

Chapter 10

Fault-Tolerant Algorithms/Protocols in Wireless Sensor Networks

Hai Liu, Amiya Nayak, and Ivan Stojmenović

Abstract Wireless sensor networks (WSNs) have wide variety of applications and provide limitless future potentials. Nodes in WSNs are prone to be failure due to energy depletion, hardware failure, communication link errors, malicious attack, and so on. Therefore, fault tolerance is one of the critical issues in WSNs. The chapter investigates current research work on fault tolerance in WSNs. We study how fault tolerance is addressed in different applications of WSNs. Five categories of applications are discussed: node placement, topology control, target and event detection, data gathering and aggregation, and sensor surveillance. In each category, we focus on the representative research works that presented algorithms and approaches in application layer to achieve fault tolerance.

10.1 Introduction

10.1.1 Background

Wireless sensor networks (WSNs) have received significant attention in recent years due to their potential applications in military sensing, wildlife tracking, traffic surveillance, health care, environment monitoring, building structures monitoring, etc. WSNs can be treated as a special family of wireless ad hoc networks. A WSN is a self-organized network that consists of a large number of low-cost and low-powered sensor devices, called sensor nodes, which can be deployed on the ground, in the air, in vehicles, on bodies, under water, and inside buildings. Each sensor node is equipped with a sensing unit, which is used to capture events of interest, and a wireless transceiver, which is used to transform the captured events back to the base

H. Liu (✉)
Department of Computer Science, Hong Kong Baptist University, Hong Kong
e-mail: hliu@comp.hkbu.edu.hk

station, called sink node. Sensor nodes collaborate with each other to perform tasks of data sensing, data communication, and data processing.

Nodes in WSNs are prone to failure due to energy depletion, hardware failure, communication link errors, malicious attack, and so on. Unlike the cellular networks and ad hoc networks where energy has no limits in base stations or batteries can be replaced as needed, nodes in sensor networks have very limited energy and their batteries cannot usually be recharged or replaced due to hostile or hazardous environments. So, one important characteristic of sensor networks is the stringent power budget of wireless sensor nodes. Two components of a sensor node, sensing unit and wireless transceiver, usually directly interact with the environment, which is subject to variety of physical, chemical, and biological factors. It results in low reliability of performance of sensor nodes. Even if condition of the hardware is good, the communication between sensor nodes is affected by many factors, such as signal strength, antenna angle, obstacles, weather conditions, and interference.

Fault tolerance is the ability of a system to deliver a desired level of functionality in the presence of faults [8]. Since the sensor nodes are prone to failure, fault tolerance should be seriously considered in many sensor network applications. Actually, extensive work has been done on fault tolerance and it has been one of the most important topics in WSNs. An early survey work can be found in [18]. However, its coverage is very limited and its references are outdated. The objective of the chapter is to investigate current research work on fault tolerance in WSNs. We study how fault tolerance is addressed in different applications of WSNs. More specifically, we address five categories of applications: node placement, topology control, target and event detection, data gathering and aggregation, and sensor surveillance. In each category, we focus on the representative research works that presented algorithms and approaches in application layer to achieve fault tolerance.

In the rest of this section, we discuss how faults happen in different levels of WSNs, and briefly introduce fault detection and recovery strategies. After that, organization of the chapter is as follows. Node placement in two-tiered WSNs is discussed in Sect. 10.2. Fault tolerance in topology control is introduced in Sect. 10.3. Target detection and event detection are introduced in Sect. 10.4. Data gathering and data aggregation are discussed in Sect. 10.5. Sensor surveillance is studied in Sect. 10.6. We conclude the chapter in Sect. 10.7.

10.1.2 Fault Tolerance at Different Levels

Five levels of fault tolerance were discussed in [18]. They are physical layer, hardware layer, system software layer, middleware layer, and application layer. On the basis of study, we classify fault tolerance in WSNs into four levels from the system point of view. More specifically, fault tolerance in a WSN system may exist at hardware layer, software layer, network communication layer, and application layer.

10.1.2.1 Hardware Layer

Faults at hardware layer can be caused by malfunction of any hardware component of a sensor node, such as memory, battery, microprocessor, sensing unit, and network interface (wireless radio). There are three main reasons that cause hardware failure of sensor nodes. The first is that sensor networks are usually for commercial use and sensor nodes are cost sensitive. Therefore, design of a sensor node will not always use the highest quality components. The second is that strict energy constraints restrict long and reliable performance of sensor nodes. For example, sensor readings may become incorrect when the battery of a sensor node reaches a certain level [26]. The third is that sensor networks are often deployed in harsh and hazardous environments, which affect normal operation of sensor nodes. The wireless radios of sensor nodes are severely affected by these environment factors.

10.1.2.2 Software Layer

Software of a sensor node consists of two components: system software, such as operating system, and middleware, such as communication, routing, and aggregation. An important component of system software is to support distributed and simultaneous execution of localized algorithms. Software bugs are a common source of errors in WSNs. One promising method is through software diversity where each program is implemented in several different versions. Since it is difficult to provide fault tolerance in an economic way at hardware level of a sensor node, numerous fault-tolerant approaches are expected at the middleware level. The majority of current applications in WSNs are simple. To adapt the real-life applications, there is a need to develop much more complex middleware for WSNs.

10.1.2.3 Network Communication Layer

Faults at network communication layer are the faults on wireless communication links. Assuming that there is no error on hardware, link faults in WSNs are usually related to surrounding environments. In addition, link faults can also be caused by radio interference of sensor nodes. For example, node a can not successfully receive a message from node b if node a is within interference range of other nodes that are transmitting messages at the same time. The standard way to enhance the performance of wireless communication is to use aggressive error correction schemes and retransmission. These two methods may cause further delay of operation. It should be pointed out that there is always a trade-off between fault tolerance and efficiency.

10.1.2.4 Application Layer

Fault tolerance can be addressed also at the application layer. For example, finding multiple node-disjoint paths provides fault tolerance in routing. The system

can switch from an unavailable path with broken links to an available candidate path. However, an approach for fault tolerance in an application can not be directly applied to other applications. It requires proper addressing of fault tolerance in different applications, on a case by case basis. On the other side, fault tolerance in application level can be used to address faults in essentially any type of resource.

10.1.3 Fault Detection and Recovery

To tackle faults in a WSN, the system should follow two main steps. The first step is fault detection. It is to detect that a specific functionality is faulty, and to predict it will continue to function properly in the near future. After the system detects a fault, fault recovery is the second step to enable the system to recover from the faults.

Basically, there are two types of detection techniques: *self-diagnosis* and *cooperative diagnosis*. Some faults that can be determined by a sensor node itself can adopt self-diagnosis detection. For example, faults caused by depletion of battery can be detected by a sensor node itself. The remaining battery of the sensor node can be predicted by measuring current battery voltage. Another example is the detection of failure links. A sensor node may detect that some link to one of its neighbors is faulty if the node does not receive any message from the neighbor within a predetermined interval. However, there are some kinds of faults that require cooperative diagnosis among a set of sensor nodes. A large portion of faults in WSNs are in this category. For example, detection method proposed in [19] is to identify faulty sensor nodes in event detection application. The detection method is based on the assumption that sensor nodes in the same region should have similar sensed value unless a node is at the boundary of the event region. The method takes measurements of all neighbors of a node and uses the results to compute the probability of the node being faulty.

The most commonly used technique for fault recovery is replication or redundancy of components that are prone to be failure. For example, WSNs are usually used to periodically monitor a region and forward sensed data to a base station. When some nodes fail to provide data, the base station still gets sufficient data if redundant sensor nodes are deployed in the region. Multiple paths routing is another example. In the case of providing single route, a requested call can not be set up or be maintained if some nodes/links along the route fail. Keeping a set of candidate routes provides high reliability of the routes for routing. It requires K -connectivity of the network if it is able to tolerate failure of $K-1$ nodes.

Fault recovery mechanism in single-hop sensor networks was studied in [5]. The proposed fault recovery scheme is to deal with failure of sensor nodes, including the sink node. The basic idea is to partition the sensor memory into two parts, namely, data memory and redundant memory. The data memory is used to store sensed data and data recovered from failures of other sensor nodes. The redundant memory is used to store redundant data for future recovery. The recovered data is distributed

in the memories of the nonfaulty sensors to be sent to the sink when it becomes available. It shows that the memory overhead is $(n + 1)/n$, provided the total number of sensor nodes in the network is n .

10.2 Node Placement in Two-Tiered Wireless Sensor Networks

Since sensor nodes are prone to failure, one approach to improving reliability and prolonging lifetime of WSNs is the introduction of two-tiered network architecture. The architecture employs some powerful relay nodes whose main function is to gather information from sensor nodes and relay the information to the sink. That is, relay nodes serve as a backbone of the network. The relay nodes are more powerful than sensor nodes in terms of energy storage, computing, and communication capabilities. The network is partitioned into a set of clusters. The relay nodes act as cluster heads and they are connected with each other to perform the data forwarding task. Each cluster has only one cluster head and each sensor belongs to at least one cluster, such that sensor nodes can switch to backup cluster heads when current cluster head is not available. In each cluster, sensor nodes collect raw data and report to the cluster head. The cluster head analyzes the raw data, extracts useful information, and then generates outgoing packets with much smaller size to the sink via multihop paths.

A fault in transmitter can cause the relay nodes to stop transmitting tasks to the sensors as well as relaying the data to the sink. Data sent by the sensors will be lost if the receiver of a relay node fails. So, a communication link fault on a sensor requires the sensor to be reallocated to other cluster heads within communication range. If faults occur in intercluster heads, the two corresponding cluster heads should be reconnected by another multihop path. Therefore, in order to handle general communication faults, there should be at least two node-disjoint paths between each pair of relay nodes in the network.

