Toward Optimal Network Fault Correction in Externally Managed Overlay Networks

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

Abstract—We consider an end-to-end approach of inferring probabilistic data-forwarding failures in an externally managed overlay network, where overlay nodes are independently operated by various administrative domains. Our optimization goal is to minimize the expected cost of correcting (i.e., diagnosing and repairing) all faulty overlay nodes that cannot properly deliver data. 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% of time, and that first checking the candidate nodes rather than the most likely faulty nodes can decrease the checking cost of correcting all faulty nodes.

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

Note: An earlier and shorter conference version of this paper appeared in IEEE INFOCOM '07 [23].

1 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 [18], [19], [32]. However, it is also desirable to *repair* the faulty components to enable them to return to their operational states. Therefore, we focus on *network fault correction*, by which we mean not only to diagnose, but also to repair all faulty components within a network. In addition, it has been shown that a network outage can bring significant economic loss. For example, the revenue loss due to a 24-hour outage of a Switzerland-based Internet service provider can be more than CHF 30 million [15]. As a result, we want to devise a *cost-effective* network fault correction mechanism that corrects all network faults at minimum cost.

To diagnose (but not repair) network faults, recent approaches like [4], [28], [36] use all network nodes to collaboratively achieve this. For instance, in hop-by-hop authentication [4], 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 [10] and involves complicated administrative management [7]. In particular, it is difficult to directly monitor and access all overlay nodes in an *externally managed network*, whose routing nodes are independently operated by various administrative domains. In this case, we can only infer the network condition from end-to-end information.

Here, we consider an *end-to-end* inference approach which, using end-to-end measurements, *infers* components that are probably faulty in forwarding data in an application-layer overlay network whose overlay nodes are externally managed by independent administrative domains. We start with a routing tree topology with a set of overlay nodes, since a tree-based setting is typically seen in destination-based routing (e.g., CAN [30] and Chord [33]), where each overlay node builds a routing 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 then monitor every root-to-leaf overlay path. If a path exhibits any "anomalous behavior" in forwarding data, then some "faulty" overlay node on the path must be responsible. In practice, the precise definition of an "anomalous behavior" depends on specific applications. For instance, a path is said to be anomalous if it fails to deliver a number of correct packets within a time window. Using the path information collected at the application endpoints (i.e., leaf nodes), we can narrow down the space of possibly faulty overlay nodes.

In the above end-to-end solution, one can tell whether a path behaves anomalously, but cannot specifically tell which and how many overlay nodes on the path are faulty. Since we cannot directly monitor and access externally managed overlay nodes, in order to correct

- *P. Lee is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. E-mail: pclee@cse.cuhk.edu.hk*
- *V. Misra and D. Rubenstein are with the Department of Computer Science, Columbia University, New York, NY 10027. E-mail: {misra, danr}@cs.columbia.edu.*

the faulty nodes, we need to contact the administrators of the corresponding domains to manually check a sequence of potentially faulty nodes and fix any nodes that are found to be actually 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 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:

- Conventional failure localization problems (e.g., [18], [19], [32]) seek to identify the most likely faulty node (i.e., the node with the highest conditional failure probability given a network with faulty nodes). Here, we show that first checking 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 node (or the set of most likely faulty nodes), we show that by first checking the candidate nodes, we can decrease the checking cost of correcting all faulty nodes, for example, by more than 30% in some scenarios.

The remainder of the paper is organized as follows. In Section 2, we describe a class of externally managed overlay networks where our network fault correction mechanism is applicable. In Section 3, we formulate the network fault correction problem. Section 4 demonstrates that naive strategies that are intuitively optimal are in fact not optimal in general. In Section 5, we introduce a potential function for identifying the candidate nodes and show the optimality results. In Section 6, we propose several heuristics for the best node inference. In Section 7, we evaluate the proposed heuristics in large-scale networks. We review related work in Section 8 and conclude the paper in Section 9.

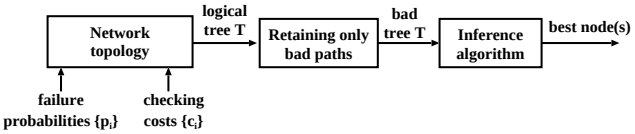


Fig. 1. End-to-end inference approach for a network fault correction scheme.

2 Externally Managed Overlay Networks

In this paper, we are interested in diagnosing and repairing faulty nodes in an *externally managed overlay network*, in which overlay nodes are independently operated by multiple *administrative domains*. By an administrative domain, we mean a single administrative authority that controls a collection of resources (e.g., routers and servers) [8]. Examples of externally managed overlay networks include Resilient Overlay Network (RON) [3], which provides routing resilience toward Internet path outages, and Service Overlay Network (SON) [14], which provides end-to-end quality-of-service guarantees. Both RON and SON deploy overlay nodes over multiple administrative domains that cooperatively accomplish certain network services. To ensure the availability of these network services, an effective network fault mechanism is therefore necessary.

Researchers also advocate the notion of security-oriented overlay architectures, such as Secure Overlay Services (SOS) [20] and Mayday [2], to defend against denial-of-service attacks. In both SOS and Mayday, data is securely tunneled over an overlay network. As shown in [20], successful data delivery is preserved with a very high probability even if a subset of overlay nodes are failed (e.g., shut down by attackers). While data may be re-routed to bypass failed nodes, robustness of data delivery will be degraded if the failed nodes are not immediately repaired because attackers can now devote resources to attacking the remaining non-failed nodes. Note that security-oriented overlay networks might be deployed over a number of end sites rather than a single ISP [20], and can be viewed as externally managed networks since each end site is independently operated. Thus, our proposed network fault correction mechanism is also essential for this type of networks.

