Palo Alto Research Center

Efficient Tracing of Failed Nodes in Sensor Networks

Jessica Staddon
Palo Alto Research Center
staddon@parc.com

Dirk Balfanz
Palo Alto Research Center
balfanz@parc.com

Glenn Durfee
Palo Alto Research Center
gdurfee@parc.com

October 1, 2002

Efficient Tracing of Failed Nodes in Sensor Networks

Jessica Staddon
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California
staddon@parc.com

Dirk Balfanz
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California
balfanz@parc.com

Glenn Durfee
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California
gdurfee@parc.com

ABSTRACT

In sensor networks, nodes commonly rely on each other to route messages to a base station. Although this practice conserves power it can obscure the cause of a measurement outage in a portion of the network. For example, when a base station ceases to receive measurements from a region of nodes it can't immediately determine whether this is because of the destruction of all the nodes in that region (due to an enemy attack, for example) or merely the result of the failure of a few nodes bearing much of the routing load. Previous solutions to this problem typically consist of re-running the route-discovery protocol, a process that can be quite expensive in terms of the number of messages that must be exchanged. We demonstrate that the topology of the network can be efficiently conveyed to the base station allowing for the quick tracing of the identities of the failed nodes with moderate communication overhead. Our algorithms work in conjunction with the existing functions of the network, requiring the nodes to send no additional messages.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*network topology, network communications, wireless communication*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*routing protocols*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Algorithms, Security, Theory

Keywords

Tracing, Routing, Security

WSNA'02, September 28, 2002, Atlanta, Georgia, USA.

1. INTRODUCTION

Nodes in sensor networks can fail for many different reasons: their batteries may be depleted, they may be accidentally destroyed, a malicious adversary may deliberate incapacitate them, *etc.* Because sensor nodes typically route measurements to the base station in a tree-like fashion (with the base station being at the root of the tree), the failure of a single node can result in a whole portion of the network becoming silent (*i.e.,* the base station ceases to receive measurements from the nodes).

If a region of the network goes silent, then the network may be unable to complete its intended function. For example, a perimeter defense system may no longer be able to detect breaches of the perimeter in the silent region. In addition, if the silence is due to the destruction of a large set of nodes, a security breach may have occurred. It is therefore critical for the base station to find out whether the recoverable failure of a small set of nodes is responsible for the outage of the larger region, or whether the silent nodes have all been destroyed, accidentally or otherwise (see Figure 1). In the former case, the network needs to adopt a new routing topology, routing around the small set of dead nodes. In the latter case, the base station should sound an alarm, since the proper functioning of the sensor network can no longer be guaranteed.

To distinguish between these cases, the base station needs to *trace* all dead nodes, *i.e.,* for each node in the network, figure out whether it is dead or alive.

Previous works on routing in sensor networks (see, for example, [15]) advocate re-running the route-discovery protocol for this purpose. This typically requires the nodes to send additional messages and consequently can be very expensive. It has also been suggested [13] that the nodes may police themselves. That is, if a node can listen-in on the neighbor it is currently routing to, it can determine that the node has failed and choose a new neighbor to route to. This process may be slow however, and is error-prone because the constrained sensor nodes cannot be expected to constantly police all their neighbors and consequently may end up routing to a new neighbor that has also failed. We demonstrate that by shifting the work load to the base station, the failed nodes can be identified quickly and efficiently using the base station's comprehensive view of the network and without a single additional message transmission by the sensor nodes.

The first step of our protocols enables the base station to learn the topology of the network. During the execution of many well known route-discovery protocols (see for example, [15, 9]) nodes learn the identities of their neighbors. To

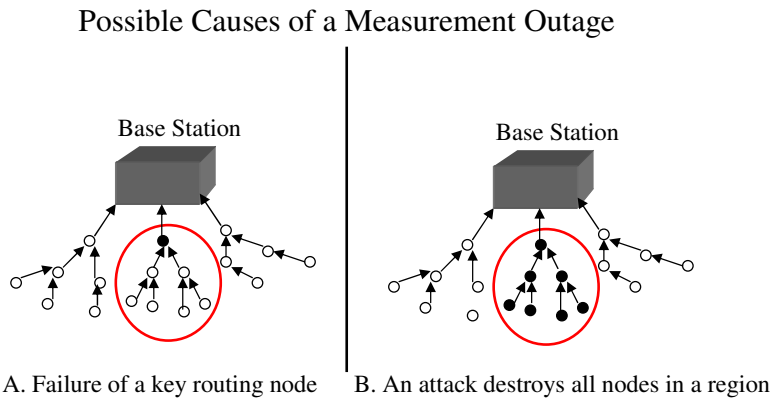

Figure 1: The two entirely different events shown in this figure both cause the region of nodes to cease sending measurements to the base station. The network, however, can presumably recover from the failure of a key routing node, while an attack on an entire region of nodes should be cause for alarm.

convey this information to the base station, each node simply appends a little bit of information about its neighbors to each of its measurements; in a constant (*i.e.,* the degree of each node) amount of time the base station has adjacency information for the entire network and hence can construct its topology. Once the base station knows the node topology the failed nodes can be efficiently traced using a simple divide-and-conquer strategy based on adaptive route update messages. We present algorithms that run in time $O(\log s)$ (where s is the number of silent nodes) and require only $O(d \log s)$ (d is number of nodes that turn out to be dead) messages in total.

