Fault Localization Using Passive End-to-End Measurement and Sequential Testing for Wireless Sensor Networks

Bing Wang, Wei Wei, Wei Zeng
Computer Science & Engineering Dept.
University of Connecticut, Storrs, CT, 06269
{bing, weiwei, wei.zeng}@engr.uconn.edu

Krishna R. Pattipati
Electrical & Computer Engineering Dept.
University of Connecticut, Storrs, CT, 06269
krishna@engr.uconn.edu

***Abstract*—Faulty components in a network need to be localized and repaired to sustain the health of the network. In this paper, we propose a novel approach that carefully combines active and passive measurements to localize faults in wireless sensor networks. More specifically, we formulate a problem of *optimal sequential testing guided by end-to-end data*. This problem determines an optimal testing sequence of network components based on end-to-end data in sensor networks to minimize testing cost. We prove that this problem is NP-hard and propose a greedy algorithm to solve it. Extensive simulation shows that in most settings our algorithm only requires testing a very small set of network components to localize and repair *all* faults in the network. Our approach is superior to using active and passive measurements in isolation. It also outperforms the state-of-the-art approaches that localize and repair all faults in a network.**

I. Introduction

Wireless sensor networks have been deployed for a wide range of applications. A deployed sensor network may suffer from many network-related faults, e.g., failure or lossy nodes or links [1], [2]. These faults affect the normal operation of the network, and hence should be detected, localized and corrected/repaired. Existing studies on sensor network fault localization use active or passive measurement (see Section II). Active measurement incurs additional monitoring traffic (a node needs to monitor itself or its neighbors, and transmit the monitoring results locally or to a centralized server). It hence consumes precious energy of sensor nodes, and may reduce the lifetime of the network. On the other hand, it has the advantage that it can exactly pinpoint the faults. Passive measurement uses existing end-to-end data inside the network. It introduces no additional traffic into the network, and hence is an attractive approach for energy-stringent sensor networks. On the other hand, it poses the challenge of fault *inference* since a faulty end-to-end behavior only indicates that some components are faulty and does not dictate exactly which components are faulty. Accurate inference from end-to-end data (i.e., locating all faults with low false positives) is not always possible because end-to-end measurement can have inherent ambiguity (see Section III).

Motivated by the complementary strengths of active and passive measurement, we propose a novel approach that uses active measurement to resolve ambiguity in passive measurement, and uses passive measurement to guide active measurement to reduce testing cost (i.e., cost incurred from active measurement, see Section III). More specifically, we formulate a problem of *optimal sequential testing guided by end-to-end data*. This problem determines an optimal testing sequence of network components that minimizes the total testing cost: it picks the first component to be tested (through active measurement), based on the test result (i.e., it is faulty or not faulty) and the end-to-end data, it determines the next component to be tested. This sequential testing continues until the identified faulty components have explained all end-to-end faulty behaviors. Since these identified faults may not have included all faults in the network, we identify all faults by solving the optimal sequential testing problem in iterations. In an iteration, based on end-to-end data in this iteration, we solve the optimal sequential testing problem to identify a set of faulty components. We then repair all the identified faulty components and start the next iteration. The iteration repeats until all end-to-end behaviors are normal. At this time, *all* faulty components have been identified and repaired.

We prove the problem of optimal sequential testing is NP-hard, and develop a greedy algorithm to solve it. We evaluate the performance of our algorithm through extensive simulation in sensor networks of static and dynamic topologies. Our simulation results show that we only need a few iterations and testing a very small subset of components to localize all faults in a network. These results demonstrate the benefits of our approach: it is superior to using active measurement alone since it selectively tests a very small subset of components; it is superior to fault inference using passive measurement since it localizes *all* faulty components while fault inference may suffer from a large number of false positives and negatives [3], [4], [5].

Our approach also outperforms two state-of-the-art approaches [6], [7] that identify and repair all faults in a network. These two approaches also run in iterations. The exhaustive inspection approach [6] differs from our approach in that in each iteration, it infers *in parallel* (rather than in sequence) a set of potential faulty components from end-to-end measurement, and tests each identified component and repairs the faulty ones at the end of the iteration. It has to test all identified components because some of them may be false positives. Hence the number of tests is at least equal to the total number of faulty components, while our approach requires much less number of tests. The approach in [7] infers the best component to be tested in each iteration, and repairs the component if necessary. Since it only tests a single component in an iteration, it may lead to a large number of iterations to localize and correct all faults. To reduce the number of iterations, the authors also consider identifying multiple faults *in parallel* in an iteration, which is in a similar spirit as exhaustive inspection [6] and suffers from the same drawbacks.

The rest of the paper is organized as follows. Section II reviews related work. Section III presents the problem setting. Section IV describes our approach. Section V presents evaluation results. Finally, Section VI concludes the paper and presents future work.

II. Related work

We consider localizing and correcting network-related faults in a deployed sensor network. Most existing studies use either active or passive measurement for this purpose.

Active measurement provides accurate view of the network at the price of introducing additional monitoring traffic into the network. Zhao et al. design a residual energy scan for a sensor network that depicts the remaining energy inside the network [8]. From the scan, a network operator can discover areas with low residual energy and take corrective actions. The same authors also propose an architecture that computes aggregates for sensor network monitoring [9]. This approach continuously collects aggregates (sum, average, count) of network properties (e.g., loss rates, energy level), triggers scan of the network when observing sudden changes in the aggregates, and further debugs the problem through detailed dumps of node states. To limit the scope of the monitoring traffic to a local area, Hsin et al. propose a distributed monitoring architecture where each node monitors its neighbors by periodically sending them probes [10]. More recently, Tolle et al. design a sensor network management system (SNMS) that allows a network operator to query the network for health information [1]. Whitehouse et al. propose Marionette, which extends SNMS by providing users the ability to call functions, and read or write variables in embedded applications [11]. Ramanathan et al. propose Sympathy, a tool for detecting and debugging failures in sensor networks [2]. Sympathy carefully selects metrics that enable efficient failure detection and includes an algorithm that analyzes root causes. Rost and Balakrishnan design a health monitoring system, Memento, that delivers state summaries and detects node failures [12]. Last, Gruenwald et al. propose a remote management system for a wide area sensor network that contains multiple and heterogenous networks [13].

Active measurement consumes precious resources of the senor network. Furthermore, malicious behaviors can mislead fault localization that solely relies on active measurement, e.g., a node may not report its status or its neighbors' status honestly, or an intermediate node may manipulate the forwarded messages or aggregates. Passive measurement using existing end-to-end data in sensor networks does not suffer from the above drawbacks. Hartl et al. use traditional network tomography techniques to infer node loss rates [3], and Mao et al. use a factor graph decoding method to infer link loss rates [4]. Both techniques, however, heavily rely on a data aggregation procedure (that is used to guarantee correlation among packets). Furthermore, their assumption of a fixed tree limits their applicability. Nguyen et al. propose lossy link inference schemes that use uncorrelated packets and take account of dynamic network topologies [5]. Their schemes, however, may lead to a large number of false positives and negatives in certain circumstances. Broadly speaking, fault inference from end-to-end data falls into network tomography, i.e., inferring internal properties through end-to-end measurement. A rich collection of network tomography techniques has been developed in the past (see [14] for a review). Most of these techniques are developed for wired networks and cannot be applied directly to wireless sensor networks. This is because most of them rely on correlated packets (through multicast or striped unicast packets) and require static topology. In wireless sensor networks, however, end-to-end data are not correlated and topology may change over time.