3 Problem Formulation

In this section, we formulate our end-to-end inference approach for network fault correction. We are interested in diagnosing and repairing faulty nodes in an externally managed overlay network, in which the overlay nodes cannot be directly accessed. In order to diagnose failures, we need to contact the administrators of the corresponding domains to manually *check* potentially faulty nodes (e.g., by conducting a set of sanity tests). Any nodes that are found to be actually faulty will then be repaired.

Figure 1 summarizes the end-to-end inference approach. We consider a logical tree as $T = (N, \{p_i\}, \{c_i\})$,

where N is the set of overlay 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. The overlay node set N provides the topological information and specifies the sequence of overlay nodes, and hence the corresponding administrative domains, along which each data packet is traversed. On the other hand, the construction of $\{p_i\}$ and $\{c_i\}$ are based on the following failure and cost models, respectively.

Failure model. Our analysis focuses on overlay node failures such that nodes cannot properly forward data. For example, nodes can delay or drop packets to be forwarded because of power outage, full transmission queues, hardware errors, or route misconfiguration. In addition, our analysis focuses on *fail-stop* failures, meaning that a node completely stops its operations upon failures. Examples of fail-stop failures include power outage or machine shutdown. Under the fail-stop model, the failure probabilities $\{p_i\}$ can then be characterized via statistical measurements of reliability indexes [17] or vulnerability modeling [13]. However, we note that constructing accurate failure probabilities for overlay nodes is difficult in practice. Thus, we also evaluate the impact on our proposed approach due to inaccurate estimates of failure probabilities (see Section 7). Here, we consider the case where each overlay node is a physical node that possesses a single identity, and hence we assume that node failures and the corresponding failure probabilities $\{p_i\}$ are all independent.

Cost model. We characterize the checking costs $\{c_i\}$ using the personnel hours and wages required for troubleshooting problems or the cost of test equipment. Thus, the checking costs can be highly varying, depending on the administrative domains in which the checked nodes resides. Note that we do not consider the cost of repairing overlay nodes, and that the total checking cost is the only cost component being considered in our optimization problem, as explained later in this section.

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 [18], but can also be non-negligible if we are concerned with how likely a node is brought down due to catastrophic events that lead to large-scale failures.

Each node in a logical tree T is 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 example, we 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. Note that the inference approach only knows whether a path is anomalous, but does not specifically know 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.

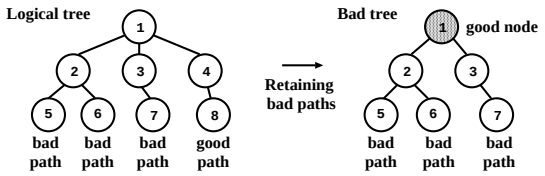


Fig. 2. Given a logical tree, we retain only the bad paths and indicate any good node. Since path $\langle 1, 4, 8 \rangle$ is a good path, it is known that nodes 1, 4, and 8 are good. Nodes 4 and 8 can be pruned from the tree, and node 1 can be indicated as good. The resulting set of bad paths will lead to a bad tree.

Since our failure model focuses on fail-stop failures, we further assume that a node exhibits the same behavior across all paths upon which it lies. With this assumption, if a node lies on at least one good path, then we infer that 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. On the other hand, if a node exhibits different behaviors across different paths, our analysis becomes more complicated, as a good path now possibly contains bad nodes as well. We pose the analysis of different behaviors of a node as future work.

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 we focus on the data-forwarding failures, the probes should represent the regular data packets that can be forwarded by overlay nodes. 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 2 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. Here, we assume that using end-to-end measurements (i.e., sending probe packets from the root to leaf nodes) to determine bad paths incurs negligible cost. Thus, 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 total 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 simultaneously inferring and checking multiple nodes in Section 7.

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:

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

We detail in Appendix A 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 [29], 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 the time scale of topology changes, the topology should remain fairly stable.

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 Appendix A, 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?*

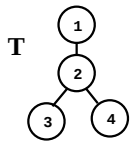
Algorithm 1 Brute-force inference algorithm

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^*

4 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 A 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.

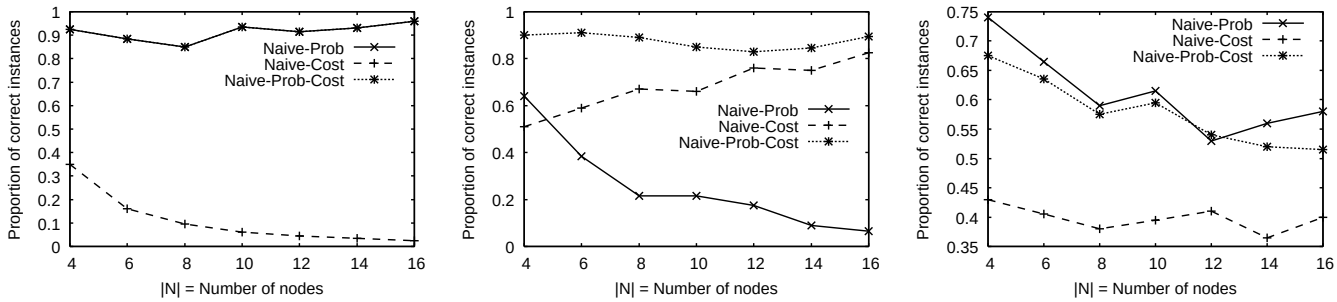
We first give a simple counter-example that disproves the above naive choices. Figure 3 illustrates a bad tree rooted at node 1 and the corresponding failure probabilities $\{p_i\}$ and checking costs $\{c_i\}$. As verified by the brute-force approach in Algorithm 1, the best node is node 2, where a 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, nor the node with the highest ratio of the conditional failure probability to the checking cost.



i	p_i	c_i	Cond. p_i	(Cond. p_i) : c_i
1	0.1	1	0.6586	0.6586
2	0.05	0.3	0.3293	1.0977
3	0.08	0.2	0.1214	0.6072
4	0.1	0.1	0.1405	1.405

Fig. 3. A counter-example of showing the best node (which is node 2) is not the one chosen by the naive choices.

