Local Monitoring and Maintenance for Operational Wireless Sensor Networks

Md Zakirul Alam Bhuiyan[†‡], Guojun Wang[†*], Jiannong Cao[‡], and Jie Wu[§]
[†]School of Information Science and Engineering, Central South University, Changsha, Hunan, P. R. China, 410083
[‡]Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong
[§]Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
[*]Corresponding author: csgjwang@mail.csu.edu.cn

***Abstract*—One of the key mechanisms underlying a wireless sensor network (WSN) is to monitor the network itself. Many existing approaches perform centralized analysis and maintenance based on a large amount of status reports collected from the WSN, while others use add-on protocols/modules that not only require extra management cost but also interrupt the normal operations of targeted WSN applications. Unlike existing work, we propose LoMoM, a new approach of Local Monitoring and Maintenance for a WSN, which combines monitoring operations for the WSN with the operations of a mobile event monitoring, in a manner that is both energy- and latency-efficient. LoMoM includes a two-part monitoring architecture: a WSN and a 3G network. Our main interest is in the WSN-part, where we address two important issues: monitoring probable anomalies/faults of the nodes, and the link failures. To achieve the event monitoring efficiently, LoMoM conducts a prompt local maintenance when such a fault occurs. Fault and event detection status reports are observed by a remote monitoring center using the 3G network. Comprehensive evaluations via both simulations and real-world experiments are conducted to validate the effectiveness of LoMoM.**

***Keywords*-Wireless sensor networks; network monitoring; monitoring architecture; local maintenance; event monitoring**

I. Introduction

Wireless sensor networks (WSNs) have been employed in a wide range of applications, such as environmental monitoring, structural health monitoring military operations, and event detection [1], [2], [3]. In an event detection application, when a detecting sensor node becomes faulty, or the status report is dropped due to various reasons, e.g., link (and route) failures, the application performance can be greatly affected. Thus, monitoring these faults and detecting them with a low latency and low energy cost are of great significance.

Some existing monitoring approaches use add-on modules [4], [5], [6], while some other approaches propose debugging or evaluation tools, and need extra hardware for node monitoring [7], [8], which may not be efficient for online debugging. Besides, many approaches to diagnosing WSNs are generally centralized, also known as *sink-based* [4], [9], [7], [10], [8]. Moreover, those approaches are not feasible in practice for a large scale resource-constrained WSN, due to the following shortages. First, they require a large number of active nodes for the monitoring function. Second, the monitoring function is carried out separately, which should be, arguably, performed in conjunction with the normal operation of an application. Third, monitoring link behaviors are not seriously focused. Fourth, it is commonly assumed to deploy a PC close to the sink; it is, however, infeasible in practice [8].

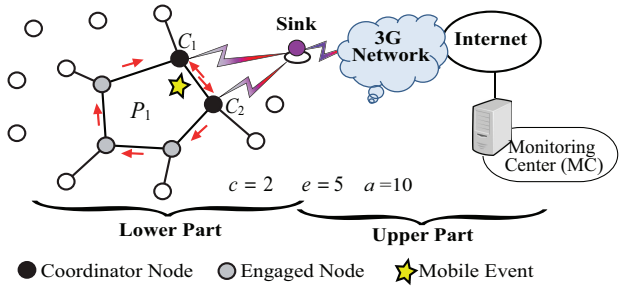

Fig. 1. The two-part monitoring architecture for WSNs, where the sensor nodes around a mobile event are monitored locally.

In response to the limitations above, in this paper we propose LoMoM. This is a novel approach to Local Monitoring and Maintenance for a WSN that performs monitoring operations for the WSN, together with the operations of a mobile event monitoring application. During the event monitoring, if there are any possible faults in the WSN, they are detected and repaired online in such a way that requires low energy cost and latency in the WSN. To achieve this, we design a two-part monitoring architecture (refer to Fig. 1) that includes two network infrastructures: a lower part (sensor nodes, coordinator nodes, and a sink) and an upper part (the sink, a 3G network, the Internet, and a monitoring center (MC)).

Our main interest is in the lower part, i.e., in the WSN. We consider a planar WSN graph embedded in the plane. The monitoring is built on decomposition of the plane. Such decomposition consists of non-overlapping polygonal-shaped forms (simply called polygons). An event is assumed to be surrounded by the perimeter of ith polygon P_i. In Fig. 1, the event lying inside P_1 can be detected as it goes across an edge (or a communication link), e.g., $<C_1, C_2>$. At the time of an edge crossing, the two nodes become *coordinators* and take the monitoring responsibilities. All the links and nodes associated with P_i are required to be monitored, meaning that the monitoring is carried out with the mobility of the event.

The problem includes two sub-problems. i) *node self-monitoring*—which enables each node to check for anomalies/faults in its own behaviors (i.e., diagnosing itself), to calculate a fault detection probability through an embedded algorithm, and to transmit it to the coordinators. ii) *Link monitoring*—which enables each node to probe for 1-hop neighbors and check the links to them, and to transmit a report to the coordinators along with the perceived behavior of the links. Besides, the coordinators have a chain of links between all the nodes in P_i. A link monitoring algorithm is

used to detect the scope of link failures. We apply Markov chain techniques in both self-monitoring and link monitoring algorithms. The sink is the final recipient of status reports, and informs the MC, who may decide to take action in response to the perceived faults. However, we enable the nodes to provide local maintenance to tolerate the faults. The MC can be located anywhere, it can even be a mobile device. To achieve this, a communication module is utilized for the upper part [11]. It is integrated into the sink, and connects the sink to the 3G network. The novel contributions of this paper are four-fold:

- We study the problem of local monitoring and maintenance for a WSN, and design a monitoring architecture in such a way that jointly offers monitoring for the WSN and one of its mobile event applications.
- We present an edge intersection algorithm to calculate the event detection probability that helps to select the coordinator nodes to lead the monitoring.
- We propose two monitoring algorithms, namely, node self-monitoring and link failure monitoring.
- We evaluate LoMoM via rigorous simulations and also a proof-of-concept system. Evaluation results, compared with state-of-the-art approaches [5], [6], [12], show that LoMoM greatly minimizes the propagation of false alarms and latency, and maximizes the WSN lifetime.