Our approach differs from existing studies on sensor network fault localization in that it carefully combines active measurement and end-to-end data. The study in [2] also uses end-to-end data together with active measurement: it uses end-to-end data to detect faults, which in turn trigger active measurement and root-cause analysis. It, however, does not utilize end-to-end data to guide selective active measurement (to reduce testing cost) as in our study. Two existing studies [6], [7] combine end-to-end measurement and active testing to identify and correct all faults in a network. Our approach outperforms these two approaches (see Sections I and V).

Last, sequential testing has been studied in the fields of machine troubleshooting, medical diagnosis and computer decision making (e.g., [15], [16], [17]). Our sequential testing problem differs from that in other fields in two important aspects: (i) our sequential tests are on individual components (instead of multiple components) with the guidance of end-to-end data that provide insights into multiple components; and (ii) we detect multiple faults instead of a single fault that is often assumed in other fields.

III. Problem setting

A. Assumptions

Consider a sensor network where sensed data are sent (periodically) from sources to a sink. As in [2], we assume the amount of end-to-end data can be used to detect faults in the network: insufficient amount of data indicates faults, while sufficient amount of data indicates that the network is operating normally. The status of a component (i.e., whether faulty or not) can be tested through active measurement, e.g.,

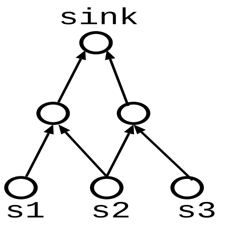


Fig. 1. An example network topology.

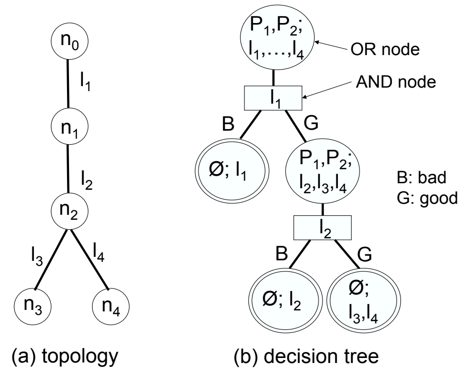


Fig. 2. An example of sequential testing. In (a), n_0 is the sink, n_3 and n_4 are the sources, and end-to-end data indicate that both paths are lossy.

by monitoring the component locally, or looking into the internal states of relevant components (e.g., using [11]). This test incurs a *testing cost*. It is due to the extra energy consumption (for monitoring and transferring the monitoring result locally or to the sink) or personnel time when a personnel is involved.

A fault in a sensor network can be of various forms. In the rest of the paper, for ease of exposition, we only consider faults in the form of lossy links (we briefly discuss lossy nodes in Section III-C). In particular, our goal is to locate *persistently* lossy links that are used in routing (we do not consider transient lossy links since they are not caused by persistent faults that need to be localized and corrected). We say link l is *lossy* or *bad* if its reception rate (defined as one minus the link loss rate) lies below a threshold, t_l. Otherwise, we say l is *not lossy* or *good*. The threshold depends on the application, and is known beforehand. We assume the losses at different links are independent of each other (as shown in [5]). Furthermore, if a link is good (or bad) on one path, then it is good (or bad) on all paths that use the link.

The routing path from a source to the sink can be static or dynamic (e.g., due to a dynamic routing technique [18]). Fig. 1 shows an example topology, where sources s_1 and s_3 use a single path; source s_2 uses two paths dynamically. We consider two settings: (1) we know *complete* path information, i.e., we know the path used by a source at any point of time; and (2) we only know *probabilistic* path information, i.e., we only know the set of paths that are used by a source and the probability to use each path. The first setting applies to static topologies, and dynamic topologies where up-to-date path information is available (e.g., obtained through a path reporting service [19] or from information embedded in data packets [18]). The second setting applies to dynamic topologies where it is too costly to obtain complete path information (e.g., it might consume too much energy).

When knowing complete path information, we define *path reception rate* as the probability that a packet traverses a path successfully. It can be estimated from end-to-end data: when n packets are transmitted along a path and m packets are received successfully, it is estimated as m/n. As mentioned earlier, we assume the amount of end-to-end data indicates whether a fault exists along a path. In particular, we assume, for path P, there exists a threshold, t_P, so that it contains at least one bad link *if and only if* its reception rate is below t_P. This assumption holds when good and bad links differ significantly. It is a reasonable assumption for wireless sensor networks since measurement studies have shown that link loss rates in sensor networks are either large or small, but rarely in between [20], [5][1]. We next illustrate when the above assumption holds. Consider a path of h links. Assume good links have reception rate at least α, while bad links have reception rate no more than β, $0 \le \beta < \alpha \le 1$. Then the above assumption holds when $\alpha^h > \beta$. This is because when the path contains no bad links, the path reception rate is at least α^h; otherwise, it is no more than β. Therefore, when $\alpha^h > \beta$, there exists a threshold (any number in (β, α^h)) so that the path contains at least one bad link if and only if its reception rate is below this threshold[2]. For convenience, we say path P is *lossy* or *bad* if its reception rate is below the threshold, t_P (in other words, it contains at least one lossy link); otherwise, it is *not lossy* or *good*.

When only knowing probabilistic path information, we define *source-sink reception rate* as the probability that a packet is sent from a source to the sink successfully. It can be estimated from end-to-end data: when n packets are sent from a source to the sink, and m packets arrive successfully, it is estimated as m/n. We again assume that there exists a threshold so that at least one link used by a source-sink pair is lossy *if and only if* the source-sink reception rate is below this threshold. Again, we say a source-sink pair (or simply a pair) is *lossy* or *bad* if its reception rate is below the threshold; otherwise, it is *not lossy* or *good*.

The above assumptions imply that all the links on a good path/pair are good, and a bad path/pair contains at least one bad link. Therefore, the potential bad links are the ones that are used by bad paths/pairs, excluding those used by good paths/pairs.

[1]Several other studies reveal links that have intermediate loss rates [18], [21], [22]. Since these types of links can have significant negative impact on the performance of upper-layer protocols [21], [23], we assume the routing protocol used in the sensor network avoids using such links, and hence most links used in the routes have either large or small loss rates.

[2]In practice, we can check whether such a threshold exists for a path by first estimating α and β (e.g., as in [5]), and then checking whether $\alpha^h > \beta$ holds. An extreme case in which the above condition holds is when $\beta = 0$, i.e., a bad link is a failure link.

