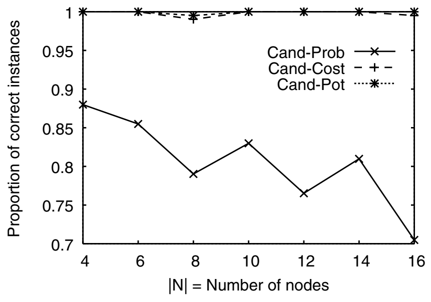
While the brute-force inference algorithm (see Algorithm 1) returns the best node, its factorial complexity prohibits its use in large-scale networks. Thus, we propose three classes of efficient heuristics for the inference algorithm that are suitable for large-scale networks. Each class of heuristics consists of two approaches: one that considers the most likely faulty nodes and one that considers the candidate nodes.

## A. Single Node Inference for a Single Tree

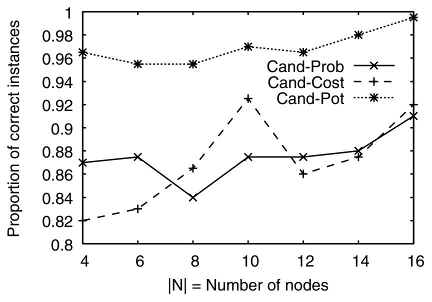
We consider two heuristics Naive-Prob (see Section III) and Cand-Pot (see Section IV), which respectively return the node that is most likely faulty given a bad tree and the candidate node with the highest potential.



(a) $p_i \sim U(0,1), c_i = 1$



(b) $p_i = 0.1, c_i \sim U(0,1)$



(c) $p_i \sim U(0,1), c_i \sim U(0,1)$

Fig. 7. Proportions of correct instances in which Cand-Prob, Cand-Cost, and Cand-Pot, start with a best node.

## B. Multiple Node Inference for a Single Tree

Instead of sequentially checking one node at a time, we can also infer and check multiple nodes *in parallel* so as to reduce the time needed to repair all bad nodes.

We first extend Naive-Prob to *Pa-Naive-Prob* (where "Pa" stands for "parallel"), which returns the most likely faulty subset $I_{pnp}$ of nodes that cover all the bad paths in a given bad tree $T$, i.e., $I_{pnp} = \arg\max_I \Pr(\mathcal{X}_I|\mathcal{T})$, where $\mathcal{T}$ is the event that $T$ is a bad tree, and $\mathcal{X}_I$ is the event that the subset $I$ of nodes are all bad and cover all bad paths in $T$. Since $\Pr(\mathcal{T}|\mathcal{X}_I)$ is one, it is equivalent to evaluate $I_{pnp} = \arg\max_I \Pr(\mathcal{X}_I) = \arg\max_I \prod_{i\in I} p_i$. We can determine $I_{pnp}$ using Algorithm 2, whose complexity is $\Theta(|N|)$.

**Algorithm 2** Pa-Naive-Prob

```
Input: a bad tree $T = (N, \{p_i\}, \{c_i\})$
 1: for all node $i \in N$ in reverse breadth-first-search order do
 2:     if node $i$ is a leaf node then
 3:         $s(i) = p_i$; mark node $i$ /* $s(i)$ denotes the score of $i$ */
 4:     else if node $i$ is a non-leaf node then
 5:         if $p_i > \prod_{j\in C_i} s(j)$ then /* $C_i$ = set of child nodes of $i$ */
 6:             $s(i) = p_i$; mark node $i$
 7:         else
 8:             $s(i) = \prod_{j\in C_i} s(j)$
 9: $I_{pnp} = \phi$; $Q = \phi$; enqueue root node of $T$ to $Q$
10: while $Q \neq \phi$ do
11:     dequeue node $i$ from $Q$
12:     if node $i$ is marked then
13:         $I_{pnp} = I_{pnp} \cup \{i\}$
14:     else
15:         enqueue all child nodes of $i$ to $Q$
16: return $I_{pnp}$
```

We also implement a candidate-based heuristic for inferring multiple nodes termed *Pa-Cand*, which returns the minimum-sized subset $I_{pc}$ of candidate nodes that cover all bad paths in a bad tree. The algorithm of finding $I_{pc}$ is shown in Algorithm 3, whose complexity is $\Theta(|N|^3)$ due to the search of candidate nodes.