An intuitive objective of relay node placement in two-tiered WSNs is to place the minimum number of relay nodes, such that some degree of fault tolerance can be achieved. A lot of work has been done on the minimum placement of relay nodes for fault tolerance in two-tiered WSNs [14, 15, 22, 28]. There are other works that study placement of sensor nodes to make a sensor network k -connected, such as [2]. It does not employ relay nodes and two-tiered architecture. However, it can be reduced to the same placement problem in two-tiered architecture by setting uniform communication ranges for both sensor nodes and relay nodes. So, we focus on relay node placement problem in two-tiered networks in this section.

There are variant definitions on the problem of minimum placement of relay nodes. Generally speaking, the problem can be described as follows. Given a set of sensor nodes that are randomly distributed in a region and their location, some relay nodes are needed to be placed on the region for forwarding data to the sink, such that each sensor node is covered by at least one relay node. The objective is to minimize the number of relay nodes that make the network k -connected (usually 2-connected is desirable).

Work in [15] assumes that the original sensor network is 2-connected and sensor nodes also participate in forwarding of the data. The objective is to guarantee that each sensor node is covered by at least two relay nodes and the network of relay nodes is 2-connected. The problem was shown to be an extension of Relay Node Double Cover problem, which has been proved to be NP-complete [11]. A polynomial time approximation algorithm was proposed. It was proved that performance of the proposed algorithm is bounded within $O(D \log n)$, where n is the number of sensor nodes in the network and D is the diameter of the network, which was defined in [15]. However, the assumption that the original sensor network is 2-connected is too strong to be applied in real applications. Moreover, it assumes that sensor nodes participate in forwarding task. Since sensor nodes usually have limited computing and communication capability, and especially very limited energy resource, it restricts application of the algorithm.

Work in [22] does not require earlier assumptions. Formal description of the problem is as follows: given a set of sensor nodes S in a region and a uniform communication radius d , the problem is to place a set of relay nodes R , such that (1) the whole network G is connected and (2) G is 2-connected. The objective of the problem is to:

$$\text{Minimize } |R|,$$

where $|R|$ denotes the number of relay nodes in R .

The authors proposed a $(6 + \varepsilon)$ -approximation solution for the case 1 of the minimum relay node placement problem (MRP-1 for short), and then proposed a $(24 + \varepsilon)$ -approximation solution for case 2 (MRP-2 for short), where ε is an arbitrary positive number and running time is polynomial when ε is fixed. The solutions were further extended to the scenario where communication radii of sensor nodes and relay nodes are different. The basic idea of the solutions is to partition the problem into two phases. The first phase is to place some relay nodes to cover all sensor nodes. The second phase is to add more relay nodes to make the whole network connected/2-connected.

The solution is based on two fundamental works. The first is the *covering with disks* problem. Given a set of points in the plane, the problem is to identify the minimum set of disks with prescribed radius to cover all the points. In [16], a polynomial time approximation scheme (PTAS) for this problem was proposed. That is, for any given error $\varepsilon \geq 0$, the ratio of the solution found by the scheme to the optimal solution is not larger than $(1 + \varepsilon)$. The running time is polynomial when ε is fixed. The scheme was called *min-disk-cover scheme*.

The other fundamental work is the *Steiner tree problem with minimum number of Steiner points* (STP-MSP). Given a set of terminals in the Euclidean plane, the problem is to find a Steiner tree such that each edge in the tree has length at most d and the number of Steiner points is minimized. Du et al. proposed a 2.5-approximation algorithm for the STP-MSP [10]. The algorithm was called *STP-MSP algorithm*. Note that sensor nodes do not participate in data forwarding. STP-MSP algorithm cannot be directly applied to the problem.

Based on earlier foundational works, the $(6 + \varepsilon)$ -approximation algorithm for MRP-1 is as follows.

Algorithm 1:

Input: S , a set of sensor nodes with locations.

ε , any given error that is larger than 0.

d , the communication radius of sensor nodes and relay nodes.

Output: G , a connected network including sensor nodes and relay nodes.

1. Use the min-disk-cover scheme to place a set of relay nodes R_1 , such that for $\forall s \in S, \exists r \in R_1$, and r covers s .
2. Use R_1 as an input of the STP-MSP algorithm to place additional relay nodes R_2 , such that G is connected.
3. Output G and the position of each relay node.

Since following theorem is the core technical part of the solution, introduction of the solution is incomplete without the theorem. We make a simple version on the original proof [22].

Theorem 1. *Let R be the solution computed by algorithm 1 and R^{opt} be the optimal solution to MRP-1. Then $\frac{|R|}{|R^{\text{opt}}|} \leq (6 + \varepsilon)$.*

Proof. Let R_1^{opt} denote the minimum set of relay nodes that cover S . Since R_1 is the solution of PTAS, thus

$$|R_1| \leq (1 + \varepsilon)|R_1^{\text{opt}}|. \tag{10.1}$$

Let R_2^{opt} denote the minimum set of relay nodes that make R_1 connected. Since R_2 is the solution of the 2.5-approximation algorithm, thus

$$|R_2| \leq 2.5|R_2^{\text{opt}}|. \tag{10.2}$$

For any $r \in R_1$, there must be at least one sensor node s , which is covered by r (Otherwise r can be removed from R_1). Consider the communication circle of s (see Fig. 10.1), there exists $v \in R^{\text{opt}}$, such that both r and v can cover s . That is, relay nodes r and v are both in the communication circle of sensor s . Thus, $d(r, v) \leq 2d$. An additional relay node u is placed in the middle point of line $v \rightarrow r$. Thus, $d(u, v) = d(r, u) \leq d$. It means node v can communicate with node r via node u . Therefore, for any relay node $r \in R_1$, an additional relay node is placed according to the earlier description, such that R^{opt} can communicate with every relay

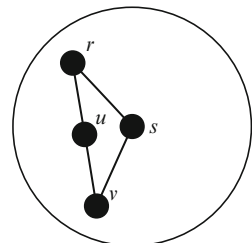


Fig. 10.1 Communication circle of sensors

node in R_1 . That is, R^{opt} and these added relay nodes make R_1 connected. Note that R_2^{opt} is the minimum set of relay nodes that make R_1 connected, and the number of added relay nodes is equal to $|R_1|$. Therefore,

$$|R_2^{\text{opt}}| \leq |R^{\text{opt}}| + |R_1|. \tag{10.3}$$

According to (10.1)–(10.3), the total number of relay nodes placed by the algorithm to MRP-1 is as follows:

$$\begin{aligned} |R_1| + |R_2| &\leq (1 + \varepsilon)|R_1^{\text{opt}}| + 2.5(|R^{\text{opt}}| + |R_1|) \\ &\leq (1 + \varepsilon)|R^{\text{opt}}| + 2.5(|R^{\text{opt}}| + (1 + \varepsilon)|R^{\text{opt}}|) \\ &\leq 6|R^{\text{opt}}| + 3.5\varepsilon|R^{\text{opt}}| \end{aligned}$$

That is, $\frac{|R|}{|R^{\text{opt}}|} = \frac{|R_1| + |R_2|}{|R^{\text{opt}}|} \leq (6 + \varepsilon)$.

The $(24 + \varepsilon)$ -approximation algorithm for MRP-2 is as follows. The basic idea is to add additional relay nodes to the connected network to make it 2-connected. \square

The approximation ratio of algorithm 2 is $(24 + \varepsilon)$. Detailed proof can be found in [22].

Algorithm 2:

Input: S , a set of sensor nodes with locations.

ε , any given error that is larger than 0.

d , the communication radius of sensor nodes and relay nodes.

Output: G , a 2-connected network including sensor nodes and relay nodes.

1. Run algorithm 1 to get a set of relay nodes R , such that $S + R$ is connected.
2. Add three backup nodes in the communication circle of each $r \in R$ as in Fig. 10.2. The set of all backup nodes in this step is denoted by R' .
3. Output G and positions of relay nodes in $R + R'$

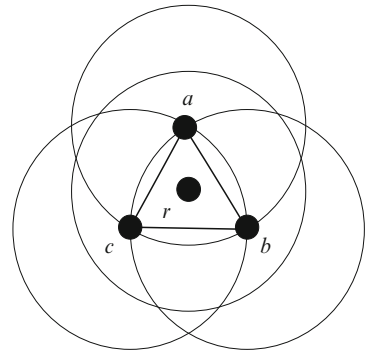


Fig. 10.2 Adding backup nodes to the communication circle of r

The works in [15] and [22] were integrated and further extended in [28]. It studied four types of fault-tolerant relay node placement problems. They are single-tiered placement with/without base stations, and two-tiered node placement with/without base stations. In single-tiered model, an edge may exist between any two types of nodes. That is, both sensor nodes and relay nodes participate in packet forwarding in single-tiered model, while only relay nodes participate in packet forwarding in two-tiered model. For each problem, a polynomial constant approximation algorithm was proposed. The ratio of performance either is smaller than, or is the same as that of the previous-best. Four problems and their corresponding algorithms will be introduced one by one.

The basic technique is the use of *steinerization*. Suppose x_i and x_j are two sensor nodes. R and r are the communication ranges of relay nodes and sensors, respectively. If $d(x_i, x_j) \leq r$, x_i and x_j can directly communicate with each other. Otherwise, x_i and x_j can be connected by deploying a minimum number of relay nodes on the line segment $[x_i, x_j]$ in the following way (called steinerizing $[x_i, x_j]$).