B. Sequential testing guided by end-to-end data

Using end-to-end data, we have narrowed down the potential lossy links to the set of links that are used by bad paths/pairs, excluding those used by good paths/pairs (since all the links on a good path/pair are good). Pinpointing which potential lossy links are indeed lossy requires testing individual links. We now motivate the need of sequential testing using an example in Fig. 2(a). This example shows two paths, $P_1 = (n_3, n_2, n_1, n_0)$ and $P_2 = (n_4, n_2, n_1, n_0)$. Suppose end-to-end data indicate that both paths are bad. When determining which links are lossy, we face the following ambiguities. First, since links l_1 and l_2 are used by both paths, they cannot be differentiated solely from end-to-end data. Second, since both paths are lossy, the lossy links can be the common links (i.e., l_1 and/or l_2), both leaf links (i.e., l_3 and l_4), or a combination of the above two scenarios. The above ambiguity can be resolved through testing individual links. An advantage of doing sequential testing is that as we reveal the status of one link, this knowledge may provide information on other links, and hence inform later decisions.

We next formulate the problem of sequential testing guided by end-to-end data. For simplicity, the formulation below assumes complete path information (the scenario where we know probabilistic path information only differs in that we use source-sink pairs instead of paths). A sequential testing problem takes the following input: (i) a set of bad paths, $P_1, \ldots, P_N$, that are identified from end-to-end data; (ii) a set of potential bad links, $l_1, \ldots, l_M$, which are the links used by bad paths but not used by good paths; (iii) a set of probabilities, $p_1, \ldots, p_M$, where $p_i \in (0, 1)$ denotes the probability that link l_i is lossy, and a set of costs, $c_1, \ldots, c_M$, where $c_i > 0$ denotes the testing cost for link l_i; and (iv) a routing dependency matrix $R = (r_{ij})_{M \times N}$ where $r_{ij} = 1$ if link l_i is used by path P_j and $r_{ij} = 0$ otherwise.

A solution to the sequential testing problem is to determine the next link to test depending on the previous link that is tested, its corresponding test result (i.e., it is lossy or not lossy), and end-to-end data, so that all bad paths are explained (i.e., each bad path contains at least one link that has been tested and found to be lossy). More conveniently, we can describe a solution to the sequential testing problem using a binary AND/OR decision tree. In the tree, an OR node represents the set of lossy paths to be explained and the potential lossy links to explain them, an AND node represents testing a link, and a leaf node represents an empty set of lossy paths to be explained (i.e., all lossy paths have been explained) and a set of bad links that has been identified to explain the lossy paths. Each AND node has two branches, leading to two OR nodes based on whether the link tested is lossy or not. If the link is lossy, the AND node branches left and the arc is marked with "B"; otherwise, the AND node branches right and the arc is marked with "G".

The *expected total testing cost* of a solution to the sequential testing problem is the sum of the expected testing costs over all links (in the input of the problem), where the expected testing cost of a link is the testing cost of this link times the probability that this link is tested. More conveniently, the expected total testing cost can be calculated from the binary decision tree: it is the sum of the expected testing costs over all AND nodes, where the expected testing cost of an AND node is the testing cost for the link tested at this AND node times the probability that this AND node is reached in the tree. An *optimal* solution to the sequential testing problem is one that leads to the minimum expected total testing cost.

We next use an example to illustrate sequential testing. Fig. 2(b) plots a binary decision tree (not necessarily the optimal one) for the example in Fig. 2(a). The root of the tree contains all lossy paths and potential lossy links. The first link tested is l_1. If l_1 is bad, sequential testing stops (since both lossy paths have been explained) and identifies one lossy link, l_1. Otherwise, the set of potential bad links is reduced to l_2, l_3, l_4, and the next link tested is l_2. If l_2 is bad, sequential testing stops and identifies one lossy link, l_2. Otherwise, sequential testing stops and concludes that links l_3 and l_4 are lossy. The expected total testing cost of this decision tree is $c_1 + (1 - p_1)c_2$ since there are two AND nodes in the decision tree; the expected testing cost of the AND node that tests l_1 is c_1, while the expected testing cost of the AND node that tests l_2 is $(1 - p_1)c_2$ (since l_2 is only tested when l_1 is good, which happens with the probability of $1 - p_1$).

Note that we may not have identified all lossy links when the above sequential testing stops. For instance, in the scenario where both l_1 and l_3 are lossy, it stops after finding link l_1 to be lossy. To identify all lossy links, we run sequential testing in iterations; at the end of an iteration, we repair all lossy links (using tools such as [2] to find out the root causes and remove the root causes) that have been found in the iteration. The iteration continues until all end-to-end behaviors are good. The reason why we do not intend to find all lossy links in one iteration is that end-to-end data in a later iteration may reveal link status at no additional testing cost. For instance, in the above example, if l_1 is lossy and the next iteration shows no lossy links, then we know that the rest of the links are good without any additional test.

Our sequential testing problem differs from existing studies [6], [5], [24] in important ways. The goal of existing studies is to find the most likely set of lossy links that explains the faulty end-to-end behaviors, while our goal is to minimize testing cost. Therefore, we may purposely test a link that is likely to be good as long as it can reduce testing cost. Our approach and the ones in existing studies can run in iterations until all lossy links are located and repaired. As we shall see (Section V), our approach requires a similar number of iterations and a much lower testing cost.

C. Discussion

The above sequential testing formulation considers lossy links. By simply replacing links with nodes, it applies to a scenario where faults are lossy nodes (similarly, by replacing links with nodes, our scheme in Section IV applies to a lossy-node scenario). Furthermore, by taking account of correlation between a node and its adjacent links (e.g., all the links adjacent to a lossy node can be lossy), the formulation can be extended to a scenario where faults include both lossy links and nodes. Further investigation of this scenario is left as future work.

The goal of our sequential testing problem is to minimize testing cost. In practice, minimizing the total number of iterations to locate and repair all faulty components can also be an important goal. Our formulation can be extended to incorporate this goal as follows. We can use a cost to represent the number of iterations required to locate and repair all faulty components, and minimize a weighted sum of this cost and the testing cost. Or we can formulate a constrained optimization problem that minimizes total testing cost under a constraint on the number of iterations, or minimizes the required number of iterations under a constraint on total testing cost. Further exploration of these problems is left as future work.

IV. Sequential Testing Scheme

In this section, we first prove that the sequential testing problem formulated in Section III is NP-hard. We then develop a greedy algorithm to solve it.

A. Complexity of the sequential testing problem

Theorem 1: Optimal sequential testing guided by end-to-end data is NP-hard.

Proof: We prove the problem is NP-hard by showing that a special instance of this problem contains a NP-hard problem (minimum set cover problem). Suppose that the testing costs for all the links are 1, the probability that a link is lossy is $1-\epsilon$, $\epsilon \in (0,1)$, and ϵ is close to 0. Let $\mathcal{P}$ denote the set of bad paths, i.e., $\mathcal{P} = \{P_1, \ldots, P_N\}$. Let $\mathcal{P}_i$ denote the set of bad paths that use link l_i, i.e., $\mathcal{P}_i = \{P_j \mid r_{ij} = 1\}$, $i = 1, \ldots, M$. It is clear that $\mathcal{P}_i \subseteq \mathcal{P}$. Suppose T is an optimal decision tree with the expected cost of C. Without loss of generality, assume that the links tested in the leftmost branch of T are $l_1, \ldots, l_m$. That is, the first link tested is l_1; if it is lossy, the second link tested is l_2, and so on. In the following, we prove that $\mathcal{P}_1, \ldots, \mathcal{P}_m$ is a minimum set cover for $\mathcal{P}$.