**Algorithm 3** Pa-Cand

```
Input: a bad tree $T = (N, \{p_i\}, \{c_i\})$
1: determine the set of candidate nodes in $T$
2: $I_{pc} = \phi$; $Q = \phi$; enqueue root node of $T$ to $Q$
3: while $Q \neq \phi$ do
4:     dequeue node $i$ from $Q$
5:     if node $i$ is a candidate node then
6:         $I_{pc} = I_{pc} \cup \{i\}$
7:     else
8:         enqueue all child nodes of $i$ to $Q$
9: return $I_{pc}$
```

## C. Multiple Node Inference for Multiple Trees

A limitation of the above heuristics is that a single logical tree only includes a subset of physical components in the entire network. In order to cover more physical components, it is important to conduct the inference algorithm on multiple bad trees, which collectively form a directed acyclic graph. In general, finding a minimum subset of nodes that cover a general set of paths can be viewed as a set-cover problem, which is NP-hard [8]. Therefore, we consider two simple heuristics termed *Mt-Pa-Naive-Prob* and *Mt-Pa-Cand* (where "Mt" stands for "multiple trees"), which respectively call Pa-Naive-Prob and Pa-Cand independently on each of the bad trees and return the union of the results.

## VI. Experiments in Large-Scale Networks

In this section, we evaluate via simulation the heuristics described in Section V in correcting faulty nodes in large-scale networks.

## A. Experimental Setup

We consider the router-level Barábasi-Albert model [3] with 200 routers and 397 links. We use the BRITE generator [18] to construct 10 topologies with random node placement and random link weights. Each topology is further assigned 20 sets of parameters using different seeds, leading to a total of 200 instances. To demonstrate how to repair faults across different layers, we are interested in monitoring routers as well as links. In each of the instances, each router and link is assigned a failure probability $p_i$ and a checking cost $c_i$ based on three distributions: (a) $p_i \sim U(0, 0.2)$ and $c_i = 1$, (b) $p_i = 0.1$ and $c_i \sim U(0, 1)$, as well as (c) $p_i \sim U(0, 0.2)$ and $c_i \sim U(0, 1)$, where $U(u_1, u_2)$, as defined in Section III, denotes the uniform distribution between $u_1$ and $u_2$.

To construct a logical tree from a physical topology (see Section II), we first create a shortest-path tree rooted at a randomly selected router based on the given link weights, and then randomly choose from the shortest-path tree a subset of $K$ paths to be included in the logical tree. Our experiments will