- If $d(x_i, x_j) \in (r, 2r]$, place one relay node at the midpoint of line segment $[x_i, x_j]$.
- If $d(x_i, x_j) > 2r$, place $1 + \lceil (d(x_i, x_j) - 2r)/R \rceil$ relay nodes on the line segment $[x_i, x_j]$, such that two relay nodes, say y_i and y_j , are at distance r from x_i and x_j , respectively, and the other $\lceil (d(x_i, x_j) - 2r)/R \rceil - 1$ relay nodes are evenly distributed on the line segment $[y_i, y_j]$.

Given $R \geq r > 0$ and a set of sensor nodes X , an edge weighted undirected complete graph $G^S(r, R, X)$, called the *steinerized graph* of (r, R, X) , consists of vertex set $V = X$ and edges with weight defined as follows:

$$c(x_i, x_j) \begin{cases} 0, & \text{if } d(x_i, x_j) \in [0, r]; \\ 1, & \text{if } d(x_i, x_j) \in (r, 2r]; \\ 1 + \lceil (d(x_i, x_j) - 2r)/R \rceil & \text{otherwise.} \end{cases} \quad (10.4)$$

Actually, the weight on each edge is the number of relay nodes needed to connected end nodes of the edge.

The approximation ratio of algorithm 3 is 14. Detailed proof can be found in [28]. The algorithm for single-tiered placement with base stations is similar to algorithm 3. The only difference is that it constructs the steinerized graph $G^S(r, R, B, X)$ in step 2, where B is the set of base stations. The approximation ratio of the algorithm for single-tiered placement with base stations was proved to be 16.

The approximation ratio of algorithm 4 was proved to be $(10 + \varepsilon)$, which was claimed to improve $(24 + \varepsilon)$ -approximation algorithm in [22]. However, it should be pointed out that both algorithm 4 and its integrated 5-approximation algorithm assume that nodes can be placed in the same position. It is not allowed in [22]. The basic idea of the 5-approximation algorithm in [23] is similar to algorithm 1. First, a minimum set of relay nodes, say set A , are placed to cover all sensor nodes by

Algorithm 3: (for single-tiered placement without base stations)

Input: $R \geq r > 0$ and set of sensor nodes $X = \{x_1, \dots, x_n\}$.

Output: Set of relay nodes $Y = \{y_1, \dots, y_l\}$.

1. Construct the steinerized graph $G^S(r, R, X)$.
 2. Compute a 2-connected minimum weight spanning subgraph G_A of $G^S(r, R, X)$ using the 2-approximation algorithm in [17]. (G_A spans all sensor nodes in X .)
 3. $l = 0$.
 4. **for** each edge $(x_i, x_j) \in G_A$ s.t. $c(x_i, x_j) \geq 1$ **do**
 5. Steiner edge (x_i, x_j) with $c(x_i, x_j)$ relay nodes: $y_{l+1}, y_{l+2}, \dots, y_{l+c(x_i, x_j)}$.
 6. $l = l + c(x_i, x_j)$
 7. **endfor**
-

Algorithm 4: (for two-tiered placement without base stations)

Input: $R \geq r > 0$, $\varepsilon > 0$, and set of sensor nodes $X = \{x_1, \dots, x_n\}$.

Output: Set of relay nodes $Y = \{y_1, \dots, y_l\}$.

1. Apply 5-approximation algorithm in [23] to place set of relay nodes $Z = \{z_1, \dots, z_k\}$, such that the resulting network is connected.
 2. Duplicate each of the relay nodes in Z to obtain Y .
-

using the min-disk-cover scheme. Second, it finds a subset of sensor nodes that are 1–1 mapped with nodes in A , and then places a set of relay nodes, say B , on the same locations of the subset. Finally, it calls STP-MSP algorithm to place a set of relay nodes, say C , to make the network connected. The set of required relay nodes is $A \cup B \cup C$.

The algorithm for two-tiered placement with base stations is similar to the 5-approximation algorithm in [23]. It was proved that the approximation ratio is $(20 + \varepsilon)$ [28].

Deploying relay nodes in heterogeneous WSNs was studied in [14]. It assumes that sensor nodes have different transmission ranges while relay nodes use the uniform transmission radius. The problem consists of two cases: (1) full fault-tolerance relay node placement, which aims to deploy a minimum number of relay nodes to establish k vertex-disjoint paths between every pair of sensor and/or relay nodes; (2) partial fault-tolerance relay node placement, which aims to deploy a minimum number of relay nodes to establish k vertex-disjoint paths only between every pair of sensor nodes. The basic idea of the proposed algorithm is similar to algorithm 3. It first constructs a steinerized weighted graph, and then applies existing algorithm for computing minimum k -vertex connected spanning graph on the weighted graph. The desired graph is achieved after steinerizing each edge in the spanning graph by placing relay nodes according to the weight of the edge.

All the aforementioned placement methods are deterministic. Stochastic node placement was discussed in [20]. It proved that even by random placement

in a unit-area square region, the probability that the network $G(V, r_n)$ is $(k + 1)$ -connected is at least $e^{-e^{-\alpha}}$ when the transmission range r_n satisfies $n\pi r_n^2 \geq \ln n + (2k - 1) \ln \ln n - 2 \ln k! + 2\alpha$ for $k > 0$ and n sufficiently large, where α is any real number.

10.3 Topology Control

Although node placement provides a method to achieving fault tolerance in a WSN, the property of fault tolerance may be not valid due to movements and energy depletion of nodes. Therefore, topology control is required to construct and maintain the property of fault tolerance in WSNs.

A fault-tolerant topology control protocol was proposed in [4]. It first constructs a Connected Dominating Set (CDS) as a backbone of the network. For each node in the CDS, it adds necessary neighbors of the node to the backbone, such that it meets the required vertex connectivity degree. The power on/off model is adopted to turn on the nodes in the backbone to meet connectivity requirement, and other unnecessary nodes are off. Period rotation is used to keep the fairness among nodes. There are several selection metrics. One is power-based selection: the node in CDS selects nodes with more power one by one till the resulting graph is local k vertex connected. Another metric is connection degree. The nodes with higher connection degree are first selected. It is because that the nodes with higher connection degree are supposed to have shorter delay. Some hybrid metrics are also discussed in [4]. Simulation results show the improvement of network lifetime with a desired vertex connectivity degree.

Most of existing works in fault-tolerant topology control aim to achieve k -vertex connectivity in the network. The objective is suitable for ad hoc networks where there is request to connect any two nodes in the network. However, data transmission in WSNs is usually in gathering and aggregation manner. It is rather important to have fault tolerance in the paths from sensors to sinks and gateway nodes, which are more powerful than sensor nodes.

The authors in [3] studied fault-tolerant topology control in heterogeneous WSN. The network consists of two types of wireless devices: a large number of sensor nodes and several resource-rich supernodes. The problem is to adjust each sensor's transmission range, such that there exist k -vertex disjoint communication paths from each sensor to the set of supernodes. The objective is to minimize the total power consumed by sensors. Three solutions were proposed. The first k -approximation algorithm consists of two steps. In the first step, a given graph is reduced to a direct graph where supernodes are merged as a root. In the second step, existing optimal solution for the Min-Weight k -OutConnectivity problem is adopted to compute the minimal transmission range of each sensor. The two steps are briefly introduced one by one.

The given graph is denoted by $G(V, E, c)$, where V is the set of nodes, E is the set of edges, and c is the set of weight of the edge (indicating the power consumed in the edge). The reduced graph is constructed as follows. All supernodes in V are

merged into one node called the *root*. Edges between sensors remain the same, and an edge between a sensor and a supernode is replaced with an edge between the sensor and the root. The weight of the edge remains the same. It should be pointed out that if a sensor is connected to more than one supernode, only the edge to the closest supernode is kept. After that, every undirected edge between two sensors is replaced with two directed arcs that point to each of them. An undirected edge between a sensor and the root is replaced with one directed arc from the sensor to the root. The process of the step is illustrated in Fig. 10.3.

The algorithm in the second step is based on the reduced graph from the first step. It applies existing optimal solution for the Min-Weight *k*-OutConnectivity problem in the reduced graph. The final transmission range of each node is the transmission range used to meet the longest edge in final result. Detail of the algorithm is as follows.

Algorithm 5:

1. Construct the reduced graph of *G*.
 2. Reverse the direction of each arc in the reduced graph and keep the weight of the arc the same.
 3. Apply the optimal solution for the Min-Weight *k*-OutConnectivity problem.
 4. Reverse back the direction of each arc.
 5. **for** each sensor **do**
 6. Adjust transmission range to meet the longest arc in the graph.
 7. **endfor**
-

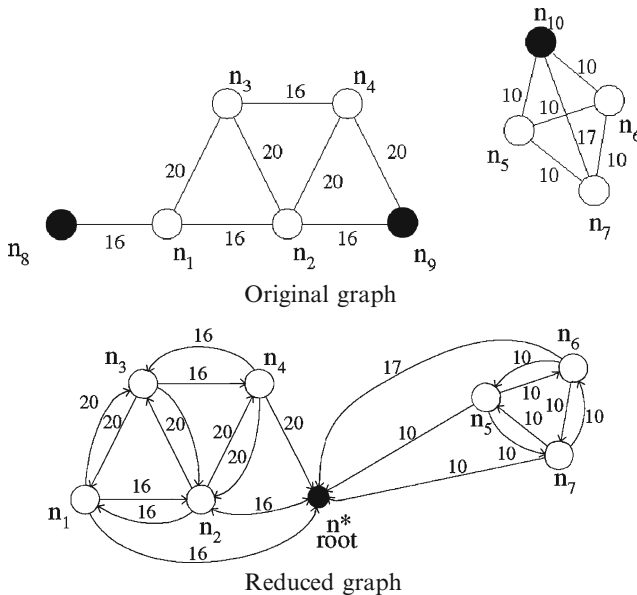


Fig. 10.3 Original graph and reduced graph

The algorithm 5 was proved to achieve performance ratio k . A greedy algorithm that minimizes the maximum transmission power of sensors was further proposed. The basic idea is to sort all edges in the reduced graph in decreasing order. For each edge in the sorted order, the edge is discarded if the k -vertex connectivity to the root is still held without the edge. The process continues until all edges are computed. Similar to algorithm 5, the final transmission range of each node is the transmission range to meet the longest edge in final graph. Distributed version of the proposed algorithms can be found in [3].