To further understand the performance of the naive choices, we evaluate three naive heuristics for the inference algorithm: (i) *Naive-Prob*, which returns the node with the highest conditional failure probability, (ii) *Naive-Cost*, which returns the node with the least checking cost, and (iii) *Naive-Prob-Cost*, which returns the node with the highest ratio of the conditional failure probability to the 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 Figure 3). Such a setting



(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. 4. Proportions of instances in which Naive-Prob, Naive-Cost, and Naive-Prob-Cost return a correct best node. Note that in (a), Naive-Prob and Naive-Prob-Cost are identical.

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 3, 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 4 illustrates the proportions of instances (out of 200) in which Naive-Prob, Naive-Cost, and Naive-Prob-Cost return correctly a best node (which may not be unique). 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 Naive-Prob and Naive-Cost (see Figures 4(a) and 4(b)), and less than 55% for Naive-Prob-Cost (see Figure 4(c)). Therefore, to determine the best node to first check, our inference algorithm should use a measure that better combines failure probabilities, checking costs, as well as the structure of the bad tree.

5 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 $\Pr(\mathcal{A}_r) = 1$.

5.1 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 [17]). 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 A, 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.

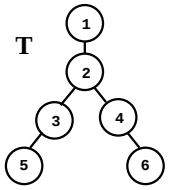
We can view the selection of candidate nodes as a mapping function that returns a single candidate node given W and T. Note that it is possible for a path in T to have multiple candidate nodes that correspond to different paths in T.

5.2 Optimality Results

We now state the main optimality result as follows. In the interest of space, the formal proofs are detailed in [22].

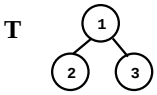
Theorem 1: Given a bad tree T, there always exists an optimal diagnosis sequence that starts with a candidate node.

In other words, the best node returned by our inference approach should be one of the candidate nodes. We



i	p_i	c_i	$\phi(i,T)$
1	0.4	1	0.67
2	0.4	1	0.67
3	0.9	1	8.46
4	0.9	0.9	9.9
5	0.9	1	8.46
6	0.4	1	0.66

Fig. 5. Counter-example 1.



i	p_i	c_i	$\phi(i,T)$
1	0.1	1	0.111
2	0.9	1	7.2
3	0.8	1	3.6

Fig. 6. Counter-example 2.

present a sketch of proof of Theorem 1 in Appendix B to illustrate the intuition of how the potential function is derived and why a candidate node should be first checked. The formal proof is presented in [22].

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

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 [17].

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.

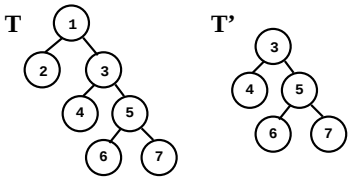
Remark: If T is a bad tree, then the root node r is also the node that has the maximum conditional failure probability since $\Pr(\mathcal{X}_r|\mathcal{T}) = \frac{p}{\Pr(\mathcal{T})} \geq \frac{\Pr(\mathcal{T}|\mathcal{X}_i)p}{\Pr(\mathcal{T})} = \Pr(\mathcal{X}_i|\mathcal{T})$ for every node i.

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.

5.3 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. In the following discussion, the optimality results are verified by the brute-force inference algorithm in Algorithm 1.

Counter-example 1: Given a bad tree, the best node is not the one with the highest potential. Figure 5 shows a bad tree T in which the candidate nodes are nodes 3 and 4. Although node 4 has the highest potential, node 3 is the best node (so is node 5). A possible optimal diagnosis sequence is $\langle 3,5,1,2,4,6\rangle$, whose expected cost 4.309, while the expected cost of any diagnosis sequence that starts with node 4 is at least 4.344.



i	p_i	c_i
1	0.939	1
2	0.806	1
3	0.042	1
4	0.979	1
5	0.326	1
6	0.992	1
7	0.997	1

Fig. 7. Counter-example 3.

Counter-example 2: Checking simultaneously all candidate nodes in a bad tree does not minimize the expected cost of correcting all faulty nodes. Figure 6 illustrates a bad tree T that has nodes 2 and 3 as the candidate nodes. To check simultaneously all candidate nodes, we can construct a diagnosis sequence $S = \langle 2,3,1\rangle$, whose expected cost is given by 2.134. However, the optimal diagnosis sequence is $S^* = \langle 2,1,3\rangle$, whose expected cost is 2.107.

To explain this counter-example, note that after node 2 has been checked, we have a new tree $T' = (N, \{p_i|p_2 = 0\}, \{c_i\})$, where setting $p_2 = 0$ is to indicate that node 2 is now known to be good (see Section 3). The potentials of nodes 1 and 3 in T' are respectively $\phi(1,T') = 0.111$ and $\phi(3,T') = 0$. Thus, node 1 is the only candidate node in T', and it must be the next node to check.

Counter-example 3: The best node for a bad tree is not necessarily the best node for a subtree. Consider a bad tree T in Figure 7. The best node is node 7, and the optimal diagnosis sequence is $\langle 7,1,2,4,3,5,6\rangle$. However, for the subtree T' rooted at node 3, if it is a bad tree, then the best node is node 4, and the optimal diagnosis sequence is $\langle 4,3,7,5,6\rangle$. The reason is that the subtree T', when residing in T, may or may not be a bad tree. Even if we have found the best node in T, the best node can still vary if we know that subtree T' is a bad tree. Therefore, 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.

5.4 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 4, 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 8 plots the results. Comparing the results to those in Figure 4, 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.

5.5 Complexity of Finding the Node with Maximum Potential

Since the computation of the potential of a node needs $\Theta(|N|^2)$ time (see Section 5.1), 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.