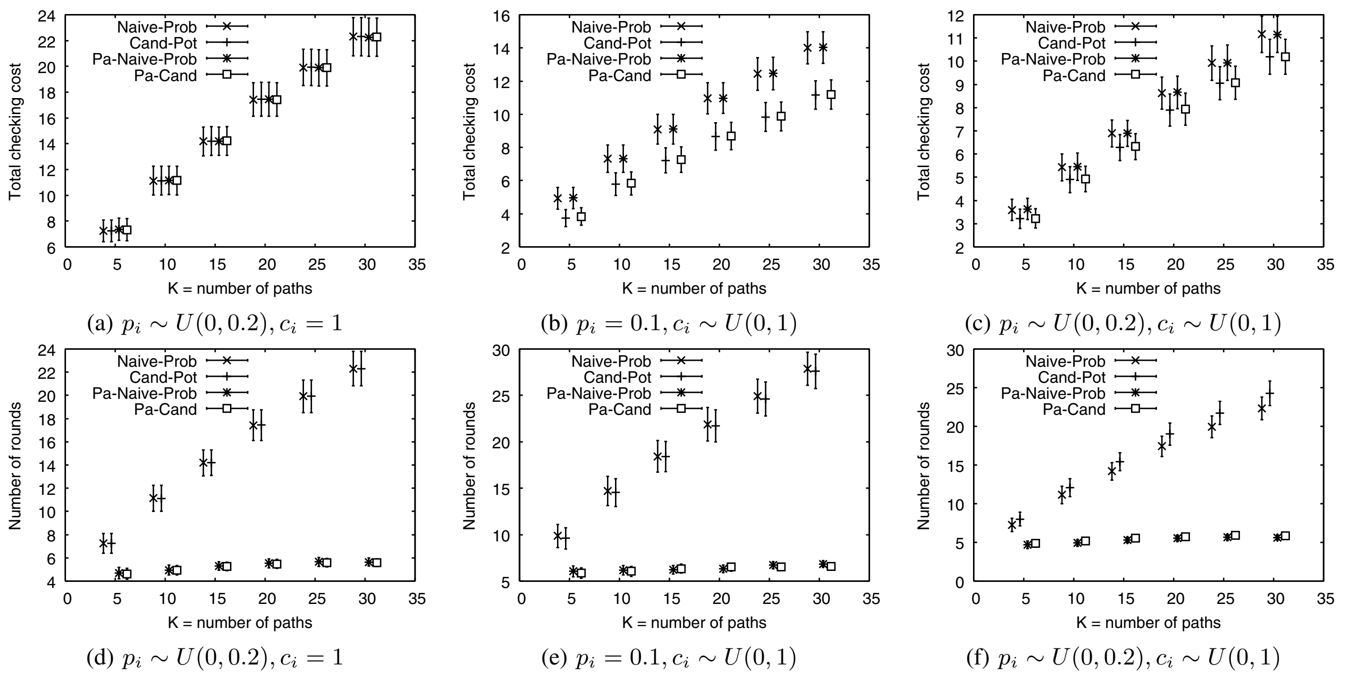


Fig. 8. Experiment 1: Comparison of Naive-Prob, Cand-Pot, Pa-Naive-Prob, and Pa-Cand in terms of the total checking cost (see (a) to (c)) and the number of rounds (see (d) to (f)), where $K$ = 5, 10, 15, 20, 25, and 30. For clarity of presentation, data points are shifted slightly on x-axis.

study the results with different values of $K$. For the heuristics that have multiple logical trees, we simply repeat the logical tree construction by selecting a random subset of routers to be the roots of the logical trees. Finding the efficient set of trees that cover all network components has been considered in [1], [4] and is beyond the scope of this paper.

Given a single or multiple logical trees, we simulate the faults by letting each router and link fail independently with its assigned failure probability. To ensure that the inference algorithm is actually executed, we require that at least one bad node resides in a logical tree. Given a logical tree, we first retain only the bad paths and form a bad tree. We then check the node (or nodes) returned from the inference algorithm that is implemented using different heuristics. For any located bad node, we "repair" it by switching it to a good node. We repeatedly update the set of bad paths and execute the inference algorithm until all bad nodes are repaired.

Due to the large network size, we cannot apply the brute-force inference algorithm in Algorithm 1 to determine the proportion of correct identification as in Sections III and IV. Therefore, our experiments mainly focus on two metrics to evaluate our heuristics:

- *Total checking cost,* the sum of costs of checking the node (or nodes) returned by the inference algorithm until all bad nodes are repaired.
- *Number of rounds,* the number of times the inference algorithm is executed until all bad nodes are repaired.

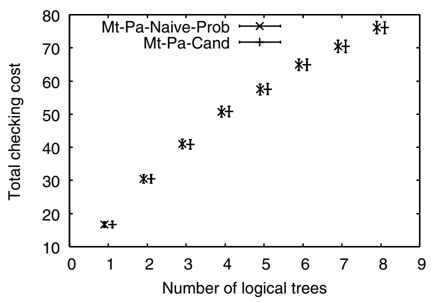
## B. Experimental Results

In the following results, each data point is averaged over 200 instances and is plotted with its 95% confidence interval.

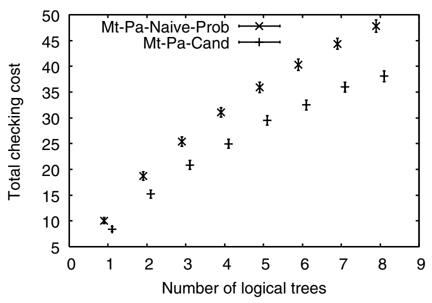
**Experiment 1 (Comparison of single-tree-based heuristics):** Figure 8 shows the performance of Naive-Prob, Cand-Pot, Pa-Naive-Prob, and Pa-Cand versus $K$, the number of paths that are included in a logical tree. As $K$ increases, more nodes are being monitored, and hence the total checking costs of all heuristics increase as well. We note that the distributions of $p_i$ and $c_i$ can influence the difference between the total checking costs of the naive heuristics (i.e., Naive-Prob and Pa-Naive-Prob) and candidate-based heuristics (i.e., Cand-Pot and Pa-Cand). When $p_i$ is varied and $c_i$ is fixed (see Figure 8(a)), both naive and candidate-based heuristics have nearly identical total checking cost. However, the candidate-based heuristics reduce the total checking costs of the naive heuristics by 21-25% when $p_i$ is fixed and $c_i$ is varied (see Figure 8(b)), and by 9-11% when both $p_i$ and $c_i$ are varied (see Figure 8(c)). This demonstrates the competence of the candidate-based heuristics when the physical components have varying checking costs (see Section II).