Overview. In Section 2 we give an overview of related work in this area. In Section 3 we explain the assumptions under which we are operating, before describing our algorithms in Section 4. We provide tracing examples in Section 5, and conclude in Section 6 by highlighting some open problems.

2. RELATED WORK

Our work is related to the problem of routing in sensor networks [11, 6, 12, 15] because we use route updates for tracing purposes. Although our primary motivation is to trace quickly in order to determine the extent of the network outage, the need to moderate power consumption does influence the route updates our algorithms choose. In addition, in some scenarios the route updates chosen by our tracing algorithms are likely to be the most efficient ones given the changing state of the network and hence, our algorithms can also help to efficiently re-route the network when recovery is possible. Our work differs from [11, 6, 12, 15] because we force the base station to do most of the routing work after the initial route-discovery protocol is executed.

As mentioned earlier, in [13] nodes police each other in order to detect misbehavior. Our approach is different because our goal is to minimize the number of messages nodes must send in order to maximize the lifetime of the network.

The idea of re-routing nodes in order to trace them is quite analogous to the problem of group testing [4]. In group testing algorithms a set of items are tested together in such a way that one failure in the set will cause the whole set to fail. The number of tests may be quite low if most of the items have not failed. We adopt this philosophy in our algorithms – if only a few nodes in the network have failed this can in general be determined much more efficiently using our algorithms than by re-running the route-discovery protocol. The security level of the network affects the parameters of the algorithms. In a higher security network, nodes may be traced more aggressively. Aggressive tracing corresponds to more route updates (or tests).

For similar reasons, our work is also somewhat related to the problem of dynamic traitor tracing (DTT) [5, 1]. However, tracing sensor nodes differs from both group testing and DTT because there are more possible outcomes from the "tests". That is, from re-routing a set of nodes the base station may learn that all the nodes are alive, at least one is dead *or* that there is a dead node in a proper subset of the original set. In addition, the base station successfully traces at least one node with each route update. We leverage these differences to achieve faster tracing times than might be expected from the simple divide-and-conquer techniques that form the basis of our algorithms.

3. MODEL

We are interested in sensor networks containing a powerful base station and n constrained sensor nodes [7]. We assume the base station is able to directly transmit messages to any node in the network, but the nodes seek to conserve power by relying on each other to route their measurements to the base station [12, 15]. We emphasize however, that as in [6] we assume the nodes are capable of sending messages across long distance, however due to the large amount of power consumed by such transmissions doing so is undesirable. Consequently, every pair of nodes are potential neighbors (*i.e.,* they can directly communicate) however, nodes prefer to route messages to their near neighbors (*i.e.,* neighbors that are within a low-power transmission range).

Our techniques are most naturally applicable to networks in which sensors send measurements in regular intervals, or *epochs* (such networks are often called *continuous*).[1] An example of such a network might be a perimeter defense

[1]Our techniques can be applied to event-driven networks, however tracing will take longer.

system in which, at the beginning of each epoch, each sensor transmits a message to the base station indicating whether or not the sensor believes the perimeter is secure.

We say a node is *dead* if it has expired. A node may cease sending measurements because it has died or because it is routing measurements to a node that has died. We provide algorithms to determine the status (*i.e.,* alive or dead) of the nodes. A node is *silent* if it is not sending measurements and its status cannot be determined given the current routing topology. We say a node has been *traced* when its status is determined. We denote the number of dead nodes in the network by d, the number of silent nodes by s, and use R, D and S to denote the sets of responsive (*i.e.,* alive), dead and silent nodes, respectively.

To make use of our tracing algorithms, two types of nodes in S must be identified: 1. Nodes that are closest to the base station (we call these *best base station neighbors*) and 2. Nodes that are closest to R (we call these *best R neighbors*). Since the exact distances between nodes in the network may be hard to determine, we instead use number of hops (where the hops are between near neighbors) as a measure of distance. For example, a node that is a near neighbor of a node in R that is a near neighbor of a near neighbor of the base station, is 1 hop from R and 3 hops from the base station. We refer to nodes in S that are neither best base station neighbors nor best R neighbors as being *internal* to S.

Using knowledge of the network topology (*i.e.,* node adjacency information) our algorithms adjust the routing topology, or system of routes, to trace nodes. Route update broadcasts (*i.e.,* new route announcements that are authenticated using a lightweight authentication scheme such as μTESLA [15]) are made by the base station. To facilitate this, we associate an identification number (ID) with each node and we often refer to a node by its ID. For example, $i_1 \rightarrow i_2 \rightarrow i_3$ is a route update indicating node i_1 should send any measurements it generates or receives to node i_2, and node i_2 should send any measurements it generates or receives to node i_3. The purpose of the route updates is to connect nodes in S to the base station, either directly or via one or more nodes in R.