The rest of this paper is organized as follows. Section II provides related work. We formulate our problem in Section III. Section IV designs the monitoring architecture. The monitoring algorithms are presented in Section V. Local maintenance is discussed in Section VI. Sections VII evaluates LoMoM through simulations and a proof-of-concept system implementation. Finally, Section VIII concludes this paper.

II. Related Work

In most applications of WSNs, the WSNs create complexities of their own. They are vulnerable to faults, due to a wide range of reasons, e.g., hardware or software component faults, energy depletion, link failures, etc. A WSN system is required to monitor itself. Existing work focus on different aspects of monitoring [4], [6], [9], [7], [10], [8], [13], but there still remain challenges in local monitoring and maintenance.

Looking into most closely related approaches, the underlying ideas of Sympathy [4], Memento [5], and DiMo [6] are that sensors periodically report their status to the sink, such as residual energy, node failures, neighbor list, and the like. Memento controls node failures in the network, which is ensured by the use of *observer* or *detector* nodes to inform the sink when a failure is detected. The sink analyzes and makes all the decisions about node failures, and reconfigures the network. Another approach, PD (post-deployment debugging) is data-centric [8], and tries to pinpoint the root-causes of network performance problems.

Although above approaches are innovative, they have some serious drawbacks: (i) there may be no node failures in many cases, while the link failure detection is not distinctly considered; (ii) the monitoring is performed separately, rather than considered with the WSN application functions; (iii) each node directly communicates to the sink, which may be reliable; however, frequent report loss brings communication overhead; (iv) they support only centralized tolerance as faults occur.

An aggregation assisted monitoring scheme (or monitoring based on hop-by-hop aggregation is proposed in [12] to monitor sensor status, such as residual energy, which presents a poller-pollee structure (PP for short). The work tackles link failures only, and studies the problem of minimizing energy cost in monitoring as a poller-pollee assignment problem, and uses hop-by-hop aggregation. Since all of the nodes are monitored simultaneously, hop-by-hop aggregation may induce energy cost and latency in every hop.

Above observations have motivated us to develop LoMoM, where coordinators make local decisions on faults, and provide local maintenance. Basically, the nodes do not wait for the remote sink's decisions for monitoring, maintenance, and event detection. The WSN is treated as a distributed system, where, in contrast to existing work, it makes the monitoring a fully distributed process. As such, this can be considered a major step towards optimized operations in WSNs.

III. Problem Formulation

A. Preliminaries

We consider a WSN as the nodes embedded in a region in the plane, and an associated communication graph $G = (E, L)$. Here, $E(G)$ is the set of vertices/nodes and $L(G)$ is the set of edges/links. The communication links are bidirectional. A mobile entity, such as an enemy vehicle, an intruder, or a mobile device user is randomly moving in the plane.

We generate the subgraph G' that consists of one or more polygons. When the event lies inside the i-th polygon P_i or approaches P_i in the d-th time interval $T_d(d = 1, 2, ...)$, the event is detected by the nodes of P_i. This means that E_i of engaged nodes and L_i of links of P_i, and associated nodes and links to P_i are monitored by the coordinators (which are the nodes that are selected as the event is about to cross an edge $l_{i,j} \in L_i$ between them).

Whenever a fault is perceived in a node, the sink has the final status about it through the coordinators. The sink uses a filter to reduce the redundant information received from the network, as to extract the precise final status. The status reports (combined with the event detection status) are sent to a server, which is equipped with the monitoring center (MC). Thus, the latency of detection notification for an actual fault is calculated as follows: (i) lower part latency—the elapsed time for a node in the WSN to decide whether a fault has occurred or not, and the time for the sink to successfully store into the buffer that the status report has been received; (ii) upper part latency—the elapsed time for the sink to successfully receive that the fault has occurred in the WSN, and the time for the MC to store into the server that the final status is received.

B. Fault Model

In LoMoM, we focus on a set of faults in WSNs that may occur during event detection and monitoring:

- Node failure: A node can fail due to power depletion, especially, when it frequently uses its maximum power

level. A node might also fail by any kind of faults in local resources, such as memory, CPU.
- Malfunctioning/application flaws—some programs may fail to run, due to software errors or hardware malfunction, but they are still capable of routing information.
- Link failure: A permanent link failure may occur, for numerous reasons, e.g., environmental influence.

C. Fault Detection Model

We monitor the above faults in two situations, **S1** and **S2**.

S1. We trace the faulty reasons by characterizing their fault patterns. Generally, a node usually has different modes of operation (e.g., active, waking, and sleeping) [14]. For simplicity, a fault is mainly estimated on the node's active mode. The active mode may further consist of several process states. We divide the active mode into three consecutive process states (see Fig. 3 for an example). We assume that each process state consists of a lot of components. Some software components produce output interacting with small, different hardware components on the sensor board. This node can be monitored by the monitoring of its component's runtime behaviors (e.g.,"1" for a successful run). The process state automation is modeled as a DMC (discrete time semi-Markov chain). By means of DMC, a node estimates the fault detection probability denoted by β.

Definition 3.1 [Node status]. *An engaged node $v_i \in E_i$ is said to be failing if at least a process state of v_i is altered and the node transmits inconsistent values. In such a case, the fault detection probability is less than expected, e.g., ≤ 0.5.*

S2: An engaged node itself may fail before a report transmission. The node may transmit the report, but there is an incident of consecutive report loss. Unfortunately, a coordinator may also make a false detection, when it does not receive the sensor fault detection probability from a node.

Definition 3.2 [Link Status] *A link is said to be failing if (i) the link to/from a node fails, (ii) a node itself fails (i.e., all of its related links then definitely fail, and status is not transmitted) or the node is unreachable (the coordinators or the sink does not receive the status in a given time bound).*

Link failures in Definition 3.2 may occur from time to time, they can be either transient or permanent. A node can make a report about a link status by monitoring its associated links of P_i as a CMC (continuous time Markov chain), and calculate a link failure probability denoted by α. The false alarm rate denoted by $F_c(h, T_d)$ is then calculated in this situation as a performance metric, where c is the number of coordinators, h is the number of hops between the node (which detect the fault) and the destination node, and T_d is the time interval.

