Passive Diagnosis for Wireless Sensor Networks

Yunhao Liu, *Senior Member, IEEE*, Kebin Liu, and Mo Li, *Member, IEEE*

***Abstract*—Network diagnosis, an essential research topic for traditional networking systems, has not received much attention for wireless sensor networks (WSNs). Existing sensor debugging tools like sympathy or EmStar rely heavily on an add-in protocol that generates and reports a large amount of status information from individual sensor nodes, introducing network overhead to the resource constrained and usually traffic-sensitive sensor network. We report our initial attempt at providing a lightweight network diagnosis mechanism for sensor networks. We further propose PAD, a probabilistic diagnosis approach for inferring the root causes of abnormal phenomena. PAD employs a packet marking scheme for efficiently constructing and dynamically maintaining the inference model. Our approach does not incur additional traffic overhead for collecting desired information. Instead, we introduce a probabilistic inference model that encodes internal dependencies among different network elements for online diagnosis of an operational sensor network system. Such a model is capable of additively reasoning root causes based on passively observed symptoms. We implement the PAD prototype in our sea monitoring sensor network test-bed. We also examine the efficiency and scalability of this design through extensive trace-driven simulations.**

***Index Terms*—Diagnosis, passive, sensor networks.**

I. Introduction

WIRELESS SENSOR NETWORKs (WSNs) have been widely studied for enabling various applications such as environment surveillance, scientific observation, traffic monitoring, etc. [14], [28]. A sensor network typically consists of a large number of resource-limited sensor nodes working in a self-organizing and distributed manner. Having made increasing efforts [6], [7], [10]–[12], [16], [18]–[20], [27], [31] on the robustness and reliability of WSNs under crucial and critical conditions, researchers, however, have done little work targeting the in-situ network diagnosis for testing operational sensor networks. It is of great importance to provide system developers useful information on a system's working status and guide further improvement to or maintenance on the sensor network.

Due to the *ad hoc* working style, once deployed, the inner structures and interactions within a WSN are difficult to observe from the outside. Existing works for diagnosing WSNs mainly rely on proactive approaches, which implant debugging agents into sensor nodes, periodically reporting the internal status information of each node to the sink, such as component failures, link status, neighbor list, and the like. For example, Zhao *et al.* [33] propose to scan the residual energy and monitor parameter aggregates including link loss rate and packet count. Such information is collected locally at each node and transmitted back to the sink for analysis. Sympathy [23] actively collects run-time status from sensor nodes like routing table and flow information and detects possible faults by analyzing node status together with observed network exceptions. The proactive information generation and retrieval exerts extra computational operations on sensors and imposes a large communication burden on a WSN, which is usually fragile at high-traffic loads. Those approaches work more like debugging or evaluation [26] tools before the system is released for use outside laboratory settings. While such tools are effective for offline debugging when sensor behavior and network scale can be strictly controlled, they may not be suitable for in-situ network diagnosis of an operational WSN since they continuously generate a large amount of traffic and aggressively consume computation, communication, and energy resources. Also, integrating those complex debugging agents with application programs at each sensor node introduces difficulties for system development.

Fig. 1. OceanSense project.

This work is motivated from our ongoing sea monitoring project [4], [30]. As shown in Fig. 1, for this project, we launched a working prototype WSN consisting of tens of nodes that float on the sea surface and collect scientific data such as sea depth, ambient illumination, pollution, and so on. Recently, in the field deployment tests, we often observed abnormal energy depletion that never occurred in the controlled laboratory experiments. We suspect that such a phenomenon is due to the usage of the MultiHopRouter (integrated in SURGE) component that frequently switches the optimized routing tree of the network owing to the highly instable environment of the sea. We also observed other problems on the sink side such as high delay of data sampling and unbalanced packet loss. Fast and accurate identification of the root causes is necessary before taking any further action such as issuing reboot messages to certain nodes or physically examining the suspicious links. With current debugging tools, it is indeed difficult to integrate

Manuscript received October 12, 2008; revised June 22, 2009 and September 17, 2009; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Agrawal. First published December 18, 2009; current version published August 18, 2010.This work was supported in part by NSFC/RGC Joint Research Scheme N_HKUST602/08, National High Technology Research and Development Program of China (863 Program) under Grant 2007AA01Z180, National Basic Research Program of China (973 Program) under Grant 2006CB303000, and NSF China Key Project 60736016 and 60533110.

The authors are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong (e-mail: liu@cse.ust.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2009.2037497

their agents with our application programs. It is even worse if we implant proactive information collectors in the network, which would inevitably speed up the depletion of energy and rapidly reduce the expected lifetime of the sensor network.

In this study, we propose an online diagnosis approach that passively observes the network symptoms from the sink. Using probabilistic inference models, this approach effectively deduces the root causes of abnormal symptoms in the network. Compared to proactive debugging tools, the passive diagnosis approach observes data from routine application packets for back-end analysis. It can also be maintained in a running system at lightweight cost, thus it is expected to accommodate the application system in a timely manner without degrading performance.

Inference-based network diagnosis methods have been widely investigated and applied in enterprise networks [5]. Various types of inference models, both deterministic and nondeterministic, have been proposed for inferring the root causes of service failures. Most models are built on expert knowledge or trained from historical data from the networks. The construction of such models can be very complicated, and once constructed, the models are often viewed as remaining unchanged for a relatively long period [5], as enterprise networks are usually stable with few dynamics in their structures. Compared to enterprise or static networks, however, sensor networks have the following unique features: 1) sensor nodes have extremely limited computational and energy resources; 2) the network topology is highly dynamic due to the instable environment and acquiring prior knowledge of the network is difficult; 3) the individual sensor nodes are error-prone. Such conditions make existing active approaches for static network diagnosis infeasible. Thus, WSNs cannot easily adapt to such slow start approaches as sensors are self-organized without any prior information on the dependencies among network elements. The high dynamics of the WSN structure also leads to the infeasibility of those inference models built from static data.

We address the above challenges as follows. First, we introduce a packet marking scheme, which marks the regular routine communicating packets to continuously reveal their communication dependencies within the network. Using the output of the scheme, the sink constructs and dynamically maintains a probabilistic inference model. This scheme works in a lightweight manner without any extra transmission in the network and can adapt to frequent network changes. Second, we employ a hierarchical inference model that captures multilevel dependencies in the network. The hierarchical model can be constructed based on incomplete information, and it is able to efficiently handle the network dynamics by updating only the changed parts. This model takes both positive and negative symptoms as input and reports the inferred posterior probability of possible root causes. Third, we design an online inference engine capable of additively reasoning the root causes such that it works even with incomplete or suspicious inputs in a nondeterministic manner. The major contributions of this study are as follows.

To the best of our knowledge, we are the first to investigate a passive method of diagnosing the wireless sensor networks.

1) According to the unique features of sensor networks, we design an efficient packet marking scheme that dynamically reveals the inner dependencies of sensor networks without injecting extra transmissions.
2) We propose hierarchical inference models that capture the multilevel dependencies among the network elements and achieve high accuracy. We further introduce a fast inference scheme that reduces the computational complexity and is thus scalable for online diagnosis in large-scale WSNs.
3) We implement our diagnosis approach, PAD, and test its effectiveness in our sea monitoring project with 24 sensors. The results of our field test show that PAD indeed helps in exploring the root causes of observed symptoms. Relying on the output of PAD, we have successfully improved our application programs.
4) We further analyze and evaluate the scalability and effectiveness of PAD design through extensive simulations under varied conditions using the trace we collect from the prototype implementation.

The rest of this paper is organized as follows. Section II introduces related work. Section III describes the framework of our system. We introduce the packet marking scheme in Section IV and discuss the two inference models based on Belief Network and Causality Diagram in Section V. In Section VI, we present our implementation and simulation results. We conclude this work in Section VII.

II. Related Work

Most existing approaches for sensor network diagnosis are proactive, in which each sensor employs a debugging agent to collect its status information and reports to the sink by periodically transmitting specific control messages. Some researchers propose to monitor sensor networks by scanning the residual energy [33] of each sensor and collecting the aggregates of parameters of sensors where in-network processing is leveraged. By collecting such information, the sink is aware of the network conditions. Some debugging systems [23], [29] aim to detect and debug software failures in sensor nodes. For example, Clairvoyant [29] focuses on debugging sensor nodes at source level and enables developers to wirelessly connect to a remote sensor in the network and execute standard debugging commands on that node including break, step, and the like. Sympathy [23] is an advanced debugging tool that detects and debugs the failures in a sensor network. It actively collects in-network information periodically from each sensor node such as neighbor list, traffic flow, and the like and analyzes the network status at the sink. By carefully selecting an optimal set of information metrics, Sympathy aims at minimizing the diagnosis cost so as to be applicable to resource-limited sensor networks. It also applies an empirical decision tree to determine the most likely root causes for an observed exception.