In the figures of this paper, known dead nodes are shaded black, responsive nodes are white and nodes that are silent and whose status has not yet been determined, are shaded gray.

ATTACK MODEL. We use an adversary model in which network operations may be disrupted by killing nodes or injecting rogue nodes into the network. We assume that the nodes are sufficiently tamper-resistant to prevent an adversary from "re-programming" a node in the original network. However, the rogue nodes can behave in any way the adversary desires. We protect against such rogue nodes by storing in each node a unique key shared with the base station. All messages from a node to the base station are authenticated using a message authentication code (MAC) (see for example, [10, 14]) based on this key.

4. TRACING IN NETWORKS WITH BASE STATIONS

Our tracing algorithms require that the base station know the near neighbors of each node. In Section 4.1 we describe how this can be achieved and in Section 4.2 we show how the base station may make use of the information.

4.1 Learning the Network Topology

The following algorithm enables the base station to learn the topology of the network with a communication overhead of $O(\log n)$ on each of the regular measurement transmissions from a node to the base station. Although the number of nodes in the network is likely to make the overhead associated with sending every route specification to the base station prohibitive, if each node merely sends a list of its near neighbors (*i.e.,* adjacency information) to the base station (rather than complete routes) the base station can do the extra work of constructing routes to facilitate fast tracing.

This adjacency information[2] is typically learned by each node during the route-discovery process [15, 3]. With some route-discovery algorithms (*e.g.,* [2]) the base station also learns the topology of the network. For use when the base station does not receive adjacency information as a byproduct of the route-discovery process, we propose the following simple algorithm. With this algorithm the network topology is learned in conjunction with the network's normal functions; that is, the sensor nodes need not send any additional messages.

ALGORITHM 1. *Network topology establishment.*
Input to the algorithm is a sensor network in which each node shares a unique key with the base station and knows its near neighbors. The result of the algorithm is that the base station learns the network's topology.

1. *In each epoch, each node attaches the ID of one of its near neighbors to its measurement for the epoch and adds the necessary authentication information. For example, for a node i that is a near neighbor of node j and is sending along a route in which the first hop is to node r, the packet may take the following general form: $ID_r||m||MAC_{k_i}(m)$ where $m = ID_i||measurement||ID_j$. (Additional information (e.g. a timestamp) may be added to m for security reasons.)*

2. *Step 1 is repeated until the base station broadcasts a message indicating it is no longer receiving new adjacency information from any of the nodes (that is, the base station has received all the topology information).*

This algorithm adds a manageable amount of overhead. Since each node would typically be sending their ID (so that the base station knows by whom the measurement was taken) and appending a MAC anyway, the additional overhead can be merely the 1 byte of ID that indicates a near neighbor (assuming that a node ID is 1 byte long, as is usually the case).

4.2 Tracing Algorithms

Our tracing algorithms make changes to the routing topology to determine the status of silent nodes. The route changes are chosen, and then announced, by the base station in such a way that the nodes nearest the base station or a responsive node are traced more quickly than internal nodes. In addition, the cost to the nodes of receiving long route update messages and making transmissions across long distances factors into the route choices.

Our algorithms follow a few simple principles:

[2]This adjacency information should be learned in an authenticated manner, but this issue is beyond the scope of this paper. See Section 6 for more on this issue.

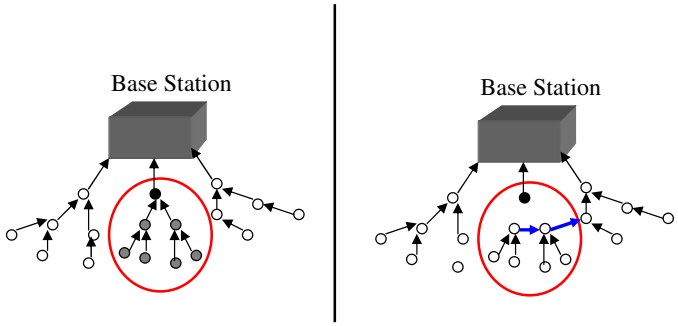

Figure 2: Although only one of the sensors is dead (indicated in black) 7 nodes have ceased sending measurements because of the routing topology. A single route update allows the base station to trace the dead node and begin receiving measurements again.

Tracing with Route Updates. In our algorithms the base station routes around known dead nodes with the goal of determining the status of the remaining silent nodes. Consider for example a network with one dead node and 6 silent (but in face, alive) nodes as depicted in Figure 2. After broadcasting a route update the base station is able to determine that none of the silent nodes are in fact dead by waiting for them to send measurements in the subsequent epoch. Note that this approach to tracing does not require any additional message transmissions by the sensors.

Tracing from the Outside In. A node that is internal to the set of silent nodes may be difficult (or expensive in terms of power) to trace if its near neighbors are dead. Hence our tracing algorithms tend to trace nodes that are near R or the base station, first. As a result, a node that appeared difficult to trace initially, may turn out to have a near neighbor in R.