First, we have $\bigcup_{i=1}^m \mathcal{P}_i = \mathcal{P}$. This is because the leftmost branch terminates with a leaf node, where all bad paths have been explained. Since the set of bad paths that the leftmost branch explains is $\bigcup_{i=1}^m \mathcal{P}_i$, we have $\bigcup_{i=1}^m \mathcal{P}_i = \mathcal{P}$. We next prove that $\mathcal{P}_1, \ldots, \mathcal{P}_m$ is a minimum set cover for $\mathcal{P}$. Suppose that the links tested in the leftmost branch are replaced by $l'_1, \ldots, l'_{m'}$. Let this new tree be T' with the expected cost of C'. It is clear that $\bigcup_{i=1}^{m'} \mathcal{P}'_i = \mathcal{P}$, where $\mathcal{P}'_i$ is the set of bad paths that use link l'_i. We only need to show that $m' \geq m$. For tree T, we have

$$C \geq 1 + (1-\epsilon) + \cdots + (1-\epsilon)^{m-1} \tag{1}$$

This is because C is no less than the total testing cost in the leftmost branch of T. In the leftmost branch, the expected cost for testing l_1 is 1, the expected cost of testing l_2 is $(1-\epsilon)$ because its testing cost is 1 and the probability that it is tested is $(1-\epsilon)$ (i.e., when l_1 is lossy). Similarly, the expected testing cost of testing l_m is $(1-\epsilon)^{m-1}$. Summing up the expected costs of all links tested in the leftmost branch, we have (1). For tree T', let C'_0 be the expected cost for testing the leftmost branch. Let T'_i denote the subtree that is connected to the right branch of the AND node that inspects l'_i (i.e., when l'_i is good), $i = 1, \ldots, m'$. Let C'_i denote the expected cost of T'_i. Then

$$\begin{aligned} C'_0 &= 1 + (1-\epsilon) + \cdots + (1-\epsilon)^{m'-1} && (2)\\ C'_i &\leq \epsilon(1-\epsilon)^{i-1}(2^{M-i} - 1) && (3) \end{aligned}$$

where (2) is for the same reason as we explained for (1), the inequality in (3) is because T'_i is at most a balanced binary tree of height $M-i-1$, with the maximum testing cost of $(2^{M-i} - 1)$. Since $C' = \sum_{i=0}^{m'} C'_i$, we have

$$\begin{aligned} C' &\leq 1 + (1-\epsilon) + \cdots + (1-\epsilon)^{m'-1} \\ &+ \epsilon(2^{M-1} - 1) + \cdots \\ &+ \epsilon(1-\epsilon)^{m'-1}(2^{M-m'} - 1) && (4) \end{aligned}$$

From the assumption that T is an optimal decision tree, we have $C \leq C'$. From (1) and (4), we can always find an ϵ close to 0 so that $C \leq C'$ can only be satisfied when $m \leq m'$. Therefore, $\mathcal{P}_1, \ldots, \mathcal{P}_m$ is a minimum set cover for $\mathcal{P}$. In summary, we have proved that a special instance of the optimal sequential testing problem embodies a NP-hard problem (minimum set cover problem). Therefore, the optimal sequential testing problem is NP-hard. ■

B. Greedy Algorithm

Since the sequential testing problem is NP-hard, we develop a heuristic scheme to solve it. Observe that, during sequential testing, as we reveal the status of one link, this knowledge may render testing some other links unnecessary. For instance, in the example in Fig. 2(a), after knowing that l_1 is lossy, testing links l_2, l_3 and l_4 becomes unnecessary; after knowing that l_1 and l_2 are good, testing l_3 and l_4 becomes unnecessary. Therefore, revealing the status of a link may lead to cost savings (from the links that do not need to be tested due to this knowledge). This saving subtracted by the testing cost of this link is the *gain* from testing this link. Our scheme is greedy in nature: each time it tests the link that provides the highest gain among all the potential bad links that need to be tested.

Before presenting our scheme, we first describe two types of links whose status can be determined without testing. We refer to them as **non-responsible** and **responsible** links, respectively. A non-responsible link is a link that we determine as good without testing (we may discover that it is bad in a later iteration), i.e., it is used by none of the bad paths to be explained (hence testing it does not provide any further information in explaining bad paths). A responsible link is a link that we determine as bad without test, i.e., there exists at least one bad path that can only be explained by this link being lossy. In the example in Fig. 2(a), after knowing that l_1 is lossy, we identify l_2, l_3 and l_4 as non-responsible links; after knowing that l_1 and l_2 are good, we identify l_3 and l_4 as responsible links.

We now describe our scheme in detail. Let $\mathcal{L}$ denote the set of potential bad links that needs to be tested. Initially $\mathcal{L} = \{l_1, \ldots, l_M\}$. Let $\mathcal{P}$ denote the set of bad paths that needs to be explained. Initially $\mathcal{P} = \{P_1, \ldots, P_N\}$. A decision in sequential testing picks a link from $\mathcal{L}$ to test, then, based on the test result, removes a set of links that does not need to be further tested from $\mathcal{L}$, and removes a set of paths that has been explained from $\mathcal{P}$. This process continues until $\mathcal{P}$ is empty. We define $\mathcal{P}_i$ as the set of bad paths that can be explained by link l_i being lossy. Then $\mathcal{P}_i$ is the set of bad paths that uses l_i, i.e., $\mathcal{P}_i = \{P_j \mid r_{ij} = 1\}$. We next describe how to adjust $\mathcal{L}$ and $\mathcal{P}$ after each test. Suppose link l_k is tested. Then we remove l_k from $\mathcal{L}$ (since it does not need to be tested further). Depending on the test result, we adjust $\mathcal{L}$ and $\mathcal{P}$ as follows.

- If l_k is bad, all the bad paths in $\mathcal{P}_k$ have been explained, and hence can be removed from $\mathcal{P}$. After removing $\mathcal{P}_k$ from $\mathcal{P}$, we may discover non-responsible links. We identify $l_i \in \mathcal{L}$ as a non-responsible link if $\mathcal{P}_i \cap \mathcal{P} = \emptyset$, i.e., none of the bad paths that needs to be explained uses l_i. After identifying non-responsible links, we remove them from $\mathcal{L}$.
- If l_k is good, after removing l_k from $\mathcal{L}$, we may discover responsible links. We identify l_i as a responsible link if there exists a bad path $P_j \in \mathcal{P}$ that can only be explained by l_i being bad (i.e., P_j uses l_i and does not use any other link in $\mathcal{L}$). After identifying responsible links, we remove these links from $\mathcal{L}$, and remove all the bad paths that use these links from $\mathcal{P}$, which may further lead to non-responsible links to be removed from $\mathcal{L}$.

