Journal of Network and Computer Applications 36 (2013) 167–177

Contents lists available at SciVerse ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

# MDiag: Mobility-assisted diagnosis for wireless sensor networks

Junjie Xiong [a,b], Yangfan Zhou [a,b,*], Michael R. Lyu [a,b], Evan F.Y. Young [b]

[a] Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen, China
[b] Department of Computer Science & Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong

ARTICLE INFO

*Article history:*
Received 18 March 2012
Received in revised form
10 August 2012
Accepted 21 September 2012
Available online 11 October 2012

*Keywords:*
Wireless sensor networks
Mobile
Smartphone
Diagnosis

ABSTRACT

Though widely employed in various applications, wireless sensor networks (WSNs) are liable to failures, especially after deployment. Since the on-site failures are difficult to reproduce, it is of critical importance to perform in-situ diagnosis. Current in-situ diagnosis methods are either intrusive or inefficient, because they either inject diagnosis agents into each sensor node or build up another network for diagnosis purpose. To tackle these issues, we propose MDiag, a mobility-assisted diagnosis approach that employs smartphones to patrol the WSNs and diagnose failures. Diagnosing with a smartphone which is not a component of WSNs does not intrude the execution of the WSNs. Moreover, patrolling the smartphone in the WSNs to investigate failures is more efficient than deploying another diagnosis network. Statistical rules are designed to guide the detection of abnormal cases. Aiming at improving the patrol efficiency, a patrol approach MSEP (maximum snooping efficiency patrol) is proposed. MSEP is designed to achieve better performance than the naive method, the greedy method, and the baseline method in increasing the detection rate and reducing the patrol time of MDiag. Experiments with real sensor nodes and emulations validate the effectiveness of MDiag in detecting anomalous cases caused by faults.



## 1. Introduction

Wireless sensor networks (WSNs) have been widely employed in various applications, such as structure monitoring, traffic surveillance, environment monitoring (Xu et al., 2004; Akyildiz et al., 2002). A wireless sensor network is usually composed of a capable base station (BS) and many resource-limited sensor nodes that self-organize into a distributed system. Recently, many efforts have been devoted to improve the reliability of WSNs in various aspects, such as simulation (Levis et al., 2003), debugging (Yang et al., 2007), and most importantly in-situ network diagnosis that observes real-time network status and enables timely network failure detection (Ramanathan et al., 2005). WSNs are liable to failures, especially after the deployment (Langendoen et al., 2006). In addition, on-site failures are difficult to reproduce (Luo et al., 2006). Hence, it is critical to perform in-situ diagnosis.

One in-situ diagnosis approach is to implant diagnosis agents into each sensor node (Ramanathan et al., 2005). However, many already-deployed WSNs are not facilitated with diagnosis functions (Langendoen et al., 2006). We have to update the software of the whole WSN so as to embed the diagnosis agents. Nevertheless, programming the agent into each sensor node will inevitably affect the normal execution of a running WSN application (Heo et al., 2010; Hui and Culler, 2004). In addition, the diagnosis agent in each sensor node is usually required to transmit specified metrics back to a centralized station for diagnosis. In a sensor node with limited resources, such diagnosis affects the on-site application performance by consuming the bandwidth and the related resources (Ramanathan et al., 2005). Furthermore, as we know, the execution of logging statements could change the actual execution of a system and then may introduce bugs (Reynolds et al., 2006). It is very costly to insert agents at each protocol layer although this would result in high granularity diagnosis. How many agents to insert is a similar problem. The more the agents are embedded, the more logs the diagnosis can obtain, and the less efficient and more intrusive the diagnosis is.

To avoid the intrusion caused by agent injection, some approaches deploy another network (such as another WSN) to monitor a WSN, but this approach is neither efficient nor cost-effective. SNTS deploys several monitoring sensor nodes (they formulate a snooping network) in the real field along with the original sensor nodes (Khan et al., 2007). The monitoring sensor nodes sniff and record the packets sent from the monitored sensor nodes, which are manually retrieved to a PC for investigation. For a large WSN, SNTS needs to deploy many monitoring sensor nodes; as a result, SNTS does not adapt to large WSNs well.

* Corresponding author at: Shenzhen Research Institute, The Chinese University of Hong Kong, Hi-tech Park, Nanshan, Shenzhen 518057, China. Tel.: +852 3943 4257.
*E-mail addresses:* jjxiong@cse.cuhk.edu.hk (J. Xiong), yfzhou@cse.cuhk.edu.hk (Y. Zhou), lyu@cse.cuhk.edu.hk (M.R. Lyu), fyyoung@cse.cuhk.edu.hk (E.F.Y. Young).

1084-8045/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2012.09.008

SNIF deploys a temporary wireless network to snoop a WSN (Ringwald et al., 2007). It is better than SNTS in that it does not need to manually fetch back the monitoring nodes, but its nodes are required to be equipped with two radio sets.

To tackle the above disadvantages, we are the first to propose a mobility-assisted diagnosis (MDiag) approach to examine failures in WSNs with an independent mobile device. Nowadays, mobile smartphones are increasingly popular. By the end of 2011, there are 6 billion mobile subscribers globally, and smartphone users show the strongest growth (Global mobile phone statistics). It is now convenient to patrol a deployed WSN with smartphones to collect packets sent from sensor nodes. Then the smartphones can perform diagnosis by analyzing these packets. Such a diagnosis process can be flexibly turned on or off, allowing users to perform an on-demand diagnosis. For WSNs that are deployed in harsh areas without human traces, mobile robots are able to carry the smartphones (Johnson et al., 2006).

A diagnosis scenario with one smartphone is shown in Fig. 1, where the dotted lines are the patrol path of the smartphone. The smartphone starts snooping sensor node 1 for a certain interval, continues traveling along the path and snooping sensor nodes 2, 3, 4, and 5 for some intervals, and then returns to the starting sensor node 1.

The most important feature of such an approach is that it is not intrusive to WSN applications because the smartphones themselves are not components of WSNs. MDiag does not need to implant diagnostic agents to the applications, and thus it does not need to modify the applications as the proactive approaches do. Hence, MDiag can be employed to scrutinize any deployed WSNs when needed without affecting the WSN performance. In addition, the mobility of smartphones allows users to apply one or several smartphones to record the packets sent from the WSN instead of many debugging sensor nodes that need laborious manual placement and retrieval. Finally, MDiag can collect all kinds of raw packets (packet created from the radio frequency chip that contains packet header information of the upper layer protocols), which is especially useful in finding more failures. To analyze raw packets of all types, the agent approaches need to inset agents at all the protocol agents, which is very inefficient. For example, MDiag can collect control packets exchanged locally, which are not transmitted back to the BS for diagnosis in Sympathy (Ramanathan et al., 2005).

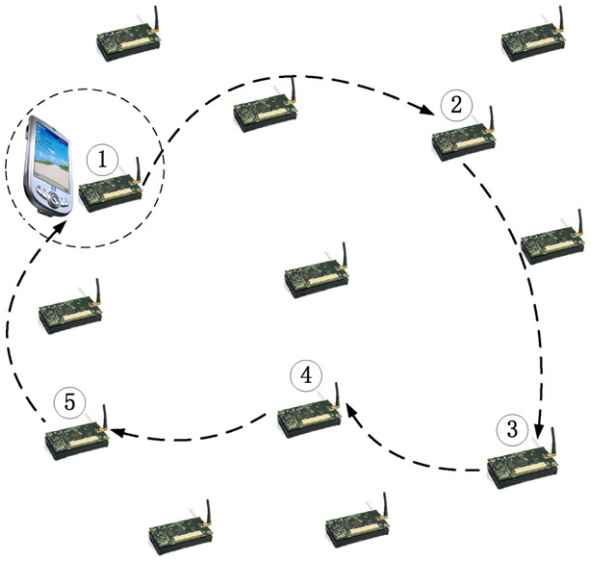


**Fig. 1.** A MDiag patrol scenario. The smartphone first visits sensor node 1, then sensor nodes 2, 3, 4, and 5.

The design of MDiag mainly faces two challenges. First, it is difficult to determine the abnormal behaviors from the collected various packets, such as routing packets, MAC layer control packets, and application data packets. We define a few statistical rules to identify anomalies based only on minimum prior knowledge of the system. Second, the patrol pattern of the smartphones influences the failure detection rate. How to design the patrol method for visiting the sensor nodes in order to collect packets is crucial and difficult. We propose a method called MSEP (maximum snooping efficiency patrol) which identifies the sensor node set to patrol and collect relevant packets for analysis. Consequently, it increases the failure detection rate. Finally, in comparison with the naive method, greedy method, and baseline method, we perform extensive experiments to demonstrate the effectiveness of MDiag and show its effective detection of several interesting failures.

