Link Scanner: Faulty Link Detection for Wireless Sensor Networks

Qiang Ma, Member, IEEE, Kebin Liu, Member, IEEE, Zhichao Cao, Member, IEEE, Tong Zhu, Member, IEEE, and Yunhao Liu, Fellow, IEEE

Abstract—In large-scale wireless sensor networks, faulty link detection plays a critical role in network diagnosis and management. Most potential network bottlenecks such as network partition and routing errors can be detected by link scan. Since sequentially checking all potential links incurs high transmission and storage cost, we propose a passive scheme Link Scanner (LS) for monitoring wireless links. As we know, to maintain a sensor network running in a normal condition, many applications in flooding manner are necessary, such as time synchronization, reprogramming, protocol update, etc. During such regular flooding processes that for other purposes originally, LS passively collects hop counts of received probe messages at sensor nodes. Based on the observation that faulty links can result in mismatch between received hop counts and network topology, LS deduces all links’ status with a probabilistic model. We evaluate our scheme by conducting extensive simulation tests. A real outdoor system is also deployed to verify that LS can be reliably applied to surveillance networks.

Index Terms—Wireless sensor networks, link detection, network diagnosis.

I. INTRODUCTION

WIRELESS sensor networks (WSNs) have been widely used in many application areas such as infrastructure protection, environment monitoring and habitat tracing. The reliability of individual links’ performance is crucial in these applications, e.g., in a surveillance network [12], the transmissions must be reliable to avoid false alarms and missed detections. Compared to the wired networks, it seems much more essential to detect link faults rather than node faults in WSNs. A wireless link itself virtually exists, which means we can’t directly observe and assess whether it performs well or not [19], [22], [25]. What is more, It proves difficult to localize the faulty links under a dynamic mal-condition in the wild, for the link quality will be significantly impacted by the natural environment like trees in the forest and flow in the ocean [3], [12]. Multi-hop networks suffer more harm than single-hop networks due to link failures. For example, a critical link may cause a large area of partition, or significantly interfere with routing protocol among the nodes, producing problems such as routing cycle and even network partition. Accordingly, compared to single-hop networks, faulty link detection becomes more difficult in the multi-hop networks due to topology features. A packet has to traverse multiple links to the sink, it is for this reason that exactly localizing a faulty link becomes really hard if only on the basis of whether the packet arrives at the sink or not. Therefore, faulty link detection becomes one of the most critical issues in multi-hop network diagnosis. Indeed, according to the status of a link, we are able to explain many failures like packet loss, routing failure, partition and so on. Notably, link performance actually reflects a network’s reliability and bottleneck if exist.

Although single link failures are more common, multiple link failures occur due to shared risks such as failure of a link while another link is under maintenance, or natural disasters that cause links traversing a region to fail. In [1], the authors use monitoring paths and cycles to localize single link and Shared Risk Link Group (SRLG) failures. They also prove that \((k+2)\)-edge connectivity\(^1\) was necessary and sufficient to uniquely localize all SRLG failures involving up to \(k\) links with one monitor. In practice, however, not all sensor networks can satisfy this strict condition, especially in the cases we spread the sensor nodes randomly in the area of interest. In addition, in most cases we are not allowed to set any more monitors after the deployment. What we expect is to utilize the rule-free probes (i.e., without computing the exact probing paths) to achieve link scan.

One of the most peculiar routing characteristics of WSN is routing dynamics. It is not surprising that a sensor node frequently changes its parent to forward packets. Unfortunately, many existing approaches just aim to detect the faulty links which had been behaving badly, but fail to offer an inspection on other unused ones, thus have no guidance to reroute when the current routing strategy is less than satisfactory.

To solve the above problems, in this work we propose Link Scanner (LS), a passive and rule-free detection approach for discovering faulty links in sensor networks. The object of LS is to provide a blacklist containing all possible faulty links. With such a blacklist, further analysis and recovery processes become possible, including (i) exploring the root causes of observed symptoms in the network, (ii) adjusting routing strategy for the related nodes, (iii) offering the spare list of links for every node. As a result, we not only achieve the goal of diagnosis, but also take a big picture of wholly link performance.

To maintain a sensor network running in a normal condition, many applications in flooding manner are necessary, such as time synchronization, reprogramming, protocol update, etc. In the flooding process, each node is expected to receive multiple probe messages through different paths. By embedding lightweight data into the flooding packet, LS passively collects hop counts of received probe messages at sensor nodes. Since faulty links may cause probes dropped, there must be mismatches between the received hop counts in sensor nodes and our expectations according to the topology. With a probabilistic and heuristics based inference model, LS analyzes the mismatches and deduces the faulty links. Specifically,

\(^1\)A network is said to be \(k\)-edge-connected if the removal of any \(k-1\) links will retain network connectivity.
the main contribution of this paper can be summarized as follows:

- To the best of our knowledge, we are the first to investigate a passive and rule-free method of detecting the faulty links including those potential but not used ones in sensor networks.
- According to the unique features of sensor networks, we design an efficient probe marking scheme that reveals the inner dependencies of sensor networks.
- LS proposes hierarchical inference models to capture the multi-level dependencies among the network elements and achieve high accuracy. We further introduce a learning-based inference scheme which increases the inspection accuracy and is thus scalable for large scale networks.
- We evaluate the performance of LS on a real testbed and extensive simulation, the results show that LS indeed detects most faulty links accurately, and helps in exploring the root causes of observed symptoms. A field study on a real outdoor deployment is also presented to verify that LS is practical to surveillance networks.