We also note that Naive-Prob and Cand-Pot, the sequential heuristics that infer one node at a time, take more rounds to execute the inference algorithm as $K$ grows. On the other hand, Pa-Naive-Prob and Pa-Cand, which return multiple nodes at a time, significantly reduce the number of rounds , while only slightly increasing the total checking cost (by less than 1% in general) as compared to their respective sequential heuristics. Furthermore, both Pa-Naive-Prob and Pa-Cand take a similar number of rounds to execute the inference algorithm, with less than one round of difference.

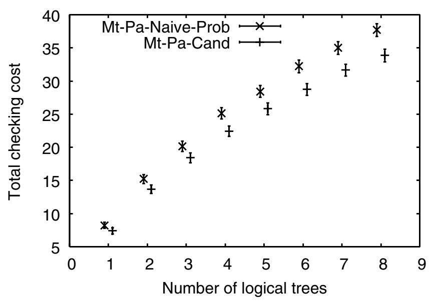
**Experiment 2 (Comparison of multi-tree-based heuristics):** Figure 9 illustrates the performance of Mt-Pa-Naive-Prob and Mt-Pa-Cand with respect to the number of logical trees, where we fix $K$ = 20 in each logical tree. Similar to Experiment 1, Mt-Pa-Cand can reduce the total checking cost of Mt-Pa-Naive-Prob, for example, by 17-21% when $p_i$ is fixed and $c_i$ is varied (see Figure 9(b)). Also, both Mt-Pa-Naive-Prob



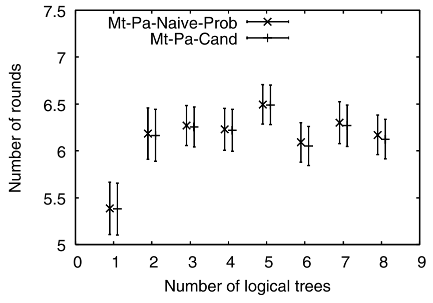
(a) $p_i \sim U(0, 0.2), c_i = 1$



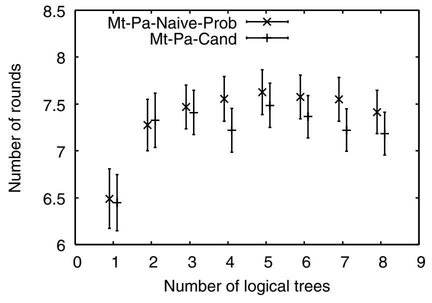
(b) $p_i = 0.1, c_i \sim U(0, 1)$



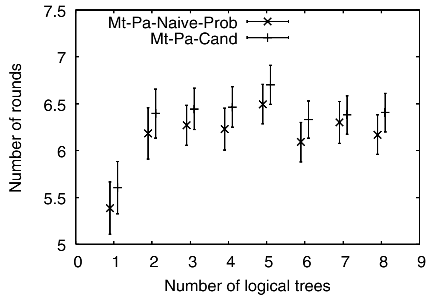
(c) $p_i \sim U(0, 0.2), c_i \sim U(0, 1)$



(d) $p_i \sim U(0, 0.2), c_i = 1$



(e) $p_i = 0.1, c_i \sim U(0, 1)$



(f) $p_i \sim U(0, 0.2), c_i \sim U(0, 1)$

Fig. 9. Experiment 2: Comparison of Mt-Pa-Naive-Prob and Mt-Pa-Cand in terms of the total checking cost (see (a) to (c)) and the number of rounds (see (d) to (f)).

and Mt-Pa-Cand call the inference algorithm with about the same number of rounds.

**Summary:** We show that the candidate-based heuristics can decrease the total checking cost of the naive heuristics that focus on most likely faults, especially when the physical components have varying checking costs. Also, multiple node inference can speed up the fault correction process, with only a slight increase in the total checking cost as compared to single node inference.