Fault tolerance of wireless networks can also be achieved by movement control of nodes. Although work in [7] is for mobile robot networks, the proposed algorithm can be easily applied to WSNs without much modification. A localized movement control algorithm was proposed to form a fault tolerant biconnected network topology from a connected network. The objective is to minimize total distance of movement of nodes.

Each node is assumed to have information of its p -hop neighbors. The p -hop subgraph of a node is defined by the graph that contains all nodes that are within p -hops from the node and all corresponding links. A node is said to be a p -hop critical node if and only if its p -hop subgraph is disconnected without the node. The distributed algorithm is executed at each node and starts as follows. At initialization stage, each node checks whether it is a p -hop critical node. If a node finds itself a p -hop critical node, it broadcasts a critical announcement packet to all its direct neighbors.

To make the network biconnected, all critical nodes should become noncritical by movement of nodes. Note that the movement of a node may create new neighbors, but it may also break some existing links. Since the p -hop subgraph of a critical node is disconnected without the node, movement of a critical node may break some current links, which results in disconnection of the network. However, the network remains connected if all current links of a noncritical node are broken. The basic idea of movement control is to move noncritical nodes while keeping critical nodes static unless they become noncritical. According to the number of critical neighbors of a critical node, there are three cases: (1) critical node without critical neighbors, (2) critical node with one critical neighbor, and (3) critical node with several critical neighbors.

In case (1), a node finds itself a p -hop critical node and does not receive any critical announcement packet from its neighbors. The p -hop subgraph of any critical node can be divided into two disjointed components without the node. The basic idea of movement control is to select two close neighbors of the node from these two disjointed components, respectively, and then move them toward each other until they become connected. See the example shown in Fig. 10.4, where node 3 in black color is a critical node and other nodes in white color are noncritical nodes. Suppose $p = 2$ in this example. Since node 3 is critical, its 2-hop subgraph is divided into two disjointed sets $A = \{1, 2, 4, 5\}$ and $B = \{6, 7, 8\}$. Suppose distance of node 5 and 8 is the minimum among all possible pairs in these two sets (actually, other selection metrics can be applied). Node 3 computes new locations of node 5 and node 8 and asks them to move to become neighbors.

Fig. 10.4 Critical nodes without critical neighbors

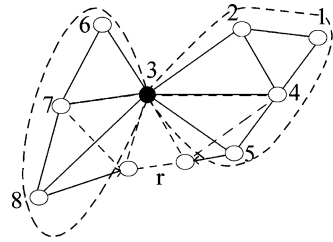
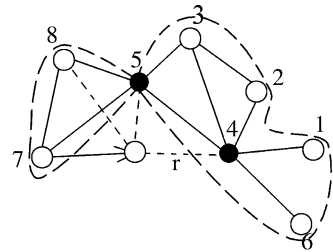


Fig. 10.5 Critical node with one critical neighbor

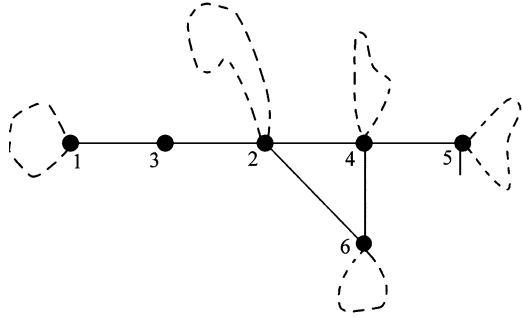


In case (2), the basic idea is to let the critical node with larger ID select one of its noncritical neighbors to move toward the other critical node. See the example in Fig. 10.5, where nodes 4 and 5 in black color are critical nodes and other nodes in white color are noncritical nodes. Since ID of node 5 is larger than ID of node 4, node 5 leads movement. Again p is assumed to equal 2. Node 5 divides its 2-hop subgraph into two disjoint sets $A = \{1, 2, 3, 4, 6\}$ and $B = \{7, 8\}$. Suppose distance of nodes 4 and 7 is the minimum for all nodes in B . Node 5 computes new location of node 7 and asks it to move toward node 4 until connected.

In case (3), some critical node has more than one critical neighbor. Note that each node sends a critical announcement packet to all its direct neighbors if it finds itself a p -hop critical node. After that, all nodes in the network know the status of their neighbors. A critical node is said to be *available* if it has noncritical neighbors and is *unavailable* otherwise. A critical node is available means that it has noncritical neighbors that are able to move. An available/unavailable critical node broadcasts an available/unavailable announcement packet to its neighbors. A critical node declares itself a *critical head* if and only if it is available and its ID is larger than the ID of any available critical neighbor, or has no available critical neighbors. The basic idea for this case is to use the pairwise merging strategy. Each critical head dominates the pair merging and selects one of its critical neighbors to pair with. For example, the available critical neighbor (if any) with largest ID is selected, or otherwise unavailable critical neighbor with the largest ID. Then the movement control for case (2) is called for each pair to compute the new topology.

See the example in Fig. 10.6, where all black nodes are critical nodes (dashed block with a node is subgraph of this node). Nodes 1, 5, and 6 are critical heads. Node 1 becomes a critical head since node 3 is unavailable. Finally, it forms three pairs: (1, 3), (5, 4), and (6, 4), dominated by nodes 1, 5, and 6, respectively. Each

Fig. 10.6 Critical node with several critical neighbors



critical head in a pair calls the movement control algorithm for case (2) to merge the pair. Pairwise merging continues until all critical nodes become noncritical, i.e., the network is biconnected.

10.4 Target and Event Detection

One of the important sensor network applications is to detect, classify, and locate specific events, and track targets over a specific region. An example is to deploy a sensor network in battle field to detect tanks. Once a tank moves into a specific area, information of the tank, such as location and speed, will be gathered and reported by the sensors that are able to sense the tank. Another example is the use of WSNs as watchers to detect fire in forest. A lot of work has been done in fault tolerance on target and event detection. In this section, we introduce some representative works in target detection and event detection, respectively.

10.4.1 Target Detection

Clouqueur et al. [6] have proposed two fault-tolerant algorithms for collaborative target detection in sensor networks in which sensor nodes can either fail due to harsh environmental conditions or maliciously. Both algorithms are based on sensor nodes sharing information to reach consensus.

The first algorithm, called *value fusion*, works as follows. Each node obtains raw energy measurements from every node, computes an average by removing the largest n and smallest n values, and compares this average to a threshold for final decision for a given n . The second algorithm, called *decision fusion*, does not work on raw measurement but rather on local decision of each sensor node. It works in the same way as the value fusion algorithm. The authors mention that there is no need for dropping the data when all nodes are known to be fault-free.

The exact agreement guarantees final consistency among fault-free nodes and therefore the faulty nodes can only degrade the accuracy of the system. System failure occurs when the bound on the number of faulty nodes acceptable is violated. The authors have measured the detection probability of both algorithms for different false alarm probabilities, number of nodes, maximum power and decay factor and have concluded that decision fusion is superior to value fusion as the ratio of faulty sensors increases. Besides having low communication cost, high precision and accuracy, both algorithms are efficient in terms of the number of faulty sensors tolerable in the network.

Ding et al. [9] have recently proposed fault-tolerant algorithms to detect the region containing targets and to identify possible targets within the target region, while catering to a stingy sensor energy budget and faulty sensors. The basic idea behind their approach to target detection is that each sensor computes the median of signal measurements such that the disturbances of extreme measurements caused by faulty sensors are filtered out. If a median exceeds certain threshold then it implies that a possible target is present. It does not, however, tell how many targets exist and where they are. A target localization algorithm is used to compute the position of each target. The task of communicating with the base station and computing target positions is delegated to a particular sensor, called the root sensor, which is the one with the local maxima. The root sensor computes the geometric center of neighboring sensors with similar observations. Target position is further refined through the use of multiple epoch observations.

Algorithm for target detection

1. Each sensor in a given neighborhood obtains its signal measurements.
2. Each sensor computes its median.
3. If the median exceeds a threshold the sensor becomes an event sensor.

Algorithm for target localization

1. Obtain the estimated signal strength from all event sensors in a given neighborhood.
2. Compute the local event sensor that has the maximum signal strength in a given neighborhood and label them root sensors.
3. For each root sensor compute the location of a target based on the geometric center of a subset of event sensors.

Algorithm for target identification

1. For each epoch, apply above Target Detection and Target Location algorithm.
2. After collecting raw data for T epochs, the base station applies a clustering algorithm to group the estimates into a final target position computation. Each group is one target.

3. If the size of a group is less than half the number of epochs (i.e., $T/2$), then with high probability this group is a false alarm; otherwise, report a target and obtain the estimate of the position of the target using the geometric center of all raw data within the group.

The algorithms proposed by Ding et al. [9] work well in dense sensor networks as median is not robust in low density, and targets must be far apart in order to identify them as independent targets. The authors assume that each sensor can compute its physical location using either GPS or some GPS-less techniques, and there is no fault in processing and transmitting/receiving neighboring measurements as well as the proper execution of algorithms.