D. Energy Cost Model and Lifetime (T)

A practical parameter to evaluate WSN performance, when studying the energy cost, is the lifetime (T) [15], [16]. We consider that a node usually supports different-modes radio interface, e.g., active, waking, and inactive [14]. We think that, the amount of energy cost required for report transmission is the key characteristic for evaluating the performance of a WSN monitoring approach.

Let R be the maximum communication range of a sensor i and ω_{send} and ω_{rcev} be the energy cost for sending and receiving data, respectively. The energy cost (ω_i) of a sensor i depends on the communication distance between any two sensors i and j, a routing model, and the amount of traffic. We adopt the most widely used transmission model [15]:

$$\omega_{send}(R) = a_1 R^{\sigma} + b_1 \quad (1)$$

which can also be normalized as $\omega_{send}(R) = R^{\sigma} + b_2$. Here, σ is an exponent parameter that is between 2 and 6, and b_2 (or b_1) is a small constant comparing with R^{σ}, and $\omega_{recv}(R) = b_2$ [15]. Thus, the energy cost of sensor i is computed as:

$$\omega_i = \sum_{\forall r_{ij}} \lambda_{r_{ij}} \cdot \omega_{send}(R) + \sum_{\forall r_{ij}} \lambda_{r_{ij}} \cdot \omega_{recv}(R) + \omega_{rest} \quad (2)$$

where r_{ij} is the path from a source sensor i to the destination sensor j, $\lambda_{r_{ij}}$ is the amount of traffic that travel along path r_{ij} within one round of monitoring, and ω_{rest} is the rest of the energy cost of i in other operations, including event sensing, computing, and monitoring operations. We can find ω_m as the maximum energy cost on i-th node in the WSN, i.e., $\omega_m = \max_{i=1,2,\ldots,N} \omega_i$. We define the system lifetime T to be the total round of monitoring data collection (including monitoring for the WSN and the mobile event) before any of the sensor nodes in the WSN runs out of its energy [17]:

$$T = \omega_r / \omega_m \quad (3)$$

where ω_r is the residual energy on i-th sensor.

E. The Problem Definition

Given a WSN $G = (E, L)$, an external sink, an MC, and a mobile event of interest, find E_i and L_i that are required to be monitored from the nodes that detect the event as it approaches to a region of the WSN, find C_i from E_i that monitors E_i and L_i, such that C_i gathers monitoring status and reports to the sink, and the sink then reports to the MC. Our objective is to minimize $F_c(h, T_d)$ and detection latency, and to maximize T.

IV. Monitoring Architecture

In this section, we describe the monitoring architecture. Our main focus is on the lower (WSN)-part of the architecture. However, we briefly describe the upper part in Section VIII.

Planar graph, e.g., Voronoi diagram and related neighborhood graph (RNG), are mostly used in the network domain. Our monitoring is built on such graphs. Particularly, we apply RNG for our purpose [18], [19], [20]. A connected network subgraph $G' \subseteq G$ is drawn without crossing edges [18], [19], is not unidirectional and disconnected. Our focus is to monitor nodes and edges from around i-th polygon P_i of G' to nodes and edges around jth polygon P_j as an event goes across the common edge between P_i and P_j [21], [22], [3].

Definition 4.1 [In- and Out-adjacent Neighbors]. *The neighbors next to a specific node, which are directly connected, are called adjacent neighbors. There are two types of them: (i) the adjacent neighbors **in**side a polygon, where the event is moving, is called in-adjacent neighbors; (ii) the adjacent neighbors, which are **out**side of a polygon, are called out-adjacent neighbors.*

In the case of an event traversal problem through polygons, we see an example of the planar graph in Fig. 2. Let E_i be

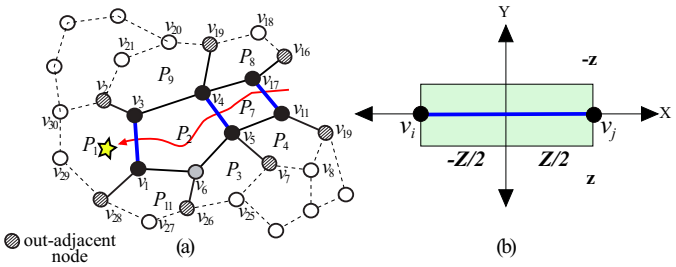

Fig. 2. Planarization techniques: a planar graph decomposed by polygons, showing the connected nodes and the edges between them that are associated to P_2; (a) event detection in a rectangular spot; (b) the entry of an event into P_i; both P_i and its edge intersection leaves a trail of edges.

the set of nodes in a polygon, i.e., $v_1, v_2, ..., v_p$, where $p \geq 3$. P_1 is a hexagon, P_2 is a pentagon, and P_7 is a tetragon. Each node in the plane has its neighboring nodes information after deployment. For example, node v_5 is connected to these nodes: (i) neighbors v_6, v_1, v_3, and v_4 in P_2; (ii) in-adjacent neighbors v_4 and v_6, and out-adjacent neighbors v_7 and v_{11}; (iii) nodes in P_7, P_4, P_3, and P_2, i.e., v_5 stores information about 4 polygons that are adjacent to it in G. Thus, a node corresponds to a number of polygons, depending on the polygon size.

The size of a polygon is defined by the number of nodes and edges around the polygon. Given a planar graph G, the average size of a polygon is given by:

$$\bar{P} \leq 2e/(e - l + 2) \tag{4}$$

where e and l are the numbers of nodes and edges of P_i, respectively. This can be achieved by the relationship between nodes, edges, and polygons as:

$$P_n + e - l = 2 \tag{5}$$

where P_n is the number of polygons corresponding to a node, according to Euler's formula [23]. This implies that our approach has cells for a planar graph, with as many edges as possible. We derive a bound for $\bar{P}$ in terms of e and l rather than l and P_n. It is expected because it is simple to count e and l in a graph, and it is not very obvious how to count P_n.