The rest of the paper is organized as follows. Section I-A presents our design and provides additional techniques to deal with several practical issues on implementation. Section II and III show the performance evaluation results from both real indoor testbed experiments and simulations. An outdoor field study is presented in section IV. Section V summarizes the related work. Section VI concludes the paper.

A. Main Design

We consider this problem in a large-scale wireless sensor network, and the network topology is known [7], [24], [4], [21]. A link exists if its length is within the communication range of radio. In our previous work [12], [11], we find that most of links, which either have a good PRR (Packet Reception Ratio) more than 90%, or have a bad PRR less than 10%. Therefore, in this work, the basic idea is to scan all the links in a flooding process. If a link is good, this probe should traverse on it, otherwise not [1], [5].

In this design, we consider both efficiency and diagnosis accuracy. First, in order to reduce the transmission overhead for each node while still obtaining sufficient information for inferring the link failures, we need to compress the local information in a certain way, then decompress and parse it at the base station. Second, LS should be able to work even if no sensor node knows its location, which also means that no computational operation at the sensor side. Most importantly, we desire to explore the failures on currently used links, as well as those unused links which potentially may fail.

B. Overview

In traditional ways of using active probes [26], [1] to detect faulty links, the monitor first maps out the paths for the active probes, then diagnoses the network according to the symptoms and results obtained through probes. In the design of LS (Figure 1), we add lightweight data in a flooding probe, and each node is expected to receive multiple probe messages through different paths. Each probe message contains a hop counter that records the number of hops from sink to current node. As faulty links may impact this process, there will be mismatches between the received hop counters in sensor nodes and the expectations based on the topology. We then analyze such mismatch with a probabilistic inference model and deduce the faulty links. The result would be a global diagnosis for the network, including how to explain the system performance as well as the prediction about the potential system bottleneck like contention and routing failures.

C. Probe Flooding

Recording all node IDs along the collection path of each data packet or calculating PRR of different paths are frequently used for estimating link qualities. Clearly, these two ways consume much overhead in information preservation and propagation. Besides, as links exist in a virtual manner, it proves difficult to observe and assess all links’ performance with fixed-path probes. They may fail to perceive those links which are not used but useful for rerouting in the future.

For a certain node, its flooding probe can be overheard by all its neighbors. For example, as shown in Fig. 2, the flooding process starts at the sink which broadcasts a probe. Node A and B hear the probe through one hop, and continuously broadcast it, so node C fetches two probes through two hops from A and B. Since node C isn’t aware of its location, it will broadcast the probe to ensure the coverage of flooding. Eventually, node A receives three probes from sink, node B and node C, respectively, i.e., the number of received probes should be equal to that of in-edges. To cover all the possible links in the network, besides the number of probes, the record also needs to distinguish the received probes. Now if A only gets two probes after the flooding process, for example, from sink and C, it means that either B fails to get any probes, or the link $B \rightarrow A$ is broken. Obviously, if B records that it actually gets the probe, we can simply infer that link $B \rightarrow A$ is broken.

A straightforward way to distinguish the probes is adding the node ID, so that each node records the parent where the probe comes from. Such an approach, being beneficial and efficient in a small network, may not work well for large-scale sensor networks, or dense networks where the number...
Sink probe is at most $C$ between receives $A$'s ($A$'s probes potentially come from $A$'s neighbors. Assume that $\text{MinHop}$ of probes of a node, i.e., maximum hop count in the received probes.

$$\text{MinHop}_{n \text{-hop}}$$ represents the minimum hop count in the received probes and $\text{MinHop}_{n \text{-hop}}$ we use the data structure in form ($n \text{-hop}$).

We observe that hop count can distinguish different probes to our recently deployed sensor network system, GreenOrbs [12], transmission overhead and information usefulness. Based on IDs. Clearly, we need to figure out a good tradeoff between the nodes, each node ought to consume 60 bytes to record all the of neighbors is over 30: even if we use 2 bytes to identify nodes, each node ought to consume 60 bytes to record all the IDs. Clearly, we need to figure out a good tradeoff between the transmission overhead and information usefulness. Based on our recently deployed sensor network system, GreenOrbs [12], we observe that hop count can distinguish different probes to some degree.

For example, if node $A$ records the corresponding hop count for every probe, it should receive one probe for each hop, i.e., one-hop probe from sink, 2-hop probe from node B and 3-hop one from node C. Recording the hop count number consumes much less resource than to record the node ID. In our design, we use the data structure in form ($\text{MinHop}, n_0, n_1, ..., n_{k-1}$), where $\text{MinHop}$ represents the minimum hop count in the received probes and $n_0$ represents the number of probes of $\text{MinHop}$-hop. Similarly, $n_i (0 \leq i \leq k-1)$ means the number of probes of ($\text{MinHop} + i$)-hop, and $\text{MinHop} + k - 1$ is the maximum hop count in the received probes.