10.4.2 Event Detection

Krishnamachari and Iyengar [19] have proposed a distributed and localized fault-tolerant event detection method for WSNs. On the basis of the observation that the sensor faults are likely to be stochastically uncorrelated, while event measurements are likely to be spatially correlated, they propose an algorithm in which each sensor node communicates with its neighbors to collect their binary decisions to correct its own decision. A majority voting scheme is shown to be the optimal decision scheme for fault correction in their work. The algorithm can be described as follows:

Let N_i be the neighbors of a sensor node i , each having a probability of failure p . Let the binary variables T_i and S_i , represent the real situation and real output of i , respectively. That is, $T_i = 0$ if the node is a normal region, and $T_i = 1$ if the node is an event region. Similarly, $S_i = 0$ if the sensor measurement indicates a normal value, $S_i = 1$ otherwise. There are four possible scenarios: $(S_i = 0, T_i = 0)$, $(S_i = 0, T_i = 1)$, $(S_i = 1, T_i = 0)$, and $(S_i = 1, T_i = 1)$. It assumes that sensor fault probability is uncorrelated and symmetric, i.e.,

$$P(S_i = 0 | T_i = 1) = P(S_i = 1 | T_i = 0) = p.$$

The binary value is determined by introducing the threshold on the real-valued readings of sensors, $0.5(m_n + m_f)$, where m_n is the mean value of normal reading and m_f is the mean of event reading.

Let $E_i(a, k)$ be the evidence such that k of its neighboring sensors report the same binary reading a as node i itself. The authors use a Bayesian fault recognition technique to determine an estimate R_i of the true reading T_i . Since it assumes that the network is deployed with high density, nearby sensors are likely to have the similar event readings unless they are on the boundary of the event region. So, the following model is adopted:

$$P(R_i = a | E_i(a, k)) = k/N.$$

Therefore, the probability with which node i can make a decision to accept its own reading S_i in face of the evidence $E_i(a, k)$ is given by

$$P_{\text{aak}} = P(R_i = a | S_i = a, E_i(a, k)) = \frac{(1-p)k}{(1-p)k + p(N_i - k)}.$$

The probability of disregarding its own reading is simply $1 - P_{\text{aak}}$. Based on this Bayesian formulation, P_{aak} and decision threshold $0 < \theta < 1$, the authors suggest three decision schemes:

Algorithm using randomized decision scheme

1. Obtain sensor readings S_j of all N_i neighbors of node i .
2. Determine k_i , the number of node i 's neighbors j with $S_j = S_i$.
3. Calculate P_{aak} .
4. Generate a random number $u \in (0, 1)$.
5. If $u < P_{\text{aak}}$, set $R_i = S_i$, else set $R_i = \neg S_i$.
(S_i is a binary variable. $\neg S_i$ is opposite value of S_i .)

Algorithm using threshold decision scheme

1. Obtain sensor readings S_j of all N_i neighbors of node i .
2. Determine k_i , the number of node i 's neighbors j with $S_j = S_i$.
3. Calculate P_{aak} .
4. If $P_{\text{aak}} > \theta$, set $R_i = S_i$, else set $R_i = \neg S_i$.

Algorithm using optimal decision scheme

1. Obtain sensor readings S_j of all N_i neighbors of node i .
2. Determine k_i , the number of node i 's neighbors j with $S_j = S_i$.
3. If $k_i > 0.5N_i$, set $R_i = S_i$, else set $R_i = \neg S_i$.

It proves that the best policy for each node is to accept its own reading if at least half of its neighbors have the same reading. Simulation results show that by using the optimal threshold decision scheme, faults can be reduced by as much as 85–95% with fault rates as high as 10%.

Krishnamachari and Iyengar's algorithm deals only with sensor faults and does not consider decision error caused by noisy measurements. Moreover, it is not known how large the neighborhood size should be.

In [24], Luo et al. have proposed a distributed approach that addresses both detection error and sensor fault simultaneously, while choosing a proper neighborhood size in order to be energy efficient and provide adequate error detection. Their approach is an improvement on the work of Krishnamachari and Iyengar [19], which provides a majority voting scheme to allow individual nodes to correct their binary decisions by communicating with their neighbors. The improvement is the additions of detection error in addition to sensor fault, as well as the determination of a proper neighborhood size to improve energy efficiency.

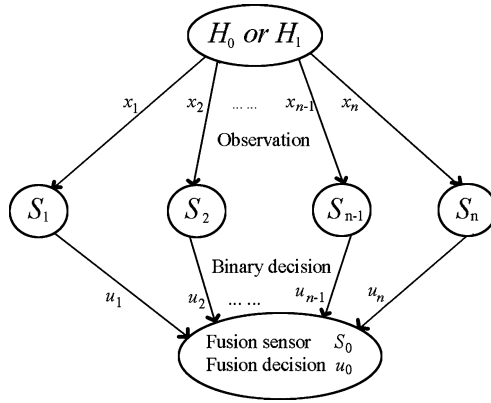


Fig. 10.7 Decision Framework

In their model, each sensor node has n neighbors and makes its binary decision independently based on its own measurement from the noisy environment. The faulty behavior of a node is considered as “event” by sensors at the location while sensor fault is treated as “no-event.” The authors consider a two-layer detection system that consists of a fusion sensor and its n neighbors. The fusion sensor makes a final decision whether an unknown hypothesis is H_0 or H_1 based on the decision from the n sensors. Let x_i denote the observation of the i th sensor, $i = 1, \dots, n$. Let u_i denote the binary decision (0 or 1) of the i th sensor, and let λ be the decision threshold common to all nodes. Based on the received sensor decisions, the fusion sensor makes the final decision u_0 as shown in Fig. 10.7 with a k -out-of- n rule for some majority k . Let $u_0 = 0$ if the fusion sensor decides H_0 and let $u_0 = 1$ if the fusion sensor decides H_1 . That is, $u_0 = 1$ if $u_1 + \dots + u_n \leq k$, and $u_0 = 0$ if $u_1 + \dots + u_n > k$.

The authors use a two-loop search algorithm to find the optimal solutions for τ ($= \ln \lambda$), g , and n for a given bound of detection error P_e and a sensor fault probability P_f . Through optimization algorithm it is possible to find the optimal decision threshold (λ) and decision majority (k), such that the probability of a detection error is minimized. In the inner loop, the optimal (τ, k) pair is obtained through numerical optimization for a fixed neighborhood size n . In the outer loop, a binary search is employed to find the minimum n that satisfies the given error bound. After optimizing the decision threshold, majority, and neighborhood size, each node then makes a decision based on the threshold and obtains the decisions of its neighbors and makes its final decision based on majority. The detection algorithm can be summarized as follows:

Algorithm for distributed detection

1. Set τ, k , and n in each sensor node. This can be done at the manufacturing time or after deployment.
2. Each sensor obtains its binary decision u_i based on its measurement and τ ($= \ln \lambda$) with threshold test.

3. Each sensor obtains the binary decisions u_1, u_2, \dots, u_n of its n neighbors and computes $u_1 + \dots + u_n$.
4. Each sensor makes its final fault-tolerant decision based on the k -out-of- n rule.

Luo et al. [24] assume that the ground truth is the same for all n neighbors of nodes i ; that is, if node i is in an event region, then so are its neighbors and vice versa. This assumption does not hold for sensors at the event boundary. Experiments have shown that this causes confusion in sensors near the boundary and leads to a decrease in detection accuracy. In practical applications, some nodes may not have enough neighbors to meet the optimization criteria. Authors suggest keeping multiple alternative (less optimal) triples or to simply deploy more sensors. Communication errors from noisy communication links can affect event detection. Solution involves modeling these errors as additional sensor faults in the detection process.

10.5 Data Gathering and Aggregation

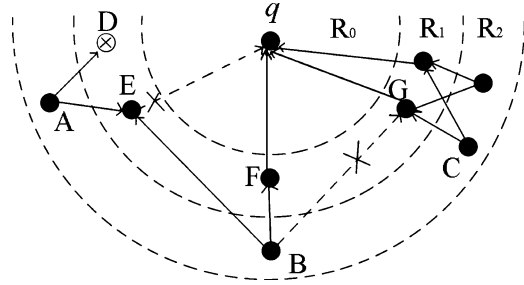
WSNs are usually used to monitor environments and collect information from sensor nodes to the sink or the querying node, which is responsible for further processing. Whatever be the application where a WSN is deployed, individual sensor readings of each sensor node needs to be collected and then reported to the sink. This is done by means of data gathering and data aggregation. Data gathering is to combine data coming from different sensor nodes, eliminate redundancy, and minimize the number of transmissions. In data aggregation, data sensed at neighboring nodes are either highly correlated or simply redundant and need to be aggregated before delivery to the upper layer. Since node failures and transmission failures are common in WSNs, fault tolerance should be considered in designing protocols of data gathering and data aggregation.

A general approach used for data gathering and data aggregation is to construct a spanning tree that is rooted at the sink and connects all nodes in the network. However, a tree topology is not robust against any node and transmission failure. An aggregation framework, called *synopsis diffusion*, that uses energy-efficient multi-path routing schemes was proposed in [25]. There are three basic functions on the synopsis.

- Synopsis generation function: $SG(\cdot)$ takes a sensor reading (including its meta data) and generates a synopsis representing that data.
- Synopsis fusion function: $SF(\cdot, \cdot)$ takes two synopses and generates a new synopsis.
- Synopsis evaluation function: $SE(\cdot)$ translates a synopsis into the final answer.

The synopsis diffusion algorithm consists of distribution phase and aggregation phase. In distribution phase, the aggregation query is flooded through the network and an aggregation topology is constructed. In aggregation phase, aggregation of individual reading of sensors is routed hop by hop toward the querying node. Details of the algorithm are as follows.