The rest of this paper is organized as follows. First, Section 2 highlights the related work. Next, Section 3 clarifies the network architecture and failure classification for MDiag. Section 4 designs the framework of MDiag while Section 5 proposes the algorithm for the smartphone patrol approaches. Section 6 presents the results of our experimental evaluation. Finally, Section 7 concludes the paper.

## 2. Related work

There has been tremendous work in finding faults that will cause network failures in WSNs. Some of them are based on simulations, such as ns-3 (http://www.isi.edu/nsnam/ns/), a general network simulator, and TOSSIM (Levis et al., 2003), a simulator for WSNs running on TinyOS (Levis et al., 2005). More advanced simulation-based failure detection tools include T-Check (Li and Regehr, 2010) and KleeNet (Sasnauskas et al., 2010). T-Check (Li and Regehr, 2010) is an event-driven simulator developed from TOSSIM, while KleeNet (Sasnauskas et al., 2010) is based on the low-level symbolic virtual machine. Given the fact that the behaviors of nodes in a simulation environment and real sensor nodes are not the same, in-situ diagnosis approaches for WSNs are indispensable.

One kind of in-situ diagnosis approaches is to embed diagnosis agents into each sensor node. For example, by injecting the diagnosis agent, Sympathy (Ramanathan et al., 2005) actively collects run-time status from sensor nodes, such as routing table and flow information, and detects possible faults after transmitting them to the BS. Nevertheless, these approaches are not adaptable for WSNs that have already been deployed without such agents. Because we have to update the software of WSNs so as to embed the agents, which unavoidably affects the running of WSN applications (Demirbas, 2008). Moreover, Sympathy asserts that it only employs one rule that insufficient data at the sink implies a failure, which means that it cannot detect certain network failures, such as overenthusiastic transmission which results in reduced battery lifetime.

For WSNs deployed with these diagnosis agents, the agent injection methods inevitably intrude the WSN applications. The agent execution may cost extra CPU computation, communication bandwidth, and energy of the sensor nodes, which imposes heavy burden on the resource-limited networks. Since Sympathy costs extra communication bandwidth and energy consumption to actively transmit the information, PAD (Liu et al., 2008) and PassiveAssertions (Romer and Ma, 2009) reduce the diagnosis efforts by piggybacking the logs in regular data packets. Self-Diagnosis (Liu et al., 2011) involves multiple local nodes in cooperation to detect WSN failures based on finite state machines. This approach reduces the communications effort of transmitting the logs to the BS. Since all the existing in-situ diagnosis approaches are intrusive to the WSNs, we propose a new

MDiag approach. Instead of changing the WSN systems, MDiag employs mobile phones to travel in deployed WSNs and to passively collect packets sent from the passing sensor nodes. The collected packets can then be utilized for diagnosis. Because mobile phones are not a part of WSNs, MDiag does not intrude the WSNs and is thus adaptable to all WSN applications.

Another type of approaches to avoid the intrusion to WSN applications is deploying another network to monitor the WSN. SNTS deploys several monitoring sensor nodes in the real field along with the original sensor nodes (Khan et al., 2007). The monitoring sensor nodes sniff and record the packets sent from the sensor nodes that they observe. Although the debugging sensor nodes can formulate a network, they cannot communicate with each other and transmit the recorded packets to the BS because their transmissions would cause interference to the original WSN. As a result, in fetching the recorded packets, all of the debugging sensor nodes have to be manually retrieved, which are quite labor-intensive. In addition, for a large WSN, SNTS should deploy many debugging sensor nodes so as to guarantee that all the original sensor nodes can be monitored, hence SNTS does not suit large WSNs.

To avoid the efforts of manually retrieving the nodes in the monitoring network, SNIF deploys a different monitoring network which is composed of nodes with two sets of radio equipment (Ringwald et al., 2007). One set is used to passively listen to the packets sent from the WSN. The other set employs a different wireless channel to transmit the sniffed packets to the BS without interfering the original WSN. Nevertheless, dual radios are generally not facilitated in the off-the-shelf sensor network products. Moreover, like SNTS, SNIF also deploys an additional diagnosis network, which is not cost-effective. With our MDiag approach, the mobility of diagnosis devices saves the cost of the large monitoring network deployment, and the employment of smartphones eliminates the extra hardware requirement of monitoring nodes.

Finally, mobile sensor nodes have already been introduced to improve WSN coverage (Liu et al., 2005; Xing et al., 2008a) while multiple, mobile base stations are also deployed to prolong the lifetime of the WSNs (Xing et al., 2008b; Shi and Hou, 2008). However, no prior work has employed mobile platforms, such as the smartphones, for diagnosing WSNs. MDiag is the first work to utilize mobile platforms for WSN diagnosis purpose.

## 3. MDiag background

### 3.1. Network architecture

We consider a network which involves a BS and static wireless sensor nodes deployed in a sensing field for monitoring temperature, humidity, noise level, etc., from the environment. A smartphone is able to receive the raw packets sent from the sensor nodes within its reception range as long as it is equipped with the same reception device as the sensor nodes or it is attached with a sensor node for snooping purpose only (Ngai et al., 2011; Rensfelt et al., 2010). To diagnose, the smartphone can patrol in the WSN field as shown in Fig. 1. The patrol can start at any time and last for any duration according to the requirement of WSN maintainers. For example, they may want to patrol a WSN once each day, or they only turn on the diagnosis for a long period when they need to ascertain and find more information about suspicious phenomena.

Intuitively, the more the smartphones are available for WSN diagnosis, the better the diagnosis effectiveness is. Nevertheless, more efforts are required to carry them in the network field. Without loss of generality, in the following we discuss the case of using one smartphone to patrol a WSN and to passively collect network packets. The case of using more smartphones can be similarly extended.

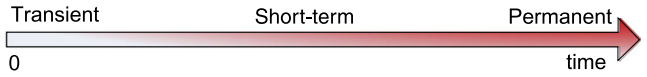


**Fig. 2.** Failure classification.

### 3.2. Failure classification

MDiag is proposed to diagnose the failures in WSNs by collecting and analyzing the packets sent by sensor nodes. Since it can only diagnose the network failure according to the packets sent out in the WSNs, it cannot detect failures that are not triggered during the execution. The failures are divided into three classes as shown in Fig. 2. The *x*-axis is failure lasting time, and it increases from left to right. The first class is transient failures that last for a very short period, such as random packet loss considered in Zhou et al. (2010). The second class lasts only for a longer period and then disappears, such as routing failures, link failures, and congestion. It is called short-term failure. The third class is permanent failures that stay in the network until they fixed or they stay for a very long time, such as node crash and incorrect resource allocation failures. We discuss some of the failures we identified in our experiments in Section 6.

The transient failures hold for a very short time and they do not affect the network behaviors distinctively, so we can hardly detect them by in-situ diagnosis. In the following, we focus on detecting the permanent failures and short-term failures.

## 4. MDiag framework

Referring to the diagnosis scenario shown in Fig. 1, we design the MDiag framework as shown in Fig. 3.

In the first step, by visiting each sensor node in the WSN for a sufficiently long time, the smartphone can get the neighboring information of each sensor node. It inputs the neighboring information to the algorithm of patrol set selection which generates the set of K sensor nodes to patrol. Note that our algorithm of patrol set selection does not require the full knowledge of the WSN topology, although during the patrol the smartphone could build up a coarse topology by locating each sensor node's approximate position with its GPS (global positioning system). During the initial WSN visit, we can also estimate the traffic load at each sensor node. Then the time that the smartphone stays along with each sensor node in the patrol set is easy to be calculated. The heavier the traffic load, the shorter the time that the smartphone requires to collect enough data packets and other control packets for the statistical analysis.

In the second step, on the basis of packet structures provided by WSN developers or system specifications, packet decoder and some statistical rules for packet analysis are applied on the collected packets.

In the third step, the sequence of packets collected during the patrol of the K sensor nodes is input to the packet decoder, which will generate statistical results. Based on the statistical rules on packet analysis, if the smartphone deducts that the network behavior is abnormal, it can produce a failure report and trigger the additional diagnosis. Further actions, such as collecting extra packets for more detailed inspection, can be taken to analyze the phenomenon and locate the potential bugs.