Much effort has been expended on network diagnosis for enterprise networks. Commercial tools [1]–[3] independently monitor servers and routers with various control messages, and alerts are automatically generated from the implanted agents in different network equipment. Those tools, being effective for diagnosing large-scale networks, are too complicated and energy-consuming for resource-constrained sensor networks. There have been some passive diagnosis approaches proposed for enterprise networks that collect a network's operational

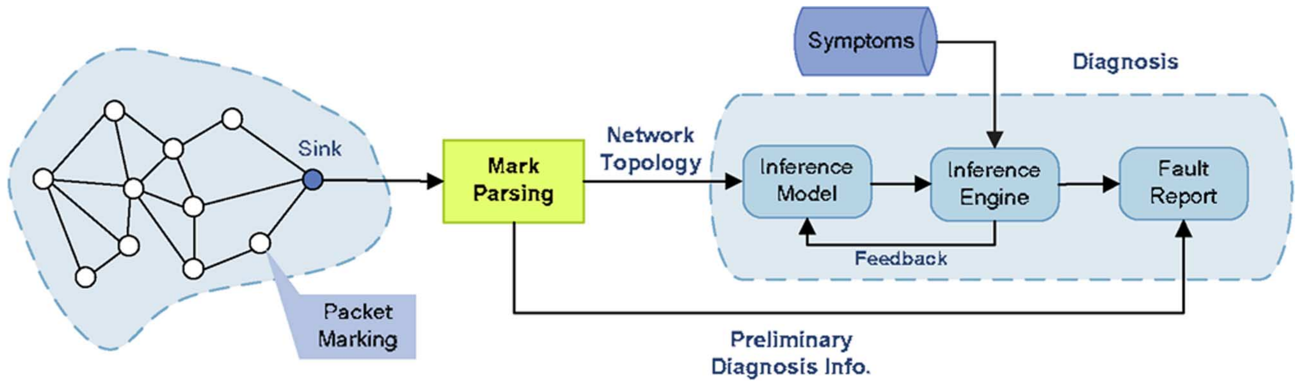


Fig. 2. PAD system overview.

status from routine data packets so as to deduce the possible root causes of exceptions by an inference model. For example, Score [17] troubleshoots via shared risk modeling. It adopts a simplified two-level graph as the inference model and formulates the problem of locating fault roots as a minimal set cover problem. Kandula *et al.* explore the bipartite graph inference model and propose Shrink, introducing a probabilistic inference scheme [15]. The bipartite graph model approximates the dependencies in enterprise networks and greatly simplifies the complexity of the inference process. Steinder and Sethi [24], [25] also assume a bipartite graph model and apply Belief Networks [21] with the bipartite graph to represent relations among links and end-to-end communications. Shi *et al.* [22] present a fault diagnosis approach for general static complex systems based on Causality Diagram. The above schemes either require preknowledge of the network dependencies, which are obtained through Shared Risk Link Groups or SNMP in a relatively stable enterprise network, or adopt simplified models to approximate the network dependencies. A WSN, however, is featured by its hierarchical multilevel structures, which can hardly be approximated by the bipartite graph model. It is also unpractical to maintain the network dependencies as stable inputs in highly dynamic and self-organized sensor networks.

The recently proposed Sherlock is the only work that adopts a multistate and multilevel inference graph for the network diagnosis [5]. They use a scoring function to derive the best explanations (root causes) for observed service exceptions. In order to avoid NP-hard computation complexity, they assume that there are at most a small constant number of failures in the enterprise network. This assumption is not valid for the unreliable and lossy WSNs. Guo *et al.* [13] tackle the problem of detecting nodes with faulty readings.

III. System Framework

We view the sensor network as a method for data acquisition in which source nodes periodically sample data and deliver them back to the sink through multihop communication. We do not assume any specific routing strategy, that is, our approach deals with networks of various communication topologies such as spanning tree or directed acyclic graph (DAG).

We design a passive diagnosis approach, PAD, for such sensor networks. PAD aims to help network managers explore the root causes of exceptions in a running sensor system. PAD implants a tiny lightweight probe into each sensor node that sporadically marks routine application packets passing by so that the sink can reassemble a big picture of the network conditions from those small clues. Nevertheless, information from marking probes is quite limited and not sufficiently accurate. PAD employs a probabilistic model to infer the statuses of unobservable network elements and reveal the root faults in the network. PAD denotes the observed abnormal situations as negative symptoms such as a long time delay of data arrival or frequent packet loss. It denotes any successful packet reception as positive symptoms. The inference model inputs both negative and positive symptoms to derive network statuses.

As illustrated in Fig. 2, PAD is mainly composed of four components: a packet marking module, a mark parsing module, a probabilistic inference model, and an inference engine. The packet marking module resides in each sensor node and sporadically marks routine application packets passing by. At the sink side, the mark parsing module extracts and analyzes the marks carried by the received data packets. The network topology can thus be reconstructed and dynamically updated according to the analysis results. The mark parsing module also generates preliminary diagnosis information such as packets loss on certain links, route dynamics, and so on. The inference model builds a graph of dependencies among network elements based on the outputs from the parsing module. Using the inference model and observed negative and positive symptoms as inputs, the inference engine is able to yield a fault report, which reveals the root causes of exceptions by setting the posterior probabilities of each network component being problematic. The inference results are also taken as feedback to help improve and update the inference model.

IV. Packet Marking

Since a sensor network has a self-organized time-varying network structure, unlike the case in an enterprise network, no prior knowledge can be obtained for constructing the inference model. Also, as a WSN topology is highly dynamic, we need to acquire the network statuses continually to maintain the topology in real time. To address the above requirements, we design a packet marking algorithm in PAD, which dynamically captures the network topology and extracts the inner dependencies among network components. Before the analysis results are directed to the inference engine for further reasoning, we can generate a preliminary diagnosis report on some basic network exceptions.

The main operation of this marking algorithm is to let sensor nodes stamp their IDs on passing data packets. Due to the size

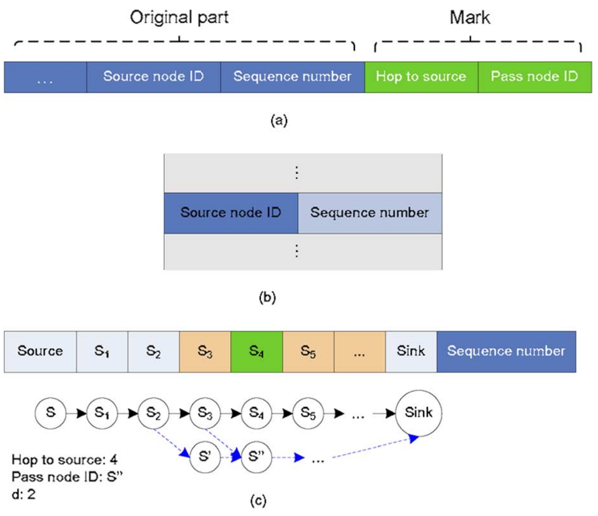


Fig. 3. The data structures for packet marking scheme. (a) A marked data packet. (b) Cache in sensor node. (c) *Path* updating.

limitations of the data packets used in sensor networks, however, the marking scheme only adds 2 bytes to each data packet that records one node ID. During the packet delivery, each packet is marked by only one selected sensor node based on a set of rules. At the sink side, the mark parsing module traces back the paths from each source node through analyzing sporadically marked packets. Through assembling the paths from different source nodes, the network topology can be reconstructed along with the regular data delivery of the system. If the network remains static, the packet marking process automatically converges and stops after the entire network topology is constructed. When network conditions vary, such as when packet loss or route changes occur, the packet marking process restarts somewhere close to the exceptional event. A strength of this design is that it does not inject any extra message into the network and strictly limits the overhead of marks attached to each data packet.

A. *Marking Scheme on Sensor Nodes*

Fig. 3(a) depicts an example of marked data packet. We assume that each original data packet contains: 1) a source node ID denoting the source node of this packet; and 2) a sequence number identifying the packet. If there is no such information recorded in the application, the marking scheme adds them to the packets. The mark added to the original packet consists of a pass node ID field that records the ID of a sensor that participates in delivering this packet and a hop to source field recording the number of hops from the source node to the marking node. When the source node issues a new data packet, it leaves the pass node ID field empty and sets the hop to source field to 0.

Every intermediate node maintains a cache for its downstream source nodes. As illustrated in Fig. 3(b), each cache entry consists of a source node ID and the sequence number of the recently received packet from the source. We call two sequence numbers of a source *continuous* if the first sequence number is larger than the latter one by 1.