Wherever the event goes across an edge $l_{i,j}$ between two nodes v_i and v_j, it can be detected. Suppose that the event is moving from P_7 to P_2 and is detected by v_4 and v_5 (black nodes in Fig. 2(a)). They are selected as the *coordinators* due to the edge intersection probability as the event goes across $l_{4,5}$. The coordinators wake up other nodes in P_2 in advance. They become *engaged nodes* before the event moves to P_2. Let a and c denote the number of connected nodes corresponding to P_i and the coordinators, respectively. In Fig. 2(b), a = 12 (i.e., the number of out-adjacent neighbors =7 and $e = 5$). The 5 engaged nodes include 2 coordinators (i.e., $c = 2$).

One of the advantages of using coordinators in P_i is that one can minimize the number of nodes for an event monitoring, i.e., minimizing the number of nodes to be monitored. For example, v_5 has 4 adjacent polygons with 11 neighboring nodes, and node v_4 has 4 adjacent polygons with 12 neighboring nodes. However, edge $l_{4,5}$ between v_4 and v_5 has only 2 polygons with 7 nodes. Thus, the number of nodes is minimal. However, the coordinators (v_4 and v_5) only need to monitor 5 nodes (v_4,v_5,v_6,v_1, and v_3), including themselves in P_2. Each engaged node (e.g., v_6) can also monitor their out-adjacent neighbors, e.g., v_{26}. v_4 and v_5 maintain a chain of links ($v_5 \rightarrow v_6$, $v_6 \rightarrow v_1$, $v_1 \rightarrow v_3$, $v_3 \rightarrow v_4$, $v_4 \rightarrow v_5$, and vice versa). Intuitively, if one wishes, more nodes (e.g., all of the out-adjacent neighbors) can be allowed to detect an event so as to get a better monitoring for that event.

A. Edge Intersection Probability

In this subsection, we briefly describe the algorithm to find out the edge, $l_{i,j}$, between node v_i and node v_j so as to select v_i and v_j as the new coordinators. The edge $l_{i,j}$ is also the common edge between P_i and P_j.

A node calculates a probabilistic value that, in turn, shows *the performance of the event detection*. Suppose that an event is currently moving through P_i. The event must go across any $l_{i,j}$ of P_i. Every node of P_i is able to compute the probabilistic value. However, we need to find v_i and v_j, which are with the maximum value. Every two nodes in P_i compute a rectangular detection spot according to the length Z of $l_{i,j}$. Consider that P_i is mapped over the X-axis, as shown in Fig. 2(b). Let z and A be the width and area of the rectangular spot, respectively. Z is directly proportional to $2z$ and $\frac{Z}{2} \leq r_s$, where r_s is the sensor sensing range. The event moves from $(-z)$ to (z).

To find an approximate location $X(x_p, y_p)$ of the event, the intensity function of v_i is defined by a sensing model [24] as:

$$I(v_i, p) = \frac{1}{[d(v_i, p)]^K}, \qquad K > 1 \tag{6}$$

where the intensity $I(v_i, p)$ derived at node v_i of P_i is related to the approximate distance $d(v_i, p)$. The exponent K in (6) affects the degree of attenuation of the measured intensity in dependence of the distance. Let $X(x_p, y_p)$ be the approximate location coordinate of the event point p over $l_{i,j}$ and I_p be the edge intersection probability from any node v_i to p, such that node v_i has the following edge intersection probability:

$$\frac{1}{A} \int_{-Z/2}^{Z/2} I(v_i, p) dx \int_{-z}^{z} dy \tag{7}$$

In order to detect any mobile event at some point $X(x_p, y_p)$ inside polygon P_i, a node v_i should meet three conditions:

- The event must be inside P_i, or close to P_i, if P_i is in the outer boundary of the network region.
- v_i must be in the active mode when the event goes through $l_{l,j}$ along in the sensing range (r_s) of v_i.
- At least one of the in-adjacent neighbors of v_i in P_i should be in the active mode at the same time that has the similar value of the intensity function obtained by (6).

The edge intersection probability of the closest node v_i to the event point p completely relies on Z and the intersection of the sensing ranges of the two nodes.

V. Local Monitoring

A. Node Self-Monitoring Algorithm

The first fundamental task in local monitoring is to observe the node's own faults, i.e., study the **S1**. We consider a decision problem of how each node probes its own faults autonomously

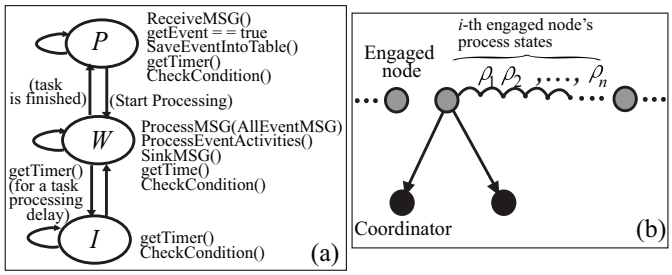

Fig. 3. (a) Some event detection tasks in the three process states, including the *CheckCondition()* function; (b) a finite set of processes of such a state as long as an event exists.

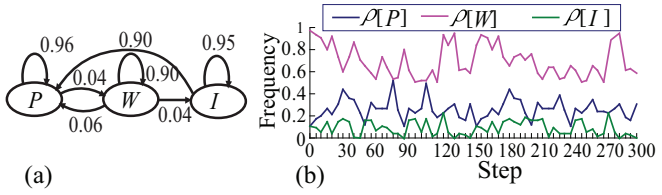

Fig. 4. (a) The outgoing probability (β) for the three states of an active nodes; (b) pmf samples obtained from a real experiment.

by monitoring its components. We address the problem as a semi-Markov decision process [25] and estimate the node fault detection probability (β). This is embedded into the node's processing core, so as to compute the probability.

In the algorithm, we define a set of necessary process states of nodes. Each such process state runs at time t_i and is involved in executing many tiny software or hardware components at discrete times $t_{i(1)}, t_{i(2)}$, and so on. If there is any fault at a specific component at some time (such as $t_{i(2)}$), this fault status can be known before the process state changes to another state. To detect a fault in advance, such a component is monitored individually. As discussed earlier, an application programmer can easily distinguish which behavior of a program, a script, or a hardware component belongs to which state, and is given a predicate (e.g., "1" for each successful run, "0" otherwise).