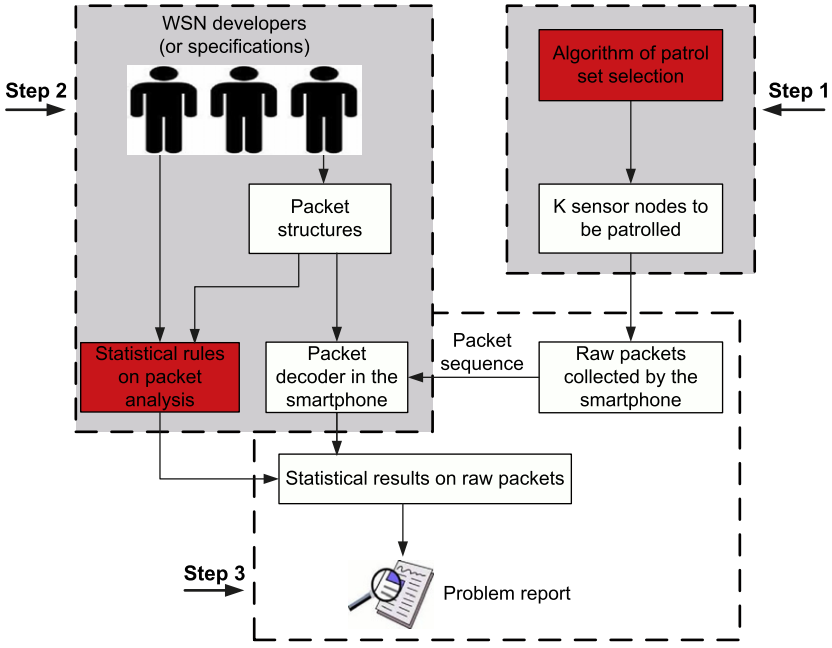


**Fig. 3.** MDiag framework.

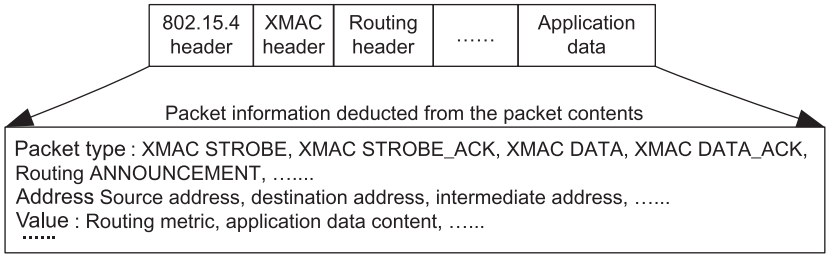


**Fig. 4.** Raw packet structure.

Next we focus on the input and output of the packet decoder and the statistical rules on packet analysis. The algorithm of patrol set selection will be discussed in the next section.

### 4.1. Packet decoder input and output

Packet decoder is used to analyze the various types of collected raw packets in the network and to output statistical packet results for inspection. First, we discuss the feature of the decoder input. Then we show the format of the decoder output.

We can collect raw packets from the radio frequency chip. Figure 4 shows a typical raw packet structure in WSNs whose MAC layer protocol is X-MAC, a frequently used MAC protocol (Buettner et al., 2006). Besides application data, the raw packet contains a lot of low-level header information, from which we can acquire the information on packet types, addresses, values, and so on. Raw packets help us analyze the network behaviors from all protocol layers, including interactions between layers.

The input of the packet decoder is a sequence of raw packets. After decoding the raw packets according to the structures shown in Fig. 4, the output is statistical results for the failure report shown in Fig. 5. It shows the sensor node neighbor information, the count of each type of packets, and so on.

```
For sensor node W1,
W1 has neighbors:
 A, B, C, ……
W1 has sent out the following types of packets to neighbor A:
 XMAC STROBE count: X
 XMAC STROBE_ACK count: Y
 XMAC DATA count: Z
 XMAC DATA_ACK count: U
 Routing ANNOUNCEMENT count: V
 ……
W1 has send out the following types of packets to neighbor B:
 ……
 For sensor node W2,
……
```

**Fig. 5.** Statistical results.

### 4.2. Statistical rules on packet analysis

The aforementioned rules describe statical packet behaviors in WSNs, therefore, they are not applicable to analyze a single packet exchange process. For example, the rules cannot be used in detecting the failure of the TCP three-way handshake caused by random packet loss because a TCP three-way handshake process does not compose a statistical packet behavior. Moreover, a random TCP packet loss does not mean application failure or performance degradation unless it happens frequently.

Because our rules are statistical and are provided by the developers or the specification, they are a subset of the specification-based rules as shown in Table 1. Unlike the state-of-the-art approach Sympathy (Ramanathan et al., 2005) which employs

**Table 1**
Our statistical rules are a subset of the specification-based rules.

| Layer | Specification-based rules | Statistical rules |
|---|---|---|
| Application layer | The value (e.g., temperature) of the application data falls within the range specified. | Yes |
| | The number of data packets exchanged among the BS and the sensor nodes should be within the specified range in a certain interval. | Yes |
| | The data communication process (such as the packet order) is based on the specification. | No |
| | … | |
| Routing layer | The routing metric value is in the specified legal range | Yes |
| | The number of each kind of routing packets sent out by the BS and sensor nodes is normal within a certain period | Yes |
| | The routing packet exchange sequence should conform to a specific routing protocol (e.g., first routing request packet, then routing reply packet) | No |
| | … | |
| MAC layer | The number of each kind of MAC packets sent out by the BS and sensor nodes is normal within a certain period | Yes |
| | The MAC packet exchange process is based on the specification | No |
| | … | |

only one rule, i.e., insufficient data at the sink implies failure, our statistical rules are based on the targets of all protocol layers. Hence our approach can detect more failures than Sympathy (Ramanathan et al., 2005). By decoding the raw packet, we can infer the following packet fields on which the statistical rules utilize:

- Packet type.
- Packet count of each type.
- Packet directions.
- Neighbor information.
- Packet value, such as routing metric of routing packet, data content of application data packet, and so on.

Without loss of generality, we take one kind of the most popular WSN applications, the data gathering application, including routing protocol *CTP* (Gnawali et al., 2009) and *XMAC* protocol (Buettner et al., 2006), as an example and give the major statistical rules in Table 2. This set of rules can be extensible according to the statistical granularity. The parameters used in the rules are represented in Fig. 5.

According to the application target, the data flow is only from the sensor nodes to the BS, and hence we derive Rules 1 and 2 in Table 2. Rule 1 means that the BS does not send data packets. Rule 2 means that other sensor nodes send the required amount of data packets.

Based on the routing protocol target, Rules 3 and 4 clarify the legal range of the routing metric, and Rule 5 means no routing fluctuation. There should be no bidirectional data exchange between two sensor nodes; otherwise, it is possible that the routing is unstable and causes the two sensor nodes to use each other as the forwarding counterpart.

Rule 6 comes from the X-MAC behavior between a pair of communicating sensor nodes, i.e., for each sensor node, the number of X-MAC STROBE_ACK packets sent should be almost equal to the number of X-MAC DATA_ACK packets sent and the number of X-MAC DATA packets received. If Rule 6 is violated, say

**Table 2**
Statistical rules for a typical WSN application.

| Layer | Statistical rules |
|---|---|
| Application layer | Rule 1. For BS, $Z$, the number of application data sent out is 0 |
| Application layer | Rule 2. For sensor nodes other than BS, $Z$ is within the application requirement |
| Routing layer | Rule 3. For BS, the legal routing metric is 0 |
| Routing layer | Rule 4. For sensor nodes other than BS, routing metric value is legal |
| Routing layer | Rule 5. For sensor nodes other than BS, no bidirectional data exchange exists, i.e., $Z*U=0$ |
| MAC layer | Rule 6. For sensor nodes other than BS, the number of each kind of MAC packets sent is normal, i.e., $Y \simeq U,X>$ or $\gg Z$ |

the number of X-MAC STROBE_ACK packets sent is far more than the number of X-MAC DATA_ACK packets sent, then the X-MAC works inefficiently.

The application mechanism, the CTP protocol, the X-MAC protocol should be working well when these rules are satisfied. The BS receives its expected data regularly, and the CTP and X-MAC protocols work efficiently without too much extra energy consumption for communications.