As shown in Algorithm 1, upon receiving a packet, an intermediate node first checks whether the packet has been marked. If yes (the pass node ID is not empty), it forwards the packet with no further operations. Otherwise, the node checks its own cache. If there is no entry for the source node ID of this packet, it marks the packet by filling the pass node ID field with its own ID. It also creates a new entry for this source node in its cache and records the sequence number for the packet. If there exists an entry in the cache for the source node and the sequence number in the packet is *continuous* with the cache entry, the intermediate node updates the cache entry with the new sequence number. To prevent duplicate marking, the intermediate node does not fill the pass node ID field, instead it increments the hop to source field in the packet by 1 and forwards the packet. If the sequence number of the packet is not *continuous* with that recorded in the cache entry, it might be due to the packet loss or routing dynamics. The intermediate node marks the packet by filling the packet pass node ID field with its own ID. The node then updates its cache entry with the new sequence number of this packet and forwards it. The sink also participates in the marking process and creates a table recording source nodes and their packet sequence numbers. Using this marking scheme, the received packet in the sink records the ID of one intermediate node in the routing path together with its hop distance to the source node. We avoid duplicate marks of the same node on the same path to save communication costs. We can further reduce the memory usage in each sensor node by organizing its cache table into bloom filters. Each intermediate node inserts and extracts the source node information on the bloom filter. The error rate introduced by the bloom filter introduces negligible adverse impact in the lossy by-nature sensor network.

Algorithm 1 Packet_Marking (packet p)

```
 1: if $p$ has been marked
 2:   return;
 3: else
 4:   check cache;
 5:   if no entry for source node of $p$
 6:     mark $p$;
 7:     create entry with *source node ID* and *sequence number* in $p$;
 8:   else if entry exists and *sequence numbers* are *continuous*
 9:     update entry with new *sequence number*;
10:     increase *hop to source* in $p$ by 1;
11:   else if entry exists and *sequence numbers* are not *continuous*
12:     mark $p$;
13:     update entry with new *sequence number*;
14:   end if
15: end if
16: return;
```

B. *Parsing the Marks*

At the sink, the mark parsing module extracts and parses the marks piggybacked from the received packets. For each source node, we keep a data structure denoted as *path* to record node IDs along the path from the source node to the sink. As shown in Fig. 3(c), a *path* contains an array of slots and each slot records a node ID along the routing path hop by hop. The *path* also has a field that records the sequence number of the latest arrived packet from each source.

On receiving a new packet, the mark parsing module checks the existence of a *path* structure associated with its source node. If there is no such *path*, it means it is the first time the sink has received packets from that source. The sink creates a new *path* for the source node and records the source node ID at the first slot. The mark parsing module then examines whether the packet has been marked (the pass node ID field has been filled). If it has been marked, the sink updates the associated slot in the *path* to be the recorded node ID according to the hop to source field in the packet.

For the packets from the recorded *path*, the parsing module operates according to the recorded sequence number. We denote d as the difference between the sequence number of the received packet and the sequence number recorded in the *path*. If the sequence number of the new packet is equal to or less than that recorded in the *path* $(d < 1)$, it means that this is a duplicate or delayed packet. As information in the duplicate and delayed packets is usually outdated and may lead to errors in the mark parsing process, we ignore marks in such packets and do not update sequence number or other slots for the *path*. As a matter of fact, according to our deployment experiences in an operational sensor network, with a relatively long sampling interval, this kind of situation $(d < 1)$ is rare. If $d >= 1$, the sequence number recorded in the *path* is updated by the newly received packet, and then other slots of the *path* are accordingly updated by parsing the mark as Algorithm 2. Normally, without packet loss, $d = 1$, and we directly add the marked node ID into the *path*. Discontinuousness of the sequence numbers $(d > 1)$ indicates that the packet loss occurs, which triggers a preliminary diagnosis report on packet loss. Besides, the number of packet losses is quantified as $d - 1$. A mismatch of the recorded pass node ID in the packet and the recorded node ID in corresponding slot in the *path* indicates a route alternation happening at the position between the hop to source recorded in the packet and its $d-1$ hops upward. If not so, the marking should have been taken earlier. The parsing algorithm then generates a preliminary report of a route switch. In such a case, the slots in recorded *path* ranging from $d-1$ hops before the hop to source position to the sink become inaccurate, so we clear all those slots.

Let us look at the example in Fig. 3(c), where a new mark is received. The pass node ID is S'', four hops away from the source. The mismatch between S'' and current node S_4 in the same position of this path indicates a route variation. Now, the issue is how to determine where the route variation occurs. If there is no packet loss, it must be node S_3 that changes its route from S_4 to S''. In this case d equals 2, indicating that one packet has just been lost. The situation can be more complicated indeed. As illustrated in Fig. 3(c), the route variation can happen at S_2; for example, S_2 changes its parent node from S_3 to S', and S' then marks the consequent packet. The packet, however, gets lost on its way to the sink, so before the next packet marked by S'' arrives, the sink cannot be aware of the route variation. Another possible case is that the route switch happens at S_3, but S_3 fails to send the consequent packet to S'', and then S'' has to mark the second packet. As the route variation happens, slots ranging from $S_3(path[hopToSource - d + 1])$ to sink are suspicious. We update S_4 to S'' and clear other slots, expecting further information. The reception of the packet without any marks triggers a preliminary report of a successful delivery. The mark parsing function is presented in Algorithm 2.

The mark parsing module constructs and updates the network topology with the recorded *path*s. Once a new packet is received, the *path* associated to its source node is updated. This indicates that all links along the current path have just participated in the transmission of a packet. For each link in the network topology, we keep a counter to count the number of transmissions experienced by this link. Such information facilitates the construction of the inference model as it tells the strength of the dependency between the parent and its successive nodes. Since links in sensor networks are usually shared by multiple paths, we do not need to collect complete path information for all paths before revealing the entire network topology. Indeed, this scheme captures the network topology with a small number of packet receptions, as demonstrated in our field experiment.

One potential issue is that when the sink fails to learn the information of some path segments and the network topology is stable, few marks are received. As a result, it will take really a long time for the sink to learn the missing path segments. Such a drawback, however, is alleviated due to the sharing feature of network links, i.e., the missing links can be recovered from other paths that share them. Such a feature definitely alleviates, but does not completely avoid, this problem. To actively eliminate such a problem, in our implementation we let the intermediate nodes periodically clear their local caches. With this operation, new marks are inserted to packets, and the path information at sink can be periodically refreshed even when the network topology is static.

Algorithm 2 Mark Parsing(packet p)

```
1: if $p.sourceNodeID$ has no associated $path$
2:   create new $path$ for $p.sourceNodeID$;
3: end if
4: $d = p.sequenceNumber - path.sequenceNumber$;
5: if $d < 1$          //duplicate packet
6:   return;
7: else
8:   $path.sequenceNumber = p.sequenceNumber$;
9: end if
10: if $d == 1$          //no packet loss
11:   if $path[hopToSource] != p.passNodeID$     //route switch
12:     $path[hopToSource] = p.passNodeID$;
13:     clear all slots in $path$ after $path[hopToSource]$;
14:     generates route switch report;
15:   end if
16: else if $d > 1$          //packet loss detected
17:   generate packet loss report;
18:   if $path[hopToSource] != p.passNodeID$     //route switch
19:     clear all slots in $path$ after $path[hopToSource - d + 1]$;
20:     $path[hopToSource] = p.passNodeID$;
21:   end if
22: end if
```

Clearly, in this design we propose to mark simple messages only, but if we insert more marks into the data packets, we obtain richer information on the network statuses and make the diagnosis process more straightforward. Nevertheless, in resource-constrained sensor networks, we have to minimize the communication overhead introduced by our diagnosis model. Therefore, we choose to only use simplified marks to additively reconstruct the network. We give details about this issue in later discussions. Compared to existing approaches, our approach with quick reactivity and fast convergence is thus more suitable for highly dynamic environments.

C. Preliminary Diagnosis Reports

Before the final diagnosis results are obtained from the inference engine, some preliminary diagnosis reports can be yielded from the mark parsing module, which help to analyze the network statuses. The preliminary diagnosis briefly infers the following reports.

1) Success delivery report. When the sink receives a packet without any mark, it indicates a successful delivery along the current path. This report tells us that the route from the source sensor node to the sink is still the same and all links along this path have just conducted a successful transmission that confirms the active state of those links.

2) Packet loss report. As described above, if the difference d between the sequence number recorded in the *path* and the sequence number of the packet is more than one, it can be inferred that the packet loss occurs. The number of packet loss is quantified as $d-1$. In this case, according to our marking scheme, the packet must have been marked by some intermediate node. This report can further locate the packet loss location if there is no route switch accompanying the packet loss.

3) Route Switch Report. The mismatch of the pass node ID in the packet and the recorded ID in the corresponding slot in the *path* indicates that the previous routing path has been altered. The position of the switch is between the hop to source recorded in the packet and its $d-1$ hop upward.