## VII. Related Work

Fault diagnosis is an important topic in network management (see survey in [22]). For example, the codebook approach [25] and the Bayesian approach [14] consider a deterministic setting where network faults are equally likely to occur. To address probabilistic faults, [12], [13], [23] seek to localize the most probable subset of faults given the observed symptoms. Katzela and Schwartz [13] show that such a problem is generally NP-hard. In view of this, Steinder and Sethi [23] formulate the problem based on bipartite belief networks and propose efficient techniques on localizing end-to-end multi-layer failures. Kandula *et al.* [12] propose to infer the most likely faulty IP links with the assumption of significantly small failure probabilities. All these schemes focus on localizing faults. In this paper, in addition to probabilistic fault localization, we take the subsequent step of repairing faults as well. We show that checking first the most likely faults does not give optimal results in general.

Fault correction has also been studied extensively in reliability engineering (see survey in [16]). The work that is closest to ours is in [11], [24], whose objective is also to minimize the cost of correcting all faulty units. However, the optimality assumes a *series* system, which is equivalent to a single path in a network setting. In contrast, we consider a routing tree whose paths are shared and dependent.

## VIII. Conclusions

We presented the optimality results for an end-to-end approach to diagnose and repair multi-layer network faults at minimum expected cost. We showed that checking first the node that is most likely faulty or has the least checking cost does not necessarily minimize the expected cost of correcting all faulty nodes. Instead, we constructed a potential function for identifying the candidate nodes, one of which should be first checked by an optimal strategy. Due to the difficulty of finding the best node from the set of candidate nodes, we proposed several efficient heuristics that are suitable for correcting fault nodes in large-scale networks. We showed that the candidate node with the highest potential is actually the best node in at least 95% of time, and that checking first the candidate nodes can reduce the cost of correcting faulty nodes as compared to checking first the most likely faulty nodes.

## References

[1] M. Adler, T. Bu, R. Sitaraman, and D. Towsley. Tree Layout for Internal Network Characterizations in Multicast Networks. In *Proc. of NGC'01*, 2001.

[2] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly Secure and Efficient Routing. In *Proc. of IEEE INFOCOM*, March 2004.

[3] A.-L. Barábasi and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, Oct 1999.

[4] Y. Bejerano and R. Rastogi. Robust Monitoring of Link Delays and Faults in IP Networks. In *Proc. of IEEE INFOCOM*, 2003.

[5] A. Bierman. Physical Topology MIB, Sep 2000. RFC 2922.

[6] R. Cáceres, N. Duffield, J. Horowitz, and D. Towsley. Multicast-based Inference of Network-Internal Loss Characteristics. *IEEE Trans. on Information Theory*, 45(7):2462–2480, November 1999.

[7] M. Coates, A. O. Hero, R. Nowak, and B. Yu. Internet Tomography. *IEEE Signal Processing Magazine*, pages 47–65, May 2002.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[9] W. Du and A. Mathur. Testing for Software Vulnerability Using Environment Perturbation. In *Proc. of the International Conference on Dependable Systems and Networks*, 2000.

[10] N. Duffield, J. Horowitz, F. L. Presti, and D.Towsley. Multicast Topology Inference from Measured End-to-End Loss. *IEEE Trans. on Information Theory*, 48:26–45, January 2002.

[11] B. Gnedenko and I. A. Ushakov. *Probabilistic Reliability Engineering*. John Wiley & Sons, Inc., 1995.

[12] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A Tool for Failure Diagnosis in IP Networks. In *ACM SIGCOMM MineNet-05*, Aug 2005.

[13] I. Katzela and M. Schwartz. Schemes for Fault Identification in Communication Networks. *IEEE/ACM Trans. on Networking*, 3(6):733–764, 1995.

[14] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. IP Fault Localization Via Risk Modeling. In *Proc. of NSDI*, 2005.

[15] P. P. C. Lee, V. Misra, and D. Rubenstein. Toward Optimal Network Fault Correction via End-to-End Inference. Computer Science Technical Report, Columbia University, May 2006. URL: http://dna-wsl.cs.columbia.edu/pubsdb/citation/paperfile/121/tr.ps.

[16] S. Lee and K. G. Shin. Probabilistic Diagnosis of Multiprocessor Systems. *ACM Computing Survey*, 26(1):121–139, March 1994.

[17] F. LoPresti, N. Duffield, J. Horowitz, and D. Towsley. Multicast-based Inference of Network-Internal Delay Distributions. *IEEE/ACM Trans. on Networking*, 10(6):761–775, Dec 2002.

[18] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. In *Proc. of MASCOTS*, Aug 2001.