## 5. Coverage-oriented smartphone patrol algorithms

By patrolling the WSN, we can collect the packets sent out by the sensor nodes in the patrol set. By analyzing these packets, we can diagnose the WSN. Collecting more packets can help detect more failures. If the packets from some sensor nodes are never collected, failures happening at these sensor nodes are less likely to be discovered. As a result, the patrol approach should try to cover all the sensor nodes in the WSN. Due to their different coverage efficiency, different patrol methods can affect the detection efficiency, especially for the short-term failures.

We do not consider the cost during the travel, hence the visiting order of the sensor nodes in the patrol set is not a concern in our patrol set establishment process. The key is the patrol set rather than the patrol path. Moreover, our experiments also verify that the difference is very small (less than 3% in most cases) for visiting the sensor nodes in the same patrol set in different order.

### 5.1. Naive method (NM)

The first method is the naive method that the smartphone visits all the sensor nodes one by one, but it needs long time to traverse the deployed WSN for only one round. If a failure happens at the last sensor node that will be patrolled, it is very likely that when the smartphone finally arrives at the sensor node, the failure phenomenon has already disappeared. Correspondingly, the failure detection rate of NM is low. If we can cover all the sensor nodes in a shorter period, then the failure missing rate would be lower.

### 5.2. Greedy method (GM)

To reduce the patrol time, we utilize the broadcast nature of wireless transmissions and thus design methods that perceive all the sensor nodes without visiting each sensor node one by one. For example, in Fig. 6, since node 2 is only neighboring to nodes 1 and 3, a smartphone that visits node 2 can also overhear the packets sent from node 1 and node 3. As a result, to patrol all the senor nodes in this topology, we only need to visit node 2 and node 4 instead of visiting all the four nodes one by one. By

patrolling the sensor nodes in the patrol set, all the sensor nodes could be snooped without visiting them one by one.

However, to find the patrol set with the minimum sensor nodes is a set cover problem, and it is shown to be NP-complete (Karp, 1972). An intuitive method is the greedy method (GM) (Johnson, 1973). To simplify the GM explanation, we choose a regular 9-node network topology as shown in Fig. 7, in which sensor node 1 is the BS, and the other eight nodes are sensor nodes. Sensor node 5's neighbors are: 2, 4, 6, and 8. The *degree* of each sensor node $v$ is the number of its neighbors, denoted as Degree($v$). For example, for node 5, Degree(5)=4, and for node 1, Degree(1)=2.

We define the *snooping efficiency* of visiting a sensor node $v$ as Degree($v$). Visiting $v$ can overhear the other (Degree($v$)−1) sensor nodes. The larger the snooping efficiency the better, because more packets can be collected on the visit. The *snooping efficiency of a patrol set S* with $K$ sensor nodes is defined as $(1/K)\sum_{v \in S}$ Degree($v$). Improving the patrol set snooping efficiency can increase the packets collected for diagnosis, and hence raise the failure detection probability.

By them our target is to improve the patrol set snooping efficiency rather than to solve the minimum set cover problem. When the sensor node coverage is satisfied, minimizing the patrol set size can increase the patrol set snooping efficiency, and increasing the patrol set snooping efficiency helps reduce the patrol set size.

In the GM algorithm, the smartphone always selects to visit the sensor node whose current degree is the largest. Each time a sensor node is selected into the patrol set, its neighbors and itself are removed from the neighbors of the remaining sensor nodes. Hence, the degrees of the remaining sensor nodes are updated for the next greedy selection. The sensor node selection continues until all the sensor nodes can be snooped.

### 5.3. Maximum snooping efficiency patrol (MSEP)

GM can result in selecting several sensor nodes with small degree. For example, the selection results with GM are shown in sub-figure (a) of Fig. 7. Sensor nodes 1, 3, 7, and 9 are visited with low snooping efficiency because their degrees are 2. In contrast, sensor node 5 is visited with high snooping efficiency. Nevertheless, since many sensor nodes are visited with low snooping efficiency, GM cannot achieve high patrol set snooping efficiency which is 2.4. Eliminating or reducing such visiting can help enhance the snooping efficiency and reduce the patrol set size. Therefore, we design a different method called maximum snooping efficiency patrol (MSEP) that can traverse all the sensor nodes with a smaller patrol set size. In comparison, its patrol set selection of the regular 9-node network is shown in sub-figure (b) of Fig. 7, which contains only three sensor nodes rather than five sensor nodes for the GM algorithm. The snooping efficiency of this patrol set is 2.67, higher than that of GM patrol set.


**Fig. 6.** Network topology I. The smartphone stays near sensor node 2.

MSEP aims at enhancing the snooping efficiency by reducing small degree node selection probability. Therefore, it can correspondingly reduce the patrol set size. On one hand, it should guarantee to cover every sensor node, including the sensor node with a small degree. Therefore, MSEP first finds $i$, the sensor nodes with the minimum degree. On the other hand, to reduce small degree node selection probability, it selects a sensor node $j$ with the largest degree from $i$'s *neighbor set*. By visiting $j$, the smartphone can also snoop $i$, and the snooping efficiency is increased. Then $j$ is added to the patrol set $M$. Nevertheless, such selection may still result in selecting sensor nodes with a small degree in the future. Hence, MSEP rewinds and corrects sensor node $j$ and $i$ selection for a few times. The details are shown in Algorithm 1.

The first step in Algorithm 1 is the sensor node selection and degree update. At line 7, sensor node $i$ with the minimum degree is selected. At line 14, sensor node $j$ with the largest degree is chosen to be included in the patrol set. To calculate how many sensor nodes remain to be patrolled, line 16 deletes $j$ and sensor nodes in its neighbor set $N_j$ from $S'$ and $W'$. Corresponding to the deletion, line 17 updates the degree of the remaining sensor nodes in $S'$ and $W'$ (i.e., the degree is decreased because of sensor node deletion).

Next is the refinement of sensor node selection. Line 19 checks whether selecting $j$ will result in the remaining sensor nodes with a small degree, e.g., degree 1. If the answer is *no*, we select the $j$ and add it to $M$ (line 22). Otherwise, we refine the sensor node selection. We select a different $j$ (back to line 14). If all sensor nodes in $H_i$ result in sensor nodes with a degree 1, remove $i$ from $S$, add it to $W$ (line 20), and repeat to find a sensor node with the minimum degree in $S$ (back to line 7). Doing such check and sensor node selection refinement can improve the snooping efficiency. In our algorithm we choose to refine the selection of

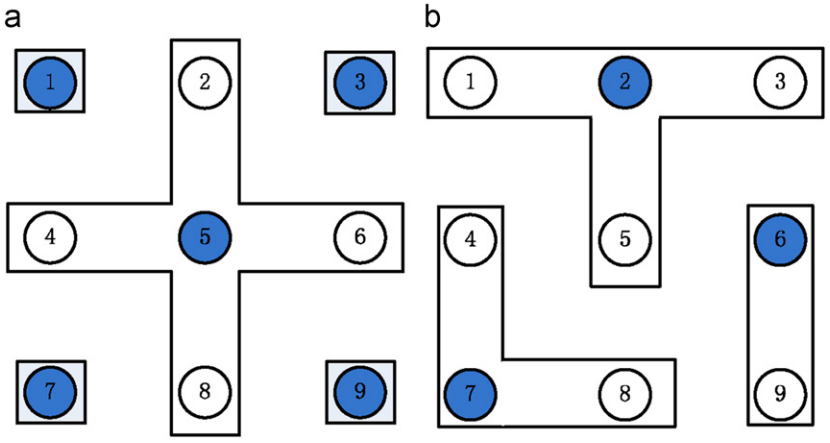


**Fig. 7.** Patrol set selection of GM (sub-figure a) and MSEP (sub-figure b). The blue circle is put into the patrol set. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

sensor nodes with degree 1. We can also refine with a larger degree, which can improve the snooping efficiency, but will cost more computation efforts.

After handling all the sensor nodes in $S$, we visit the sensor nodes in $W$. The difference is that when handling sensor nodes in $W$, we do not take refining actions on sensor nodes with degree 1.