V. Probabilistic Inference

The packet mark parsing module provides a coarse abstraction and incomplete report. At the sink, the successive probabilistic inference helps to reveal the inner dependencies among different network elements in the sensor network and expose the hidden root causes of the exterior symptoms. Network elements are inner correlated, for example, the crash of an upstream node causes all its children to disconnect from the sink. In contrast, simultaneous congestion of multiple paths may indicate a high probability of a malfunction at a common link. Based on such observations, we explore the dependencies among network elements (link status, sensing function, path status, etc.) on the constructed communication topology and encode them with a probabilistic model. Exterior symptoms like delay or loss of data samples are considered as inputs. When specific symptoms are observed by our inference algorithm, we can deduce the probability of the failures of each network element and find the most probable root causes in real time.

Most existing inference schemes for static enterprise networks use the simplified bipartite graph or tree-based inference model. As the network topologies in sensor networks are highly dynamic and no prior knowledge can be acquired in advance, it is difficult to apply the models for static networks in sensor networks. Instead, we apply a hierarchical inference model to capture the inner dependencies in sensor networks. The hierarchical model is good for encoding indirect dependencies with its hierarchical structure and can be constructed without complete information. Also, being assembled by many subparts, it can easily handle the network dynamics efficiently by updating the changed parts only. We first apply the Belief Network [21] as our inference model. Belief Network is a well-known probabilistic model that has been widely used in research domains like artificial intelligence and system engineering. In Belief Network, each possible root cause or symptom is represented by a variable. Each variable might have multiple values (e.g., 1 for a link in active state and 0 for in trouble). Causal relationships between different variables are denoted as directional arcs. Inferences can be conducted on this model to deduce the probability of particular values to our interested variables once the values of some other variables have been observed (e.g., symptoms like the high delay of data samplings). To further speedup the process, we propose a simplified inference model, Causality Diagram. According to the characteristics of sensor networks, we can design a simplified Causality Diagram that accurately approximates the inference results and reduces the overhead.

A. Belief Network

A Belief Network (or Bayesian Network) is a directed acyclic graph (DAG) that represents a set of variables and their probabilistic relationships. Each vertex in the graph denotes a random variable. In the rest of this paper, we use "vertex" and "variable" interchangeably. A directional arc from vertex X_1 to X_2 indicates a causal relation between the two variables in which the variable associated with the starting vertex X_1 acts as the cause and the variable of X_2 is the effect. The cause X_1 is called a parent of the outcome X_2. The strength of the relation between a parent and its child is defined by the conditional probabilities. We then formulate a Belief Network as a binary (G, P), where $G = \{V, E\}$ is a DAG and $P = \{P_i\}$ specifies a conditional probability distribution (CPD) in G. Here, $V = \{V_i\}$ represents the set of vertices in G, and $E = \{E_j\}$ denotes all arcs (or edges). P_i specifies the conditional probability distribution of each variable given its parents. When the value domain of variable is discrete, the CPD can be represented as a conditional probability table (CPT).

Given certain evidence (values of some variables), the Belief Network can answer three major types of queries [21]: 1) posterior probability assessment; 2) maximum posterior hypothesis; and 3) most probable explanation. The first type of query, which estimates posterior probabilities of certain variables given some evidence variables, best fits our requirements in this work.

B. Inferring Through Belief Network

Our inference model automatically constructs and maintains a Belief Network from the output of the mark parsing module. The inference engine accordingly infers from this model hidden statuses of the network. In our PAD approach, the Belief Network structure is assembled from the current network topology obtained from the mark parsing module.

1) Constructing a Belief Network: Fig. 4(a) depicts a simple example topology composed of a sink and three sensor nodes. The directional edge between two nodes denotes a wireless link and the direction of data transmitting along the link. There are five types of variables in our Belief Network, each of which has the value domain of $\{Up, Down\}$ that denotes a normal or abnormal working status, respectively.

For each source node, we add a variable D_i to the Belief Network, which denotes the status of the data reception of the

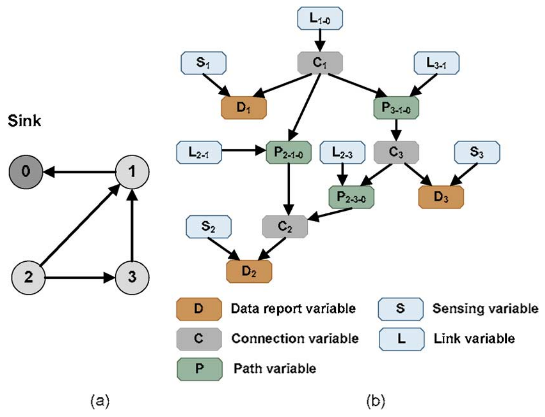
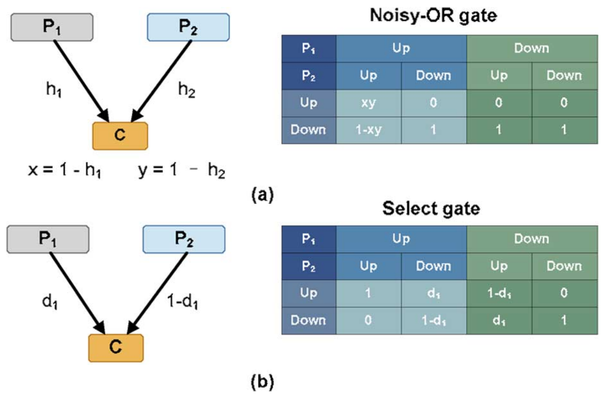


Fig. 4. Belief Network constructed from the communication topology. (a) Network topology. (b) Belief Network.



Noisy-OR gate

P_1	Up		Down	
P_2	Up	Down	Up	Down
Up	xy	0	0	0
Down	1-xy	1	1	1

Select gate

P_1	Up		Down	
P_2	Up	Down	Up	Down
Up	1	d_1	$1-d_1$	0
Down	0	$1-d_1$	d_1	1

Fig. 5. CPTs of (a) *noisy-OR* and (b) *Select* gates.

source node. For example, if the sink observes a long time delay in the data reception from a source, the corresponding D_i variable of this node will be set to *Down*. Note that, in many applications, some sensor nodes do not sample data, but only relay messages for other nodes. Some of the nodes simply relay packets for other sensors, so there are no data reception variables for those nodes. The status of the data report variable D_i depends on two parent variables, the sensing variable S_i and the connection variable C_i. The sensing variable *Si* indicates the sensing function of the corresponding source node, and the connection variable C_i describes the condition of the network connectivity from the source node to the sink. We add two arcs from S_i and C_i to D_i to represent the dependencies between them. S_i and C_i are thus called the parent variables of D_i in the Belief Network. Both the sensing functionality and the network connectivity condition will affect the success of the data reported from the source node.

The connectivity from a source node to the sink relies on one or more paths connecting them. For example, node 2 in Fig. 4(a) can choose to deliver packets through two parents, node 1 and node 3, so in the corresponding Belief Network, the connection variable C_2 has two parent variables P_{2-1-0} and P_{2-3-0}. They are called path variables. The subscript of each path variable sequentially denotes the ID of the start node on the path, the ID of the next hop node from the start node, and the ID of the end node on the path. As illustrated in Fig. 4(b), the status of each path variable depends on two parent variables. One is the link variable on the first hop from the start node, and the other is the connection variable of its parent node. The link variable L_{m-n} represents the communication conditions of a wireless link between two nodes m and n.

We connect each pair of variables that has a dependency with a directional arc from the cause variable to the outcome variable. Eventually, we obtain a hierarchical network composed of these five types of variables in which dependencies among network elements are encoded. Among the five types of variables, the statuses of the link and sensing variables are hidden from the exterior observations that most need to be inferred. The path and connection variables are intermediate variables that are usually combinational results of other parent variables. The data report variables are outputs of the mark parsing module that we directly observe at the sink. The Belief Network structure consisting of the variables is automatically maintained and updated when network topology and communication conditions vary over time.

2) Inference on Belief Network: Once the Belief Network structure is constructed, a critical issue is how to assign CPTs for variables that specify the conditional probabilities between parents and their children. Different logistic relations between parents and their children lead to different methods for calculating the CPT. For example, the sensing variable and connection variable affect their children variable of data report in a logical *OR* manner, i.e., if one of them is in the *Down* state, the data report variable should be switched onto the *Down* state. Due to the diverse routing schemes and high dynamics in sensor networks, a sensor may maintain multiple parents for relaying its data. Consequently, in our inference model, multiple path variables affect the same connection variable in *Select* mode where the status of selected paths will determine the status of the connection variable. In PAD, we employ the *noisy-OR* gate [21] and *Select* gate [5] to encode these operations.