Subdividing for Faster Tracing. Multiple route update broadcasts may be made in each epoch in order to trace dead nodes more quickly. The silent nodes are subdivided into disjoint sets and a new route (that terminates in a node in R or the base station) is broadcast for each set. Hence, the first problem when choosing new routes is deciding how to subdivide S. We consider two approaches. The first approach, which we term *subdividing by best base station neighbors* partitions the nodes into sets such that each set contains a node that is the fewest hops from the base station as any node in S, along with as many of the node's children (as defined by the *routing* topology) as possible so as to arrive at roughly equally size sets. If this is insufficient to create the number of sets required by the algorithm we proceed similarly with the nodes that are the next closest to the base station, *etc.* Finally, if any nodes in S are not added to one of the sets by this process, we add them to the existing set to which they are closest (where the closeness of a node to a set is determined by measuring the minimum number of hops from the node to a member of the set).

When subdividing by best base station neighbors, the route through a set may be able to use much of the initial routing topology and merely modify the part of the route that contains dead node(s). In this way, route updates may be kept quite short, however the maximum hop length in a route may be quite large if a node that is the minimum number of hops from the base station doesn't have a near neighbor in R and is far from the base station (despite being as close to the base station as any node in S).

In order to reduce the average transmission distance between neighbors, we also consider *subdividing by best R neighbors*. The goal of subdividing in this way is to ensure that nodes whose near neighbors are all in S won't be forced to make a possibly expensive transmission to a node in R. To this end, we subdivide the nodes into approximately equally sized sets (using the *network* topology to construct the sets) such that each set contains a node with a near neighbor in R. If this doesn't yield as many sets as the algorithm requires, we weaken the requirement and permit sets with a member that is near a near neighbor of R, *etc.*, while still aiming for equally sized sets. Subdividing in this way helps to ensure that all transmission distances within each route are small, but it may require long route update broadcasts as it may be difficult to preserve much of the initial routing topology.

Figure 3 illustrates how the two approaches to subdividing might work in practice. We emphasize that these subdivision techniques are *heuristics* rather than precise algorithms. The particular network topology should be taken into account to alternate between the approaches or perhaps, to rule out one of them entirely (*e.g.,* subdividing by best R neighbor may not make much sense if the network forms a tree of depth one).

The first algorithm uses simple divide-and-conquer techniques to trace the dead nodes.

Algorithm 2. *Subdivision-Based Tracing.*

Input to the algorithm consists of the network node topology, an integer $a > 1$ representing the number of equally sized pieces a group of nodes will be subdivided into in each iteration, the initial set S of silent nodes, the initial set D of dead nodes and the initial set R of responsive nodes (of course, R can be determined from S and D). The output of the algorithm consists of a result set R of responsive nodes and a result set D of dead nodes, such that the union of sets R and D is all the nodes in the network.

Let $t = 1$ and $S = \cup_{i=1}^{t} S_i = S_1$ denote the current set of silent nodes that cannot be determined to be in R or D given the initial route topology.[3]

[3]Recall that S does not include nodes immediately known to be dead, such as nodes 19 and 20 in Figure 4. Those nodes make up the initial set D.

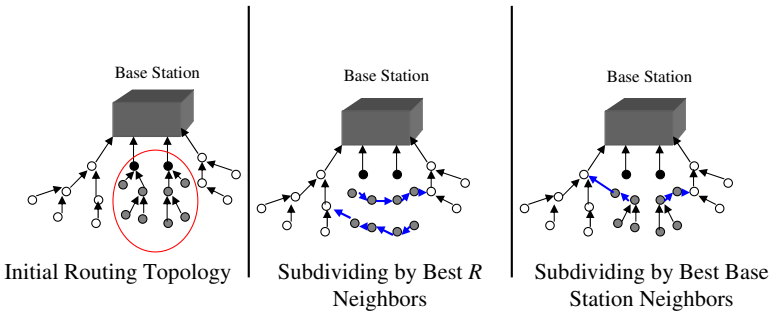

Figure 3: When 10 nodes become silent the 2 nodes adjacent to the base station (marked in black) are known to be dead. Subdividing the remaining nodes once by best R neighbors requires 2 route updates, each of length 5. Subdividing by best base station neighbors requires two updates (one with a large hop to R) each of length 3.

1. *For $j = 1, \ldots, t$, if $|S_j| \geq a$ then subdivide S_j into a equally sized subsets (roughly) using best base station neighbors or best R neighbors subdividing. If $1 \leq |S_j| < a$ then S_j is not subdivided. The collection of subdivided sets along with those that were not subdivided form the current collection $\{S_j\}_j$. Set t to be the number of nonempty sets in the collection.*

2. *For $j = 1, \ldots, t$, the base station chooses a best base station neighbor or best R neighbor (according to the choice of subdividing method) in S_j uniformly at random and broadcasts a route going through the other nodes in S_j to this node and to a node in R or the base station (whichever is closer). If a node in R is chosen then the route from the node in R to the base station remains the same.*

3. *In the following epoch, the base station learns the status of some of the nodes in S. For $j = 1, \ldots, t$, all the nodes in S_j that are responsive (i.e. those from which the base station is receiving measurements) are added to R. All nodes in S_j that are routing directly to a node in R or the base station but aren't responsive, are added to D. For each $S_j = \emptyset$ decrement t and adjust the indices of the remaining sets, $\{S_j\}_j$, appropriately. Provided $t > 0$, go to step 1, else quit as all the nodes have been traced.*

See Section 5 for an example run of this algorithm.