During sequential testing, each time we pick the link that provides the maximum gain for the current $\mathcal{L}$ and $\mathcal{P}$. When given $\mathcal{L}$ and $\mathcal{P}$, the gain from testing a link is obtained as follows. For link $l_k \in \mathcal{L}$, let G_k^b denote the cost saving when knowing l_k is bad; let G_k^g denote the cost saving when knowing l_k is good. To obtain G_k^b, we assume l_k is bad and identify a set of non-responsible links as described earlier. Then G_k^b is the sum of the testing costs of all the non-responsible links thus identified. Similarly, to obtain G_k^g, we assume l_k is good and identify a set of responsible and non-responsible links as described earlier. Then G_k^b is the sum of the testing costs of all the responsible and non-responsible links thus identified. Let G_k be the expected gain from knowing the status of link l_k. Then $G_k = p_k G_k^b + (1 - p_k) G_k^g - c_k$, where p_k is the probability that l_k is bad, and c_k is the cost of testing l_k.

Our scheme is summarized in Algorithm 1, where $\mathcal{L}_b$ denotes the set of lossy links discovered in an iteration, $\mathcal{L}$ is the set of potential bad links that needs to be tested, and $\mathcal{P}$ is the set of bad paths to be explained. Lines 1-3 initialize $\mathcal{L}_b = \emptyset$, $\mathcal{L} = \{l_1, \ldots, l_M\}$, and $\mathcal{P} = \{P_1, \ldots, P_N\}$, respectively. Lines 4-11 identify responsible and non-responsible links, and adjust $\mathcal{L}_b$, $\mathcal{L}$ and $\mathcal{P}$ accordingly before testing any link. Afterwards, the algorithm runs in a loop and stops when all lossy paths have been explained (i.e., $\mathcal{P}$ is empty). In the loop, it calculates

Algorithm 1 Sequential Testing (Greedy Scheme)

```
1:  $\mathcal{L}_b = \emptyset$
2:  $\mathcal{L} = \{l_1, \ldots, l_M\}$
3:  $\mathcal{P} = \{P_1, \ldots, P_N\}$
4:  **for** $\forall l_i \in \mathcal{L}$ s.t. $l_i$ is a responsible link **do**
5:      $\mathcal{L} = \mathcal{L} \setminus \{l_i\}$
6:      $\mathcal{L}_b = \mathcal{L}_b \cup \{l_i\}$
7:      $\mathcal{P} = \mathcal{P} \setminus \mathcal{P}_i$
8:  **end for**
9:  **for** $\forall l_i \in \mathcal{L}$ s.t. $l_i$ is a non-responsible link **do**
10:     $\mathcal{L} = \mathcal{L} \setminus \{l_i\}$
11: **end for**
12: **while** $\mathcal{P} \neq \emptyset$ **do**
13:     **for** $\forall l_i \in \mathcal{L}$ **do**
14:         calculate the gain from knowing the status of $l_i$, $G_i$
15:     **end for**
16:     $k = \arg\max_i G_i$
17:     test link $l_k$
18:     $\mathcal{L} = \mathcal{L} \setminus \{l_k\}$
19:     **if** link $l_k$ is lossy **then**
20:         $\mathcal{L}_b = \mathcal{L}_b \cup \{l_k\}$
21:         $\mathcal{P} = \mathcal{P} \setminus \mathcal{P}_k$
22:         **for** $\forall l_i \in \mathcal{L}$ s.t. $l_i$ is a non-responsible link **do**
23:             $\mathcal{L} = \mathcal{L} \setminus \{l_i\}$
24:         **end for**
25:     **else**
26:         **for** $\forall l_i \in \mathcal{L}$ s.t. $l_i$ is a responsible link **do**
27:             $\mathcal{L} = \mathcal{L} \setminus \{l_i\}$
28:             $\mathcal{L}_b = \mathcal{L}_b \cup \{l_i\}$
29:             $\mathcal{P} = \mathcal{P} \setminus \mathcal{P}_i$
30:         **end for**
31:         **for** $\forall l_i \in \mathcal{L}$ s.t. $l_i$ is a non-responsible link **do**
32:             $\mathcal{L} = \mathcal{L} \setminus \{l_i\}$
33:         **end for**
34:     **end if**
35: **end while**
36: repair all links in $\mathcal{L}_b$
```

the expected gain for each link that needs to be tested, and tests the one with the highest gain. The calculation of the gain for a link is as described earlier and is presented formally in the Appendix. The algorithm then adjusts $\mathcal{L}$, $\mathcal{P}$ and $\mathcal{L}_b$ according to the test result: lines 20-24 are the adjustment when the tested link is bad; lines 26-33 are the adjustment when the tested link is good. At the end, the algorithm repairs all the links that are found lossy (i.e., those in $\mathcal{L}_b$).

We next use an example to illustrate our scheme. The topology is shown in Fig. 3, where node n_0 is the sink and nodes $n_3, \ldots, n_7$ are the sources. Fig. 3 shows eight links, $l_1, \ldots, l_8$. The ground truth is that links l_1, l_6, l_7 and l_8 are bad. The cost for testing a link is 1 unit. The probability that a link is lossy is $p = 0.2$. We first assume that we know complete path information. In this case, end-to-end data indicate that all paths are lossy. We denote the bad paths

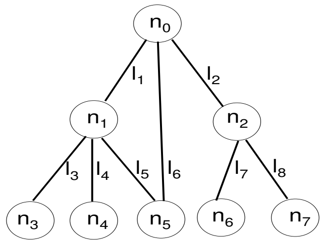


Fig. 3. An example to illustrate the greedy scheme.

as $P_1 = (l_3, l_1)$, $P_2 = (l_4, l_1)$, $P_3 = (l_5, l_1)$, $P_4 = (l_6)$, $P_5 = (l_7, l_2)$, and $P_6 = (l_8, l_2)$. From the routing matrix, for link l_i, we obtain $\mathcal{P}_i$, the set of bad paths that use l_i. Then $\mathcal{P}_1 = \{P_1, P_2, P_3\}$, $\mathcal{P}_2 = \{P_5, P_6\}$, $\mathcal{P}_3 = \{P_1\}$, $\mathcal{P}_4 = \{P_2\}$, $\mathcal{P}_5 = \{P_3\}$, $\mathcal{P}_6 = \{P_4\}$, $\mathcal{P}_7 = \{P_5\}$, and $\mathcal{P}_8 = \{P_6\}$. At the beginning, $\mathcal{L}_b = \emptyset$, $\mathcal{L} = \{l_1, \ldots, l_8\}$, and $\mathcal{P} = \{P_1, \ldots, P_6\}$. Without any testing, we identify l_6 as a responsible link since it is only used by P_4, and P_4 is lossy. Therefore, we remove l_6 from $\mathcal{L}$, remove $\mathcal{P}_6 = \{P_4\}$ from $\mathcal{P}$, and add l_6 to $\mathcal{L}_b$. For the rest of the links, we calculate their gains as $G_1 = 3p + 3(1-p) - 1 = 2$, $G_2 = 2p + 2(1-p) - 1 = 1$, $G_3 = G_4 = G_5 = 3(1-p) - 1 = 1.4$, and $G_7 = G_8 = 2(1-p) - 1 = 0.6$. The expected gain G_1 is obtained as follows. If l_1 is bad, then P_1, P_2 and P_3 are explained and hence we do not need to test links l_3, l_4 and l_5, leading to a saving of 3; if l_1 is good, then l_3, l_4 and l_5 are bad without the need of test, leading to a saving of 3. Therefore, $G_1 = 3p + 3(1-p) - 1 = 2$. The expected gain G_3 is obtained as follows. If l_3 is bad, then it leads to a saving of 0; if l_3 is good, then l_1 must be bad, which makes testing l_4 and l_5 unnecessary (since P_2 and P_3 have been explained by l_1), and hence the total saving is 3. Therefore, $G_3 = 3(1-p) - 1 = 1.4$. The gains for other links are obtained in a similar manner. Since testing l_1 provides the highest gain, we choose to test l_1. The result is that l_1 is lossy. We therefore remove l_1 from $\mathcal{L}$, remove $\mathcal{P}_1 = \{P_1, P_2, P_3\}$ from $\mathcal{P}$, and add l_1 to $\mathcal{L}_b$. We then identify l_3, l_4 and l_5 as non-responsible links and remove them from $\mathcal{L}$. For the rest of the links (i.e., l_2, l_7, l_8), we again calculate their gains, and select to test l_2 since it provides the highest gain. The result is that l_2 is good. We therefore remove l_2 from $\mathcal{L}$, identify l_7 and l_8 as responsible links, and hence add them to $\mathcal{L}_b$ and remove the paths that they use from $\mathcal{P}$. At this time, $\mathcal{P}$ is empty, the iteration ends, and all lossy links have been identified and repaired (end-to-end data in the next iteration will indicate that all paths are good). To summarize, our algorithm uses one iteration and two tests to identify all lossy links.