Fig. 5(a) shows the CPT in a *noisy-OR* gate, where any one of the parent variables in *Down* status results in the *Down* status of the child variable. In Fig. 5, h_1 and h_2 represent the noisy property that means even if both parent variables are in the *Up* status, the child variable still has a chance to fail (in *Down* status). In PAD, *noisy-OR* gates exist in several cases such as when the sensing and connection variables affect the data report variables, the link and connection variables affect the path variables, and so on. The relation between multiple path variables and a connection variable is represented by the *Select* gate as illustrated in Fig. 5(b). Here, d denotes the dependency strength of each parent, and in the case of Fig. 5(b), d is the probability that the child connection variable selects a certain path to relay data. Thus, the probability that a connection variable is in the *Up* status is given by

$$\Pr(\text{child} = Up \,|\, P_1, P_2, \ldots P_i \ldots P_n) = \sum_{p_i = Up} d_i.$$

In the Belief Network, each *noisy-OR* gate connects two parent variables to a child variable, so the CPT calculation is quick. The *Select* gate might connect more parent variables to a child variable, but the maximum number of parent variables for one gate is bounded by the number of neighbors for a sensor node. The number of neighbors is normally treated as a constant. Hence, the CPT calculation for *Select* gate is also

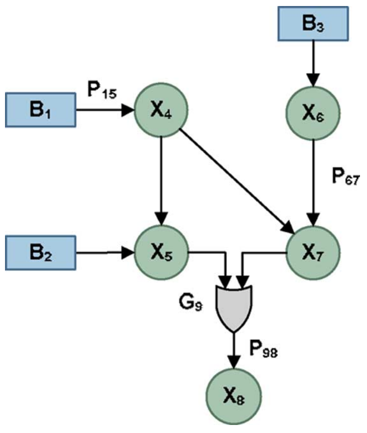


Fig. 6. The Causality Diagram.

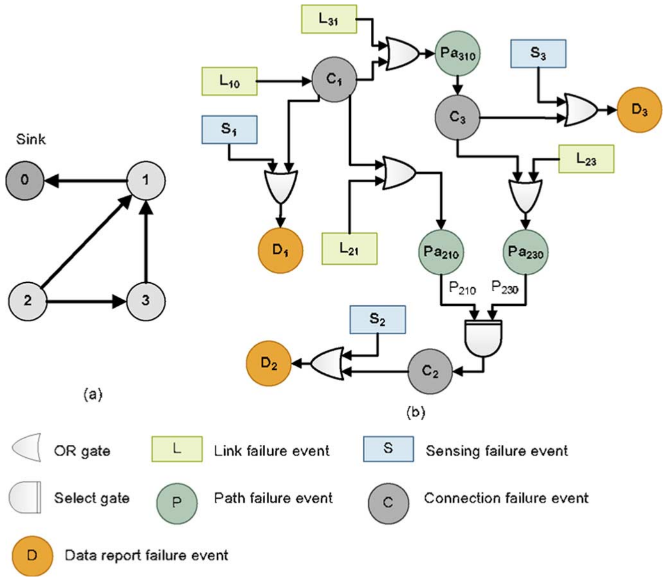


Fig. 7. The Causality Diagram constructed from the communication graph. (a) Network topology. (b) Causality Diagram.

efficient. In the initial stages, the prior fault probability distribution of the link and sensing variables are assigned according to experience data. The value of each d_i is assigned by estimating the percentage of transmissions delivered through each path in a connection. Such information is input from the mark parsing module.

The outputs of the inference process are the status estimations about the link and sensing variables. Such estimations reveal deeper understanding of the network operation. For example, a single link failure might be caused by environmental interference to the wireless communications, and multiple weak links relating to one sensor node might indicate a faulty node. We have more discussions in Section VI on how we detect the network faults from the output of our inference process.

C. Fast Inference Scheme

The Belief Network model is a widely used tool in dealing with inference tasks that achieve high performance even with incomplete or suspicious inputs. The inference process in a general Belief Network, however, is NP-Hard [8], and even some approximate approaches have been proven to be NP-hard [9]. While previous studies in comparatively stable enterprise networks are able to simplify [24], [25] the Belief Network into bipartite graphs or polytrees, the hierarchical multilevel characteristic of sensor networks makes it impractical. To speed up the inference for large-scale sensor networks, we further propose a new inference model based on the Causality Diagram [32].

Similar to Belief Network, Causality Diagram is a graphic inference model. Instead of conditional probability, Causality Diagram uses dependency strength to represent the relationships between vertices and exploit logistic computation in the probabilistic inference process.

As shown in Fig. 6, a Causality Diagram is a directed graph consisting of four types of elements including basic events, intermediate events, arc events, and logic gates. Each vertex or arc in a Causality Diagram denotes an event. Rectangles like B_1 denote the basic events that are independent causes of other events. Circular vertices like X_5 represent intermediate events that can be outcomes or causes of other events. An arc connecting two vertices is called an arc event that specifies a causal relation between the two events. The associated strength on an arc denotes the probability that the parent event affects its child event. Note that if there is no additional parameter on an arc, it means that the parent event has an impact on the child event at a probability of 1. The logic gates like G_5 specify how multiple parent events jointly influence one child event.

Taking the same example network topology in Fig. 4, Fig. 7 shows how to construct a Causality Diagram for our inference engine. Different from that in Belief Network, each vertex in a Causality Diagram denotes a fault event. Those vertices without parents (rectangular in shape) are basic events that are independent root causes. Other circular vertices denote intermediate events.

The traditional inference algorithm is NP-hard [32] on general Causality Diagrams and is thus infeasible for our approach. Nevertheless, in this design, due to the characteristics of WSNs, we are able to use the specifically defined *OR* and *Select* gates to model the dependency relationships between node behaviors. In our model, the *OR* gate represents the causal relationship where the occurrence of any of the parent events will cause the child event. *Select* gate describes the relationship that the child event is affected by one of its parent event according to a certain probability distribution. This enables us to apply a fast inference scheme, leveraging the particular structure of our model. Our scheme contains four stages:

1) We represent each intermediate event by its first-order cut set (CS_1) expression.

2) We adopt an early disjointing mechanism. Before generating the final cut sets (CS_f) expressions, we directly disjoint the CS_1 expressions. Based on the definition of the *Select* gate, the cut sets in a CS_1 expression of the connection failure events are already exclusive.

3) We calculate final disjoint cut sets (DCS_f) expressions by iteratively replacing intermediate events in each expression. Since all negative events generated from step 2 are basic events, we avoid the complex *NOT* operations and the replacement process can be operated efficiently.

4) We estimate the posterior probabilities of user specified events. Given observed events E (evidences), we can calculate

the posterior probability of interested events H (root causes). $E = E_1 \cap E_2 \cap \ldots \cap E_k$. According to the Bayesian formula

$$\Pr(H \mid E) = \frac{\Pr(HE)}{\Pr(E)} = \frac{\Pr(H \cap E)}{\Pr(E)} = \frac{\Pr(H \cap E_1 \cap E_2 \cap \cdots \cap E_k)}{\Pr(E_1 \cap E_2 \cap \cdots \cap E_k)}.$$

Both H and E_i have been expressed as DCS_f, and the result of operating logic *AND* on two DCS_f expressions is still a DCS_f expression. Hence, expressions on numerator and denominator are both DCS_f, and the posterior probability of H can be calculated algebraically.

D. Characterizing the Faults

After the inference process, both the packet mark parsing module and the inference engine output the fault reports about the sensor network statuses. In this section, we discuss how PAD further characterizes the faults in the network through analysis of the fault reports. In PAD, we trace the fault reasons by characterizing their fault patterns as follows.

1) Physical damage. In many field applications, physical damage might occur and destroy a portion of or the entire hardware of sensor nodes.

2) Software crashes. Software crashes include local problems on the sensor node such as a send queue overflow or busy CPU in those nodes that are physically intact. PAD detects the sensor nodes in a software crash by both the diagnosis information from the mark parsing module and the posterior probability estimations from the inference engine.

3) Network congestion. Network congestion relates to a group of sensors or traffic flows. The occurrence of network congestion usually leads to a high packet loss rate within the influenced region. Due to such a feature of this type of faults, the observed symptoms are usually temporal and distributed across a large time and space span.

4) Environmental interferences. Environmental interference can significantly degrade the performance of WSNs even without any internal problems within the WSN itself. The environment interference usually has high spatial correlation.

5) Application flaws. As the application programs might contain flawed components, the network might suffer from some inefficiency that does not lead to system crashes but consumes computational or communicational resources.

VI. Evaluation

We conduct comprehensive simulations and implement field experiments to evaluate the performance of PAD. For the implementation, we used the BNJ implementation of the Belief Network inference as part of our inference engine. We implemented the packet marking scheme for TelosB motes on the TinyOS platform with nesC language. We implemented the mark parsing module on the java based back end.