LEMMA 1. *In Algorithm 2, d dead nodes can be identified amongst s silent nodes in time $O(\log(\frac{s}{a}) + a)$ epochs with $O(ad)$ route updates per iteration.*

Proof: In each iteration of Algorithm 2, $S_j \neq \emptyset$ if in the previous epoch, S_j was of size at least 2 and contained a dead node. Hence, the number of subdivisions (and hence, route updates) in a given epoch is no more than ad. It remains to determine the number of epochs that the algorithm runs in the worst case. The algorithm is from the divide-and-conquer paradigm and can be represented by an a-ary tree. The only additional wrinkle is that we're guaranteed to learn the status of at least one node from every route update, so the size of the sets, S_j, shrinks by at least one in every epoch. Consequently, in epoch i, the sets are of size at most, $\frac{s-(\frac{a^i-1}{a-1})+1}{a^i}$. Setting this quantity to be at most $a-1$ and solving for i, we get $i \geq \frac{\log[\frac{(s+1)(a-1)}{((a-1)^2+1)}]}{\log a}$, and since at most $a-1$ additional epochs are required to trace a set of size $a-1$, we have the claim of the theorem. □

The primary goal of the previous algorithm is fast tracing (as a function of a). Even when employing Algorithm 2 with best R neighbors subdivision it may be possible to achieve shorter transmission distances by choosing to subdivide only the sets that are sufficiently close to R. The following algorithm describes how this may be done using best R neighbors subdivision (although subdivision by best BS neighbors can easily be substituted). We describe the algorithm probabilistically, although deterministic versions are also possible.

The base station assigns each node a probability according to its proximity to R (*i.e.,* the minimum number of hops from the node to a node in R) and its proximity to the base station: if the node is close to R *or* the base station it has a probability close to 1. Since R changes as the algorithm is executed, it may make sense to adjust a node's probability. For example, the base station may simply use the smaller of a node's hop distance to the current set R and the node's hop distance to the base station, to determine its probability (in which case the notion of a probability may be unnecessary). Or, an approach that is less computationally expensive for the base station but may have the same effect, is to calculate the probabilities based on hop distance to the *initial* set R and renormalize periodically (*i.e.,* to ensure that there are always nodes in S with probability 1).

The purpose of using probabilities is to increase the likelihood that the status of the nodes closest to R will be determined most quickly, potentially creating routes through which other nodes may be traced with less power consumption. To this end, the probabilities associated with the nodes in a set determine the probability that the set is subdivided.[4] Of course, any delay in subdividing means tracing takes longer. However, tracing the nodes closer to R may be enough to determine a large-scale network outage, and if so, many unnecessary route updates that would have depleted the healthy part of the network can be avoided. On

[4]Since there are many possible sets, only some of which will appear during an execution of the algorithm, we choose to assign probabilities to the nodes rather than the sets.

the other hand, if only a few nodes are dead, the algorithm will eventually output a new set of routes with short hop lengths, and hence, it may be unnecessary to re-run the route-discovery protocol.

ALGORITHM 3. *Adaptive Subdivision-Based Tracing.*

Input to the algorithm consists of the network node topology, an integer $a > 0$ representing the number of equally sized sets a group of nodes may be subdivided into, the initial set S of silent nodes, the initial set D of dead nodes, the initial set R of responsive nodes and for each node, i, a probability p_i, $0 \leq p_i \leq 1$, which is used to determine whether or not a set of nodes is subdivided. The output of the algorithm consists of a set R of responsive nodes and a set D of dead nodes. The union of sets R and D is all the nodes in the network.

Let $S = \cup_{i=1}^{t} S_i$ denote the current set of silent nodes that cannot immediately be determined to be in R or D. Set t=1, $S = S_1$ initially.

1. *For $j = 1, \ldots, t$, if $|S_j| \geq a$ subdivide S_j into equally sized subsets (roughly) using subdivision by best R neighbors with probability $\max_{r \in S_j} p_r$. That is, with probability 1-$\max_{r \in S_j} p_r$, S_j is not subdivided. The collection of subdivided sets along with those that were not subdivided form the current collection $\{S_j\}_j$. Set t to be the number of sets in the collection.*

2. *For $j = 1, \ldots, t$, the base station chooses a best R neighbor in S_j uniformly at random and broadcasts a route going through the other nodes in S_j to this node and then to a node in R or the base station (whichever is closer). If a node in R is chosen then the route from the node in R to the base station remains the same.*

3. *In the following epoch, the base station learns the status of some of the nodes in S. For $j = 1, \ldots, t$, all the nodes in S_j that are responsive (i.e. those from which the base station is receiving measurements) are added to R. All nodes in S_j that are routing directly to a node in R but aren't responsive are added to D. For each $S_j = \emptyset$ decrement t and adjust the indices of the remaining sets, $\{S_j\}_j$, appropriately. Based on the routing topology and S, R and D, the base station may want to adjust the probabilities $\{p_i\}_i$. Provided $t > 0$, go to step 1, else quit as all the nodes have been traced.*

Algorithm Analysis: Because this algorithm is adaptive and may be used quite differently from Algorithm 2, an analysis along the lines of the proof of Lemma 1 is not necessarily an accurate performance measure. However, to illustrate how such an analysis might be made we provide it for the non-adaptive, non-biased case, that is, the probabilities are equal and fixed for all nodes. Then we briefly discuss the performance of the algorithm in more likely scenarios.