Fig. 10.8 An example for synopsis diffusion



During query distribution phase, nodes in the network form a set of rings around the querying node, say q , according to their distance to q . q is in ring R_0 and a node is in ring R_i if i hops away from q . The query aggregation period is divided into epochs and aggregation process is executed once at each epoch. It is assumed that nodes in different rings are loosely time synchronized and are allocated specific time intervals when they should be awake to receive synopsis. For example, in Fig. 10.8, node q is in R_0 . There are five nodes in R_1 (including one fault node during the aggregation phase), and four nodes in R_2 . At the beginning of each epoch, each node in the outermost ring (R_2 in the example) generates its local synopsis $s = SG(r)$, where r is the sensor reading relevant to the query answer. The node broadcasts its synopsis to all neighbors. In general, a node in ring R_i wakes up at its allocated time, generates its local synopsis $s = SG(\cdot)$, and receives synopses from all nodes within transmission range in ring R_{i+1} . Upon receiving a synopsis s' , a node updates its local synopsis as $s = SF(s; s')$. The updated synopsis s is broadcast at the end of allocated time of the node. Thus, the fused synopses propagate layer by layer toward node q , which returns $SE(s)$ as the answer to the query at the end of the epoch. We can see that synopsis of node B can reach the querying node even if the transmissions $E \rightarrow q$ and $B \rightarrow G$ fail. However, since both nodes D and transmission $E \rightarrow q$ are failure, synopsis of node A can not be received by node q .

By dividing a network into a set of rings, data aggregation can be done over arbitrary topologies. Since multiple paths are used to route data to the querying node, another challenge of the aggregation algorithm is to support duplicate-sensitive aggregates. It exceeds the scope of the chapter. Interested readers can find more details in [25].

Two fault-tolerant schemes for duplicate-sensitive aggregation in WSNs were presented in [13]. The schemes use the available path redundancy to deliver a correct aggregate result to the sink. The basic idea is as follows. When a packet is lost between two sensors because of a link error, it is possible that one or more other sensors have correctly overheard the packet. If some of them have not yet transmitted their own values, they correct the error by aggregating the missing value into theirs. Since the lost packet is aggregated with another packet, error recovery does not cause extra overhead.

It assumes that a network is static in each query process. The track topology of the network is formed in layers, which is similar to Fig. 10.8. The only difference is that some edges may exist among nodes in the same track/layer (see Fig. 10.9).

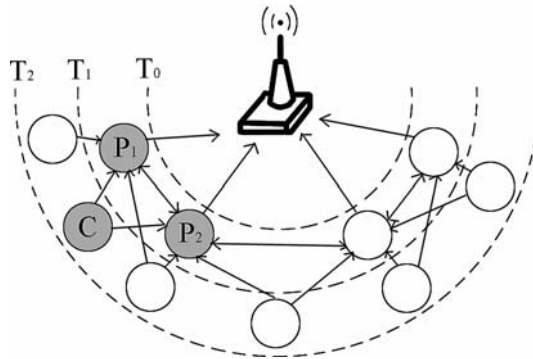


Fig. 10.9 Track topology

Edges are classified into three types: primary, backup, and side edges. Primary and backup edges are between adjacent layers (between a sensor node and its parents). Side edges are within the same layer (among parents). It assumes that errors in primary, backup, and/or side edges are independent. Each sensor selects one parent (and correspondingly one edge) as its primary parent and zero or more parents as backups. Primary edges form a spanning tree and are used as long as no communication error occurs. If an error occurs in a primary edge, data may be successfully delivered by some backup edges. The missing value is aggregated at most once by using side edges. Therefore, there is no duplicated value that is aggregated and reported to the sink. It should be noted that a sensor can be a primary parent for some children and at the same time a backup parent for some others.

Each parent attaches a bit vector to data messages it sends. The bit vector contains the IDs and bit positions of children whose values have been correctly received and aggregated. Each parent broadcast its bit vector. By overhearing the bit vectors over side edges, backup parents know link errors when one or more children are missing from the bit vector. Each parent determines the bit positions of its children inside the other parents' bit vectors during topology construction. The bit vector of a primary/backup parent contains two bits for each child. One is the *e-bit* that indicates error in the child's primary edge. The *e-bit* is set to 1 if the primary parent does not receive messages from a child. Overhearing the primary parent's signal, a backup parent sets its *e-bit* to 1 as well to propagate the error signal. The other is the *r-bit* that indicates that the sensor is correcting or helping correct the error.

See that in Fig. 10.9, there are three layers. Sensor C in layer T_2 has two parents: primary parent P_1 and backup parent P_2 . C transmits a message to P_1 and P_2 overhears the message. If there is no error, only P_1 aggregates the message. Suppose there is a link error in primary edge $C \rightarrow P_1$, P_2 will receive the bit vector of P_1 over the side edge $P_1 \rightarrow P_2$. P_2 finds that C is missing, and aggregates C 's value into its own to correct the error.

Data gathering is another important and fundamental operation in WSNs. A delay/fault-tolerant data gathering scheme for mobile sensor networks was studied in [27]. The scheme consists of two components: data transmission and queue

management. Data transmission is to determine when and where to transmit data messages based on the *delivery probability*. Queue management is to determine which messages are to be transmitted or dropped based on the *fault tolerance*. These two important parameters are introduced in the following.

Delivery probability: It reflects the likelihood that a sensor can deliver data messages to the sink. The decision on data transmission is based on delivery probability. Let ξ_i denote the delivery probability of a sensor i . ξ_i is initialized with zero and updated upon an event of either message transmission or timer expiration. More specifically, if there is no message transmission within an interval of Δ , the timer expires and it generates a timeout event. It means that the sensor could not transmit any data messages during Δ . Thus, its delivery probability should be reduced. Whenever sensor i successfully transmits a data message to another node k , ξ_i should be updated to reflect its current ability in delivering data messages to the sinks. Since end-to-end acknowledgment is not employed in the scheme due to its low connectivity, sensor i does not know whether the message transmitted to node k will eventually reach the sink or not. Therefore, it estimates the probability of delivering the message to the sink by the delivery probability of node k , i.e., ξ_k . More specifically, ξ_i is updated as follows,

$$\xi_i = \begin{cases} (1 - \alpha)[\xi_i] + \alpha\xi_k, & \text{transmission;} \\ (1 - \alpha)[\xi_i], & \text{timeout,} \end{cases} \quad (10.5)$$

where $[\xi_i]$ is the delivery probability of sensor i before it is updated, and $0 \leq \alpha \leq 1$ is a constant employed to keep partial memory of historic status. If k is the sink, $\xi_k = 1$, because the message is already delivered to the sink. Otherwise, $\xi_k < 1$.

Fault tolerance: It indicates the importance of a message by duplicate copy of the message. Different from other gathering schemes where the packets are deleted after they are transmitted, sensors in the scheme may still keep a copy of the message after its transmission to other sensors. Therefore, multiple copies of the message may be created and maintained by different sensors in the network. Such redundancy for fault tolerance indicates the importance of a given message. Each message is assumed to carry a field that keeps its fault tolerance. Let F_i^j denote the fault tolerance of message j in the queue of sensor i . There are two approaches to define the fault tolerance of a message in [27].

The first is delivery probability-based approach. The fault tolerance of a message is defined to be the probability that at least one copy of the message is delivered to the sink by other sensors in the network. The fault tolerance of a message is initialized to be zero at the beginning. Suppose sensor i multicasts a data message j to Z nearby sensors, denoted by N_z , $1 \leq z \leq Z$. The multicast transmission essentially generates totally $Z + 1$ copies. Let $F_{N_z}^j$ denote the fault tolerance of message j transmitted to sensor N_z . It can be computed as follows:

$$F_{N_z}^j = 1 - \left(1 - [F_i^j]\right) (1 - \xi_i) \prod_{m=1, m \neq z}^Z (1 - \xi_{N_m}). \quad (10.6)$$

The fault tolerance of the message at sensor i is updated as

$$F_i^j = 1 - \left(1 - [F_i^j]\right) \prod_{m=1}^Z (1 - \xi_{N_m}), \quad (10.7)$$

where $[F_i^j]$ is the fault tolerance of message j at sensor i before multicasting. The fault tolerance of each message is updated according to (10.2) and (10.3). In general, the more times a message has been forwarded, the more copies of the message are created. It will thus increase its delivery probability, which results in stronger fault tolerance.

The second approach is called message hop count-based approach, where fault tolerance is defined according to the hop count of the message. Let h_j denote the number of times that the message j has been forwarded. A message with larger h_j usually has more copies in the network. Fault tolerance in this approach is defined with $F_i^j = h_j^2/H^2$, where H is the maximum hop count. For a new message, $F_i^j = 0$. since $h_j = 0$. If a message has just been sent to the sink, $F_i^j = 1$. Simulation results in [27] show that the approach based on the message hop count is less accurate than the delivery probability-based approach.

Based on the delivery probability, process of data transmission is as follows. Suppose sensor i has a message j at the top of its data queue ready for transmission. When it moves into the communication ranges of a set of Z sensors, sensor i first learns their delivery probabilities and available buffer spaces via handshake. Message j is transmitted to node N_z , $1 \leq z \leq Z$, if $F_{N_z}^j > F_i^j$ and there are available buffers in N_z . Then, delivery probability of node i is updated according to equation (10.3). The process continues until the updated F_i^j is larger than a predetermined threshold.