6 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 different groups of efficient heuristics for the inference algorithm that are suitable for large-scale networks. Each heuristic belongs to one of the two classes: (i) *naive heuristics*, which consider only the most likely faulty nodes based on the conditional failure probability distribution, and (ii) *candidate-based heuristics*, which consider the candidate nodes based on both the conditional failure probability as well as the checking cost distributions.

6.1 Single Node Inference for a Single Tree

We consider two heuristics Naive-Prob (see Section 4) and Cand-Pot (see Section 5), which respectively return the node that is most likely faulty given a bad tree and the candidate node with the highest potential.

6.2 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.

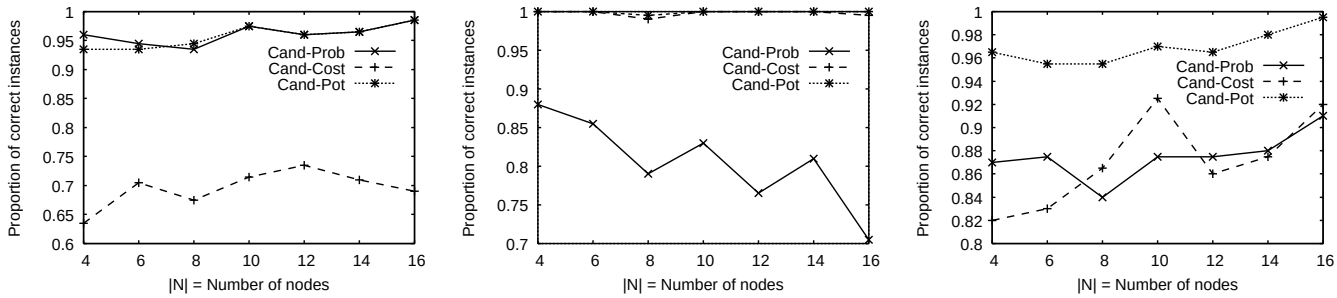
Algorithm 3 Pa-Cand

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}

6.3 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 [11]. Therefore, we consider two multi-tree-based 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.

Note that in typical overlay networks, an overlay node may possess multiple logical identities, or may be shared by multiple overlay networks. Thus, multiple failed points may actually correspond to the failure of a single physical node, and this implies that node failures



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

are correlated. Our multi-tree-based heuristics provide a preliminary attempt to illustrate how we can correlate node failures in multiple tree-based topologies.

7 Experiments in Large-Scale Networks

In this section, we evaluate via simulation the naive heuristics (i.e., Naive-Prob, Pa-Naive-Prob, and Mt-Pa-Naive-Prob) and the candidate-based heuristics (i.e., Cand-Pot, Pa-Cand, and Mt-Pa-Cand) described in Section 6 in correcting faulty nodes in large-scale networks.

7.1 Experimental Setup

We use the BRITE generator [27] to generate random overlay networks based on the router Waxman model. In our prior conference version [23], we also consider the router-level Barbási-Albert model [5]. While unstructured overlay networks (e.g., Gnutella [9]) generally follow the Barbási-Albert model, structured networks (e.g., CAN [30] and Chord [33]) are similar to random networks in the sense that nodes have similar degrees [12]. As shown in our simulation results, both graph models lead to similar observations. Thus, we focus on the Waxman model in this paper.

Each network topology that we generate contains 200 nodes and 800 links. We treat the nodes and links as overlay nodes and overlay links, respectively. We then 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. Our simulation focuses on monitoring overlay nodes so as to demonstrate the importance of first checking candidate nodes. In each of the instances, each overlay node is assigned a failure probability p_i and a checking cost c_i. Unless otherwise specified, p_i and c_i follows one of the 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 4, denotes the uniform distribution between u_1 and u_2.

To construct a logical tree, we first create a shortest-path tree rooted at a randomly selected router based on the given overlay 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 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], [6] and is beyond the scope of this paper.

Given a single or multiple logical trees, we simulate the faults by letting each overlay node 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.

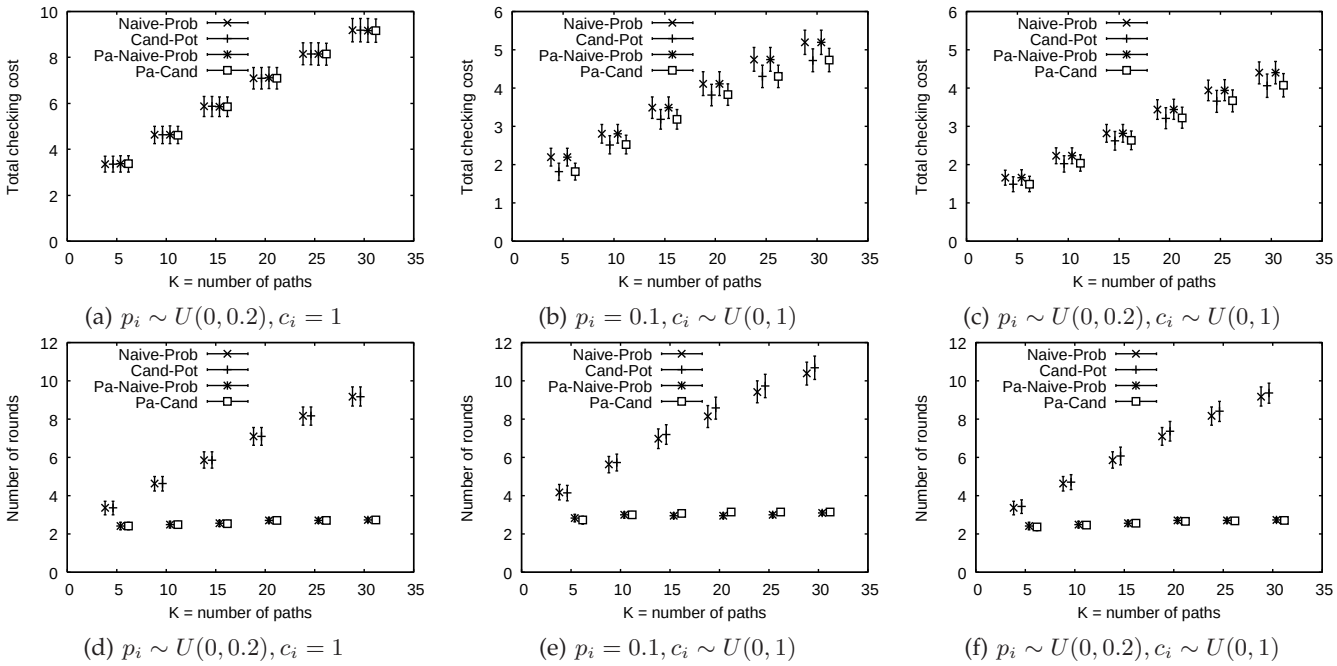
Due to the large network size, we cannot apply the brute-force inference algorithm in Algorithm 1 to determine the proportion of correct identification as in Sections 4 and 5. Therefore, our experiments mainly focus on two metrics to evaluate our heuristics:

- *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.

7.2 Experimental Results

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

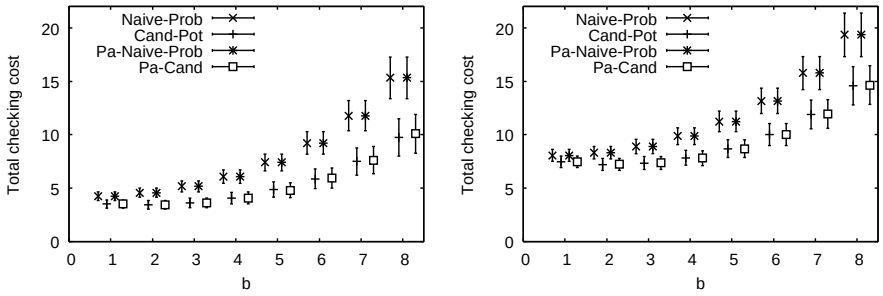
Experiment 1 (Comparison of single-tree-based heuristics). We first consider the performance of the heuristics when we only monitor a single logical tree. Figure 9 shows the performance of Naive-Prob, Cand-Pot, Pa-Naive-Prob, and Pa-Cand versus K, the number of paths that are included in a logical tree. As K increases, more nodes are being monitored, and hence



(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 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.



(a) $K = 5$ (b) $K = 20$

Fig. 10. Experiment 2: Comparison of single-based heuristics in terms of the total checking cost under a contrived checking cost distribution.

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 9(a)), both naive and candidate-based heuristics have nearly identical total checking cost. This also confirms the effectiveness of conventional fault localization approaches, whose focus is to identify the most likely faults without considering the variance of the cost component. However, the candidate-based heuristics reduce the total checking costs of the naive heuristics when c_i is varied, for instance, by 8-18% when p_i is fixed and c_i is varied (see Figure 9(b)), and by 7-12% when both p_i and c_i are varied (see Figure 9(c)). It should be noted that such cost reduction can imply significant revenue saving, given that the revenue loss due to a network outage is generally huge [15]. This demonstrates the competence of the candidate-based heuristics when the physical components have varying checking costs (see Section 3).

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 single-tree-based heuristics under a contrived checking cost distribution). We point out that some distributions of checking

costs can lead to a large performance improvement of the candidate-based heuristics over the naive heuristics. Intuitively, if checking costs are fixed for all nodes in a bad tree, then both naive and candidate-based heuristics prefer to first check the nodes near the root node, as such nodes have higher conditional failure probabilities On the other hand, if the nodes around the leaf positions have small checking costs, then they may be preferred to be first checked by the candidate-based heuristics based on the potential calculation, but not by the naive heuristics, which do not consider checking costs. We examine this observation in this experiment using a contrived checking cost distribution.

For a given logical tree T that we monitor, we first identify the *depth* d_i of each node i, i.e., the path length from root to node i (we let $d_r = 0$ for root r). Also, we let d_{max} be the maximum depth of all nodes in T. Then we let the checking cost c_i of node i be $c_i = (1.5 - d_i/d_{max})^b$, where b is some exponent that we vary in our evaluation. Note that as b increases, nodes closer to the root node will have increasingly higher checking costs than those near the leaf nodes. Here, we set $p_i \sim U(0, 0.2)$, and consider the cases where $K = 5$ and $K = 20$ in a logical tree.

Figure 10 shows the total checking costs of different heuristics versus b. The improvement of the candidate-based heuristics over the naive heuristics increases with b. Specifically, when $b \geq 4$, Cand-Pot (resp. Pa-Cand) reduces the total checking cost of Naive-Prob (resp. Pa-Naive-Prob) by more than 30% when $K = 5$ and by more than 20% when $K = 20$.

Experiment 3 (Comparison of multi-tree-based heuristics). We now evaluate the performance of the heuristics when we monitor multiple logical trees. Figure 11 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 10-16% when p_i is fixed and c_i is varied (see Figure 10(b)), and by 7-11% when both p_i and c_i are varied. Also, both Mt-Pa-Naive-Prob and Mt-Pa-Cand call the inference algorithm with about the same number of rounds.

Experiment 4 (Inaccuracies in failure probability estimation). In general, it is difficult to accurately estimate the failure probabilities of all overlay nodes. In this experiment, we study how our heuristics are affected subject to the inaccurate failure probability estimation. We let p_i and p_i' be the actual and estimated failure probability for node i, respectively. We let $p_i \sim U(0, 0.2)$. We then consider three cases (i) $p_i' = p_i$ (i.e., no error), (ii) $p_i' = 0.1$, and (iii) $p_i' = 0.2$. We assume that $c_i \sim U(0, 1)$.