In the non-adaptive, non-biased setting, there exists a probability p, such that for all i, $p_i = p$ and is fixed for the duration of the algorithm. We know from Lemma 1 that if each set is always subdivided the algorithm will take no more than r epochs where r is $O(\log(s/a)+a)$. We now need to account for the fact that in each epoch a set is *not* subdivided with probability $1-p$. When a set is not subdivided its size will still decrease somewhat in each epoch because the algorithm always removes from the set any silent nodes that are directly transmitting to a node in R and re-routes the remaining nodes in the set. However, to get a rough upper bound on the running time of the algorithm it suffices to ignore this subtlety and assume that in each epoch the size of the set stays the same with probability $1-p$. Hence, the probability that the algorithm takes $r+i$ epochs to complete is at most $\binom{r+i-1}{r}(1-p)^i$. An upper bound on the expected run-time is:

$$\begin{aligned}
\sum_{i=0}^{\infty} \binom{r+i-1}{i}(1-p)^i(r+i) &= r\sum_{i=0}^{\infty}\binom{r+i}{i}(1-p)^i \\
&< r\sum_{i=0}^{\infty}(1+i)^r(1-p)^i \\
&< r + r\int_1^{\infty}(1+x)^r(1-p)^x dx
\end{aligned}$$

Using standard calculus techniques the final term above can be strictly upper bounded by:

$$r + \frac{r}{|\log(1-p)|}\left(\frac{\frac{r^{r+1}}{|\log(1-p)|^{r+1}} - 1}{\frac{r}{|\log(1-p)|} - 1}\right).$$

Note that as p converges to 1 this algorithm becomes identical to Algorithm 2 with best base station neighbor subdivision, and consequently the upper bound on the expected running time converges to r. Of course, as p becomes smaller, the probability of subdivision decreases and the expected running time grows. To complete the analysis in this setting, note that the expected number of route updates per iteration is at most: $(ap+1-p)d = [(a-1)p+1]d \leq ad$.

To see how this algorithm can be used to improve upon Algorithm 2 consider a network in which the s silent nodes roughly form a binary tree. In such a network, there may be a threshold number of dead nodes which is grounds for sounding an alarm. This algorithm may discover that the threshold number of nodes are dead without extraneous route updates by setting $p_i = 0$ for the nodes that are far away from R and the base station. Further, if all the nodes are to be traced, and only a few nodes are in fact dead, this algorithm can enable this to be done with more power-efficient routing and in only moderately longer time. In Section 5, an example along these lines is presented. □

5. TRACING EXAMPLES

To illustrate how the tracing algorithms may be used in practice we now discuss applying them to an example network in which 20 of the nodes are silent. The example network is partially depicted in Figure 4; only the silent nodes and their nearest responsive neighbors are shown (the complete topology of the network is not needed to demonstrate the algorithms). We consider two situations: 1. Widespread node failure: all of the silent nodes have failed, 2. Sporadic node failure: a few of the important routing nodes have expired. The examples illustrate that if the security requirements are such that situation 1 must be ruled out as quickly as possible then Algorithm 2 is the best choice. However, with a little more time Algorithm 3 may be used with potentially less power consumption.

In the following we consider how Algorithm 2 with best base station neighbors subdividing and Algorithm 3 may perform (both with $a = 2$) first when the network has suf-

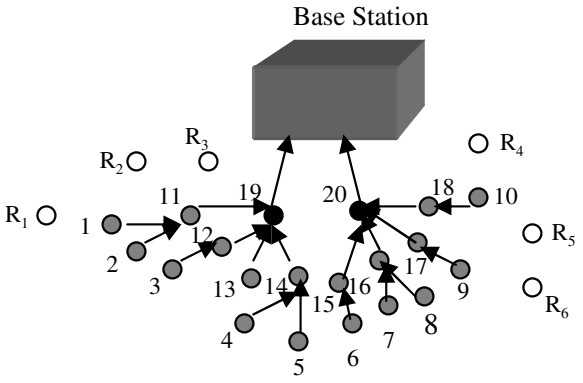

Figure 4: This small network contains 2 nodes that are known to be dead and 18 silent nodes. If nodes 1-18 are dead, Algorithm 3 performs no better than Algorithm 2. However, if only a few of the nodes are dead Algorithm 3 can trace the entire network in only moderately more time than Algorithm 2 and with less power consumption.