[19] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: Detecting and Isolating Malicious Routers. In *Proc. of the IEEE Conference on Dependable Systems and Networks (DSN)*, June 2005.

[20] J. Moy. OSPF Version 2, April 1998. RFC 2328.

[21] M. Rabbat, R. Nowak, and M. Coates. Multiple Source, Multiple Destination Network Topomography. In *Proc. of IEEE INFOCOM*, 2004.

[22] M. Steinder and A. Sethi. A Survey of Fault Localization Techniques in Computer Networks. *Science of Computer Programming, Special Edition on Topics in System Administration*, 53(2):165–194, Nov 2004.

[23] M. Steinder and A. S. Sethi. Probabilistic Fault Localization in Communication Systems Using Belief Networks. *IEEE/ACM Trans. on Networking*, 12(5):809–822, Oct 2004.

[24] I. A. Ushakov. *Handbook of Reliability Engineering*. John Wiley & Sons, Inc., 1994.

[25] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High Speed and Robust Event Correlation. *IEEE Communications Magazine*, 34(5):82–90, May 1996.

[26] S. Q. Zhuang, D. Geels, I. Stoica, and R. H. Katz. On Failure Detection Algorithms in Overlay Networks. In *Proc. of IEEE INFOCOM*, March 2005.

## Appendix I
## Calculation of Conditional Failure Probabilities

To compute the conditional failure probability of a node given a bad tree $T$, we first denote the nodes in $T$ by 1 to $|N|$ in breadth-first-search order. Let $T_i$ be the subtree rooted at node $i$, for $1 \le i \le |N|$. Thus, $T = T_1$. Let $C_i$ be the set such that $k \in C_i$ if node $k$ is a child of node $i$. Let $\mathcal{X}_i$ be the event that node $i$ is bad (i.e., $\Pr(\mathcal{X}_i) = p_i$). Let $\mathcal{T}_i$ be the event that subtree $T_i$ is bad. Thus, the conditional failure probability of node $i$ given a bad tree $T$ is:

$$\Pr(\mathcal{X}_i|\mathcal{T}_1) = \frac{\Pr(\mathcal{T}_1|\mathcal{X}_i)p_i}{\Pr(\mathcal{T}_1)} \quad \text{(by Bayes' rule)},$$

where

$$\Pr(\mathcal{T}_i) = p_i + (1-p_i)\textstyle\prod_{k\in C_i}\Pr(\mathcal{T}_k), \forall\, i \in [1, |N|],$$

$$\Pr(\mathcal{T}_i|\mathcal{X}_j) = \begin{cases} \Pr(\mathcal{T}_i), & \text{if } i > j \\ 1, & \text{if } i = j \\ p_i + (1-p_i)\prod_{k\in C_i}\Pr(\mathcal{T}_k|\mathcal{X}_j), & \text{if } i < j. \end{cases}$$

The idea here is that subtree $T_i$ is a bad tree if (i) node $i$ is bad or (ii) node $i$ is good and each of its child subtrees is a bad tree. Using dynamic programming [8], we can compute the conditional failure probability of a node in $\Theta(|N|^2)$ time.

## Appendix II
## Proofs

Part of the notation used in the proofs is defined in Section IV. In addition, we say nodes $i$ and $j$ are *ancestrally related* if node $i$ is either an ancestor or descendant of node $j$ (i.e., there exists a path in $T$ that covers both nodes $i$ and $j$), or *ancestrally unrelated* otherwise. To illustrate the ancestral relationships within a tree, we use the tree in Figure 5 as an example. Node 1 (resp. node 3) is an ancestor (resp. descendant) of node 3 (resp. node 1). Node 3 is ancestrally related to node 1, but is ancestrally unrelated to node 4.

**Sketch of Proof of Theorem 1**: Consider two diagnosis sequences $S_j = \langle j, k, l_3, \cdots l_{|N|}\rangle$ and $S_k = \langle k, j, l_3, \cdots, l_{|N|}\rangle$. If nodes $j$ and $k$ are ancestrally unrelated, then both $S_j$ and $S_k$ have the same expected cost since having checked one node has no impact on whether we will check another node on a different path.

