36

Fault Tolerance in Wireless Sensor Networks

Farinaz Koushanfar
University of California at Berkeley

Miodrag Potkonjak
University of California at Los Angeles

Alberto Sangiovanni-Vincentelli
University of California at Berkeley

36.1 Introduction

36.1.1 Motivation

The reliability of computer, communication, and storage devices was recognized early as one of the key issues in computer systems. Since the 1950s, techniques that enhance the reliability of computer and communication systems have been developed in academia and industry. It has been recognized that as complexity of computing and communication devices increases, fault tolerance will gain more importance. Surprisingly, it has never been the major design objective, mainly because reliability of individual components has been increasing at a much more rapid pace than was expected. In addition, creative packaging and cooling schemes have tremendously reduced the stress factor on computation and communication systems.

The only component of fault tolerance that has received a great deal of attention in industry is offline testing. The modern testers are $10+ million systems that are contributing increasingly to the cost of modern microprocessors. In addition, the percentage of logic that supports testing has been rapidly increasing in the last 10 years, from less than 1% to more than 5% of the total transistor count.

The rapid growth of the Internet was the first major facilitator of renewed interest in fault tolerance and related techniques such as self-repair [8, 9, 10, 13]. Because the Internet requires a constant mode of operation, a special effort has been made to develop fault-tolerant data canters. The emergence of

wireless sensor networks will further increase the importance of fault tolerance while at the same time imposing a number of unique new conceptual and technical challenges to fault tolerance researchers.

At least three major groups of reasons support research in fault-tolerant sensor networks receiving significant attention. The first one is related to the technology and implementation aspects. At least two components of a sensor node, sensors and actuators, will directly interact with the environment and be subjected to a variety of physical, chemical, and biological forces. Therefore, they will have significantly lower intrinsic reliability than integrated circuits in fully enclosed packaging. In addition, wireless sensor nodes are exceptionally complex systems in which a variety of components interact in a complex way.

Furthermore, hundreds or maybe thousands of these nodes will form a distributed embedded network system that will handle a variety of sensing, actuating, communicating, signal processing, computation, and communication tasks. Wireless sensor networks will often be deployed as consumer electronic devices that will put significant constraints on cost and therefore quality of used components. More importantly, nodes operate under strict energy constraints that limit energy budgets dedicated to testing and fault tolerance.

The second reason is that applications will be equally as complex as the involved technology and architectures. More importantly, sensor networks will often operate in an autonomous mode without a human in the loop; security and privacy concerns will often prevent extensive testing procedures. This will adversely affect not only testing and fault tolerance but also related tasks such as debugging, in which reproduction of specific conditions during which a fault occurred will be difficult. Also, applications will require that sensor nodes are often deployed in uncontrolled and sometimes even hostile environments. Finally, and maybe most importantly, many applications of sensor networks will be safety critical and could have an adverse impact on humans and the environment, particularly when actuators are used.

The final reason is that, because wireless sensor networks are a new scientific and engineering field, the best way to address a particular problem is not quiet clear. In this situation, it is also difficult to predict accurately the best way to treat fault tolerance within a particular wireless sensor network approach. In addition, technology and envisioned applications for wireless sensor networks are changing at a rapid pace. For example, if one considers power consumption, each particular scheme will depend significantly on the relative power consumption of different approaches. Specifically, if communication energy is significantly higher than computation energy, it is important to develop localized algorithms that will require only a limited amount of communication.

Therefore, with respect to fault tolerance, it is important to consider schemes that conduct error detection using only local information. If one wants to ensure fault tolerance during the sensor fusion, the goal is to design fault-tolerant techniques that do not significantly increase the communication overhead. On the other hand, if the computation energy is significantly higher than the communication requirements, it is a good idea to support communication resources at one node with computation resources at other nodes. It is preferable to develop fault-tolerant sensor fusion approaches that require little additional computation regardless of any additional communication requirements.

36.1.2 Objectives

The primary goal of this chapter is to survey the field of fault tolerance in sensor networks. Fault tolerance is considered at four different levels of abstraction, starting from hardware and system software and going to the middleware and application layers. Fault tolerance is examined at each level of six individual components of a node: computing engine; communication and storage subsystems; energy supply; sensors; and actuators. It is also considered at the level of a node, as well as at the network level. Finally, resiliency against errors, where wireless sensor networks are treated as embedded distributed systems, is discussed.

Three aspects of fault tolerance are considered: fault models, error detection and diagnosis techniques, and resiliency mechanisms. In order to provide in-depth treatment of specific approaches, two case studies will be presented: one on error detection in sensor networks and one on heterogeneous built-in self-repair (BISR)-based fault tolerance. Finally, in order to provide a global vision, discussion will center

on the relationship to sensor networks and traditional fault tolerance techniques as well as a set of predictions of future research directions in this field.