We show that each node is modeled by an embedded discrete time semi-Markov chain (DMC) for monitoring process states. A DMC is defined by *a set of states* and *transition probabilities* between the states. A semi-Markov decision process is one that changes states in accordance with a Markov chain, but takes a discrete time between changes. We focus on the node's active mode operations in a discrete-time manner.

Therefore, we divide the active mode into three consecutive process states: *Preprocessing (P)* is the initial state that waits or prepares for new tasks; *Working (W)* state that mainly processes the tasks; *Idle (I)* is the idle period of time during processing a task (as in Fig. 3(a) and 3(b)). When a process commences at time t_1, it is in the *P* state. We estimate the probability at each process state transition. Since the algorithm is embedded into a node to be executed in each discrete time, there is no need of any neighbor or the sink's interaction in the algorithm execution.

The ability to make such an independent decision saves a large amount of energy, because synchronization requires transmission among nodes in P_i or to the sink. Starting from a process in Λ of a certain state, when each node changes its state, β is calculated by executing a function CheckCondition(β) through the above procedures. Note that the tasks (e.g., as listed in Fig. 3(a)) for event detection run in parallel with CheckCondition(β). We measure the frequencies obtained by each process state execution of a node, and normalize these frequencies, so that the sum of the frequencies is 1. However, the frequency has a variance. The normalized frequencies are the point estimate of the probabilities. See Fig. 4 for the process state transition values that depict a probability, as well as the pmf samples obtained from our real experiment with 20 Imote2 sensors. Since an active state of a node is a DMC, the status of the node is known by $\beta = (\Lambda, Q)$.

Each coordinator aggregates received status reports. If β <0.5, a node is faulty. It possibly fails if β becomes lower as time goes on. β =0 indicates that a node fails. If a node fails, its in-adjacent neighbors in P_i do not receive the message. As a result, it can also report "0" to the coordinators by default. When a node detects its own faulty behavior, it may fail at a later time. By that time, the sink is made aware in advance about the faulty behaviors, as β becomes lower with time.

B. Link Monitoring Algorithm

The second fundamental task is to monitor link failures. Although **S1** hints at a future failure, it is still not sufficient for monitoring **S2**. A coordinator or the sink can make a decision on a link considering **S2**. Monitoring the link behavior is ignored in most existing work [4], [5], [6], [9], [7], [8], [13].

In order to calculate link failure detection probability, we apply the CMC (Continuous-time Markov Chain) technique. We consider that each link is a part of a chain between two nodes. Each coordinator has the *start* and the *end* of the chain of links. Each node individually monitors the links to its 1-hop neighbors, including the out-adjacent neighbors. The chain can be longer when a node uses multi-hop communication (especially, in case of a large polygon) and be in a continuous time, rather than discrete time. However, different from the DMC technique, there is no internal component tracking in the CMC. Each state (or node v_i) can talk to another state (node v_2) at some time. When v_1 can successfully talk, we say that there is a connection between the two nodes, and the link between them is available at the time.

A state is assumed as a node and state to state means a link $l_{i,j}$ or $<v_i, v_j>$ between nodes v_i and v_j. A link status between v_i and v_j is known by the transition. If there is a single link between v_i and v_j, failure of the single link will result in a false alarm, e.g., v_i sends a message to its 1-hop neighbor v_j. All the links on the route between v_i and v_j are checked by the CMC. We count how many hops are used on the route.

We assign a transition status for each chain. Let λ and μ be the transitions between node v_i to v_j, and node v_j to v_i, respectively. We assign statuses as follows. (i) For a single hop communication, "1" is for a direct successful chain from v_i to v_j and "1" is for v_j to v_i, otherwise, it is "0" for both directions. (ii) For a two-hop communication, "11" is for a successful chain from v_i to v_j; otherwise, it is "01" or "10"

for a broken link, i.e., there is an alternative chain available; (iii) "00" implies a link failure.

Let t be the time for reporting from an engaged node to a coordinator in T_d, and the sink needs to get a fault status. Thus, we have coordinators and engaged nodes ($e \geq c \geq 1$). Let h be the number of hops from an engaged node to a coordinator. Without loss of generality, we find the false alarm rate $F_c(h, T_d)$, and we take T_d/t as an integer.

In LoMoM, each node hits its neighbors and checks its link to the nodes in P_i and to the out-adjacent neighbors. In order to hit a link through a CMC, the detection probability denoted by α is set to calculate $F_c(h, T_d)$ for a link status given by:

$$\alpha = \mu(\lambda + \mu)^{-1} \tag{8}$$

We observe the link status between nodes in two cases.

Case 1: When an engaged node in P_i is monitored by a single coordinator ($c = 1$) and there is a single-hop ($h = 1$) from the coordinator to the engaged node or the sink. Let $s(t_0)$ denote the link state at t_0, $P[s(t + t_0) = 1|s(t_0) = 1]$ is the transition probability. Hence, $F_1(h, T_d)$ is computed by:

$$\begin{aligned} F_1(h, T_d) &= F_1(h, t)(P[s(t + t_0) = 1|s(t_0) = 1])^{(T_d/t)-1} \\ &= F_1(h, t)\left[\frac{P[s(t+t_0)=1, s(t_0)=1]}{P[s(t_0)=1]}\right]^{(T_d/t)-1} \\ &= F_1(h, t)\left[\frac{F_1(h,2t)}{F_1(h,t)}\right]^{(T_d/t)-1} \\ \text{where} \quad F_1(h, t) &= 1 - \alpha(t)^h \end{aligned} \tag{9}$$

Case 2: When the size of P_i is large, each node of P_i is monitored by two or more coordinators ($c \geq 2$) and there is a multi-hop ($h \geq 2$) from a coordinator to the engaged node or the sink. Hence, $F_c(h, T_d)$ is computed by:

$$F_c(h, T_d) = (1 - \alpha(t)^h)^{T_d/t} \tag{10}$$

VI. Discussion of Local Maintenance

A. Basic Fault Monitoring and Tolerance

Normally, the monitoring is carried out in the case of $c = 2$ in P_i. However, the monitoring is also carried out in the case of $c \geq 2$ in P_i, when the size of P_i is large (e.g., $e > 6$), due to (i) a highly dense or sparse network, (ii) some regions that are denser than others relying on the WSN deployment, and (iii) application environments. We also verify the case that if all of a sudden, $c = 1$ at some point of time, i.e., in the incidence of a coordinator failure detection. This is an extreme situation in LoMoM. At the moment when $c = 1$, another coordinator can detect it, can take up the role of the failed coordinator, and tolerate this situation by connecting an in-adjacent neighbor.