In many cases, only 4 bytes of data is required to record by a node, i.e., ($\text{MinHop}, n_0, n_1, n_2$). For any node, say $A$, all $A$'s probes potentially come from $A$'s neighbors. Assume that $A$'s $\text{MinHop}$ is $C$, then any of $A$'s neighbor, say $B$, possibly receives $A$'s ($C$+1)-hop probe such that the hop count of $B$'s probe is at most $C$+2. So $A$ only needs to record the hop count between $C$ and $C$+2 (including $C$ and $C$+2). However, some corner cases happen due to asymmetric links as illustrated in Fig. 3. Node $N_k$ ($k > 2$) loses node $N$'s 2-hop probe but gets a k-hop probe along the path $N_1 \rightarrow N_2 ... \rightarrow N_k$ which is its only probe. So $N_k$ should broadcast a ($k$+1)-hop probe to its neighbors. If $N$ actually receives this probe, it records (1,1,0,...,1), where there are $k$-1 ‘0’s. This kind of asymmetric information extremely improves the inference accuracy but costs much transmission overhead. To reduce the cost, we code each byte in the record as follows:

- If the first bit is ‘0’. That means it indicates a number of hop count.
- Else the first bit is ‘1’. Then the remaining 7 bits indicate the number of successive ‘0’ in the record.

With this design, in Fig. 3, the $k$-1 successive ‘0’s can be represented by one byte “1x_5x_4x_3x_2x_1x_0”, where “0x_5x_4x_3x_2x_1x_0” equals to $k$-1 after binary-decimal conversion. This design proves feasible as we assume that there are no more than “01111111”, i.e., 127 probes traverse the same hop count, and thus no more than 127 ‘0’s in the record.

In practice, however, much information vagueness still exists when inferring faulty links with hop count. For example, in Fig. 2, assume that both $A$ and $B$ perform as our expectation (i.e., (1,1,1,1)), and $C$ only gets one 2-hop probe (i.e., (2,1)). In this case, what we can accurately infer is that $C$ misses a probe from $A$ or $B$, that is, one of link $B \rightarrow C$ and link $A \rightarrow C$ is broken. To solve this problem, the most straightforward metric is link length, i.e., without any information provided by other nodes, we can simply deduce that the longer link may result in probe loss. In a large-scale network with a more intricate performance, simply relying on the relationship between link length and PRR is hardly accurate, thus we need to consider more link features based on real system observations.

D. DLP

DLP (Disjunctive Logic Programming) is a formalism representing indefinite information. Similar to Prolog, language statements consist of facts, inference rules, strong constraints and weak constraints. In DLP, we initiate some facts based on real observation, which are definitely true and used to explore potential possibilities, or models in a disjunctive logic program. In addition, we need to enumerate both the strong constraints and weak constraints. Each strong constraint only presents a conjunction of facts, while a weak constraint additionally assigns a numeric cost to the conjunction. The program will prune the models which violate any strong constraint, then rank the remaining models according to the weak constraints they violate, finally outputs the lowest cost model of inferred facts generated from the observation facts and inference rules.

The specific DLP implementation used is DLV [9]. In DLV, we set the disjunctive inference rules as follows:

$$fact_1 \lor fact_2 \lor ... \lor fact_k : - fact.$$

In this rule $fact$ should be pre-defined as a possible truth either input as an observation, or inferred by others. $fact_i (1 \leq i \leq k)$ are disjunctive, which means $fact$ must infer one and only one of $\{fact_1, fact_2, ..., fact_k\}$. Besides, we can define a strong constraint as:

$$- fact_1, fact_2, ..., fact_k.$$

and a weak constraint as:
E. Probabilistic Reasoning

In order to detect all the potential links even if they haven’t been used, we consider every link, provided the physical distance of the corresponding two nodes is within effective communication range. In our expectation, every link should generate two probe records (i.e., two-way broadcast). If the collection record mismatches with our expectation, we can judge that some links must fail to deliver the probes. Due to the incompleteness and vagueness of information, some metrics have to be defined to infer the potential solutions.

1) Data Pre-processing: Raw record data is converted into observation facts for DLP, including topology features and probe records. Topology features mainly depict the physical location and link length, thus initialize the expected collection results during flooding. Probe records describe the observation facts, i.e., for node S and hop count C, how many probes with hop count C does S receive in the flooding? With topology knowledge, we can further group the neighbors of S by hop counts. For example in Fig. 2, we divide A’neighbors into three groups. If we find that the probe with hop count 2 is lost, we can say that B fails to send the probe to A due to some certain reasons. Normally, one group contains more than one node. For node C, its neighbors A and B are in one group since the probes from both of them are 2-hop. When C’s records show that only one 2-hop probe is received, we realize the group gets one less probe, but not attribute to one specific node’s fault.