Figure 12(a) shows the performance of the single-tree-based heuristics Naive-Prob and Cand-Pot versus K (as in Experiment 1). As expected, the inaccurate estimation of failuire probabilities increases the total checking costs of all heuristics. However, Cand-Pot still reduces the total checking costs of Naive-Prob in all cases (by 14-18%). Figure 12(b) shows the performance of the multi-tree-based heuristics Pa-Naive-Prob and Pa-Cand (as in Experiment 2, where we also set K= 20 here), and Pa-Cand still gives smaller total checking costs than Pa-Naive-Prob for different estimates of failure probabilities.

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 overlay nodes 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. Furthermore, we observe similar improvement of the candidate-based heuristics over the naive heuristics even though the estimates of failure probabilities are inaccurate.

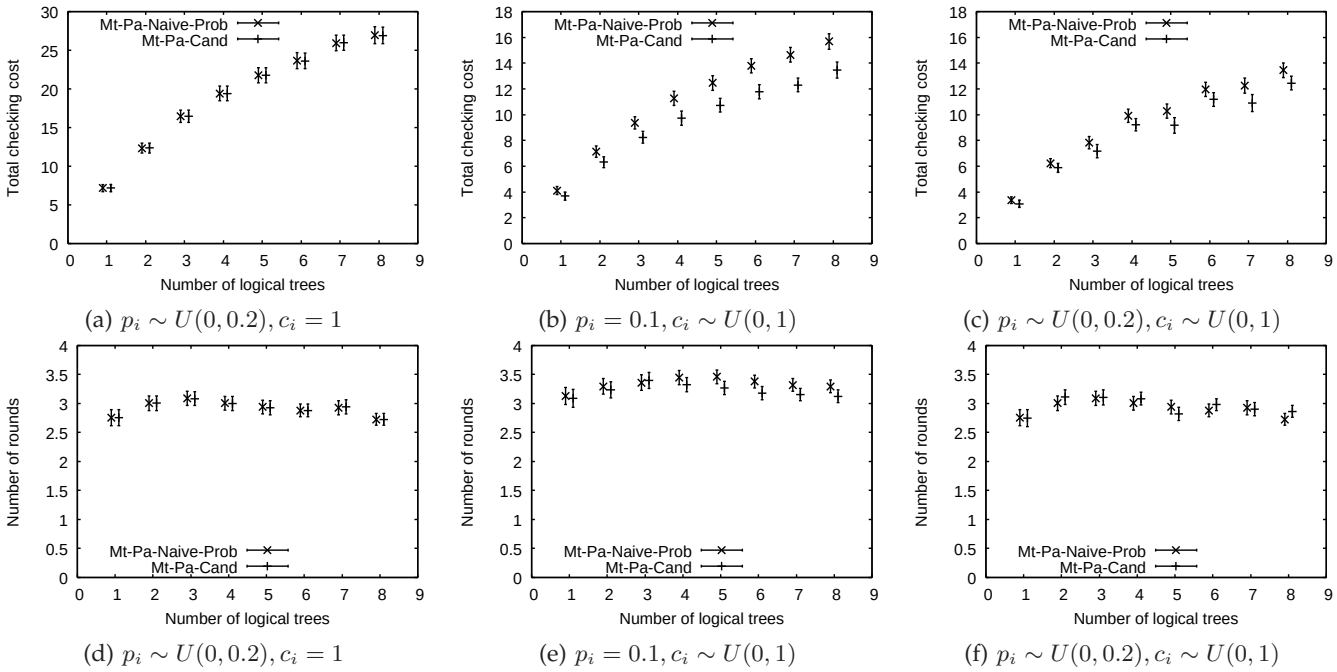
8 Related Work

Our end-to-end inference solution is inspired by [7], [10], [16], [26], 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.

Fault diagnosis is an important topic in network management (see survey in [31]). For example, the codebook approach [35] and the Bayesian approach [21] consider a deterministic setting where network faults are equally likely to occur. To address probabilistic faults, [18], [19], [32] seek to localize the most probable subset of faults given the observed symptoms. Katzela and Schwartz [19] show that such a problem is generally NP-hard. In view of this, Steinder and Sethi [32] formulate the problem based on bipartite belief networks and propose efficient techniques on localizing end-to-end multi-layer failures. Kandula *et al.* [18] 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.

References [25], [36] consider various dimensions of performance of failure detection in overlay networks, in which nodes periodically probe their neighbors to determine failures. Our work considers the case where nodes are externally managed and they need to be manually checked in order to determine failures.

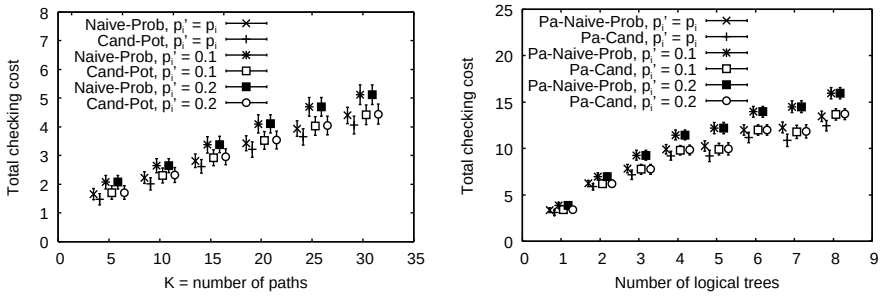
Fault correction has also been studied extensively in reliability engineering (see survey in [24]). The work that is closest to ours is in [17], [34], whose objective is also to minimize the cost of correcting all faulty units. However, the optimality assumes a *series* system, which is equivalent to a single path in a network setting. In contrast, we consider a routing tree whose paths are shared and dependent.



(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. 11. Experiment 3: 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)).



(a) Single-tree case (b) Multi-tree case

Fig. 12. Experiment 4: Comparison of naive and candidate-based heuristics in terms of the total checking cost with inaccurate failure probability estimation.