Node Failure Type	Best Tracing Time	Best Ave. Transmission Distance	Smallest No. of Route Updates
Widespread	Algorithm 2	No Difference	No Difference
Sporadic	Algorithm 2	Algorithm 3	Algorithm 3

Figure 5: Algorithm 2 always subdivides and re-routes so it traces most quickly. Algorithm 3 can allow for route updates with shorter transmission distances and fewer route update broadcasts when node failure is not widespread.

fered widespread failure and then when the failure is sporadic.

WIDESPREAD NODE FAILURE. Initially, $D = \{19, 20\}$ and $S = \{1, \ldots, 18\}$. In the first step of Algorithm 2, nodes in S are subdivided into the sets $S_1 = \{1, 2, 3, 4, 5, 11, 12, 13, 14\}$ and $S_2 = \{6, 7, 8, 9, 10, 15, 16, 17, 18\}$. The route updates[5] $14 \rightarrow 13 \rightarrow 12 \rightarrow 11 \rightarrow R_2$ and $15 \rightarrow 16 \rightarrow 17 \rightarrow 18 \rightarrow R_4$ are broadcast. In the next epoch, the base station learns that 11 and 18 are dead and adds them to D. Subdividing again yields the following sets: $S_1 = \{1, 2, 3, 12\}$, $S_2 = \{4, 5, 13, 14\}$, $S_3 = \{6, 7, 8, 15, 16\}$ and $S_4 = \{9, 10, 17\}$, and route updates: $1 \rightarrow 2 \rightarrow 3$, $12 \rightarrow R_3$, $14 \rightarrow 13 \rightarrow R_3$, $15 \rightarrow 16 \rightarrow \textit{Base Station}$, and $17 \rightarrow 10 \rightarrow R_4$. This allows the base station to determine in the next epoch that $12, 13, 16$ and 10 are dead. Subdividing again, yields: $S_1 = \{1, 2\}$, $S_2 = \{3\}$, $S_3 = \{4\}$, $S_4 = \{5, 14\}$, $S_5 = \{6, 15\}$, $S_6 = \{7, 8\}$, $S_7 = \{9\}$ and $S_8 = \{17\}$, with the following route updates: $2 \rightarrow 1 \rightarrow R_1$, $3 \rightarrow R_2$, $4 \rightarrow \textit{Base Station}$, $14 \rightarrow \textit{Base Station}$, $15 \rightarrow \textit{Base Station}$, $7 \rightarrow 8 \rightarrow R_6$, $9 \rightarrow R_6$ and $17 \rightarrow \textit{Base Station}$. In the next epoch, the base station is able to add $1, 3, 4, 8, 9, 14, 15$ and 17 to R leaving only $\{2, 5, 6, 7\} = S$. Subdividing into $S_1 = \{2, 5\}$ and $S_2 = \{6, 7\}$, the base station determines that 2 and 7 are dead with route updates $5 \rightarrow 2 \rightarrow R_1$ and $6 \rightarrow 7 \rightarrow R_6$. Subdividing one last time allows the base station to trace the remaining nodes in the next epoch.

Because all the nodes are dead, Algorithm 3 requires just as many power-consuming route updates but potentially takes a lot longer because in each epoch, each S_j is not subdivided with non-zero probability.

SPORADIC NODE FAILURE. We now consider the same network when only nodes $11, 12, 13, 14, 18, 19$ and 20 are dead. Once 19 and 20 are added to D, both algorithms subdivide into $S_1 = \{1, 2, 3, 4, 5, 11, 12, 13, 14\}$ and $S_2 = \{6, 7, 8, 9, 10, 15, 16, 17, 18\}$, and the base station announces route updates $14 \rightarrow 13 \rightarrow 12 \rightarrow 11 \rightarrow R_2$, $15 \rightarrow 16 \rightarrow 17 \rightarrow 18 \rightarrow R_4$. In the following epoch, the base station learns that 18 and 11 are dead and adds them to R. Again, with high probability, the algorithms proceed in the same way by subdividing[6] to form, $S_1 = \{1, 2, 3, 12\}$, $S_2 = \{4, 5, 13, 14\}$, $S_3 = \{6, 7, 8, 15, 16\}$ and $S_4 = \{9, 10, 17\}$, and announces route updates: $1 \rightarrow 2 \rightarrow 3$, $12 \rightarrow R_3$, $14 \rightarrow 13 \rightarrow \textit{Base Station}$, $15 \rightarrow 16 \rightarrow \textit{Base Station}$, $17 \rightarrow 10 \rightarrow R_4$. In the following epoch, the base station adds 13 and 12 to D. At this

[5]There is flexibility in the choice of responsive node to route to. Since our purpose is simply to illustrate the algorithms we choose the responsive node that appears closest in Figure 4, however load balancing is also an important consideration.

[6]Algorithm 3 will also subdivide each set with high probability because each set contains a node close to R.