An engaged node schedules a report to a coordinator in every t. A coordinator gathers reports from each of its engaged nodes at every ct, and forwards (in the case of a confirmed fault) the report to the sink in every ct, $ct \leq T_d$. t varies from microseconds to seconds, depending on the event movement or on the WSN application requirements.

B. Discussion on Local Maintenance

Our approach is adaptive to network dynamics, and supports local maintenance as a link, or if a node fault occurs. If one or more nodes fail, or even if all of the engaged nodes in i-th P_i fail, that does not significantly affect the correctness of the performance of the mobile event application. Sensor fault and removal, connectivity hole, obstacle, etc., can be tackled with local operations, which does not require the sink's mediation.

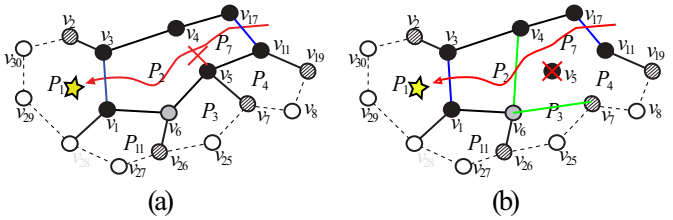

Fig. 5. An example of the local maintenance: single link failure; (b) single node failure.

See Fig. 5 for examples. Suppose that there is a permanent fault on a link $l_{i,j}$, e.g., $l_{4,5}$ in Fig. 5(a). The current coordinators (v_{17} and v_{11}) can monitor this failure, and can take actions on this failure. At least three options are available for calculating edge intersection probability (i.e., detecting the event further): (i) v_5 connects v_{17} in P_7; (ii) v_5 connects v_3 in P_2 if option (i) does not help, especially when the event moves faster; or (iii) v_5 communicates the neighbors in P_7 and P_2 to merge P_7 and P_2 into one, i.e., the nodes v_{17}, v_{11}, v_5, v_6, v_1, v_3, and v_4 are involved in the event detection. Consider that there may be a free space (or a hole) that can be the sensing area of a single node (say, v_5) failure (see Fig. 5(b)) as the coordinators (v_{17} and v_{11}) can monitor this failure. In this respect, our approach still has the ability to track the event. The surviving neighboring nodes, including v_4 (which would be a new coordinator if v_5 was not failed), and in- and -out adjacent neighbors are already in connection with each other. They adjust the polygon P_7 locally by creating links $l_{7,6}$ and $l_{6,4}$. Thus, the event is detected further.

VII. Performance Evaluation

A. Simulation

1) Methodology and Simulation Scenarios: In this section, we evaluate the benefits of LoMoM through extensive simulations. We use the OMNeT++ simulation tool. Our objectives in the simulations are on the following metrics:

- Study of the false alarm rate ($F_c(h, T_d)$).
- Fault detection rate—the *detection rate* is defined by the rate between the number of faults detected and the number of all faults occurred in a WSN.
- Detection latency (the upper and lower parts latency).
- Network lifetime (T), where T is normalized to 1 (one).

We consider two scenarios, which are set in different sizes and parameters. We think that the outcomes in the two scenarios may provide an insight into the performance of LoMoM in different large-scale WSN settings. Scenario I and Scenario II are comprised of 200 nodes and 800 nodes, respectively. In all of the simulation runs, the nodes are randomly distributed in 2D planar fields of 200m $\times$ 200m for Scenario I and 800m $\times$ 800m for Scenario II, respectively.

All transmitted status reports in the WSN are recorded by the sink. We enable the sink to use a filtering technique [26] to filter the redundant information before sending to the MC. Although the filtering is suggested only for event detection,

TABLE I
$F_c(h, T_d)$, h = 3 TO 5, DIFFERENT PPRS

PRR	80%	85%	90%	95%	98%
PP	0.061	0.041	0.0288	0.02	0.012
DiMo	0.113	0.091	0.071	0.05	–
Memento	0.151	0.122	0.107	0.071	–
LoMoM-C	0.043	0.029	0.019	0.011	0.007
LoMoM	0.025	0.017	0.009	0.006	0.004

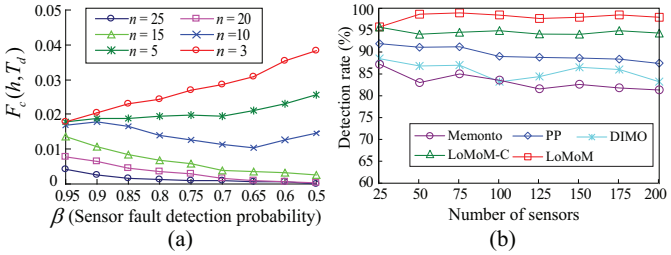

Fig. 6. (a) $F_c(h, T_d)$ vs. β; (b) the detection rate vs. N.

we implement it for both monitoring operations. When a node receives a message, it generates a message with an 50-bytes payload in Scenario I and an 80-bytes payload in Scenario II. $c \leq 3$, $n \leq 10$, $h \leq 6$, and $e \leq 6$ in Scenario I, and $c \leq 5$, $n \leq 16$, $h \leq 12$, and $e \leq 8$ in Scenario II. We model each sensor with six discrete power levels in the interval {-10dBm, 0dBm}, as the Imote2's power settings is tuned within the IEEE 802.15.4. The amount of energy cost required by the levels is between 11.2mA and 17.4mA.

At a fixed simulation time, we inject different types of faults (up to 20% of the nodes) randomly into the WSN, and let the nodes use our algorithms to generate detection reports. This invalidates the sensing and radio capabilities of 10% (5%+5%) of the nodes, and provides minimum power to 10% of the nodes so that they fail during runtime. We compare these nodes with the list of the nodes that have not been made faulty.