In the program of DLP, we refine the facts like what described in Table I. Most of facts are from the records, like e(A, B) and distance(A, B, Length), completely depend on prior physical topology knowledge. noProbe(A) is determined by the collection records, for checking whether A’s record exists at the sink or not. Based on these facts, we set some inference rules to describe the relationships between them:

\[
g(A, HopCount) : \neg e(B, A, HopCount).
\]

<table>
<thead>
<tr>
<th>FACT</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>node(N, X, Y)</td>
<td>a node with ID as N and location (X, Y)</td>
</tr>
<tr>
<td>e(A, B, HopCount)</td>
<td>a directional link from A to B, in which the probe is HopCount-hop.</td>
</tr>
<tr>
<td>lost(A, B)</td>
<td>e(A, B, HopCount) exists but B did not receive A’s probe.</td>
</tr>
<tr>
<td>g(A, HopCount)</td>
<td>node A’s neighbor group with hop count of HopCount.</td>
</tr>
<tr>
<td>size(g(A, HopCount), n)</td>
<td>the expected number of neighbors in g(A, HopCount) is n.</td>
</tr>
<tr>
<td>inGroup(B, g(A, HopCount))</td>
<td>node B is in g(A, HopCount).</td>
</tr>
<tr>
<td>noProbe(A)</td>
<td>node A has not received any probe yet.</td>
</tr>
<tr>
<td>distance(A, B, Length)</td>
<td>the physical distance between A and B is Length</td>
</tr>
<tr>
<td>lostCount(A, HopCount)</td>
<td>the number of A’s lost probes with hop count as HopCount.</td>
</tr>
<tr>
<td>asymLink(A, B)</td>
<td>asymmetric links: Link A → B performs much better than Link B → A.</td>
</tr>
<tr>
<td>faultyLink(A, B)</td>
<td>Link A → B is detected as a faulty link.</td>
</tr>
</tbody>
</table>

TABLE I: DLP facts

The first rule tells that, if there exist two nodes, A and B, the distance between which is within the communication range, and the corresponding probe is of HopCount-hop. So A’s neighbor group of HopCount, i.e., g(A, HopCount) should be generated by the program. What not described in this rule is the number of probes should be equal to the size of this group. The second rule divides the neighbors into groups, narrowing the inference results when some probes are lost. Actually, our program includes 32 other rules generated by the broadcasting protocols, physical topology information and heuristic experiences. Note that, under some environments, some rules will be removed due to the environmental factors impact the transmission dramatically.

What we desire to know is the potential facts inferred by the observation facts. To be precise, currently we roughly know how many probes are lost during the flooding process. Furthermore, the missing probe’s hop counts are known. What follows is to infer that which specific neighbor in the group fails to broadcast the probe, so as to delve deeply into the exact faulty links. The fundamental inference rule is:

\[
\text{noProbe}(A) \lor \text{faultyLink}(A, B) : \neg \text{lost}(A, B).
\]

It means that, when A is proved to not send its probe to B, there are two disjunctive possibilities. One is A has not received any probe from others; the other one is link A → B performs poor. These two reasons may also be both true. In DLP, however, one and only one of them will be regarded as truth, and the other is ignored. Actually, we can easily judge whether A has probe or not in the collected reports.

2) Cost Function: Strong constraints are inviolable truths. Whatever models generated by DLP should not contain any conjunction of facts in a strong constraint. LS sets two strong constraints:

\[
: \neg \#\text{count}(A: \text{lost}(A, B), \text{e}(A, B, \text{HopCount})) \neq \text{lostCount}(B, \text{HopCount}).
\]

\[
: \neg \text{noProbe}(A), \text{not lost}(A, B), \text{e}(A, B, \text{HopCount}).
\]

The first constraint guarantees that every lost probe of node B should be traced to its specific neighbor. In DLP, this constraint requires to traverse every combination of neighbors in each group. The second one presents a simple logical contradiction, that is, if node A has no probes, every one of its neighbors must not receive any probes from A.

Weak constraints are based on observed correlations between the links corresponding to engineering practices. Each practice holds as a general rule of thumb, but may be violated by an individual model. Thus the model that violates the fewest
practices seems to be the best approximation of reality. Here we list the weak constraints in order of importance.

1) One node’s fault may cause its neighbors simultaneously fail to receive the probe from it. That is, a faulty node results in all its out-going links poor simply because there is no probe being sent out.

2) Similar with the first one, one node may simultaneously lose its neighbors’ probes. That is, a faulty node results in all its in-going links poor.

3) Regional correlation. Many studies show that environment factor can significantly impact link performance. At the same time, regional channel collision sometimes leads to a critical degradation of link qualities.

4) Symmetric links. Although there are a few asymmetric links in the network, most pairs of links perform consistently, especially for small packets.

5) The longer the distance between two nodes, the less the PRR, especially when the PRR is less enough, the tendency becomes more obvious.

The cost for a model is assigned based on the number of practices violated, weighted by the above importance of the practice. It also proves possible for DLP to output multiple equal-cost violated models. To give another group of possible solutions, after each round DLP generates a set of models, we then add one strong constraint in which the fact conjunction is set as same as the output into the program, hence the program will first eliminate the previous solutions. In conclusion, DLP programs suffer from three sources of complexity: (1) the exponential number of answer set candidates; (2) the difficulty of checking whether a candidate M is an answer set; (3) the difficulty of determining the optimality of the answer set.