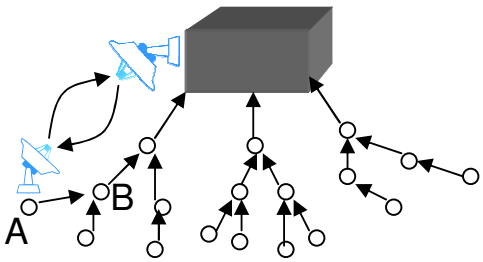

Figure 6: Disrupting a routing protocol

point, the algorithms diverge. Algorithm 2 continues subdividing each set, forming: $S_1 = \{1, 2\}$, $S_2 = \{3\}$, $S_3 = \{4\}$, $S_4 = \{5, 14\}$, and announces route updates: $2 \rightarrow 1 \rightarrow R_1$, $3 \rightarrow R_3$, $4 \rightarrow R_3$ $14 \rightarrow$ *Base Station*. In the following epoch, the base station will learn 14 is dead and will trace 5 by routing it to 4. On the other hand, Algorithm 3 may not do as much subdividing because of the associated node probabilities. For example, Algorithm 3 may form the sets: $S_1 = \{1, 2\}$, $S_2 = \{3\}$, $S_3 = \{5, 4, 14\}$, and announce the same route updates: $2 \rightarrow 1 \rightarrow R_1$, $3 \rightarrow R_3$, $4 \rightarrow 14 \rightarrow$ *Base Station*. In the next epoch the base station still needs to trace 4 and 5, but can accomplish this with shorter transmission distances by announcing the route updates: $4 \rightarrow 3$ and $5 \rightarrow 6$. Note that this saves the expensive routing of 4 to the R_3, which Algorithm 2 announced.

We summarize the advantages of each algorithm in the Figure 5.

6. OPEN PROBLEMS

We've shown that with minimal overhead the network topology can be conveyed to the base station and subsequently the cause of network partition can be determined with a small amount of communication. As mentioned earlier however, our algorithms contain some flexibility due to the unpredictability of sensor network topologies. Consequently, it is difficult to provide precise bounds on the lengths of the route update messages. Although receiving messages is not as costly as sending them, it does consume power, hence it is important to bound. We plan to implement the algorithms to better assess this value.

Our work assumes that a secure route-discovery protocol is available. Most routing protocols establish routes based on proximity, *i.e.,* nodes learn who their neighbors are by determining whose messages they can hear.

If an adversary can simply amplify signals of certain nodes during route-discovery then it can forge proximity information. Figure 6, for example, shows an adversary who is intercepting, and then amplifying, messages both from the base station and sensor node A. As a result, A will believe it close to the base station and will not forward its packets to B, as it should.

Note that this attack works even if the route-discovery messages are encrypted and authenticated. The adversary simply amplifies all (encrypted) messages, including all authentication information. As far as message exchanges are concerned, this results in a setup that is indistinguishable from one where A is indeed very close to the base station.

To complete the attack, the adversary stops amplification after the routing topology is established. A will always send its messages directly to the base station, but will fail to reach it. The attacker, therefore, has effectively disconnected A from the rest of the network.

In [8], Hu *et al.* independently describe this attack (termed a "wormhole") and provide an initial attempt to address the problem.

Acknowledgements

We wish to thank Jie Liu, Jim Reich, Hao-Chi Wong and Feng Zhao for helpful discussions.

7. REFERENCES

[1] Omer Berkman, Michal Parnas, and Jiri Sgall. Efficient dynamic traitor tracing. In *Proc.* 2000 *Symp. on Discrete Algorithms*, pages 586–595, 2000.

[2] P. Buonadonna, J. Hill, and D. Culler. Active message communication for tiny networked sensors. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01)*, April 2001.

[3] A. Cerpa and D. Estrin. Ascent: Adaptive self-configuring sensor networks topologies. In *Proceedings of the 21th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, New York, NY, June 2002.

[4] D. Du and F. Hwang. *Combinatorial Group Testing and its Applications*. World Scientific Publishing, Singapore, 1993.

[5] Amos Fiat and Tamir Tassa. Dynamic traitor tracing. *Journal of Cryptology*, 14(3):211–223, 2001.

[6] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS*, 2000.

[7] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for network sensors. In *Proceedings of ASPLOS*, 2000.

[8] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Wormhole detection in wireless ad hoc networks. Technical Report TR01-384, Rice University Department of Computer Science, June 2002.

[9] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM-00)*, pages 56–67, N. Y., August 6–11 2000. ACM Press.

[10] H. Krawczyk, M. Bellare, and R. Canetti. RFC 2104: HMAC: Keyed-hashing for message authentication, February 1997. Status: INFORMATIONAL.

[11] Q. Li, J. Aslam, and D. Rus. Hierarchical power-aware routing in sensor networks. In *Proceedings of the DIMACS Workshop on Pervasive Networking*, May 2001.

[12] A. Manjeshwar and D. P. Agrawal. Teen: A routing protocol for enhanced efficiency in wireless sensor networks. In *1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing (IPDPS)*, April 2001.

[13] Sergio Marti, T. J. Giuli, Kevin Lai, and Mary Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (Mobicom '2000)*, pages 255–265, Boston, MA, August 2000.

[14] A. J. (Alfred J.) Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.

[15] A. Perrig, R. Szewczyk, V. Wen, D. Cullar, and J. Tygar. Spins: Security protocols for sensor networks. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking (Mobicom '2001)*, 2001.