On the other hand, if nodes $j$ and $k$ are ancestrally related, then given that $T$ is a bad tree and that node $j$ (resp. $k$) has been checked in $S_j$ (resp. $S_k$), we skip node $k$ (resp. $j$) and save cost $c_k$ (resp. $c_j$) if and only if checking node $j$ (resp. $k$) reveals a good path in $T$. This requires the following conditions to hold: (i) node $j$ (resp. $k$) is bad, (ii) node $k$ (resp. $j$) is good, and (iii) there exists a path $W_{jk}$ containing nodes $j$ and $k$ such that all nodes other than nodes $j$ and $k$ on $W_{jk}$ are good. Let $\mathcal{W}_{jk}$ be the event that condition (iii) holds. Note that $\mathcal{W}_{jk}$ are independent of $\mathcal{X}_j$ and $\mathcal{X}_k$ since $\mathcal{W}_{jk}$ corresponds to the nodes other than nodes $j$ and $k$. Thus,

$$\begin{aligned} & E[\text{cost of } S_j|\mathcal{T}] - E[\text{cost of } S_k|\mathcal{T}] \\ = \; & -c_k \Pr(\mathcal{X}_j, \overline{\mathcal{X}_k}, \mathcal{W}_{jk}|\mathcal{T}) + c_j \Pr(\overline{\mathcal{X}_j}, \mathcal{X}_k, \mathcal{W}_{jk}|\mathcal{T}) \\ = \; & \tfrac{1}{\Pr(\mathcal{T})}\big[-c_k \Pr(\mathcal{T}|\mathcal{X}_j, \overline{\mathcal{X}_k}, \mathcal{W}_{jk}) \Pr(\mathcal{X}_j, \overline{\mathcal{X}_k}, \mathcal{W}_{jk}) \; + \\ & c_j \Pr(\mathcal{T}|\overline{\mathcal{X}_j}, \mathcal{X}_k, \mathcal{W}_{jk}) \Pr(\overline{\mathcal{X}_j}, \mathcal{X}_k, \mathcal{W}_{jk})\big] \text{(by Bayes' rule)} \\ = \; & \tfrac{c_j c_k \Pr(\overline{\mathcal{X}_j}) \Pr(\overline{\mathcal{X}_k}) \Pr(\mathcal{W}_{jk})}{\Pr(\mathcal{T})}\big[-\tfrac{\Pr(\mathcal{T}|\mathcal{X}_j,\overline{\mathcal{X}_k},\mathcal{W}_{jk}) \Pr(\mathcal{X}_j)}{c_j \Pr(\overline{\mathcal{X}_j})} \; + \\ & \tfrac{\Pr(\mathcal{T}|\overline{\mathcal{X}_j},\mathcal{X}_k,\mathcal{W}_{jk}) \Pr(\mathcal{X}_k)}{c_k \Pr(\overline{\mathcal{X}_k})}\big] \\ & \text{(since } \mathcal{X}_j, \; \mathcal{X}_k, \text{ and } \mathcal{W}_{jk} \text{ are independent)} \\ = \; & \tfrac{c_j c_k \Pr(\mathcal{W}_{jk}) \Pr(\overline{\mathcal{X}_j}) \Pr(\overline{\mathcal{X}_k})}{\Pr(\mathcal{T})}\left[-\phi(j,T) + \phi(k,T)\right] \quad \cdots \text{(*)}. \end{aligned}$$

To understand (*), without loss of generality, let node $j$ be an ancestor of node $k$. The event $\mathcal{W}_{jk}$ implies that all ancestors of node $j$ are good, and the event $\mathcal{W}_{jk}\cap\overline{\mathcal{X}_j}$ implies that all ancestors of node $k$ (including node $j$) are good. Given that node $j$ (resp. node $k$) is bad, whether its descendants are good has no impact on $\Pr(\mathcal{T}|\mathcal{X}_j, \mathcal{A}_j)$ (resp. $\Pr(\mathcal{T}|\mathcal{X}_k, \mathcal{A}_k)$). Therefore, $\Pr(\mathcal{T}|\mathcal{X}_j, \mathcal{A}_j) = \Pr(\mathcal{T}|\mathcal{X}_j, \overline{\mathcal{X}_k}, \mathcal{W}_{jk})$ and $\Pr(\mathcal{T}|\mathcal{X}_k, \mathcal{A}_k) = \Pr(\mathcal{T}|\overline{\mathcal{X}_j}, \mathcal{X}_k, \mathcal{W}_{jk})$.

From (*), in order that node $j$ should be checked before node $k$, the expected cost of $S_j$ should be no greater than that of $S_k$, or equivalently, $\phi(j,T) \ge \phi(k,T)$. Therefore, it is intuitive to first check a node with high potential. ■