**Algorithm 1.** MSEP algorithm

1: Node set $S=\{$all the nodes$\}$
2: Patrol set $M=\emptyset$
3: Candidate node set $W=\emptyset$
4: $S'=S, W'=W$
5: $k=0$ //$k$ is the size of set $M$
6: **while** $S!=\emptyset$ **do**
7: Choose a node $i\in S$ with the minimum degree //from neighbor information
8: Let $H_i=N_i\bigcup i$ // $N_i$ is node $i$'s neighbor set
9: **repeat**
10: $S'=S, W'=W$
11: **if** $H_i==\emptyset$ **then**
12: break
13: **end if**
14: $\forall$ nodes $\in H_i$, choose $j$ with the largest degree
15: Delete $j$ from $H_i$
16: Delete $j$ and $N_j$ from $S'$ and $W'$
17: Update Degree(v), $\forall v\in S'$ and $W'$
18: **until** Do not exists $v\in S'$ with Degree(v) $==1$
19: **if** $H_i==\emptyset$ and $\exists v\in S'$ with Degree(v) $==1$ **then**
20: Delete $i$ from $S$, add $i$ to $W, W'=W, S'=S$
21: **else**
22: Add $j$ to $M, W=W', S=S', k=k+1$
23: **end if**
24: **end while**
25: **while** $W!=\emptyset$ **do**
26: Choose a node $i\in W$ with the minimum degree
27: $\forall$ nodes $\in N_i\bigcup i$, choose $j$ with the largest degree
28: Add $j$ to $M$, Delete $j$ and $N_j$ from $W, K=K+1$
29: Update Degree(v), $\forall v\in W$
30: **end while**

## 6. Evaluations

In evaluating the patrol set size and the snooping efficiency, we perform extensive simulations in the most popular freeware network simulator ns-3 (http://www.isi.edu/nsnam/ns/). The results demonstrate that our MSEP requires smaller patrol size than GM, and it achieves higher snooping efficiency. The results imply that MSEP can obtain higher failure detection. In the simulation, NM is not compared with MSEP because its patrol set size is far too larger than that of GM and MSEP.

In evaluating failure detection effectiveness, for permanent failure detection, we carry out the evaluations with 4 TelosB sensor nodes (http://www.willow.co.uk/TelosB_Datasheet.pdf) and 1 Andriod smartphone (Rensfelt et al., 2010), in which the transmission range of the sensor nodes is about 20–30 m indoor. We perform emulations on sensor node operating system Contiki-2.4 (Dunkels et al., 2004) for short-term failure detection, in which the transmission range of the emulated sensor nodes is 50 m. In all the experiments, an existing application, data collection application runs with underlying routing protocol *CTP* (Gnawali et al., 2009) and *XMAC* protocol (Buettner et al., 2006). We use real failures encountered in our experiments and also failures found in Contiki, and passively collect the raw packets sent from the radio frequency chip. The received packets can be stored at the smartphone which is equipped with larger memory. Then based on the packet format, we provide a packet decomposition program (small enough to fit in smartphones) to analyze the received packets online and then generate the failure report.

Besides NM and GM, we also implement a baseline method called RM-$K$. RM-$K$ randomly selects $K$ sensor nodes to form the patrol set. It does not care whether such set can cover all the sensor nodes, i.e., visiting the sensor nodes in the set can snooping all the sensor nodes in the WSNs. We repeat the random patrol set selection for 100 times and use the averaged results for RM-$K$.

### 6.1. Patrol set size and snooping efficiency

In this section, we perform extensive simulations of MSEP and GM algorithms in ns-3. We use the setdest tool in ns-3 to randomly generate connected network topologies of the following sizes: 25, 100, 225, 400, 625, 900, 1225, and 1600 nodes. SMAC (Ye et al., 2002), the classical MAC protocol for WSNs, is employed. Each sensor node generates application data periodically. Both DSR (dynamic source routing) and static routing are used. The reception range is 50 m.

Figure 8 shows that the patrol set size of the MSEP is smaller than that of GM. It follows that the patrol set snooping efficiency of MSEP is higher than that of GM when both MSEP and GM guarantee the coverage of the WSNs. Figure 9 also verifies this

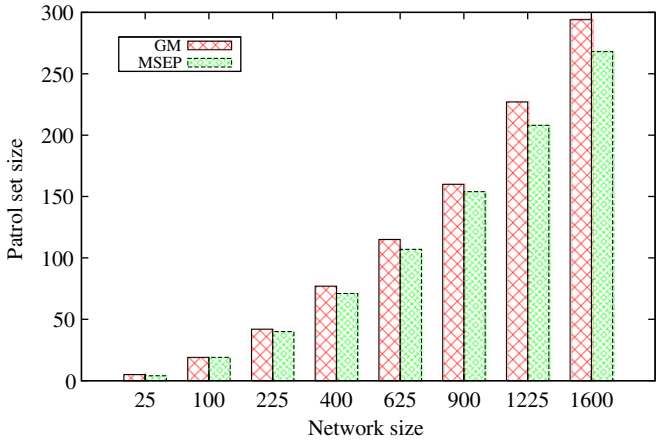


**Fig. 8.** Patrol set size.

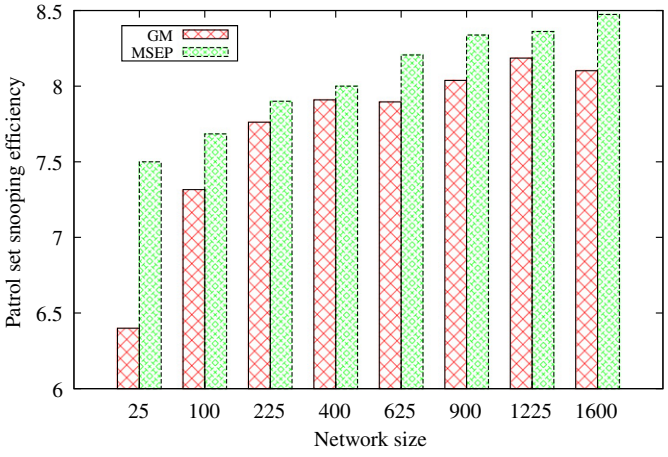


**Fig. 9.** Patrol set snooping efficiency.

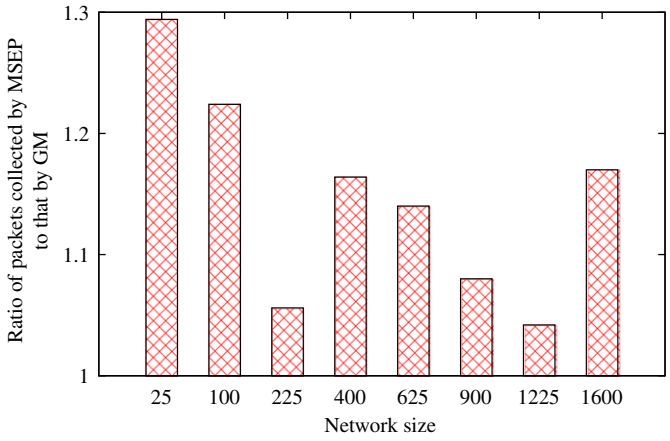


**Fig. 10.** Ratio of packets collected by MSEP to packets collected by GM.

point. The theoretical patrol set snooping efficiency can be calculated after knowing the degrees of each sensor node in the patrol set. Next by running the WSN application in ns-3, we calculate the packets that are snooped by the MSEP and GM in the same period, through which we can get the practical patrol set snooping efficiency in simulations. The results are shown in Fig. 10. The ratio of packets collected by MSEP to packets collected by GM is consistent with the ratio of the snooping efficiency of MSEP and GM. Because the packets collected are of various kinds and the staying time at different visited sensor node is different, the actual total count of packets collected in simulations may vary from those in the theoretical cases. Although small fluctuations exist between the theoretical and practical scenarios, they both verify that MSEP achieves higher snooping efficiency than GM. We observe that the ratios for the 25-node and 100-node networks are much larger than those for the other networks. The reason is that the node density of the 25-node and 100-node network (1 sensor node per 51.84k sq m) is lower than that of the rest of the networks (1 sensor node per 46.24 sq m). Hence the snooping efficiency of them is also smaller than that of the other networks as shown in Fig. 9. In larger networks with randomly deployed sensor nodes, higher density is required in order to guarantee that the networks are connected without holes. This means that MSEP is much better than GM, especially in WSNs with low node density.

### 6.2. Permanent failure detection

Initially, for the purpose of demonstrating an existing WSN application in Contiki, we build up and run it with four telosB sensor nodes and one smartphone shown in Fig. 6. Since the indoor transmission range of TelosB is about 20–30 m, we separate two neighboring sensor nodes for about 18 m apart. We do not expect any failure because the topology is simple and the application is provided by the developers. However, the failure still happens.