A. Simulations

We first examine the effectiveness and efficiency of PAD through simulations. We simulate a sensor network in which sensor nodes are deployed on a two-dimensional space, with the sink located at the center. Sensors periodically generate data and deliver to the sink through multihop routes. Two routing schemes are applied in the simulation. Various types of faults are inserted into links or nodes according to different test settings. We use a cutoff threshold to detect the failures. In the following tests, if the output posterior probability of a certain network element (for example a link) to be faulty is higher than 50%, we will regard this element as failure.

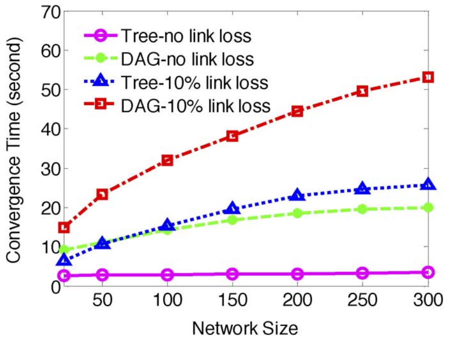


Fig. 8. Convergence time on varying network sizes.

We apply two metrics for estimating the performances of inference models, the detection ratio and false positive ratio. Detection ratio is the ratio between the number of faults founded and the number of all faults. False positive ratio is the ratio between the real failures detected and all failure reports generated by our diagnosis system.

1) The Efficiency of the Packet Marking Scheme: We evaluate the convergence time of the packet marking scheme under various network conditions. In these tests, we simulate a data acquisition network using both spanning tree-based routing and DAG-based routing schemes. Each source node samples environment data and generates a new packet every 2 s. Different routing schemes lead to different types of topologies. The notation *Tree* denotes a spanning tree topology rooted at the sink, and the notation *DAG* denotes a multipath routing strategy where each sensor node has multiple parents. Besides the routing topologies, the link loss rate also impacts the topology reconstruction. Thus, we evaluate the performance of our approach with different link loss rates. Here, no link loss indicates that all packet transmissions are guaranteed to deliver, and 10% link loss means that each link has 10% packet loss rate. Under the latter setting (10% link loss), it is difficult for a source node far away from the sink to deliver its packets to the sink since a packet has high probability to get lost on a long path.

In the first test, we measure the convergence time of reconstructing the entire network topology. The results are shown in Fig. 8. Without packet loss, the topology reconstruction can be accomplished very quickly. According to the results in Fig. 8, the complete topology of a sensor network with up to 300 nodes can be rebuilt within 20 s, which is very efficient. The *DAG* network has a more complicated topology than *Tree*, so the marking scheme requires more time to figure it out. When there is link loss, the convergence time of both topologies increases since many packets together with the marks get lost on their way to the sink. As the network size increases, the growing speed of convergence time is less than the linearity, indicating a good scalability of this approach. In the second test, we assume that the network topology has already been constructed at sink. Then,

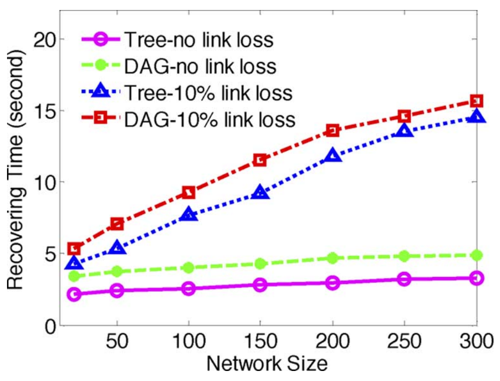


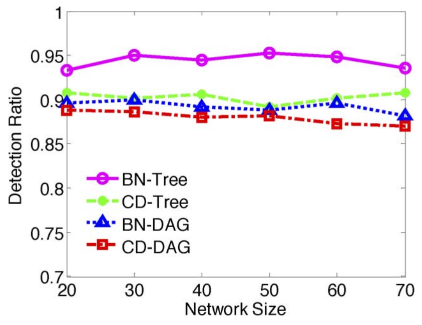
Fig. 9. Recovering time on varying network sizes.

we manually inject route dynamics into the network and measure the average time needed for capturing these alternations. From the results shown in Fig. 9, we find similar trends as that in Fig. 8. The recovering time increases when there is packet loss. Besides, packet loss can occur and lead to a mark at some hop before the route alternation, which delays the operation for recording the route alternation. Relatively more time is required for detecting topology alternations on the more complicated *DAG* network than the *Tree* network. The growing speeds of all curves are less than linearity, and our scheme captures the dynamics in a large network with up to 300 nodes in around 15 s. According to the above results, our approach has a fast convergence speed and is able to capture the network dynamics quickly. Our method is also shown to be scalable with varied network size.

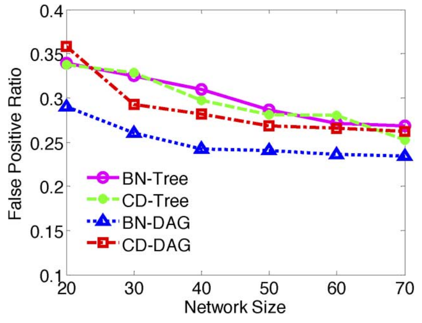
2) The Performance of Inference Models: We then evaluate the performance of the two inference models with four different groups of tests. We inject artificially created errors into the network and let both inference models generate fault reports according to the posterior probability estimations.

We first inject sensing failures into sensor nodes and compare the detection ratio and false positive ratio of both models. We randomly invalidate the sensing capabilities of 10% of the nodes. *BN-Tree* and *BN-DAG* denote inference results of the Belief Network model on the spanning tree topology and *DAG* topology. *CD-Tree* and *CD-DAG* represent the inference results of Causality Diagram model. We vary the network size from 20 to 70 nodes. Fig. 10(a) plots the detection ratios, where we can see both models achieve detection ratios higher than 85% in most situations. Belief Network model has a slightly higher detection ratio than Causality Diagram model as it adopts exactly accurate inference. Fig. 10(b) shows the false positive ratio of the two models. We see that for both models, the false positive ratio decreases as the network size increases. By analyzing the false reports, we find that most false positive reports relate to the leaf nodes. As those nodes lie on the boundary of the network field and do not relay data for others, if they do not report data to sink, there are few clues as to whether it is due to a sensing failure or a communication failure. As the network size increases, the percentage of boundary nodes decreases, so the false positive ratio becomes lower.

We then inject node failure of both sensing and communication errors into sensor nodes; see Fig. 11 for the results. As the network size increases, the detection ratio decreases and the false positive ratio increases at the beginning (when the number

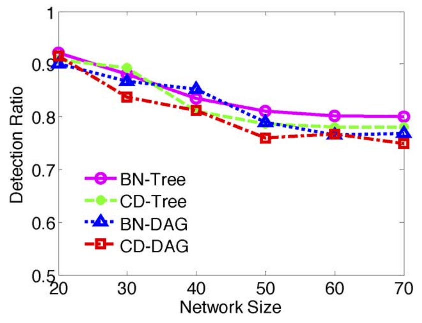


(a)

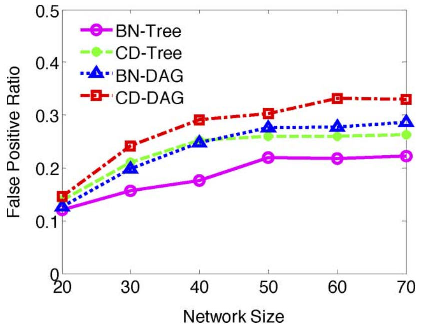


(b)

Fig. 10. (a) Sensing failure detection ratio. (b) Sensing failure false positive ratio.



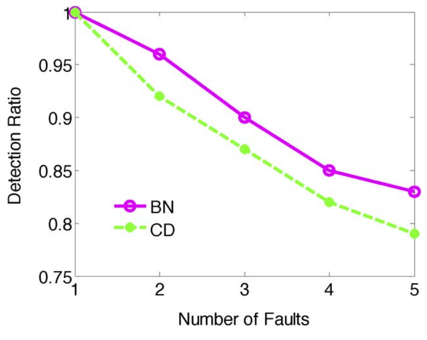
(a)



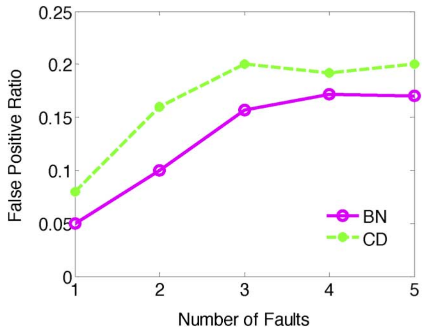
(b)

Fig. 11. (a) Node failure detection ratio. (b) Node failure false positive ratio.

of nodes is around 50). Then, after the network size grows beyond 50, the detection ratio and false positive ratio become relatively stable with the network size. The performance degradation at the beginning part of the test is mainly because the system performance of extremely small networks is much better than that of general settings. For example, when the network contains only several nodes, the detection ratio can be as high as more than 90% and false positive ratio can be very low. Such



(a)



(b)

Fig. 12. (a) Detection ratio for multiple faults. (b) False positive ratio for multiple faults.

a result is due to the extremely simple structure, which is easy to perform our inference. When the network is relatively large, the performance of our system degrades but still maintains at the high end (with more than 75% detection ratio and around 30% false positive). The system performance remains stable as the network size further increases. According to the results, our system maintains a high performance with increased network size.