Comparison. We consider three prominent approaches, namely, Memento [5], PP [12], and DiMo [6], which are proposed for monitoring general WSNs. We also implement our approach in a semi-centralized (or semi-distributed) manner, called LoMoM−C, to see the performance of the approach in both semi-centralized and distributed scenarios.

2) Simulation Results: We first evaluate $F_c(h, T_d)$ in the five approaches under PRR (packet reception rate) between 80% and 99%, as shown in TABLE I. Then, we analyze $F_c(h, T_d)$ with different values of β in Fig. 6(a). Among all of the approaches, Memento clearly suffers from false alarms, due to a higher amount of packet losses. However, it improves this situation as PRR decreases, according to their technique for recovery. DiMo uses a method that is not supported by all the approaches, as it often allows multiple retransmissions for the observers (i.e.,faults detectors) and redundant relays. In LoMoM−C, the sink directly receives the monitoring status. There is no local coordinator; the sink analyzes whether or not there is a fault. LoMoM achieves more than 98% PRR and more than 96% detection accuracy when $h = 1$.

Now, we illustrate the results of faults detection in Fig. 6(b) under scenario I. We vary the network size from 25 nodes to 200 nodes. LoMoM successfully detects more than 98% of the

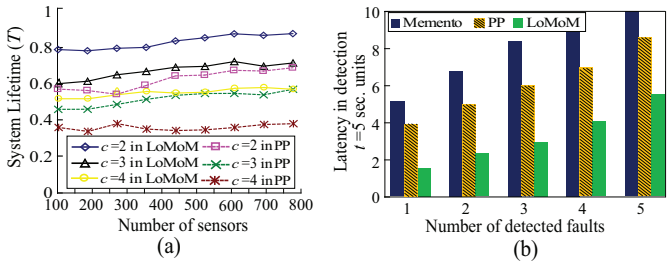

Fig. 7. (a) T vs. c; (b) latency between the fault occurrence and its detection.

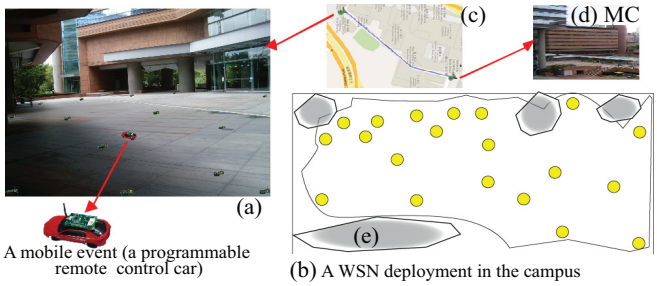

Fig. 8. Outdoor WSN deployment: (a) the deployment site; (b) Google map indicating the distance between the MC and the deployment site; (c) the location of the MC.

faults in most situations, the detection rate keeps stable as $e \geq 5$, and the number of sensors increases while the detection rate is slightly affected (from 3% to 4% on average) in LoMoM−C. PP and DiMo achieve detection rates between 90% and 85%, while Memento achieves more than 80% in a 55-node network.

Next, we examine T, as shown in Fig. 7(a). We can see that LoMoM maximizes T much more significantly (about 40% to 60%) than does PP. Also, an interesting observation is that sensor density does not affect T so much. Fig. 7(b) shows that the detection latency is affected in both Memento and PP, due to higher h, packet-loss, and reporting frequency, while LoMoM takes less time. In LoMoM, if a coordinator fails or the report is lost, the report from another coordinator is received at the sink. Moreover, the latency is decreased by the reduced number of acknowledgments.

B. Proof-of-Concept System Implementation

In this section, we describe the proof-of-concept system of LoMoM, via field studies in the HongKong PolyU campus, as shown Fig. 8. The experiments were conducted thrice over three different days. It lasted about 8 hours in each day. A programmable remote control car was employed as a mobile event, as shown in Fig. 8(a). We implement the system on top of the Intel Imote2, using TinyOS 2.0.1 [27]. We deploy 20 Imote2 plus 1 sink Imote2. We add some enhancement to the sink Imote2 for the upper part of the architecture: it is integrated by a type of communication module [11] that helps transmitting monitoring data to the MC over a 3G network.

The MC (a laptop) contains the server that stores the received reports. The distance between the locations of the MC (placed in our lab) and the deployment site, as shown in Fig. 8, is around 750 meters. We inject 4 types of faults into four sensors as follows. The communication fault (restarting radio) and sensing faults (invalidating sensing module) are injected

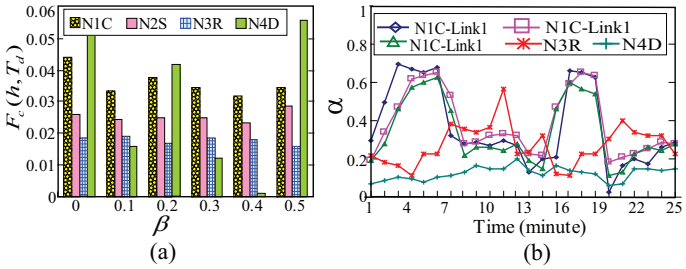

Fig. 9. (a) $F_c(h, T_d)$ vs. β; (b) achieved α (the link failure detection probability) at different times.

every two minutes into two sensor nodes (denoted by N1C and N2S) respectively. They are then mixed into the network. One sensor (N3R) is set to restart for every minute. Another sensor (N4D), preloaded with a local decision program, is set in a way in which it makes false decisions every three decisions. We mark the faulty nodes' locations. The car is moved around their vicinity of the locations. Each experiment is repeated ten times. Each sensor is allowed to record the fault detection packets (about β or α) in its buffer. We retrieve them later for the purpose of analysis.

Fig. 9(a) shows the estimated $F_c(h, T_d)$ vs. β. The sensor (N4D) shows abrupt changes at different times, since it provides wrong decisions on fault detection. Fig. 9(b) illustrates the four curves showing the probability of the three links' status associated with node N1C, and it also indicates the faults in those links. We can see that LoMoM correctly captures the periodical communication failures of N1C, It achieves more than 97% detection rate, and around 96% accuracy in PRR.