3) Learning-based Scheme: As explained above, the basic inference model consists of input facts, rules and constraints. Input facts include topology information (i.e., nodes’ location and links’ length) and collected records. Rules describe the inner relationships between facts including input facts and hidden facts, while the constraints are used to exclude inaccurate inferences (i.e., strong constraints) and optimize possible inferences (i.e., weak constraints). Besides, feedback information could be provided by fault reports. Generally, feedback information is divided into two categories. One is facts which directly point out the links’ performance, thus the next inference is able to narrow the scope of lost probes. The other one is parameter adjustment, which focuses on weak constraints, including importance order and link features.

For example, if there shows about 30% asymmetric links exist in the network, hence we can update the corresponding weak constraint, and thus the next inference can adjust the network parameters. Our evaluation shows that this learning-based scheme significantly increases the inference accuracy.

II. Testbed Evaluation

We first evaluate LS through a real indoor testbed consisting of 60 TelosB motes (Figure 4), where the network diameter is 7 hops. The sensor nodes are fixed on a mobile wall. Besides of power supply, the wire is used to send and collect network information wholly independent of wireless channel. Every node records the transmitters of received probes, and send this record through wire to the back-end. These records compose the ground-truth.

Two metrics are used for evaluating LS: false negative rate (i.e., miss detection rate) and false positive rate (i.e., false alarm rate). False negative rate is defined as the proportion of faulty nodes which are detected as normal, while false positive rate is defined as the proportion of normal nodes which are detected as faulty. Because the inferring process is based on record collection, fault reports may be different if the collections vary. To examine the reliability of LS, we conduct 100 experiments for each group of comparison. We also take an external record as ground truth for comparison through USB serial channel, so as not to impact the regular flooding and collecting process.

In the first experiment, we examine how network scale affects the fault report. Intuitively, more nodes indicate more links. In the second experiment we change the network density, thus change the potential possibilities of probe transmissions. From the angle of DLV, this may bring in more solution combinations, further disturb regular inference. What is more, a dense network may significantly degrade the probe flooding, hence impact the expected record for the real topology. Unlike the previous two, the third experiment utilizes the former fault reports to help the current inference model eliminate vagueness and correct the weak constraints on the link correlation.

A. Network Scale

We run a series of experiments with 20, 40, 60 motes, respectively, under the same network density and no feedback from the fault report (i.e., 100 experiments are totally independent). As described in Fig. 5, the more nodes in the network, the more difficulties existing in the inference. False negative rate stays about 7.3% in 20-mote network, and increases to 11.4% and 13.8% when network scale enlarges to 40 and 60 motes. False positive rate also follows this trend. As we can see, when the network includes 60 nodes, one out of every 7 links reported as normal is actually bad, while one out of even every 9 links reported as faulty is normal in fact. For each node, once it misses one probe, the more neighbors, the more combinations of solutions in DLV’s inference model. Furthermore, a large-scale network suffers from other matters like channel collision and routing error. From this set of results, LS is proved to perform reliably under three network scales.

B. Network Density

Network density can also significantly change the network topology. A dense network suffers more channel collision and
Besides, some transmissions exceed our expectation of link length. If there is no important constraints to address these issues. "Irregular" phenomena, which seriously impacts the inference of the probe but the closer ones fail. Indeed, there exist many such overflow or channel collision, some far away neighbors receive the probe from node 27 (dash line). The closer nodes (i.e., node 9, 10, 17) are supposed to broadcast their probes to node 27, successfully. When the network is becoming denser, nodes may get more neighbors and experience more mismatch cases. Notably, it proves difficult to figure out these cases, since they act almost randomly and independently. Figure 9b shows the fault report. Relying on higher level weak constraints, LS judges that node 27’s 2-hop probes are from node 10 and 16, but not node 7 and 8. Inversely, we correctly judge that node 8, 10, 16, 17 have received the probe from node 27. For node 8, its record shows that it has received 6 3-hop probes. Most of its neighbors broadcast 2-hop probe, and its closest 3-hop neighbors are node 4, 18, 19, 27, 26, 25 and 24. However, there is not any node has received probes from node 18, thus we judge that node 18’s probe must disappear in the flooding procedure.

C. DLV Feedback

From these two groups of experiments, it proves that a multiple-node, dense network may bring much vagueness into the DLV inference model, leading to a biased fault report. Because DLV generates the optimal inference result just based on its facts and rules. The facts include node location, link distance and records, while the rules show the inner relationship between the links (detailed in section I-E). Unlike the previous experiments, we run 100 experiments which are not independent. That is, every fault report generates a feedback for the next DLV program. The experiments try three schemes for feedback: weak constraints, facts, and both. In our implementation, there are four weak constraints. The feedback of weak constraints optimizes the cost functions, like importance order and proportions of asymmetric links in the network. The feedback of facts are more straightforward and emphasized, including the bad links and nodes found, and thus makes the program totally avoid to obey the rules, and eliminates inaccurate inferences by the rules. Figure 7 clearly shows that whatever feedback of constraints are provided by the fault report, it only produces little or even no benefit in the next inference. By contrast, feedback of facts concretely increases the diagnosis accuracy.