In previous tests, the number of faults in the network is fixed to a constant percentage. In this test, we vary the number of faulty nodes to study the effect of simultaneous faults on the inference performances. In this test, we simulate a network with 25 sensor nodes. The number of simultaneous faults varies from 1 to 5. The experiment is conducted with the spanning tree-based routing scheme. The results are shown in Fig. 12. We find that when more faults simultaneously occur, the detection ratio decreases and the false positive ratio increases. This is because the existence of multiple errors introduces mutual interference in the inference model and degrades the performance. As Fig. 12 shows, a single error is easy to detect, and five simultaneous errors lead to worse results.

The last group of simulations compares the computational efficiency of the two inference models. As shown in Fig. 13, the computation overhead of Belief Network is much larger than Causality Diagram, and their difference increases quickly as the network size increases. Although as previous simulation shows, Causality Diagram model provides approximated inference with less accuracy than the Belief Network model, and it largely reduces the computational overhead and is thus more viable for practical usage.

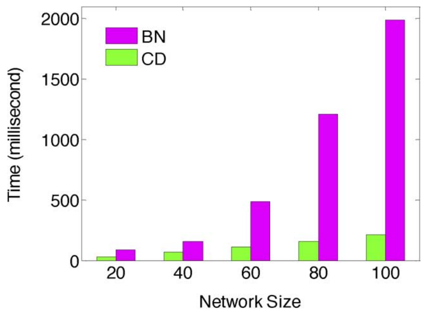
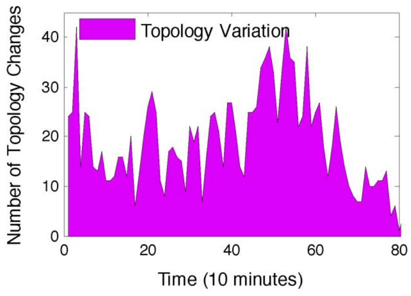
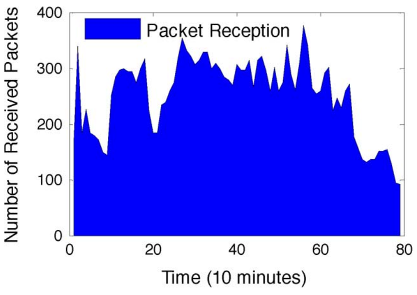


Fig. 13. Computation time of two inference models.



(a)



(b)

Fig. 14. (a) Topology variation statistics over time. (b) Packets reception statistics over time.

B. Implementations and Field Experiments

We implement and test the effectiveness of the PAD approach through a field study in our sea monitoring sensor network system [30]. The experiment is conducted over a long period of more than three months, and in this section, we fetch and analyze a five-day data trace that consists of a segment of 22 416 received packets as well as the marks in them.

1) Observations in the Field Study: The analysis results from PAD confirm our concern about the energy efficiency of the system. We indeed observe extraordinarily high frequency of topology variations in the sensor network. Fig. 14 compares the topology variations with the packet receptions during the sea monitoring system operation. In order to measure the speed of network topology changes, we count the number of topology variations in every 10-minute interval. In our test, the topology variations include situations where new links are added to the network or existing links are invalidated. From results in Fig. 14(a), we find that every 10 min there are topology variations of 10 to more than 40 times that in the network. As shown in Fig. 14(b), however, there is no apparent correspondence between the topology variations and the packet receptions. Thus, most of the topology variations occur but do not significantly

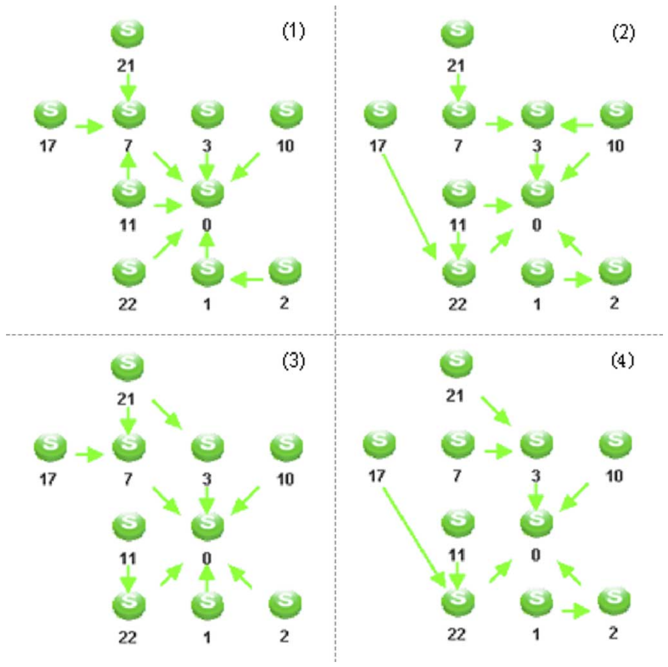


Fig. 15. Topology evolutions over time in field study.

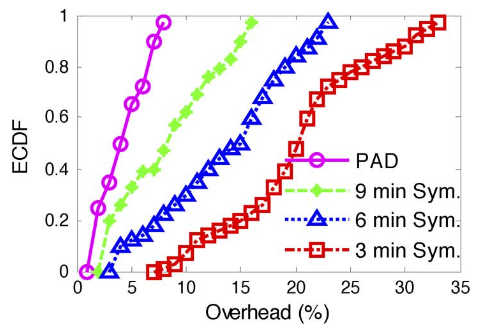


Fig. 16. System overhead (PAD versus Sympathy).

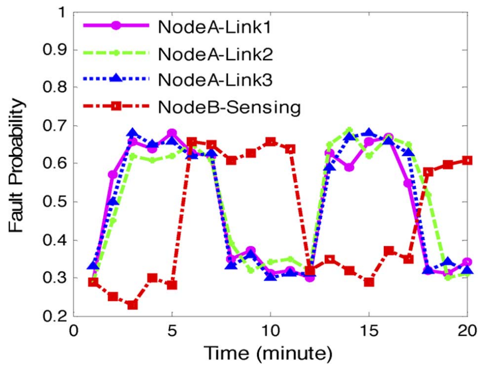


Fig. 17. Diagnosis results for detecting the manually injected faults in our field study.

improve the network communication quality. Fig. 15 exhibits a group of topology snapshots of a certain region in the network. The interval between each pair of consecutive subfigures is 2 min. According to the algorithm used in the MultiHopRouter component, the topology variations indeed always incur large traffic overhead in the network.

This observation confirms our concern that the network quickly depletes the node energy due to the frequent route switches, while most of them occur because of the instability of link quality between the floating sensor nodes. Clearly, frequent route switches may lead to high energy cost that largely constrains the lifetime of our monitoring system. We improve our application program by setting adequate redundancy in measuring the link quality and switching the routes. Currently, our system has been operating neatly with much fewer unnecessary routing dynamics.

2) Traffic Overhead: Through analyzing the received packets, we compare the extra overhead introduced by PAD and Sympathy. Sympathy aims to collect all necessary diagnosis information and determine the root causes with a role-based method. If all information can be sufficiently obtained in time, it achieves exact accuracy. On the other hand, frequent information collection leads to tremendous traffic burden on operational networks. We use empirical cumulative distribution functions (ECDFs) to quantify the overhead. In Fig. 16, the x-axis denotes the ratio of the diagnosis overhead to the total network traffic, and the y-axis denotes the ECDF. For example, a point with value (0.3, 0.8) on x- and y-axes respectively indicates the fact that 80% of the time, the diagnosing transmission dominates less than 30% of the total network traffic. A curve to the left represents a small cumulative overhead. Different curves for the Sympathy approach denote the cases of different report intervals in sending the diagnosis metrics. As Fig. 16 shows, PAD significantly outperforms Sympathy in terms of the traffic overhead. Due to the vast traffic overhead, Sympathy is more suitable for lab experiment-based debugging but not feasible for diagnosing an instant working system. If we use a long sample interval for collecting diagnosis information, Sympathy will have smaller overhead but definitely miss much network status information and cannot output instant diagnosis results. Instead, our passive diagnosis approach does not rely on the active information collection and can work even with incomplete information.