VIII. Conclusion and Future Work

We proposed LoMoM, a comprehensive local monitoring and maintenance approach for WSNs. Our novel contributions in this approach include: (i) the monitoring architecture, where the monitoring is performed online and from a remote place; (ii) both the node self-monitoring and the link monitoring techniques; (iii) the design of joint monitoring operations for a WSN and its mobile event application, where the monitoring for WSNs is carried out with the event mobility. Through simulations and proof-of-concept experiments, our approach shows the remote network monitoring facilities with energy- and latency-efficiency. Particularly, it achieves more than 97% detection rate and prolongs the system lifetime from 40% to 70%, while it doubles the event detection probability, compared to existing work. Investigating the performance of event monitoring and analyzing the costs of local maintenance by using polygons in WSNs will be our future work.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61272151 and 61272496, the International Science & Technology Cooperation Program of China under Grant Number 2013DFB10070, and the "Mobile Health" Ministry of Education - China Mobile Joint Laboratory (MOE-DST No. [2012]311). This work is also supported by HKRGC under GRF grant PolyU5106/11E and HK PolyU Niche Area Fund 1-BB6C, and by NSF under grants ECCS 1231461, ECCS 1128209, CNS 1138963, CNS 1065444, and CCF 1028167.

References

[1] G. Wang, M. Z. A. Bhuiyan, J. Cao, and J. Wu, "Detecting movements of a target using face tracking in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.91 2013.
[2] M. Z. A. Bhuiyan, J. Cao, G. Wang, and X. Liu, "Energy-efficient and fault-tolerant structural health monitoring in wireless sensor networks," in *Proc. of IEEE SRDS*, 2012, pp. 301–310.
[3] M. Z. A. Bhuiyan, G. Wang, J. Cao, and J.Wu, "Energy and bandwidth-efficient wireless sensor networks for monitoring high-frequency events," in *Proc. of IEEE SECON*, 2013.
[4] N. Ramanathan, K. Chang, R. Kapur, L. Girod, and E. Kohler, "Sympathy for the sensor network debugger," in *Proc. of ACM SenSys*, 2005.
[5] S. Rost and H. Balakrishnan, "Memento: A health monitoring system for wireless sensor networks," in *Proc. of IEEE SECON*, 2006.
[6] A. Meier, M. Motani, S. Hu, and K. Simon, "DiMo: Distributed node monitoring in wireless sensor networks," in *Proc. of MSWiM*, 2008.
[7] D. Yu, "DiF: A diagnosis framework for wireless sensor networks," in *Proc. of IEEE INFOCOM*, 2010.
[8] Z. Chen and K. G. Shin, "Post-deployment performance debugging in wireless sensor networks," in *Proc. of IEEE RTSS*, 2009.
[9] K. Romer and J. Ma, "PDA: Passive distributed assertions for sensor networks," in *Proc. of ACM/IEEE IPSN*, 2009.
[10] K. Liu, M. Li, Y. Liu, M. Li, Z. Guo, and F. Hong, "Passive diagnosis for wireless sensor networks," in *Proc. of ACM SenSys*, 2008.
[11] D. Pan, Y. Yuan, D. Wang, Y. Peng, and X. Peng, "Demo: A long-range high-rate communication module for Imote2," in *Proc. of IEEE INFOCOM*, 2011.
[12] C. Liu and G. Cao, "Distributed monitoring and aggregation in wireless sensor networks," in *Proc. of IEEE INFOCOM*, 2010.
[13] K. Liu, Q. Ma, X. Zhao, and Y. Liu, "Self-diagnosis for large scale wireless sensor networks," in *Proc. of IEEE INFOCOM*, 2011.
[14] M. Z. A. Bhuiyan, G. Wang, and J. Wu, "Target tracking with monitor and backup sensors in wireless sensor networks," in *Proc. of IEEE ICCCN*, 2009, pp. 1–6.
[15] S. Olariu and I. Stojmenovic, "Design guidelines for maximizing lifetime and avoiding energy holes in sensor networks with uniform distribution and uniform reporting," in *Proc. of IEEE INFOCOM*, 2006.
[16] I. Dietrich and F. Dressle, "On the lifetime of wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 5, no. 1, pp. 1–39, . 2009.
[17] A. Sankar and Z. Liu, "Maximizing lifetime routing in wireless ad hoc networks," in *Proc. of IEEE INFOCOM*, 2004.
[18] B. Leong, S. Mitra, and B. Liskov, "Path vector face routing: Geographic routing with local face information," in *Proc. of IEEE ICNP*, 2005.
[19] J. Cartigny, F. Ingelrest, D. Simplot, and I. Stojmenovic, "Localized LMST and RNG based minimum-energy broadcast protocols in Ad Hoc networks," *Ad hoc Networks (Elsevier)*, vol. 3, no. 2, pp. 1–16, 2005.
[20] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker, "Lazy cross-link removal for geographic routing," in *Proc. of ACM SenSys*, 2006.
[21] G. Wang, M. Z. A. Bhuiyan, and L. Zhang, "Two-level cooperative and energy-efficient tracking algorithm in wireless sensor networks," *Wiley's Concurrency and Computation: Practice & Experience*, vol. 22, no. 4, pp. 518–537, 2010.
[22] M. Z. A. Bhuiyan, G. Wang, and J. Wu, "Polygon-based tracking framework in surveillance wireless sensor networks," in *Proc. of IEEE ICPADS*, 2009, pp. 174–181.
[23] M. D. Berg, M. V. Kerveid, M. Overmars, and O. Schwarzkof, *Computational Geometry: Algorithms and Applications*. Heidelberg: Springer-Verlag, 2008.
[24] M. Waelchli, M. Scheidegger, and T. Braun, "Intensity-based event localization in wireless sensor networks," in *Proc. of IFIP WONS*, 2006.
[25] S. M. Ross, *Stochastic Processes*. New York: Jonn Wiley and Sons Inc., 1996.
[26] S. Lai, J. Cao, and X. Fan, "TED: Efficient type-based composite event detection for wireless sensor networks," in *Proc. of IEEE DCOSS*, 2011.
[27] http://docs.tinyos.net/index.php/Imote2.