Based on the fault tolerance, the process of queue management is as follows. Each sensor has a data queue that contains data messages ready for transmission. A message with smaller fault tolerance means that the message is forwarded less times and has less copy in other nodes. Therefore, the message is more important and should be transmitted with a higher priority. This is done by sorting the messages in the queue with an increasing order according to their fault tolerance. The message with the smallest fault tolerance is always at the top of the queue and is transmitted first. There are two cases that a message is dropped. First, if the queue is full and fault tolerance of a new message is larger than that of the message at the end of queue, the message is dropped. Otherwise, the message at the end of the queue is replaced with the new message. Note that the new message should be inserted into the queue at appropriate position (not always at the end of the queue). Second, if the fault tolerance of a message is larger than a threshold, the message is dropped to reduce transmission overhead. It supposes that the message will be delivered to the sinks with a high probability by other sensors in the network. A message will be dropped immediately if it already reaches the sink.

10.6 Sensor Monitoring and Surveillance

Sensor monitoring/surveillance is different from target/event detection discussed in previous section. Target/event detection is usually to detect presence and status change of targets and events, while sensor monitoring/surveillance is usually to monitor static targets and interested area in applications. In sensor monitoring/surveillance application, a popular strategy for increasing the reliability is to deploy more nodes than strictly necessary. Since there are strict energy constraints on sensor nodes, usually only a portion of sensor nodes is active to maintain system operations while others go to sleep. Therefore, it requires a schedule that determines which ones remain active and which ones may go to sleep. To ensure proper operation of the network, sleeping nodes should frequently monitor active nodes. Those crashed nodes will be replaced once they are detected. On the other hand, nodes should remain sleeping as much as possible to save the energy. If the energy consumed in the monitoring process is too high, spare nodes may exhaust their batteries before they are needed.

The optimal monitoring period in fault-tolerant sensor networks was studied in [1]. A schedule algorithm called Sleep-Query-Active (SQA) was proposed. It is to ensure that the network remains connected and the lifetime of the network is maximized. It assumes that nodes are aware of their locations and uses the information to divide a two-dimensional space into grids. The distance of two farthest points in any two adjacent grids must be smaller than the communication range of sensor nodes R (see Figure 10.10a). Thus, the cell side, r , should satisfy $r \leq R/\sqrt{2}$. It is to make sure of the connectivity of the network if more than one node exists in each grid.

It further assumes that each node in the network can only be in one of two states: either sleeping or active, as illustrated in Fig. 10.10b. The purpose of the wait state is to desynchronize nodes that start at the same time. Nodes in the network send *discovery messages* in following situations: (1) when they enter active state, (2) periodically when they are in the active state (to overcome the loss of messages), and (3) in active state when they receive a discovery message from a node with lower rank. Here, the ranks of nodes are determined by the estimated node active

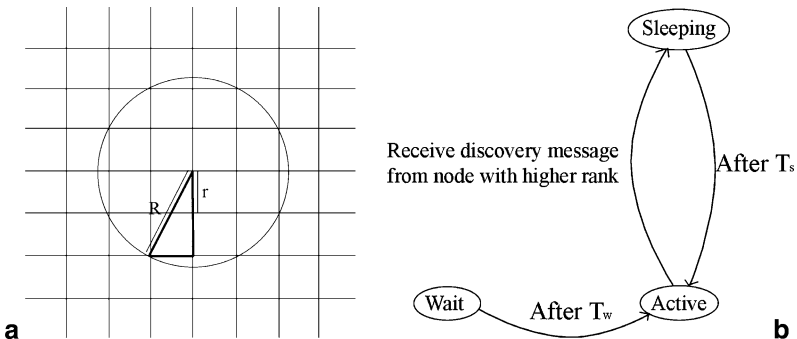


Fig. 10.10 Sleep-Query-Active algorithm

time, i.e., the remaining energy. That is, higher rank means longer expected lifetime. Whenever a node in active state receives a discovery message from a node with higher rank, it immediately sets up a timer to wake up and goes to sleeping state. The sleeping timeout, T_s , is treated as the monitoring period. Each time a node goes to sleep, it picks the value for T_s from an interval with uniform probability. Selection of proper T_s is done via extensive simulations since a theoretical approach to determine T_s is a task of great difficulty [1].

Area coverage problem in sensor networks was addressed in [12]. The problem is to schedule sensor nodes to be active or sleeping, such that both connectivity and full area coverage are achieved. The objective is to achieve similar performance in terms of ratio of active sensors in a given round as the best existing localized solution, while significantly reducing the number of messages for making decision at each node. It assumes that sensing radio and communication radio are different, and sensor nodes are time synchronized. The authors proposed four variants of protocols, which rely on low communication overhead in order to be applied for highly dense networks. Overview of the proposed protocols is as follows.

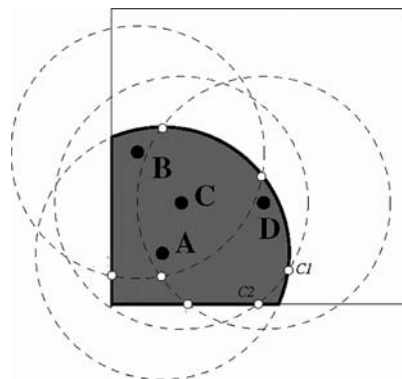
Each node selects a random timeout and listens to messages sent by other nodes before the timeout expires. Once timeout ends, the neighbor table of a node contains every node that already made decision with shorter timeout. The node evaluates the status of coverage and connectivity and decides to be active if its sensing area is not fully covered or connectivity requirement is not satisfied. Otherwise, the node goes to sleep mode. The node announces its decision to its neighbors. Note that a node may hear from more active neighbors after it decides to be active. The node may change its mind by sending retreat message to its neighbors if its sensing area becomes fully covered or connectivity requirement becomes satisfied.

Framework of the proposed protocols consists of four components: timeout computation, coverage evaluation, connectivity conservation, and decision announcement. They are introduced one by one.

In timeout computation, it is assumed that any two neighboring nodes would select different random numbers, so that two nodes never attempt to send messages at the same time to avoid collisions. Actually, a node may make decision to go to sleep before the timeout expires once it receives enough information from neighbors and the sensing area is fully covered already. It saves computation on useless messages.

A node decides to sleep if its sensing area is fully covered by a set of connected nodes with lower timeout values. The coverage evaluation is to study how to judge that the sensing area of a node is fully covered. The covering criterion is based on the theorem that states that a sensing area is fully covered by a set of covering disks if every intersection point of two covering disks inside the area is covered by another covering disk. To deal with border nodes, the coverage criterion is extended to take into account the intersections of sensing areas and the deployment or monitoring area. Intersection of a node's sensing area and the monitoring area is called the node's *revised sensing area*. For example in Fig. 10.11, node A 's revised sensing area is the shaded area. A could get into sleep mode since circle centered at C covers all intersection points created by other circles and revised sensing area of A , while $C1$ and $C2$ are covered by circle centered at D .

Fig. 10.11 Coverage evaluation



In connectivity conservation, a simple rule is that the connectivity is guaranteed so long as the sensing area is fully covered and the communication range is at least twice the sensing range. In the case that the communication range is less than double of the sensing range, a node can decide to turn off if its sensing area is fully covered by connected neighbors.

After verifying the coverage condition, each node decides whether or not to send a message. Messages contain geographic position of nodes and activity status. Four variants of the protocols are differentiated by the messages sent by nodes.

Positive-only: Each node that decides to be active sends exactly one message. Nodes that decide to sleep do not send any message.

Positive and negative: Each node sends exactly one message. A positive or a negative acknowledgment are for an active or sleep status, respectively.

Positive and retreat: Same as positive-only except that a node that has already decided to be active can later switch to sleep status based on messages from newly announced active nodes. Such nodes send one retreat message.

Positive, negative, and retreat: Any decision of a node will be transmitted. Each node sends one message corresponding to the original decision on active or sleep status. Nodes with originally positive decision may switch to sleep mode later and send one retreat message.

Simulation results show that the last variant of the protocols has the best overall performance among four variants, in terms of less percentage of active nodes and longer lifetime of the network. However, it generates more messages than others.

The surveillance nature of sensor networks requires a long lifetime. A maximal lifetime problem in sensor surveillance systems was studied in [21]. Given a set of targets and sensors and a base station (BS) in an area, the sensors are used to watch (or monitor) the targets and collect sensed data to the BS. Each sensor has an initial energy reserve, and a fixed surveillance range and an adjustable transmission range. A sensor can watch at most one target at a time. A target can be inside the surveillance range of several sensors. A typical example is the use of camera to

continuously watch some targets, such as cargo containers. In some applications, a target needs to be watched by several sensors at any time for fault tolerance, or for localization of a target. Without loss of generality, each target is supposed to be watched by k , $k \geq 1$, sensors at any time. Since sensors are usually redundantly deployed, the problem is to schedule a subset of sensors to be active at a time to watch the targets and find the routes for the active sensors to send data back to the BS, such that each target should be watched by k sensors at any time and the lifetime of the entire sensor network is maximized. The *lifetime* is the duration up to the time when there exists one target that cannot be watched by k sensors or data can not be forwarded to the BS due to the depletion of energy of the sensor nodes.

The proposed optimal solution consists of three steps. In the first step, it formulated the problem with linear programming (LP) technique to compute the upper bound on the maximal lifetime and a workload matrix. The objective of the second step is to find the detailed schedule for sensors to watch targets based on the workload matrix. The basic idea is to represent the workload matrix as a bipartite graph and then apply the perfect matching technique to decompose the workload matrix into a sequence of schedule matrices. Each perfect matching corresponds to a schedule matrix. The process of finding perfect matching is continued until the workload matrix is completely decomposed. The last step to construct sensor surveillance trees for each session is based on the schedule matrices and data flow computed from the LP formulation.

10.7 Thoughts for Practitioners

Link fault is a common problem in real deployments of WSNs. To improve reliability of communications, cross-layer design could be adopted for specific applications of WSNs. For example, placement of redundant sensor nodes than necessary is a solution to achieve fault tolerance of the network. In MAC layer and communication layer, one standard solution is to increase reliability of wireless communication in adoption of error correction mechanisms and acknowledgement mechanism. In software layer, it requires to avoid software bugs in both OS and middleware. Based on difference in applications, techniques and algorithms that are introduced in the chapter could be employed.