Specifically, in the packets which sent from sensor node 2 to sensor node 3, the ratio of XMAC STROBE_ACK packets to XMAC DATA_ACK packets is about 52, while in fact the ratio should be around 1 according to Rule 6. This means that node 2 agrees to receive XMAC DATA packets from node 3 for 52 times, but it only tells node 3 that it receives the expected data packets once.

This is a performance degradation failure because it costs a lot of control packet exchange to transmit one required data packet and the energy consumption is very high. It can hardly be detected at the application layer because the BS receives the expected data packets. Although this problem does not fail the application in this simple topology, it will break down the application when the network size increases. In this experiment, the smartphone can detect the failure by staying near any sensor node. It finds that the collected packets violate Rule 6.

It is very surprising that this failure is caused by the 'printf' statements in the application program. As we know, for the program running in the PC, inserting a few 'printf' statements can hardly affect the normal execution. Nevertheless, it damages the WSN application execution. In this WSN application, every time a sensor node is going to send out a data, it will first print out a message 'Sending'. By analyzing the actual running process of the sensor node, we notice that the 'printf' statement will issue the serial port interrupt, which will post a process to print out a character. Printing out several characters means more processes posted in the CPU queue. As a result, the 'printf' statement will take up so many CPU resources that the CPU is too busy to handle the packet transmission and reception in time. As a result, the packets are dropped after the sensor node receives them at the physical layer rather than being dropped on the way.

Forgetting to comment out the debugging 'printf' statements is very common. Writing redundant statements is also common, which may fail the WSN applications due to consumption of limited resources. This kind of failures is not easy to be identified because the programs are correct in the aspect of function and logic. As performance degradation failure, they may not be manifested in simple scenarios, such as small-scale testbed. Nevertheless, since MDiag is able to collect packets of all types, it can analyze the WSN application and detect failures at all protocol layers. It can help find failures that do not just occur at the application layer.

### 6.3. Short-term failure detection

The patrol approaches are very crucial in detecting short-term failures. For the sake of easy identification of correct behaviors, we use the regular network topology shown in Fig. 11, in which node 1 and node 2 are 40 m apart considering that the transmission range of the emulated sensor node is 50 m. In fact, our MDiag also works very well with sensor nodes randomly deployed because our algorithm of patrol sensor node set selection relies only on neighbor information of each sensor node rather than on the precise topology and locations.

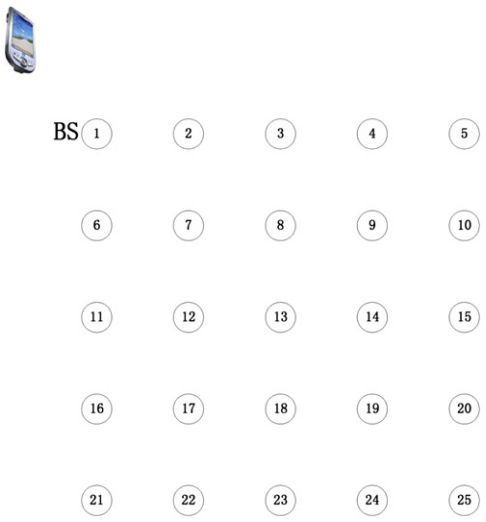


**Fig. 11.** Network topology II. The smartphone patrols the sensor nodes in the network.

#### 6.3.1. Short-term failure explanation

Then we run the existing application in a larger network shown in Fig. 11. We find that the routing is very unstable when the WSN starts to run. Many bidirectional data exchange happens, which violates Rule 5. After a long time the routing stabilizes, i.e., no bidirectional data exchange. Due to node crash and application fix, sensor node reboot is not rare in real deployment. We reboot a sensor node during the application running process. Because such reboot takes a very short time, according to the basic mechanisms of routing protocol *CTP*, we expect that the routing may remain unchanged or the routing of the surrounding sensor nodes may change a little. Nevertheless, the reboot results in routing fluctuation in a large range and for a long time. Rule 5 is violated again, i.e., bidirectional data exchange happens.

This failure in the routing protocol CTP is caused by not initializing the routing value of each sensor node to the maximum value while the routing value of the BS is initialized as 0. According to the original value in the memory, the routing value is 0 at the initial stage. As a result, a rebooted sensor node will broadcast its routing value as 0. This means that it is the BS whereas it is actually not. In this way, routing disturbance will occur in the network when a sensor node is rebooted. Although the routing will become abnormal after the correct routing value from the BS arrives at the rebooted sensor node and its surrounding sensor nodes, failures last for a period of time because the wrong routing value will propagate for a certain range. Specifically, the routing fluctuation caused by a sensor node reboot is a short-term failure.

At time 600 s, the network stabilizes, and node 15 is rebooted to repeat our previous failure detection process. To get the ground truth, we analyze all the packets sent out from time 600 s to 1600 s and find many abnormal cases (*ACs*): bidirectional data packet exchange between a pair of sensor nodes, which disobeys Rule 5. The abnormal cases can be classified in two aspects. First, in respect of lasting time, some of them last for a much longer time than the others. We name them *long AC* and *short AC*. Second, for some ACs, the bidirectional data packet exchange happens frequently throughout their lasting period while for the other ACs, it only happens for a very few times. We called them *frequent AC* and *infrequent AC*. For example, R represents a datum in the opposite direction of a datum D. In 200 s, the pattern of a frequent AC can be DRDRDRDRDR and an infrequent AC can be DDDDDRRRRR. Since patrolling the WSN cannot snoop all the packets sent out by the network, a frequent and long AC is easy to be discovered with high probability.

People are interested in knowing whether reboot happens, but they always ignore the performance during reboot. In fact, many potential bugs are trigger in corner situations, such as sensor node reboot, random packet loss, packet duplication, and sensor node crash (Sasnauskas et al., 2010). In addition, not initializing variables is a frequently-made mistake, though its negative influence is not always obvious. When such kind of frequently-made mistakes meet the corner cases, failures are triggered. Nevertheless, some of the failures do not persist for a very long time. Hence, the BS may still receive its expected data at the application layer. In this way, other methods that are unable to collect packets of all types will have a lower chance to detect the failures.

#### 6.3.2. Short-term failure detection results

To evaluate the failure detection ability of MSEP, we compare the AC detection probabilities of NM, GM, and MSEP and RM-*K* because detecting any AC means detecting failure. In this experiment, the patrol set of NM, GM, and MSEP contains 25, 10, and 7 sensor nodes respectively. RM-7's patrol set consists of seven randomly selected sensor nodes while RM-10's consists of 10. To collect three (the data packet number can be adjusted) data packets at each snooped node, the patrol time of NM, GM, MSEP is around 625, 260, and 180 s respectively.

To compare the approaches, we first compare their performance in finding one AC by filtering out all the other ACs. Then we check their probability in finding at least one AC from all the ACs. To do so, we select seven representative ACs as shown in Fig. 12. Among them exist long, short, frequent, and infrequent ACs. AC1, AC2, and AC3 are long ACs that last for about 500 s or 600 s, while AC4, AC5, AC6, and AC7 are short ACs that last for about 200 s. AC1, AC3, AC4, and AC5 (red pillars) are frequent AC. AC2, AC6, and AC7 (yellow pillars) are infrequent ACs.