D. Topology Study

Figure 8 shows the topology of 40 nodes in our experiments, including some probe traces. As we can see, due to queue overflow or channel collision, some far away neighbors receive the probe but the closer ones fail. Indeed, there exist many such "irregular" phenomena, which seriously impacts the inference if there is no important constraints to address these issues. Besides, some transmissions exceed our expectation of link length.

For example, as illustrated in Fig. 9a, the 2-hop probes received by node 27 are from node 7 and 8 (real line), respectively, and node 8, 10, 16, 17 receive the probe from node 27 (dash line). The closer nodes (i.e., node 9, 10, 17) are supposed to broadcast their probes to node 27, successfully. When the network is becoming denser, nodes may get more neighbors and experience more mismatch cases. Notably, it proves difficult to figure out these cases, since they act almost randomly and independently. Figure 9b shows the fault report. Relying on higher level weak constraints, LS judges that node 27’s 2-hop probes are from node 10 and 16, but not node 7 and 8. Inversely, we correctly judge that node 8, 10, 16, 17 have received the probe from node 27. For node 8, its record shows that it has received 6 3-hop probes. Most of its neighbors broadcast 2-hop probe, and its closest 3-hop neighbors are node 4, 18, 19, 27, 26, 25 and 24. However, there is not any node has received probes from node 18, thus we judge that node 18’s probe must disappear in the flooding procedure.
In practice, however, it is rare that nodes fail to broadcast or receive probes in a flooding procedure. Therefore, in most cases we need to infer faulty links on the basis of link features. That is why we need the weak constraints. In the large-scale simulation study we will give a more specific comparison in allusion to the characteristics of weak constraints in the inference model.

III. LARGE-SCALE SIMULATION STUDY

As the testbed experiments can only investigate a limited design space in terms of the network scale and corner cases, we further conduct a large-scale simulation study. In this evaluation, we pay more attention on the importance order of weak constraints, as well as their respective inner parameters.

A. Simulation Setup

In the simulation design, 2000 sensor nodes are randomly deployed on a 2500m × 2500m map, and the communication range is 100m, thus the network density is average 10 neighbors for each node when the nodes are uniformly distributed. We use the logarithmic distance path loss model [18] to simulate the received signal strength \( S \), which can be formulated as:

\[
S_i \propto -10\beta \log\left(\frac{d_i}{d_0}\right) + X_i,
\]

where \( \beta \), the signal fading factor, is set to 4 as in [27]. \( d_i \) is the link length while \( d_0 \) is the reference distance set to 1m. We also add a random noise factor \( X_i \) in the simulated environment, which follows a 0-mean normal distribution with variance \( \sigma^2 \) where \( \sigma \) is set to 4. A link’s quality is decided by its received signal strength, an extra random factor and other manual settings. However, we emphasize the inference procedure but not to simulate a complicate scene for data collection. We manually set some links disconnect, according to link quality and the ratio of asymmetric links. These links could be the ground-truth for comparison. Compared to real testbed evaluation, the simulation results seem more reasonable on the basis of network topology. Hence we primarily discuss about the inference model itself, and study the impact of importance order of weak constraints and some individual specific weak constraints in DLV. For each setup we take 1000 runs.

B. Impact of Importance Order

In our expectation, with different importance orders, DLV generates different solutions. For example, if we desire to emphasize the importance of the correlation between link length and link quality, the inference model significantly depends on network topology. By contrast, if we believe that some bad nodes probably exist in the networks and their self-contained hardware faults definitely cause links failures, we can rank this fact the highest. Otherwise, it does not make sense that we are discussing about link features and correlations while there are even no links in the networks.

Figure 10a, 10b describes LS’s fault report under four importance orders. We use "12345" to represent the order we list in section I-E, then vary the order to "45123", "54123", and "34512" in the simulation, in order to verify our expectation. As we can see, LS achieves a false negative rate around 10.2% as well as a false positive rate around 9.5%. Under the order "45123" and "34512", LS fails to accurately explore some links’ real performance. They both put constraint 4 before constraint 5, 1, and 2. So the inference model mainly considers about asymmetric links in the network and takes less concern on some existing bad nodes. What is more, the order "54123" regards the link length as the most critical factor to judge a link’s performance, which strongly differs from the real system observation.

In addition, we try to observe the variations by removing some weak constraints from the inference model at the beginning. First, we remove the weak constraints about that most faulty links are caused by bad nodes. Then we remove the weak constraint defined based on the observation about symmetric links in the network, which implies that a link probably performs well if its reverse link is good. We find that the impact of constraint 1 and 2 is so obvious, nearly 17% difference. As mentioned above, a sensor node’s own hardware failure or program error may cause all of its links even its neighbors’ links destructive. Without these two constraints in the interference model, DLV fails to utilize the inner relationship between links and nodes. For example, it should be seriously challenged if the final fault report says that only one or two neighbors have received node A’s probe but node A actually has 20 neighbors around. Constraint 4 also partially impacts the inference results in the simulation where we set about 20% asymmetric links.