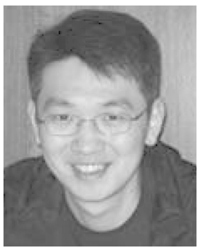
3) Diagnosis for the Problematic Network: In this experiment, we artificially inject sensing and communication faults into two sensor nodes respectively and mix them into the network. We let the two nodes interchangeably work under normal and exceptional statuses. We turn off the wireless radio of one sensor node (NodeA) every 5 min and invalidate the sensing module of the other node (NodeB) every 5 min. Each fault remains for 5 min. As shown in Fig. 17, the dash-dotted curve with square plots represents the inferred posterior fault probability of the sensing functionality in NodeB. The inference result accurately captures the periodical faults of the sensing module in NodeB. The three other curves denote the inferred fault probabilities of three links associated with NodeA, which indicate the faults in those links. We can see that PAD correctly captures the periodical communication failures of NodeA. According to Sympathy performance report, Sympathy is able to detect any failure injected into the network if the system parameters are properly set. By comparing the outputs of our online diagnosis tool and the log information from deployed sensors, we find that PAD achieves more than a 90% detection ratio and around 80% accuracy, but with significantly reduced overhead.

VII. Conclusion and Future Work

Although there have been many approaches proposed for debugging the operation of sensor network systems in a controlled laboratory, few works have been done toward an in-situ diagnosis tool for monitoring the statuses of operational systems in the field. In this paper, we propose PAD, a passive diagnosis approach that can be efficiently implemented and applied to a normally working sensor network system providing in-situ network diagnosis. The proposed lightweight packet marking scheme collects necessary hints without injecting extra traffic overhead to the original system. The probabilistic inference model residing at the sink captures the unique features of the sensor networks and yields accurate results. The inference engine works well even with incomplete or suspicious inputs in a nondeterministic manner. We implement our diagnosis approach and validate its effectiveness in a field test in our sea monitoring project. The sea monitoring project is an ongoing project. We are currently utilizing PAD as an important diagnosis tool to detect possible faulty components in the system and guarantee its correct operations. On the other hand, we are relying on such a platform to further test the effectiveness and efficiency of PAD and hope to improve it according to our future observations.

REFERENCES

[1] *HP Openview*, [Online]. Available: http://www.openview.hp.com
[2] *IBM Tivoli*, [Online]. Available: http://www.ibm.com/software/tivoli
[3] *Microsoft Operations Manager*, [Online]. Available: http://www.microsoft.com/mom
[4] *OceanSense: Sensor Network for Sea Monitoring*, [Online]. Available: http://www.cse.ust.hk/~liu/Ocean/index.html
[5] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *Proc. ACM SIGCOMM*, 2007, pp. 13–24.
[6] X. Bai, D. Xuan, Z. Yun, T. H. Lai, and W. Jia, "Complete optimal deployment patterns for full-coverage and k-connectivity $(k <= 6)$ wireless sensor networks," in *Proc. ACM MobiHoc*, 2008, pp. 401–410.
[7] J. Cao, L. Zhang, J. Yang, and S. K. Das, "A reliable mobile agent communication protocol," in *Proc. IEEE ICDCS*, 2004, pp. 468–475.
[8] G. F. Cooper, "Probabilistic inference using belief networks is NP-hard," Stanford Knowledge Systems Laboratory, Tech. Rep., 1987.
[9] P. Dagum and M. Luby, "Approximately probabilistic reasoning in Bayesian belief networks is NP-hard," *Artif. Intell.*, pp. 141–153, 1993.
[10] Q. Fang, J. Gao, and L. J. Guibas, "Locating and bypassing routing holes in sensor networks," in *Proc. IEEE INFOCOM*, 2004, vol. 4, pp. 2458–2468.
[11] R. K. Ganti, P. Jayachandran, H. Luo, and T. F. Abdelzaher, "Datalink streaming in wireless sensor networks," in *Proc. ACM SenSys*, 2006, pp. 209–222.
[12] B. Gedik, L. Liu, and P. Yu, "ASAP: An adaptive sampling approach to data collection in sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 12, pp. 1766–1783, Dec. 2007.
[13] S. Guo, Z. Zhong, and T. He, "FIND: Faulty node detection for wireless sensor networks," in *Proc. ACM SenSys*, 2009, pp. 253–266.
[14] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh, "Energy-efficient surveillance system using wireless sensor networks," in *Proc. ACM MobiSys*, 2004, pp. 270–283.
[15] S. Kandula, D. Katabi, and J.-P. Vasseur, "Shrink: A tool for failure diagnosis in IP networks," in *Proc. MineNet*, 2005, pp. 173–178.
[16] K. Klues, G. Hackmann, O. Chipara, and C. Lu, "A component-based architecture for power-efficient media access control in wireless sensor networks," in *Proc. ACM SenSys*, 2007, pp. 59–72.
[17] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, "IP fault localization via risk modeling," in *Proc. USENIX NSDI*, 2005, pp. 57–70.
[18] S. Lim, C. Yu, and C. R. Das, "Rcast: A randomized communication scheme for improving energy efficiency in MANETs," in *Proc. IEEE ICDCS*, 2005, pp. 123–132.
[19] H. Liu, P. Wan, C.-W. Yi, X. Jia, S. A. M. Makki, and N. Pissinou, "Maximal lifetime scheduling in sensor surveillance networks," in *Proc. IEEE INFOCOM*, 2005, vol. 4, pp. 2482–2491.
[20] Y. Liu, Q. Zhang, and L. Ni, "Opportunity-based topology control in wireless sensor networks," in *Proc. IEEE ICDCS*, 2008, pp. 421–428.
[21] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.
[22] S. Qingxi, W. Hongchun, and Z. Qin, "Intelligent fault diagnosis technique based on causality diagram," in *Proc. WCICA*, 2004, vol. 2, pp. 1751–1755.
[23] N. Ramanathan, K. Chang, L. Girod, R. Kapur, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," in *Proc. ACM SenSys*, 2005, pp. 255–267.
[24] M. Steinder and A. S. Sethi, "Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms," in *Proc. IEEE INFOCOM*, 2002, vol. 1, pp. 322–331.
[25] M. Steinder and A. S. Sethi, "Probabilistic fault localization in communication systems using belief networks," *IEEE/ACM Trans. Netw.*, vol. 12, no. 5, pp. 809–822, Oct. 2004.
[26] M. Varshney, D. Xu, M. Srivastava, and R. Bagrodia, "SenQ: A scalable simulation and emulation environment for sensor networks," in *Proc. IEEE/ACM IPSN*, 2007, pp. 196–205.
[27] J. Wu and S. Yang, "SMART: A scan-based movement-assisted sensor deployment method in wireless sensor networks," in *Proc. IEEE INFOCOM*, 2005, vol. 4, pp. 2313–1324.
[28] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *Proc. ACM SenSys*, 2004, pp. 13–24.
[29] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse, "Clairvoyant: A comprehensive source-level debugger for wireless sensor networks," in *Proc. ACM SenSys*, 2007, pp. 189–203.
[30] Z. Yang, M. Li, and Y. Liu, "Sea depth measurement with restricted floating sensors," in *Proc. IEEE RTSS*, 2007, pp. 469–478.
[31] H. Zhai and Y. Fang, "Impact of routing metrics on path capacity in multi-rate and multi-hop wireless ad hoc networks," in *Proc. IEEE ICNP*, 2006, pp. 86–95.
[32] Q. Zhang, "Probabilistic reasoning based on dynamic causality trees/diagrams," *Rel. Eng. Syst. Safety*, vol. 46, pp. 202–220, 1994.
[33] J. Zhao, R. Govindan, and D. Estrin, "Residual energy scan for monitoring sensor networks," in *Proc. IEEE WCNC*, 2002, vol. 1, pp. 356–362.



Yunhao Liu (SM'06) received the B.S. degree in automation from Tsinghua University, Beijing, China, in 1995, and the M.S. and Ph.D. degrees in computer science and engineering from Michigan State University, East Lansing, in 2003 and 2004, respectively.

He is now an Associate Professor and Post-Graduate Director with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong. His research interests include sensor network, P2P and Internet computing, and pervasive computing.

Dr. Liu has been a Member of the Association for Computing Machinery (ACM) since 2006.

Kebin Liu received the B.S. degree from the Department of Computer Science, Tongji University, Shanghai, China, and the M.S. degree in computer science from Shanghai Jiaotong University, Shanghai, China.

He is a joint Ph.D. student with the Department of Computer Science and Engineering, Shanghai Jiaotong University, and the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong, under the supervision of Dr. Yunhao Liu. His research interests include sensor networks and distributed systems.

Mo Li (M'06) received the B.S. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2004.

He is currently working toward the Ph.D. degree with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong. His research interests include sensor networking and pervasive computing.