Figures 13 and 14 statistically show the detection probability of AC1 and AC3, two long and frequent ACs. In this scenario, the lasting of all the ACs is no longer than 600 s, hence we do not consider working for an interval longer than 600 s. Generally, as the working time of the smartphone increases from 200 s to 600 s, the detection probabilities of all methods increase because more packets can be snooped. The detection probability of RM-7 and RM-10 is similar and they represent the average level. NM is worse than RM-7 and RM-10, i.e., its performance is below the average. MSEP is better than other methods no matter how long the smartphone works. Notice that GM performs worse than NM. Because GM always chooses the sensor nodes with the largest

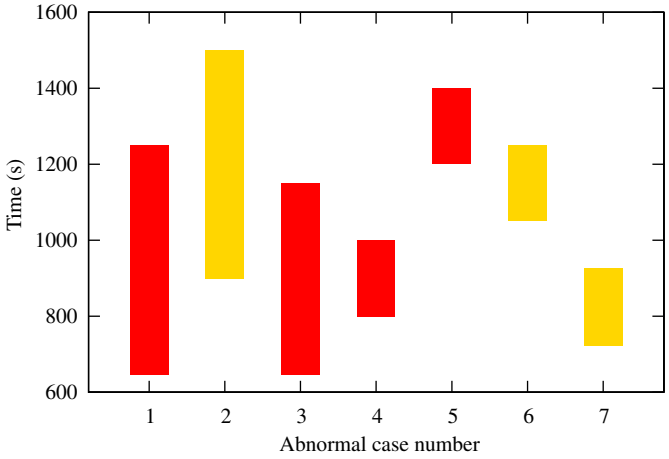


**Fig. 12.** Lasting time of the abnormal cases.

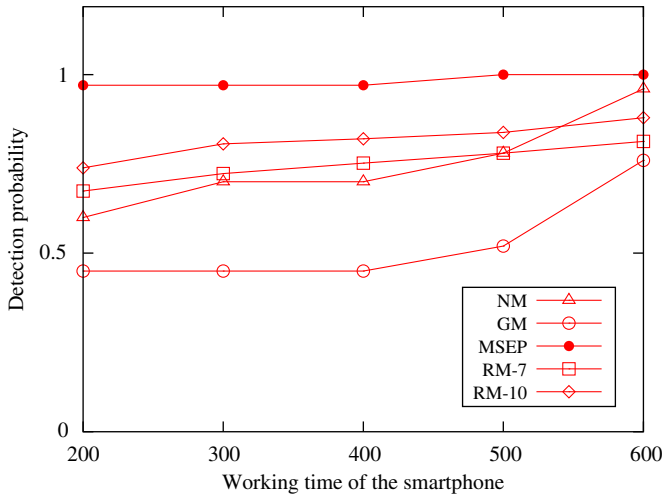


**Fig. 13.** Detection probability of AC1.

current degree, it will lead to local optimum rather than global optimum. GM is not an ideal method because it cannot always outperform NM.

In detail, for MSEP, the patrol time is 180 s, hence the smartphone can patrol the WSN once within 200 s and about three times within 600 s. Nevertheless, the patrol time of NM is 625 s, and hence NM can only patrol all the sensor nodes for about one time within 600 s. This results in a much lower detection probability of NM than MSEP when the working time of the smartphone is 200 s. So do GM, RM-7, and RM-10. When the working time is 600 s, all the methods can patrol the WSN for at least one time. Because AC1 and AC3 are long and frequent, they are easy to be discovered by all the methods with high probability. Compared with other methods, MSEP can save the working time a lot while still maintaining high detection probability.

Figure 15 plots the detection probability of AC2. Since AC2 is infrequent, the detection probability of AC2 is much lower than those of AC1 and AC3, especially when the smartphone working time is less than 500 s. Specifically, when the smartphone working time is less than 400 s, all methods can hardly collect enough packets to detect AC2. Generally, the detection probabilities of all the other methods are lower than that of MSEP. Unlike the case of AC1 and AC3, in the case of AC2, GM performs better than NM though it is still below RM-7 and RM-10.

Figure 16 statistically shows the detection probabilities of AC4, AC5, AC6, and AC7 when the working time is 200 s. Because they are short AC lasting for about 200 s, the smartphone only works for 200 s. Since AC4 is more frequent than AC5, the detection probability of AC4 is higher than that of AC5. AC6 and AC7 are infrequent AC while AC4 and AC5 are frequent AC, hence the detection probabilities of AC6 and AC7 are lower than those of AC4 and AC5. For all the short AC, MSEP achieves higher detection probability than the other methods. GM is better than NM for AC5 and AC6. It is worse for AC4 and AC7. The performance of RM-7 and RM-10 is similar. GM and NM are below the average level while MSEP is above the average level.

In summary, Fig. 17 demonstrates that the detection probability of all ACs for MSEP is higher than that of all the other methods. In addition, since more ACs exist, the detection probabilities of all approaches are high when the smartphone working time approaches 600 s. Furthermore, Fig. 17 shows that if MSEP is employed, the smartphone only needs to work for about 300 s to achieve high detection probability. When the ACs are not as frequent as the ACs in this experiment, using MSEP to work for a

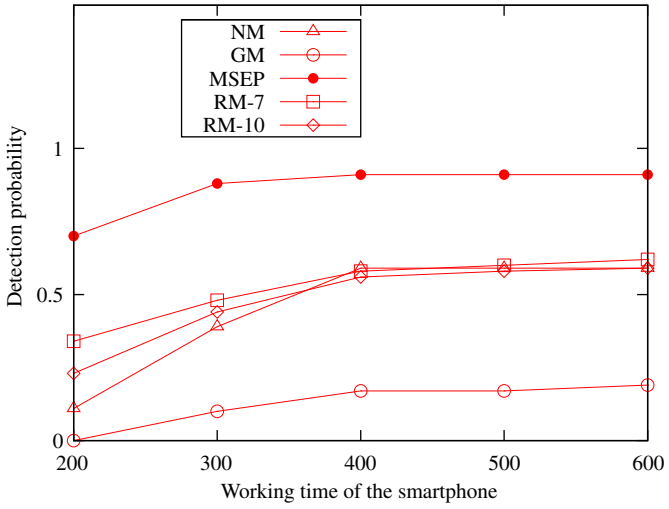


**Fig. 14.** Detection probability of AC3.

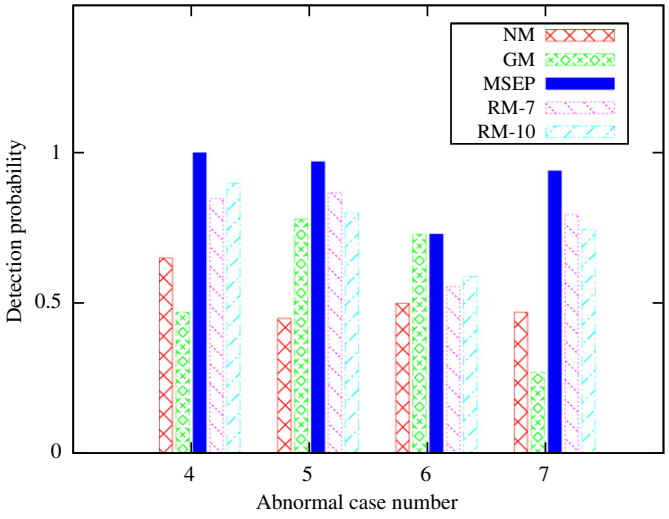


**Fig. 16.** Detection probability of AC4, AC5, AC6, and AC7 when the working time is 200 s.

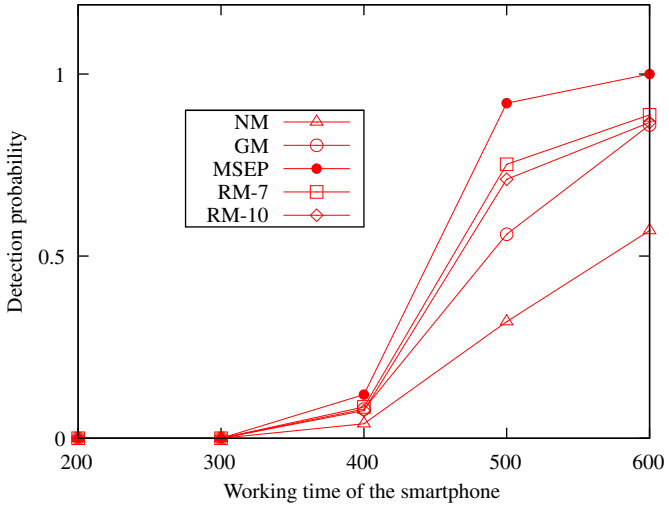


**Fig. 15.** Detection probability of AC2.

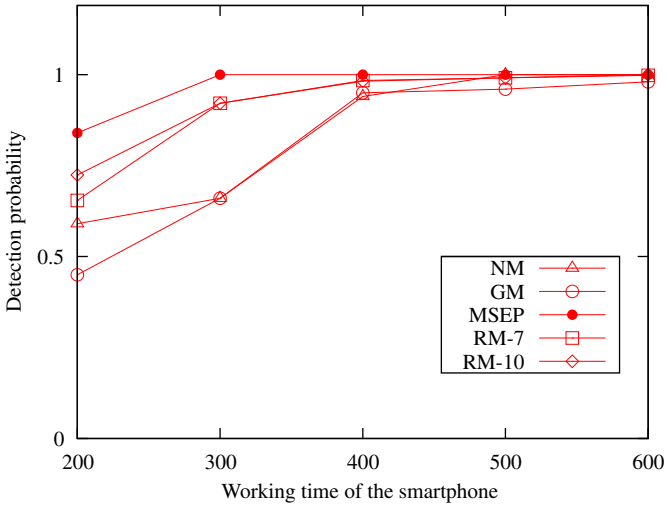


**Fig. 17.** Detection probability of all ACs.

longer time can increase the detection probability a lot as shown in Fig. 15. In a word, MSEP can reduce the smartphone patrol time and increase the failure detection rate.