C. Impact of Asymmetric Links

In this section, we delve deeply into the parameter settings in the weak constraint about asymmetric links. In the simulation setup, there are 20% asymmetric links in the network. We initialize DLV’s prior knowledge about asymmetric links number to 10%, 20%, 30% and 40%, respectively. DLV potentially regards a link as asymmetric link with a possibility of this manual prior knowledge. As described in Fig. 10c and 10d, LS’s accuracy fluctuates when the prior knowledge mismatches with the ground truth, but keeps relatively accurate false negative rate and false positive rate. We believe that this constraint’s low importance more or less eliminates the impacts because the first three important constraints have covered its potential conclusive decision, which can be also explained by the system evaluation on DLV feedback.
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2421353, IEEE Transactions on Wireless Communications

In section II we evaluate LS under an indoor environment, discussing how different network topologies impact the inference results. Then we conduct a large-scale simulation study to examine LS by adjusting the parameters in the program. In this section, we deploy a real outdoor system to verify that LS can be reliably applied to surveillance networks. 80 sensor nodes are deployed in a 75m × 20m forest, as illustrated in Fig. 11, in a 5 × 16 manner, which makes it easy to locate the nodes and compute the links’ length. Nodes’ transmission power is set as 15 \(^3\) which guarantees each node has 10 neighbors on average. Each node records the transmitters of received probes, then we read the memory of them to get the ground-truth.

\(^3\)In TinyOS, every transmission is assigned with signal power from 1 to 31. The higher power, the better and stronger signal.

IV. FIELD STUDY

A. Implementation

The program flow for each node is shown in Fig. 12. After receiving a probe, the sensor node updates its local MinHop by recording the minimum hop. Then MinHop is embedded into the broadcasting probe. In addition, by counting the number of probes for different hops, a record (detailed in Section I-C) is generated and collected by the sink. LS incurs memory overhead on RAM and ROM for data and program storage, respectively. (i) As mentioned in section I-A, every report is usually generated in the form \(\text{MinHop}, n_0, n_1, n_2\), i.e., only 4 bytes storage consumption. (ii) To evaluate LS’ ROM overhead, we implement a simple collection application using CTP. Then we compare this benchmark and the same one with LS module. The original benchmark consumes 18270 bytes ROM while the LS-embedded version consumes 22440 bytes, which indicates the LS module consumes approximately 4.1KB ROM, which is acceptable compared to 48KB ROM in telosB. Besides, no computation overhead is needed as the probe is broadcasted.

Table II and III are the preliminary knowledges about the number of asymmetric links in the network, which help us set the algorithm parameters. We take three scenes: flat, forest, and indoor, and observe the correlations between the number of asymmetric links and PRR, as well as packet size. For example, in Table II, if an indoor link’s PRR is only 10%, then 21.8% of its links are asymmetric. Here we regard a link as asymmetric if the difference of PRR between the corresponding links is larger than 0.1. The ratio in the bracket is under the assumption that a link is asymmetric if the difference is larger than 0.2.

B. Basic Observations

First, we observe the distribution of faulty links. Due to hidden terminals and receiving queue overflow, a node with more neighbors is more likely to lose packets, thus degrades its links’ performance. In this field study, the nodes are divided into 4 groups according to the number of their neighbors, and we record each probe’s sender ID to identify the sources of received probes for each node. Figure 13a shows the ratio for each group. For those nodes have less than 9 neighbors, each of them loses 1.5 probes on average, while the nodes with more than 12 neighbors lose 4 probes, averagely.

In addition, we compare every pair of nodes to find out the asymmetric pairs, i.e., only one of two nodes has received the other’s probe. Similarly, we divide the nodes into 4 categories according to the number of neighbors. Figure 13b compares the ground truth and the inference results. For those nodes have
neighbors, the numbers are 3.2 and 3.7, respectively, and thus significantly. In contrast, for those nodes with more than 12 neighbors, the number of asymmetric links could not change the detection report, which means that tiny difference of knowledge about 9 neighbors in ground truth is 1.2, while 1.3 in LS's (i.e., LS before and after learning). Figure 13b shows that faulty links in the fault reports of two inference models explain the benefit generated by learning, we compare the decreases the false positive rate from 15.5% to 13.2%. To its accuracy for those nodes with more than 12 neighbors, i.e., detects the links more accurately. Before learning, LS has a false negative rate as 17%, while it achieves to 15.4% after we modify the program according to Fig. 13b by adjusting the prior knowledge about the number of asymmetric links for different groups of nodes.

C. Performance Evaluation

We also evaluate LS in terms of false negative rate and false positive rate. We try to add the basic observations into the inference model, to help the program specify different ratios of asymmetric links for different groups of nodes. Figure 13c compares the false negative rate before and after the learning. As we can see, for those nodes with less than 9 neighbors, the false negative rates are both 8.9%. In contrast, LS improves its accuracy for those nodes with more than 12 neighbors, i.e., detects the links more accurately. Before learning, LS has a false negative rate as 17%, while it achieves to 15.4% after we modify the program according to Fig. 13b by adjusting the prior knowledge about the number of asymmetric links for different group of nodes.