The next two sections provide relevant preliminary information. After that, fault tolerance is discussed at the node and network levels. Next, two case studies address in a more comprehensive way several technical details with respect to fault tolerance in sensor networks. Finally, future directions along the three dimensions of fault tolerance are suggested.

36.2 Preliminaries

36.2.1 Sensor Network

A wireless sensor network is a system of small, wirelessly communicating nodes in which each node is equipped with multiple components. In particular, each node has a computation engine; communication and storage subsystems; a battery supply; and sensing and, in some cases, actuating devices. Such a network is envisioned to integrate the physical world with the Internet and computations. The power supply on each node is relatively limited, and frequent replacement of the batteries is often not practical because of the large number of nodes in the network. Therefore, energy is the most constraining factor on the functionality of these networks. In order to save energy, nodes only use short-range communications, which have been proven to consume much less energy than long-range communications [44]. Short-range communication between the nodes implies localized interaction in the network.

There is a need to model the different components of a sensor network. Sensor networks are often abstracted and mapped into a graph in which each vertex corresponds to a wireless node and an edge corresponds to the communication between two nodes. If the communication between the nodes is bidirectional, the mapped graph of the network will be nondirected. However, if this communication is asymmetric, then the mapped graph becomes directed. The communication model between the nodes can be one to one, or one to many. In the one-to-one model, each node sends and receives messages from only one of the communication edges. In the one-to-many model, each message sent out by a node can be heard by all of its neighbors. Because of the great variety of different sensors, in terms of their functionality and in terms of their underlying technologies, providing a reasonable and practical model for sensors and actuators is a very complex task.

Many potential applications are envisioned for sensor networks. For example, they can be used in a battlefield, where they can detect and spy on enemies or support the positive forces. Also, they can be used in intelligent security systems in buildings and security-critical applications. They can be used for habitat-monitoring applications in which they can monitor and study changes in phenomena for a long time. A number of comprehensive surveys on sensor networks have been conducted [1, 24, 45].

36.3 Example of Fault Tolerance in a Sensor Network System

The problem of fault-tolerant multimodal sensor fusion for digital binary sensors can be informally introduced using the example shown in Figure 36.1(a) and (b). A sensor network recognition system is deployed in an office to identify people in that office as they walk in through the main door. Six people, named A, B, C, D, E, and F, work in the office. The system consists of two different types of sensors: (1) a height sensor, which is a set of light sensors in series; and (2) a voice recognition sensory system that requires everybody entering the room to speak a given pass phrase into a microphone. Figure 36.1(a) shows the selected identification characteristics of people in the office. Figure 36.1(b) shows the same characteristics mapped to a two-dimensional plot.

It is easy to see that the system can distinguish between two persons, P_1 and P_2, if they fall into different squares, when mapped to the chart shown in Figure 36.1(b). If all of the sensors work properly, each person will naturally fit into a different square, according to the figure. For most of the cases, even if one of the height sensors or voice sensors fails, the recognition of the right person is still possible. This is

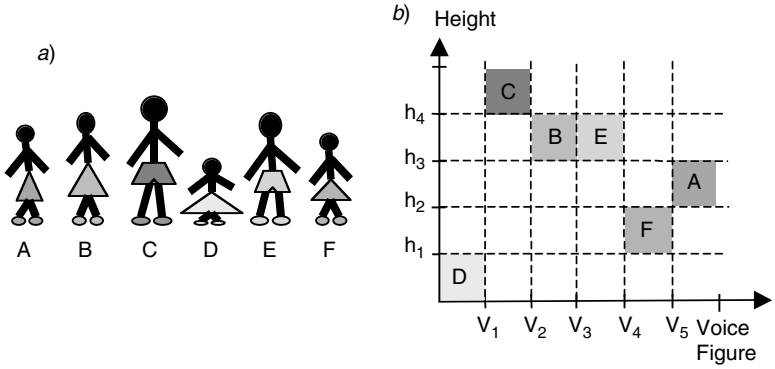

FIGURE 36.1 Example of a multimodal sensor network system. (From Koushanfar, F. et al., *IEEE Sensors*, 2, 1491–1496, June 2002. With permission.)

accomplished using heterogeneous fault tolerance, in which a failed sensor of one type can be replaced by the functionality of a sensor of another type. However, for the case of persons B and E, who are the same height, voice figure is the only way to distinguish the two persons, so the system does not have any fault tolerance to the failure of sensor v_3 that distinguishes between the objects B and E. If the office had only five people (excluding either B or E), then it would be completely fault tolerant. Complex sensor network systems can be designed in such a way that the heterogeneous fault tolerance scheme can make the system resilient to the failure of a specified number of sensors of specified modality.