## 7. Conclusions

In this paper, we propose a mobility-assisted diagnosis method called MDiag to diagnosis failures in WSNs with smartphones. The advantages of this approach are multi-fold: MDiag does not intrude the WSNs and is more efficient than deploying another network for diagnosis purpose. In addition, MDiag can help the BS find more failures because it can snoop all kinds of packets that are sent out. Aiming at the targets of all protocol layers, we design statistical rules to guide the abnormal phenomena determination. MSEP algorithm is further proposed to improve the detection rate and reduce the patrol time of MDiag. The permanent failure detection experiments demonstrate that MDiag can help discover more failures than BS-centralized methods. The experiments on short-term failure detection show that MSEP algorithm suits WSNs better than NM, GM, and baseline method RM-*K*.

## Acknowledgments

This work was substantially supported by the National Natural Science Foundation of China (Project No. 61100077), the National Basic Research Program of China (973 Project No. 2011CB302603), the Shenzhen Basic Research Program (Project No. JC201104220300A), and the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Nos. CUHK 415311 and N CUHK405/11).

## References

Akyildiz I, Su W, Sankarasubramaniam Y, Cayirci E. Wireless sensor networks: a surveyComputer Networks 2002;38:393–422.

Buettner M, Yee G, Anderson E, Han R. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In: Proceedings of the 4th international conference on embedded networked sensor systems (SenSys); 2006.

Demirbas M. A transactional framework for programming wireless sensor/actor networks. In: Proceedings of the international conference on information processing in sensor networks (IPSN); 2008.

Dunkels A, Gronvall B, Voigt T. Contiki—a lightweight and flexible operating system for tiny networked sensors. In: Proceedings of the 29th annual IEEE international conference on local computer networks (LCN); 2004.

Global mobile phone statistics, 〈http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats〉.

Gnawali O, Fonseca R, Jamieson K, Moss D, Levis P. Collection tree protocol. In: Proceedings of the 7th international conference on embedded networked sensor systems (SenSys); 2009.

Heo J, Gu B, Eo SI. Energy efficient program updating for sensor nodes with flash memory. In: Proceedings of the 2010 ACM symposium on applied computing; 2010.

Hui JW, Culler D. The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the 2nd international conference on embedded networked sensor systems (SenSys); 2004. p. 266–79.

Johnson DS. Approximation algorithms for combinatorial problems. In: Proceedings of the 5th annual ACM symposium on theory of computing; 1973.

Johnson D, Stack T, Fish R, Flickinger DM, Stoller L, Ricci R, et al. Mobile emulab: a robotic wireless and sensor network testbed. In: Proceedings of the IEEE international conference on computer communications (INFOCOM); 2006.

Karp RM. Reducibility among combinatorial problems; 1972.

Khan MMH, Luo L, Huang C, Abdelzaher T. SNTS: sensor network troubleshooting suite. In: Proceedings of the IEEE international conference on distributed computing in sensor systems (DCOSS); 2007.

Langendoen K, Baggio A, Visser O. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In: Proceedings of the international workshop on parallel and distributed real-time systems; 2006.

Levis P, Lee N, Welsh M, Culler D. TOSSIM: accurate and scalable simulation of entire tinyos applications. In: Proceedings of the 1st international conference on embedded networked sensor systems (SenSys); 2003. p. 266–79.

Levis P, Madden S, Polastre J, Szewczyk R, Woo A, Gay D, et al. TinyOS: An operating system for sensor networksAmbient Intelligence 2005;5:115–48.

Li P, Regehr J. T-Check: Bug finding for sensor networks. In: Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks (IPSN); 2010. p. 174–85.

Liu B, Brass P, Dousse O. Mobility improves coverage of sensor networks. In: Proceedings of the 6th international symposium on mobile ad hoc networking and computing (MobiHoc); 2005.

Liu K, Li M, Liu Y, Li M, Guo Z, Hong F. Passive diagnosis for wireless sensor networks. In: Proceedings of the 6th international conference on embedded networked sensor systems (SenSys); 2008. p. 1132–44.

Liu K, Ma Q, Zhao X, Liu Y. Self-diagnosis for large scale wireless sensor networks. In: Proceedings of the 30th IEEE international conference on computer communications (INFOCOM); 2011.

Luo L, He T, Zhou G, Gu L, Abdelzaher TF, Stankovic JA. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog. In: Proceedings of the 25th IEEE international conference on computer communications (INFOCOM); 2006.

Ngai EC-H, Huang H, Liu J, Srivastava MB. Oppsense: Information sharing for mobile phones in sensing field with data repositories. In: Proceedings of the IEEE communications society conference on sensor, mesh and ad hoc communications and networks (SECON); 2011.

Ramanathan N, Chang K, Kapur R, Girod L, Kohler E, Estrin D. Sympathy for the sensor network debugger. In: Proceedings of the 3rd international conference on embedded networked sensor systems (SenSys); 2005. p. 255–67.

Rensfelt O, Hermans F, Larzon L-A, Gunningberg P. Sensei-uu: A relocatable sensor network testbed. In: Proceedings of the 5th ACM international workshop on wireless network testbeds, experimental evaluation and characterization; 2010.

Reynolds P, Killian C, Wiener JL, Mogul JC, Shah MA, Vahdat A. Pip: detecting the unexpected in distributed systems. In: Proceedings of the 3rd symposium on networked systems design and implementation (NSDI); 2006.

Ringwald M, Romer K, Vitaletti A. Passive inspection of sensor networks. In: Proceedings of the IEEE international conference on distributed computing in sensor systems (DCOSS); 2007.

Romer K, Ma J. PDA: Passive distributed assertions for sensor networks. In: Proceedings of the 8th ACM/IEEE international conference on information processing in sensor networks (IPSN); 2009. p. 337–48.

Sasnauskas R, Landsiedel O, Alizai MH, Weisez C, Kowalewskiz S, Wehrle K. KleeNet: discovering insidious interaction bugs in wireless sensor networks before deployment. In: Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks (IPSN); 2010. p. 186–96.

Shi Y, Hou YT. Theoretical results on base station movement problem for sensor network. In: Proceedings of the IEEE international conference on computer communications (INFOCOM); 2008.

Telosb datasheet, 〈http://www.willow.co.uk/TelosB_Datasheet.pdf〉.

The network simulator version 3, 〈http://www.isi.edu/nsnam/ns/〉.

Xing G, Wang J, Shen K, Huang Q, Jia X, So HC. Mobility-assisted spatiotemporal detection in wireless sensor networks. In: Proceedings of the international conference on distributed computing system (ICDCS), Genova, Italy; 2008a. p. 784–94.

Xing G, Wang T, Jia W, Li M. Rendezvous design algorithms for wireless sensor networks with a mobile base station. In: Proceedings of the 9th international symposium on mobile ad hoc networking and computing (MobiHoc); 2008b.

Xu N, Rangwala S, Chintalapudi KK, Ganesan D, Broad A, Govindan R, et al. A wireless sensor network for structural monitoring. In: Proceedings of the 2nd international conference on embedded networked sensor systems (SenSys); 2004. p. 13–24.

Yang J, Soffa ML, Selavo L, Whitehouse K. Clairvoyant: a comprehensive source-level debugger for wireless sensor networks. In: Proceedings of the 5th international conference on embedded networked sensor systems (SenSys); 2007. p. 189–203.

Ye W, Heidemann J, Estrin D. An energy-efficient MAC protocol for wireless sensor networks. In: Proceedings of the 21st IEEE conference on computer communications (INFOCOM); 2002. p. 1567–76.

Zhou Y, Chen X, Lyu MR, Liu J. Sentomist: Unveiling transient sensor network bugs via symptom mining. In: Proceedings of the international conference on distributed computing system (ICDCS), Genova, Italy; 2010. p. 784–94.