We observe similar improvement of false positive rate in Fig. 13d. The evaluations before and after learning stay the same for the nodes with less than 9 neighbors, i.e., both are 7.4%. In contrast, the links of nodes with more than 12 neighbors are more accurately detected. Learning process decreases the false positive rate from 15.5% to 13.2%. To explain the benefit generated by learning, we compare the faulty links in the fault reports of two inference models (i.e., LS before and after learning). Figure 13b shows that the average number of asymmetric links of nodes with less than 9 neighbors in ground truth is 1.2, while 1.3 in LS’ report, which means that tiny difference of knowledge about the number of asymmetric links could‘t change the detection significantly. In contrast, for those nodes with more than 12 neighbors, the numbers are 3.2 and 3.7, respectively, and thus the learning process improves the evaluation by adjusting the corresponding knowledge in the inference model.

V. RELATED WORK

Network diagnosis has been extensively studied in recent years. Existing approaches can be broadly divided into two categories: debugging tools and inference schemes. This work belongs to the later category. Clavoyant [23] is a notable tool which focuses on debugging sensor nodes at the source-level, and enables developers to wirelessly connect to a remote sensor and execute debugging commands. Declarative Tracepoints [2] allows the developers to insert a group of action-associated checkpoints at runtime, which are programmed in an SQL-like declarative language. Existing inference-based diagnosis schemes for WSNs like Sympathy [17] or Emstar [6] rely heavily on an add-in protocol that periodically reports a large amount of network information from individual sensor nodes to the sink, introducing huge overhead to the resource constrained and traffic sensitive sensor network. In order to minimize the overhead, some researchers propose to establish inference models by marking the data packets [13], [16], and then parse the results at the sink to infer the network status, or conduct the diagnosis process in local areas [14]. Steinder and Sethi [20] apply Belief Network with the bipartite graph to represent dependencies among links and end to end connections, then the root causes can be deduced by conducting inference on the Belief Network. [15] explores the bottleneck nodes in a WSN, and [10] enhances the network visibility by analysing the events and status in history.

Besides, most approaches actively design their probes to fetch desired information for faulty link detection [1], especially in the managed enterprise WLANs and wireless mesh networks, where the monitors are easy to deploy. For each cycle, a node is required to monitor the cycle’s performance. [8] develops a non-adaptive fault diagnosis through a set of probes where all the probes are employed in advance. The authors in [7] propose a failure detection scheme, in which monitors are assigned to each optical multiplexing and transmission section. These approaches usually compute the probe paths according to different network symptoms, so as to combine the network topology to infer the link status. For a large scale sensor network, however, deploying monitors in the wild not only increases the cost, but also needs to guarantee sustainable management. Sniffers can be used to collect the information. Indeed, to use sniffer also needs to take into account the cost of maintenance and other deployment details like coverage and timeline accordance.

VI. CONCLUSION

A wireless network often contains a large number of links which virtually exist in the air, but we can never directly observe whether they perform well or not. We proposes a passive and low-cost link scanning scheme LS for faulty link detection. LS infers all links statuses on the basis of data collection from a prior probe flooding process, in which we leverage hop count to reflect the in/out-going link performances. In the inference model, we use DLP to describe the inner relationship among
the links, and finally output the optimal fault report with some constraints, which reversely generates a feedback for DLP’s next computation. We evaluate our algorithm through a testbed consisting of 60 TelosB sensor motes and an extensive simulation study, while a real outdoor system is deployed to verify that LS can be reliably applied to surveillance networks.

ACKNOWLEDGMENT

This study is supported in part by the NSF China Distinguished Young Scholars Program 61125202 and NSF China No. 61472219.

REFERENCES


Qiang Ma received the BE degree in Department of Computer Science and Technology at Tsinghua University, China, and his Ph.D. degree in Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His research interests include sensor networking, network diagnosis and management and mobile computing. He is a member of the IEEE.

Kebin Liu received his BS degree in Department of Computer Science from Tongji University, and an MS degree in Shanghai Jiaotong University, China. He received his PhD degree in Department of Computer Science and Engineering at Shanghai Jiaotong University, China. His research interests include sensor networks and distributed systems. He is a member of the IEEE.

Zhichao Cao received the BE degree in Department of Computer Science and Technology at Tsinghua University, China, and his Ph.D. degree in Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His current research interest is wireless sensor networks and mobile network. He is a member of the IEEE.

Tong Zhu received his BS degree in Department of Computer Science and Technology from Nanjing University, China, and his Ph.D. degree in Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His research interests include sensor networking, measurement, and mobile computing. He is a member of the IEEE.

Yunhao Liu received his BS degree in Automation Department from Tsinghua University, China, and an MA degree in Beijing Foreign Studies University, China. He received an MS and a Ph.D. degree in Computer Science and Engineering at Michigan State University, USA. He is now IEEE Fellow, Chang Jiang Professor and Dean of School of Software, Tsinghua University, China.