Let us look at the above example again assuming that we only know probabilistic path information. In this case, we consider five source-sink pairs, all identified as bad from end-to-end data. Denote these bad pairs as $P_1 = (l_3, l_1)$, $P_2 = (l_4, l_1)$, $P_3 = (l_5, l_6, l_1)$, $P_4 = (l_7, l_2)$, and $P_5 = (l_8, l_2)$. The greedy algorithm will test l_1 and then l_2, and identify l_1, l_7, l_8 as bad links. After these links are repaired, in the next iteration, we find that only P_3 is lossy. Since P_3 uses three links, l_1, l_5 and l_6, and l_1 and l_5 are used by good pairs, we identify l_6 as bad. To summarize, our algorithm uses two iterations and two tests to identify all lossy links.

V. Performance evaluation

We evaluate the performance of our greedy algorithm through extensive simulation (using a simulator that we developed) in a sensor network. This network is deployed in a 10 unit $\times$ 10 unit square. A single sink is deployed at the center, and 500 other nodes (sources and/or relays) are uniformly randomly deployed in the square. The transmission range of each node is 3 units. At a given point of time, the paths from the sources to the sink form a reversed tree. In the tree, node n_1 has a directed link to n_2 if n_2 is in the transmission range of n_1, and n_2 forwards data for n_1.

We say a link is bad if its transmission rate lies below 0.8 (i.e., its loss rate is above 0.2); otherwise, it is good. We use two loss models. In the first model, a good link has transmission rate of 0.99 and a bad link has transmission rate of 0.75 (this model is also used in [4], [5]). In the second loss model, a good link has transmission rate uniformly distributed in $[\alpha, 1]$ and a bad link has transmission rate uniformly distributed in $[0, \beta]$, $\alpha > 0.8$ and $\beta < 0.8$. We only describe the results under the second loss model; the results under the first one are similar.

We assume the losses at a link follow a Bernoulli or Gilbert process. Under Bernoulli process, a packet traversing a link is dropped with a probability that is equal to the link loss rate. Under Gilbert process, a link is in a good or bad state. When in a good state, the link does not drop any packet; while in a bad state, the link drops all packets. The transition probabilities between good and bad states are chosen so that the average loss rate is what assigned to the link. We only show the results under Bernoulli loss process; the results under Gilbert loss process are similar (since our algorithm uses average loss rates and hence is not sensitive to the loss process).

We assume all the links have the same testing costs of 1 unit. If a network has been operating for a long time, the probability that a link is lossy can be estimated from historical data. Otherwise, the probability is unknown and one may assume that all links have the same probability of being lossy (as in [6]). In the following, we assume the latter case, and let p denote the probability that a link is lossy. We set p to 0.2, 0.4, 0.6, or 0.8 and find that our scheme is not sensitive to the choice of p. The results below are for $p = 0.2$.

We consider both static and dynamic routing. Under static routing, the paths from the sources to the sink are fixed, and we know the complete path information. Under dynamic routing, the routing tree from the sources to the sink is chosen randomly from multiple trees at every time unit, and we assume a path report service that reports path information periodically to the sink. When this service runs at every time unit, it provides complete path information; when it runs at coarser time scales, it provides probabilistic path information and the probability to use a route is estimated from the frequency that this route is used. For each topology, we vary the percentage of lossy links from 1% to 30%, and randomly choose lossy links. For simplicity, we assume that once a lossy link is repaired, it remains good (although our scheme can handle the case where a repaired link becomes lossy again in a later iteration).

We compare our sequential testing scheme with two existing studies [6], [7] that also localize and repair all faults in a network (Section I describes them briefly). One approach in [7] tests a single link in each iteration, which requires a large number of iterations to localize and repair all lossy links (the number of iterations increases linearly with the number of bad paths, and it can require 30 iterations for only 30 bad paths; while in all the settings we investigate, the average number of iterations under our scheme is below 6, even for over 600 bad paths). To reduce the number of iterations, [7] proposes another approach that tests multiple links in parallel. This approach is similar in spirit to the exhaustive inspection approach [6], and under our setting (i.e., all links have the same probability of being lossy and the same testing cost), the best heuristic using this approach is essentially the same as exhaustive inspection. In the following, we only present the comparison results of our scheme and exhaustive inspection[3].

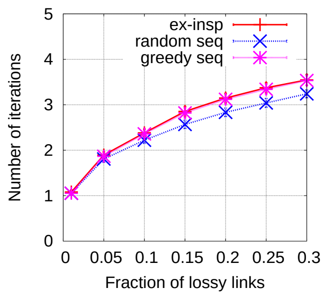
We also compare our greedy scheme with a baseline sequential testing scheme, which differs from the greedy scheme in that it randomly selects a link (from the remaining potential lossy links) to test, instead of choosing the one that provides the maximum gain. We refer to this baseline scheme as *random sequential testing*.

We next report the results under static and dynamic routing respectively. The performance metrics we use are the *number of iterations* required to identify all lossy links and *total testing cost*. Since the testing cost of each link is 1 unit, the total testing cost equals to the number of link that are tested. We normalize the total testing cost by the number of lossy links in the network, and refer to the resulting metric as *normalized testing cost*.