9 Conclusions

We present the optimality results for an end-to-end inference approach to correct (i.e., diagnose and repair) probabilistic network faults at minimum expected cost. One motivating application of using this end-to-end inference approach is an externally managed overlay network, where we cannot directly access and monitor nodes that are independently operated by different administrative domains, but instead we must infer failures via end-to-end measurements. We show that first checking 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. In view of this, we construct 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 propose several efficient heuristics that are suitable for correcting fault nodes in large-scale overlay networks. We show 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.

Appendix A
Calculation of Two Quantities

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 $1 \leq i \leq |N|$. 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 [11], we can compute the conditional failure probability of a node in $\Theta(|N|^2)$ time.

Expected Cost of a Diagnosis Sequence. Let $S = \langle l_1, l_2, \cdots, l_{|N|}\rangle$ be a diagnosis sequence on $T = (N, \{p_i\}, \{c_i\})$. Let $\mathcal{T}$ be the event that T is a bad tree. Let $\mathcal{T}_{l_i}^D$ be the event that the subtree rooted at node l_i is a bad tree after nodes $l_1, \cdots, l_{i-1}$ have been examined (i.e., checked or skipped). Let $\mathcal{A}_{l_i}^D$ be the event that every ancestor j of node l_i, such that $j \in \{l_{i+1}, \cdots, l_{|N|}\}$ (i.e., node j is examined after node l_i), is a good node.

When node l_i is to be examined, nodes $l_1, \cdots, l_{i-1}$ are known to be good nodes. To account for the presence of any known good node i in a bad tree T during the calculation, we can set its failure probability $p_i = 0$. Thus,

$$\Pr(\mathcal{T}_{l_i}^D) = \Pr(\mathcal{T}_{l_i}|p_{l_1} = \cdots = p_{l_{i-1}} = 0), \text{ and}$$
$$\Pr(\mathcal{A}_{l_i}^D) = \Pr(\mathcal{A}_{l_i}|p_{l_1} = \cdots = p_{l_{i-1}} = 0),$$

where $\mathcal{T}_{l_i}$ is the event that the subtree rooted at node l_i is bad, and $\mathcal{A}_{l_i}$ is the event that all ancestors of node l_i are good.

If the subtree rooted at node l_i remains a bad tree or at least one of the ancestors of node l_i is a bad node, then node l_i still lies on bad paths only and it has to be checked. Thus,

$$\begin{aligned}&\Pr(\text{node } l_i \text{ is checked}\big|\mathcal{T}, \text{nodes } l_1, \cdots, l_{i-1} \text{ known to be good})\\ =&\Pr(\mathcal{T}_{l_i}^D \cup \overline{\mathcal{A}_{l_i}^D}|\mathcal{T}).\end{aligned}$$

Suppose that T is a bad tree. Thus, the expected cost of S given that T is a bad tree is:

$$\begin{aligned}&\textstyle\sum_{i=1}^n c_{l_i}\Pr(\mathcal{T}_{l_i}^D \cup \overline{\mathcal{A}_{l_i}^D}|\mathcal{T})\\ =&\textstyle\sum_{i=1}^n c_{l_i}\big[1 - \Pr(\overline{\mathcal{T}_{l_i}^D} \cap \mathcal{A}_{l_i}^D\big|\mathcal{T})\big]\\ =&\textstyle\sum_{i=1}^n c_{l_i}\big[1 - \Pr(\mathcal{A}_{l_i}^D\big|\mathcal{T}) + \Pr(\mathcal{T}_{l_i}^D \cap \mathcal{A}_{l_i}^D\big|\mathcal{T})\big]\\ =&\textstyle\sum_{i=1}^n c_{l_i}\big[1 - \frac{\Pr(\mathcal{T}\big|\mathcal{A}_{l_i}^D)\Pr(\mathcal{A}_{l_i}^D)}{\Pr(\mathcal{T})} + \frac{\Pr(\mathcal{T}\big|\mathcal{T}_{l_i}^D,\mathcal{A}_{l_i}^D)\Pr(\mathcal{T}_{l_i}^D)\Pr(\mathcal{A}_{l_i}^D)}{\Pr(\mathcal{T})}\big].\end{aligned}$$

Note that $\mathcal{T}_{l_i}^D$ depends on node l_i and its descendants, while $\mathcal{A}_{l_i}^D$ depends on the ancestors of node l_i. Thus, $\mathcal{T}_{l_i}^D$ and $\mathcal{A}_{l_i}^D$ are independent. We can obtain $\Pr(\mathcal{T}|\mathcal{A}_{l_i}^D)$ by setting $p_j = 0$ for every ancestor j of node l_i such that $j \in \{l_{i+1}, \cdots, l_{|N|}\}$. and computing $\Pr(\mathcal{T})$ using the modified failure probabilities. We can compute $\Pr(\mathcal{T}\big|\mathcal{T}_{l_i}^D, \mathcal{A}_{l_i}^D)$ in the same way, except that we additionally set $p_{l_i} = 1$ for the event $\mathcal{T}_{l_i}^D$. Also, we can compute $\Pr(\mathcal{T}_{l_i}^D)$ and $\Pr(\mathcal{A}_{l_i}^D)$ as previously stated. Computing $\Pr(\mathcal{T})$ takes $\Theta(|N|^2)$ time. Thus, the complexity of computing the expected cost is $\Theta(|N|^3)$.