However, there is a trade-off between fault tolerance and efficiency of the network. Many fault-tolerant techniques and algorithms may cause extra energy consumption, increase of overhead, transmission collision, and delay of operation. Therefore, balance point should be carefully analyzed and determined depending on the task of networks and objective of applications.

To apply WSNs in safety critical applications, security threats must be addressed during all operational phases of a fault-tolerant system. However, most current approaches do not include security measures.

10.8 Conclusions and Directions for Future Research

The goal of the chapter is to investigate current research work on fault tolerance in WSNs. We studied how fault-tolerant techniques were addressed in node placement, topology control, target and event detection, data gathering and aggregation, and sensor surveillance. We focused on the application layer and introduced representative works in each application. Actually, there are other applications where fault tolerance attracts attention, such as clustering, time synchronization, gateway assignment, etc.

Although extensive works have been done on fault tolerance in each layer of the WSN system, cross-layer solutions are expected in future. Use of the resource could be more efficient if resource can be properly integrated and scheduled in different layers. Therefore, cross-layer solutions are expected to have better performance than current solutions.

A new trend of WSNs is to cooperate or integrate with other wireless device/systems, such as actuator networks and RFID system. For example, there are an increasing number of applications that require the network system to interact with the physical system or environment via actuators. That is, it requires the use of sensor networks along with actuators to build wireless sensor and actuator networks (WSANs). Although fault tolerance techniques for WSNs could be reused in WSANs, there are new challenges that require new solutions. For example, when an actuator fails, the sensors that report their data to the actuator may either switch to another actuator or directly pass the data to the sink.

Acknowledgments This research is supported by NSERC Collaborative Research and Development Grant CRDPJ 319848-04 and the UK Royal Society Wolfson Research Merit Award.

Terminologies

Fault tolerance. The ability of a system to deliver a desired level of functionality in the presence of faults.

Self-diagnosis. A sensor node determine faults by itself.

Cooperative diagnosis. Several sensor nodes determine faults by cooperation.

Covering with disks. Given a set of points in the plane, the problem is to identify the minimum set of disks with prescribed radius to cover all the points.

Steiner tree problem with minimum number of Steiner points (STP-MSP). Given a set of terminals in the Euclidean plane, the problem is to find a Steiner tree such that each edge in the tree has length at most d and the number of Steiner points is minimized.

Dominating Set (DS). A subset of the vertices of a graph if every vertex in the graph is either in the subset or is adjacent to at least one vertex in the subset.

Connected Dominating Set (CDS). A connected DS.

p-hop subgraph. Of a node is defined by the graph that contains all nodes that are within p -hops from the node and all corresponding links.

p-hop critical node. The node is said to be p -hop critical node if and only if its p -hop subgraph is disconnected without the node.

Available. A critical node is said to be available if it has noncritical neighbors.

Critical head. A critical node is said to be a critical head if and only if it is available and its ID is larger than the ID of any available critical neighbor, or has no available critical neighbors.

Questions

1. From system point of view, in which layers could faults happen in WSNs? And which layer does the chapter focus on?
2. What are the basic methods for fault detection and recovery in WSNs?
3. What is the basic idea of [22] to place the minimum number of relay nodes for WSNs?
4. What is the best known result for *covering with disks* problem?
5. What is the best known approximation ratio for STP-MSP?
6. Please find DS and CDS in Fig. 10.5.
7. Suppose $p = 1$, please find p -hop critical nodes in original graph of Fig. 10.3.
8. What are the basic methods in target/event detection?
9. What is the difference between data gathering and data aggregation?
10. What is the difference between sensor monitoring/surveillance and target/event detection?

References

1. F. Araújo, and L. Rodrigues. "On the Monitoring Period for Fault-Tolerant Sensor Networks," *LADC 2005*, LNCS 3747, São Salvador da Bahia, Brazil, October 2005.
2. J.L. Brediny, E.D. Demainez, M.T. Hajiaghayiz, and D. Rus, "Deploying Sensor Networks with Guaranteed Capacity and Fault Tolerance," *MobiHoc 2005*, urbana-champaign, IL, 2005.
3. M. Cardei, S. Yang, and J. Wu, "Algorithms for Fault-Tolerant Topology in Heterogeneous Wireless Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 4, pp. 545–558, 2008.
4. Y. Chen, and S.H. Son, "A Fault Tolerant Topology Control in Wireless Sensor Networks," *Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, 2005.
5. S. Chessa, and P. Maestrini, "Fault Recovery Mechanism in Single-Hop Sensor Networks," *Computer Communications*, vol. 28, issue 17, pp. 1877–1886, 2005.
6. T. Clouqueur, K.K. Saluja, and P. Ramanathan, "Fault Tolerance in Collaborative Sensor Networks for Target Detection," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 320–333, 2004.

7. S. Das, H. Liu, A. Kamath, A. Nayak, and I. Stojmenovic, "Localized Movement Control for Fault Tolerance of Mobile Robot Networks," *The First IFIP International Conference on Wireless Sensor and Actor Networks (WSAN 2007)*, Albacete, Spain, 24–26 Sept. 2007.
8. M. Demirbas, "Scalable Design of Fault-Tolerance for Wireless Sensor Networks," PhD Dissertation, The Ohio State University, Columbus, OH, 2004.
9. M. Ding, F. Liu, A. Thaler, D. Chen, and X. Cheng, "Fault-Tolerant Target Localization in Sensor Networks," *EURASIP Journal on Wireless Communications and Networking*, 2007.
10. D. Du, L. Wang, and B. Xu, "The Euclidean Bottleneck Steiner Tree and Steiner Tree with Minimum Number of Steiner Points," *Computing and Combinatorics, Seventh Annual International Conference COCOON 2001, Guilin, China, Aug. 2001 Proceedings*.
11. R.J. Fowler, M.S. Paterson, and S.L. Tanimoto, "Optimal Packing and Covering in the Plane are NP-complete," *Information Processing Letter*, vol. 12, pp. 133–137, 1981.
12. A. Gallais, J. Carle, D. Simplot-Ryl, and I. Stojmenovic, "Localized Sensor Area Coverage with Low Communication Overhead," *Proc. of IEEE Sensor'06*, Daegu, Korea, Oct. 2006.
13. S. Gobiari, S. Khattab, D. Mosse, J. Brustoloni, and R. Melhem, "Fault Tolerant Aggregation in Sensor Networks Using Corrective Actions," *Third Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, vol. 2, pp. 595–604, 2006.
14. X. Han, X. Cao, E.L. Lloyd, and C.C. Shen, "Fault-Tolerant Relay Node Placement in Heterogeneous Wireless Sensor Networks," *INFOCOM 2007*.
15. B. Hao, H. Tang, and G.L. Xue, "Fault-Tolerant Relay Node Placement in Wireless Sensor Networks: Formulation and Approximation," *High Performance Switching and Routing (HPSR 2004)*, pp. 246–250, 2004.
16. D.S. Hochbaum, and W. Maass, "Approximation Schemes for Covering and Packing in Image Processing and VLSI," *Journal of the ACM (JACM)*, vol. 32, issue 1, pp. 130–136, Jan. 1985.
17. S. Khuller, and B. Raghavachari, "Improved Approximation Algorithms for Uniform Connectivity Problems," *Journal of Algorithms*, vol. 21, pp. 214–235, 1996.
18. F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli, "Fault Tolerance in Wireless Sensor Networks," in *Handbook of Sensor Networks*, I. Mahgoub and M. Ilyas (eds.), CRC press, Section VIII, no. 36, 2004.
19. B. Krishnamachari, and S. Iyengar, "Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 241–250, 2004.
20. X.Y. Li, P.J. Wan, Y. Wang, and C.W. Yi, "Fault Tolerant Deployment and Topology Control in Wireless Networks," *MobiHoc 2003*, Annapolis, MD, 2003.
21. H. Liu, P.J. Wan, and X. Jia, "Maximal Lifetime Scheduling for Sensor Surveillance Systems with K Sensors to 1 Target," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 12, Dec. 2006.
22. H. Liu, P.J. Wan, and X. Jia, "On Optimal Placement of Relay Nodes for Reliable Connectivity in Wireless Sensor Networks," *Journal of Combinatorial Optimization*, vol. 11, pp. 249–260, 2006.
23. E. Lloyd, and G. Xue, "Relay Node Placement in Wireless Sensor Networks," *IEEE Transactions on Computer*, vol. 56, pp. 134–138, 2007.
24. X. Luo, M. Dong, and Y. Huang, "On Distributed Fault-Tolerant Detection in Wireless Sensor Networks," *IEEE Transactions on Computers*, vol. 55, no.1, pp. 58–70, 2006.
25. S. Nathy, P.B. Gibbons, S. Seshany, and Z.R. Anderson, "Synopsis Diffusion for Robust Aggregation in Sensor Networks," *SenSys'04*, November 3–5, 2004, Baltimore MD.
26. G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," In *SenSys'05: Proceedings of the third International Conference on Embedded Networked Sensor Systems*, New York, 2005.
27. Y. Wang, and H. Wu, "DFT-MSN: The Delay/Fault-Tolerant Mobile Sensor Network for Pervasive Information Gathering," *INFOCOM 2006*, Barcelona, Spain, 2006.
28. W. Zhang, G.L. Xue, and S. Misra, "Fault-Tolerant Relay Node Placement in Wireless Sensor Networks: Problems and Algorithms," *INFOCOM 2007*, Anchorage, AL, 2007.