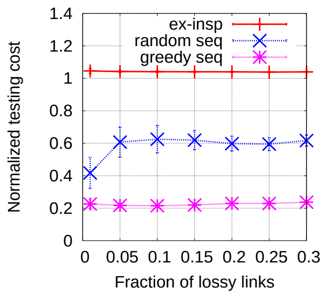
A. Static routing

We build a routing tree as a spanning tree rooted at the sink. The leaves of the tree are the sources. Each intermediate node in the tree has the number of branches uniformly distributed in $[1, B]$, where $B = 10$ or 5. A tree generated using $B = 5$ is "taller" and "thinner" than one generated using $B = 10$. A good link has reception rate uniformly distributed in $[0.95, 1]$; a bad link has reception rate uniformly distributed in $[0, 0.60]$. In this setting, we can find a threshold to determine whether a path is good or bad. This is because for a path of h links, its reception rate is at least 0.95^h if it is good and is at most 0.60 otherwise. Since $0.95^h > 0.60$ ($h \leq 5$ when $B = 10$ and $h \leq 7$ when $B = 5$), we can find a threshold to determine whether this path is good or bad. In particular, we use $(0.95^h + 0.60)/2$ as the threshold. In each iteration, a source sends 400 packets

[3]When implementing exhaustive inspection, we use the inference algorithm in [24], which is applicable to general topologies and is the same as that in [6] for a tree topology (the algorithm in [6] only considers trees).

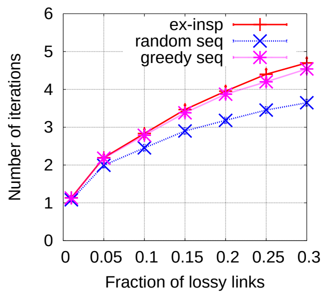


(a) Number of iterations.

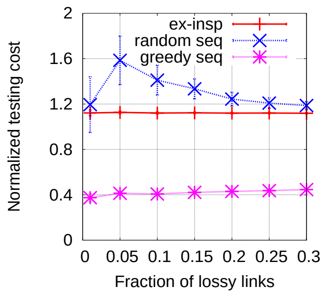


(b) Normalized testing cost.

Fig. 4. Simulation results under static routing, $B = 10$.



(a) Number of iterations.



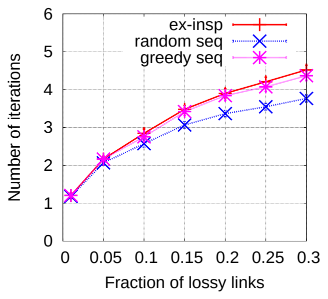
(b) Normalized testing cost.

Fig. 5. Simulation results under static routing, $B = 5$.

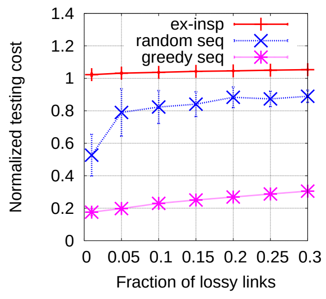
to the sink; the reception rate of a path is estimated from these packets (when using 400 packets, the decision on whether a path is good or bad is correct with probability close to one in our setting). Each confidence interval below is obtained from 150 simulation runs, using five randomly generated routing trees and 30 simulation runs in each tree.

Fig. 4 plots the results when $B = 10$. The results of exhaustive inspection, and greedy and random sequential testing are plotted in the figure. We observe that all three schemes require only a few iterations to localize all lossy links. Under exhaustive inspection, the normalized testing cost is at least 1 (it is higher than 1 since some identified links are false positives). Under the two sequential testing schemes, the normalized testing costs are below 1. Furthermore, the normalized testing cost under greedy sequential testing is much lower than that under random sequential testing. It around 0.2 for all the settings, indicating that only a small fraction of links (0.002 to 0.06 when the fraction of lossy link changes from 0.01 to 0.30) needs to be tested to localize all lossy links.

Fig. 5 plots the results when $B = 5$. As expected, the number of iterations and the normalized testing cost are larger than those when $B = 10$ for all the three schemes, since end-to-end data in a "taller" and "thinner" tree posses a larger amount of ambiguity in determining the status of individual links. In this case, all three schemes still need only a few iterations to localize all lossy links; greedy sequential testing still maintains low normalized testing cost (around 0.4 for all the settings), while both exhaustive inspection and random sequential testing have normalized testing costs above 1.

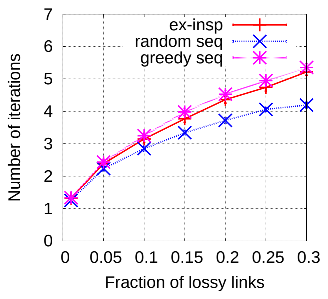


(a) Number of iterations.

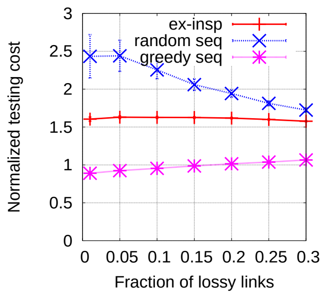


(b) Normalized testing cost.

Fig. 6. Simulation results under dynamic routing when knowing complete path information, $B = 5$.



(a) Number of iterations.



(b) Normalized testing cost.

Fig. 7. Simulation results under dynamic routing when knowing probabilistic path information, $B = 5$.

B. Dynamic routing

Under dynamic routing, we assume the routing tree from the sources to the sink is chosen randomly with equal probability from two trees at every time unit. This is motivated from the measurement results that for each source, a small number of paths carry most traffic in a sensor network under dynamic routing [5]. The two routing trees have the same set of leaves (which are the sources), and differ in the structure of intermediate nodes (we form the second tree by randomly changing the parents of the nodes in the first tree). Again, an intermediate node has the number of branches uniformly distributed in $[1, B]$, $B = 10$ or 5. In all the topologies we generated, above 94% and 91% of the sources use two paths when $B = 10$ and 5, respectively, the rest of the sources use one path. For a source that uses two paths, the path lengths are the same. Again, each confidence interval below is obtained from 150 simulation runs, over five randomly generated topologies and 30 simulation runs in each topology.

When knowing complete path information, the settings for good and bad links are the same as those in the static routing. In each iteration, a source sends 800 packets to the sink (if a source uses two paths, approximately 400 packets are sent on each path). Fig. 6 plots the results when $B = 5$ (the results when $B = 10$ are slightly better, figure omitted). Again, all three schemes require a few iterations to localize all lossy links. The normalized testing costs under all three schemes are lower than those under static routing (see Fig. 5). This is because the path diversity when using two routing trees leads to less ambiguity than that when using a single routing tree. Last, the normalized testing cost under greedy sequential testing is much lower than that under the other two schemes. It is below 0.3 for all the settings.

When knowing probabilistic path information, a good link has reception rate uniformly distributed in $[0.99, 1]$; a bad link has reception rate uniformly distributed in $[0, 0.60]$. In this setting, we can find a threshold to determine whether a source-sink pair is good or bad. This is because, for a source using two paths with equal probability, each path of h links, the source-sink reception rate is at least 0.99^h when the source-sink pair is good, and is at most $(1+0.6)/2$ otherwise. Since $0.99^h > (1+0.6)/2$ ($h \leq 5$ when $B = 10$ and $h \leq 7$ when $B = 5$), we can find a threshold to determine whether the source-sink pair is good or bad. In particular, we choose the threshold to be in the middle of $(1 + 0.6)/2$ and 0.99^h. In each iteration, a source sends 800 packets to the sink; and the path report service sends 40 path reports (it runs every 20 time units) to the sink for each source. Fig. 7 plots the results when $B = 5$ (the results when $B = 10$ are slightly better, figure omitted). Again, all three schemes require a few iterations, and greedy sequential testing incurs much lower testing costs than the other two schemes. Observe that the testing costs under all three schemes are much larger than those when knowing complete path information. This is due to a larger amount of ambiguity when not knowing the exact path information. Under greedy sequential testing, the number of links tested is close to the number of lossy links (the normalized testing cost is close to 1).