APPENDIX B
PROOFS

Part of the notation used in the proofs is defined in Section 5. 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 3 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,

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

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, $\phi(j,T) \geq \phi(k,T)$. Therefore, it is intuitive to first check a node with high potential. ∎

Proof of Corollary 1: Note that if T is a single path, then $\Pr(\mathcal{T}|\mathcal{X}_i, \mathcal{A}_i) = 1$ for all i. Thus, the best node will be

the only candidate node whose potential is $\max_i \frac{p_i}{c_i(1-p_i)}$. ∎

Proof of Corollary 2: Let node r be the root node. Then $\Pr(\mathcal{T}|\mathcal{X}_r, \mathcal{A}_r) = 1$. Thus, its potential $\phi(r,T)$ is no less than that of any other non-root node in T. Therefore, node r is the only candidate node in T and is hence the best node to be first checked. ∎

Proof of Corollary 3: Consider a two-level tree T whose root node r is attached to $|N|-1$ nodes. If node r has the maximum conditional failure probability given that T is a bad tree, then for every leaf node $i \neq r$, we have $\Pr(\mathcal{X}_r|\mathcal{T}) \geq \Pr(\mathcal{X}_i|\mathcal{T}) \Leftrightarrow \frac{p_r}{\Pr(\mathcal{T})} \geq \frac{p_r p_i + (1-p_r)\prod_{j\neq r} p_j}{\Pr(\mathcal{T})}$ $\Leftrightarrow \frac{p_r}{c(1-p_r)} \geq \frac{\prod_{j\neq r} p_j}{c(1-p_i)} \Leftrightarrow \phi(r,T) \geq \phi(i,T)$. Thus, the root node also has the maximum potential, implying that it is the only candidate node. By Theorem 1, it is also the best node.

On the other hand, suppose that some leaf node $k \neq r$ has the maximum conditional failure probability. Suppose the contrary that the optimal diagnosis sequence S^* does not start with node k. Hence, let $S^* = \langle l_1, l_2, \cdots, l_{j-1}, k, l_{j+1}, \cdots, l_{|N|}\rangle$. We want to show that moving node k to the front of S^* does not increase the expected cost of S^*. We consider two cases:

- *Case 1: Node l_1 is the root node r.* Consider the diagnosis sequence $S' = \langle k, r, l_2, \cdots, l_{j-1}, l_{j+1}, \cdots, l_{|N|}\rangle$. Note that for any node l_m, where $m = 2, \cdots, j-1, j+1, \cdots, |N|$, if it is checked (resp. skipped) in S^*, it will also be checked (resp. skipped) in S', and vice versa. Thus, we skip node r (resp. k) in S' (resp. S^*) and save cost c if and only if node r (resp. k) is good. The difference between the expected cost in S' and S^* is $E[\text{cost of } S'|\mathcal{T}] - E[\text{cost of } S^*|\mathcal{T}]$ = $-c\Pr(\overline{\mathcal{X}_r}|\mathcal{T}) + c\Pr(\overline{\mathcal{X}_k}|\mathcal{T}) \leq 0$, since $\Pr(\mathcal{X}_k|\mathcal{T})$ is maximum.
- *Case 2: Node $l_1 \neq r$ is a leaf node.* In S^*, we claim that node l_2 is the root node r. After l_1 is checked, the next node in S^* will be a candidate node in $T' = (N, \{p_i|p_{l_1} = 0\}, \{c_i\})$. Since every leaf node in T' has zero potential, the root node r is the only candidate node, and hence $l_2 = r$. We then construct a diagnosis sequence $S'' = \langle k, r, l_1, l_3, \cdots, l_{j-1}, l_{j+1}, \cdots, l_{|N|}\rangle$. Similar to Case 1, the difference of the expected cost in S'' and S^* is $E[\text{cost of } S''|\mathcal{T}] - E[\text{cost of } S^*|\mathcal{T}] = -c\Pr(\overline{\mathcal{X}_{l_1}}|\mathcal{T}) + c\Pr(\overline{\mathcal{X}_k}|\mathcal{T}) \leq 0$, since $\Pr(\mathcal{X}_k|\mathcal{T})$ is maximum.

Since there exists a diagnosis sequence that starts with node k and has no greater expected cost than does S^*, node k is a best node. ∎

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their useful comments on improving this paper. This work was supported in part by the National Science Foundation under grant numbers CNS-0626795 and CCR-0615126.

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] D. Andersen. Mayday: Distributed Filtering for Internet Services. In *4th Usenix Symposium on Internet Technologies and Systems*, Mar 2003.

[3] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, Oct 2001.

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

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

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

[7] 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.

[8] K. Carlberg. Emergency Telecommunications Services (ETS) Requirements for a Single Administrative Domain, Jan 2006. RFC 4375.

[9] Clip2. The Gnutella Protocol Specification v0.4. Available on http://www.limewire.com.

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

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

[12] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Proc. of NDSS*, 2007.

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

[14] Z. Duan, Z.-L. Zhang, and Y. Hou. Service Overlay Networks: SLAs, QoS, and Bandwidth Provisioning. *IEEE/ACM Trans. on Networking*, 11(6):870–883, Dec 2003.

[15] T. Dübendorfer, A. Wagner, and B. Plattner. An Economic Damage Model for Large-Scale Internet Attacks. In *Proc. of IEEE WETICE*, Jun 2004.

[16] 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.

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

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

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

[20] A. Keromytis, V. Misra, and D. Rubenstein. SOS: An Architecture for Mitigating DDoS Attacks. *IEEE JSAC, Special Issue on Service Overlay Networks*, 22(1), January 2004.

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

[22] 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.

[23] P. P. C. Lee, V. Misra, and D. Rubenstein. Toward Optimal Network Fault Correction via End-to-End Inference. In *Proc. of IEEE INFOCOM*, May 2007.

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

[25] Z. Li, L. Yuan, P. Mohapatra, and C.-N. Chuah. On the Analysis of Overlay Failure Detection and Recovery. *Computer Networks*, 51:3823–3843, 2007.

[26] 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.

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

[28] 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.
[29] M. Rabbat, R. Nowak, and M. Coates. Multiple Source, Multiple Destination Network Tomography. In *Proc. of IEEE INFOCOM*, 2004.
[30] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, 2001.
[31] 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.
[32] 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.
[33] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, 2001.
[34] I. A. Ushakov. *Handbook of Reliability Engineering*. John Wiley & Sons, Inc., 1994.
[35] 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.
[36] 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.