VI. Conclusions and Future Work

In this paper, we formulated an optimal sequential testing problem that carefully combines active and passive measurements for fault localization in wireless sensor networks. This problem determines an optimal testing sequence of network components based on end-to-end data to minimize testing cost. We proved that this problem is NP-hard and proposed a greedy algorithm to solve it. Extensive simulation demonstrated that our scheme only requires a few iterations and testing a small subset of links to identify all lossy links in a network.

As future work, we are pursuing in the following directions: (1) evaluating the performance of our approach under other scenarios, e.g., for heterogeneous loss probabilities that change over time, and for different faulty component distributions inside the network (e.g., the location of faulty components follows a more clustered distribution instead of uniform random distribution), (2) developing more scalable approaches to reduce the overhead in obtaining complete path information under dynamic topologies, and (3) developing alternative sequential testing algorithms.

Appendix

Algorithm 2 describes how to calculate gain, G_k, for link $l_k \in \mathcal{L}$ in the greedy algorithm. Lines 3-7 obtain the cost saving when assuming l_k is bad. Lines 8-17 obtain the cost saving when assuming l_k is good. Last, line 19 obtains the gain.

Algorithm 2 **Calculate gain, G_k, for link $l_k \in \mathcal{L}$**

```
1:  $G_k^b = G_k^g = 0$
2:  $\mathcal{L}' = \mathcal{L}, \mathcal{P}' = \mathcal{P}$
3:  $\mathcal{P} = \mathcal{P} \setminus \mathcal{P}_k$, $\mathcal{L} = \mathcal{L} \setminus \{l_k\}$
4:  **for** $\forall l_i \in \mathcal{L}$ s.t. $l_i$ is a non-responsible link **do**
5:      $\mathcal{L} = \mathcal{L} \setminus \{l_i\}$
6:      $G_k^b = G_k^b + c_i$
7:  **end for**
8:  $\mathcal{L} = \mathcal{L}', \mathcal{P} = \mathcal{P}'$, $\mathcal{L} = \mathcal{L} \setminus \{l_k\}$
9:  **for** $\forall l_i \in \mathcal{L}$ s.t. $l_i$ is a responsible link **do**
10:     $\mathcal{L} = \mathcal{L} \setminus \{l_i\}$
11:     $\mathcal{P} = \mathcal{P} \setminus \mathcal{P}_i$
12:     $G_k^g = G_k^g + c_i$
13: **end for**
14: **for** $\forall l_i \in \mathcal{L}$ s.t. $l_i$ is a non-responsible link **do**
15:     $\mathcal{L} = \mathcal{L} \setminus \{l_i\}$
16:     $G_k^g = G_k^g + c_i$
17: **end for**
18: $\mathcal{L} = \mathcal{L}', \mathcal{P} = \mathcal{P}'$
19: $G_k = pG_k^b + (1-p)G_k^g - c_k$
20: return $G_k$
```

Acknowledgment

This work was partially supported by NSF CAREER award 0746841 and Qualtech Systems Incorporated. The authors would like to thank Yoo-Ah Kim, Jim Kurose, Prashant Shenoy, and the anonymous reviewers for their helpful comments and suggestions.

References

[1] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *EWSN*, January 2005.
[2] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," in *SenSys*, November 2005.
[3] G. Hartl and B. Li, "Loss inference in wireless sensor networks based on data aggregation," in *IPSN*, April 2004.
[4] Y. Mao, F. R. Kschischang, B. Li, and S. Pasupathy, "A factor graph approach to link loss monitoring in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, April 2005.
[5] H. X. Nguyen and P. Thiran, "Using end-to-end data to infer lossy links in sensor networks," in *Proc. of IEEE INFOCOM*, 2006.
[6] N. Duffield, "Network tomography of binary network performance characteristics," *IEEE Transactions on Information Theory*, vol. 52, December 2006.
[7] P. P. Lee, V. Misra, and D. Rubenstein, "Toward optimal network fault correction via end-to-end inference," in *Proc. of IEEE INFOCOM*, May 2007.
[8] J. Zhao, R. Govindan, and D. Estrin, "Residual energy scans for monitoring wireless sensor networks," in *WCNC*, March 2002.
[9] J. Zhao, R. Govindan, and D. Estrin, "Computing aggregates for monitoring wireless sensor networks," in *SNPA*, May 2003.
[10] C.-F. Hsin and M. Liu, "A distributed monitoring mechanism for wireless sensor networks," in *Proc. of ACM Workshop on Wireless Security (WiSe)*, 2002.
[11] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, , and D. Culler, "Marionette: Providing an interactive environment for wireless debugging and development," in *IPSN*, 2006.
[12] S. Rost and H. Balakrishnan, "Memento: A health monitoring system for wireless sensor networks," in *IEEE SECON*, 2006.
[13] C. Gruenwald, A. Hustvedt, A. Beach, and R. Han, "SWARMS: A sensornet wide area remote management system," in *TridentCom*, 2007.
[14] A. Adams, T. Bu, R. Caceres, N. Duffield, T. Friedman, J. Horowitz, F. L. Presti, S. Moon, V. Paxson, and D. Towsley, "The use of end-to-end multicast measurements for characterizing internal network behavior," *IEEE Communications Magazine*, May 2000.
[15] M. R. Garey, "Optimal binary identification procedures," *SIAM Journal on Applied Mathematics*, vol. 23, pp. 173–186, September 1972.
[16] K. R. Pattipati and M. G. Alexandridis, "Application of heuristic search and information theory to sequential fault diagnosis," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 4, pp. 872–887, 1990.
[17] K. R. P. V. Raghavan, M. Shakeri, "Optimal and near-optimal test sequencing algorithms with realistic test models," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 29, no. 1, pp. 11–26, 1999.
[18] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *SenSys*, November 2003.
[19] T. Schmid, H. Dubois-Ferriére, and M. Vetterli, "Sensorscope: experiences with a wireless building monitoring," in *Proc. of Workshop on Real-World Wireless Sensor Networks*, 2005.
[20] N. Reijers, G. Halkes, and K. Langendoen, "Link layer measurements in sensor networks," in *MASS*, (Fort Lauderdale, FL), October 2004.
[21] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex behavior at scale: an experimental study of low-power wireless sensor networks," Tech. Rep. UCLA/CSD-TR 02-0013, February 2002.
[22] M. Zuniga and B. Krishnamachari, "An analysis of unreliability and asymmetry in low-power wireless links," *ACM Transactions on Sensor Networks*, vol. 3, June 2007.
[23] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, "Models and solutions for radio irregularity in wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, pp. 221–262, May 2006.
[24] H. X. Nguyen and P. Thiran, "The boolean solution to the congested IP link location problem: Theory and practice," in *Proc. of IEEE INFOCOM*, (Alaska, USA), 2007.