36.4 Classical Fault Tolerance

Fault tolerance emerged early as a major concern during design of digital computing systems. For example, the Bell Lab's relay-based computer was performing multiple computations for the same inputs using the same program, in order to compare results and detect potential temporary malfunctioning. Also, UNIVAC 1, the first computer commercially available in 1951, utilized parity checking and arithmetic unit replication to enhance its reliability. At the same time, fault tolerance received research attention in the mid 1950s. Moore and Shannon [36] and von Neumann [42] conducted studies on how to design systems that preserve functionality after a subset of components experiences failures. Also, embedded computing systems such as Bell Lab's electronic switching system had extensive support for fault tolerance, mainly through enhanced serviceability features [16]. Furthermore, Apollo's mission to the Moon used triplicated computers in order to maximize the chances of success [46].

During the 1970s, research on fault tolerance started to diverge along several lines. Initially, the two most important and influential were fault tolerance in VLSI-based systems and fault tolerance in distributed systems [30, 31, 38, 40]. More recently, fault tolerance in data bases [7, 22], the Internet (e.g., reliable multicast and reliable distributed storage, peer–peer), and self-repair have been topics of major importance. In VLSI systems, fault tolerance has been addressed at all levels of abstraction, including circuit, logic, register, transfer, program, and system levels. It is common that reliable system design is discussed within three stages of life of a product: design, manufacturing, and operational. Before going into more technical details, it is important to define the basic entities.

Error is the manifestation of a fault inside a program. It is important to note that error can occur not only at the fault site, but also at some distance. Fault is an incorrect state of hardware or a program as a consequence of a component's failure. Permanent faults are continuous and stable in time. For example, permanent hardware faults are consequences of irreversible physical alteration within a component. An intermittent fault is one that has only occasional manifestation due to unstable characteristics of the hardware, or as a consequence of a program being in a particular subset of space. Finally, a transient fault is the consequence of temporary environmental impact on otherwise correct hardware. For example, often the impact of cosmic radiation may be transient.

Fault tolerance considers three main types of concerns: fault models; fault detection and diagnosis; and resiliency mechanisms. Each level of abstraction has its own types of faults [47]. For example, at the gate level, several fault models have been successfully used in the testing phase. One model is "stuck" where the logical value on interconnect gate or pin is permanently set to a value stuck at one or stuck at zero. Another model is the "bridging" fault model, in which two or more neighboring signal lines are physically connected introducing wired AND or wired OR functions, depending on the logic family used. Shorts and opens are another class of faults and correspond to missing or additionally introduced connections, respectively. Finally, in unidirectional faults, an error occurs in the same logical direction when a certain single failure occurs. Typical examples are an open circuit in a memory-select line resulting in a particular word incorrectly read as all ones, instead of all zeros. It is interesting and important to note that almost all testing approaches assume a single fault model, regardless of which type of fault is considered.

Reliability techniques compromise the following phases:

- Fault confinement establishes limits of fault effects over a particular area; therefore, contamination of other areas is prevented.
- Fault detection is a phase in which it is recognized that an unexpected event has occurred.
- Fault latency is the time that passes between the fault occurrence and the moment when the fault is detected.

Traditionally, fault detection techniques are classified into offline and online detections. Most often, for offline detection, special diagnostic programs are employed during idle periods of time or using multiplexing with a regular mode of operation. Online detection targets real-time fault identification and is performed simultaneously with a real work load. Typical online detection techniques include parity checking, duplication, and triplication.

- Diagnosis is a stage at which the exact occurrence of a fault is attributed to a specific atomic piece of hardware.
- Reconfiguration is the stage entered after diagnosis at which the system is restructured in such a way that faults do not have an impact on the correct output. Graceful degradation is a reconfiguration technique in which performance of the system is reduced, but the correct functionality is preserved.
- Recovery is a stage at which an attempt is made to eliminate the effects of faults. The two most widely used recovery techniques are fault masking and retry. The fault masking approach uses redundant correct information to eliminate the impact of incorrect information. In retry, after the fault is detected, a new attempt to execute a piece of a program is made in the hope that the fault is transient.
- Restart is the stage invoked after the recovery of correct, undamaged information. In cold restart, a complete resetting of the system is conducted.
- Repair is the stage during which the failed component is substituted with the operational component.

A number of excellent surveys, special issues, and textbooks on fault tolerance are available [25, 32].

36.5 Fault Tolerance at Different Sensor Network Levels

36.5.1 Physical Layer

The physical layer is responsible for establishing communication in a given medium between two nodes. Typical tasks at this level include modulation–demodulation and encoding–decoding. Traditionally, fully hardwired solutions have been used in order to minimize cost and maximize energy efficiency. A software radio is a wireless communication device in which parts or all of the physical layer functions are realized in software [33].

Software radios are a way to extend programmability into the physical layer and to enable adaptation to channel conditions. It has been demonstrated that adaptive link layer techniques can significantly improve the performance of wireless networks [17–19, 21]. The first commercially available radio was the speakeasy device [29]. Comprehensive surveys about software radios and related technologies have been conducted [34, 35, 41]. Currently, the major commercial application driver is incompatibility between the number of cellular and PCS communication standards. The primary reason for deployment of hardware and software radios has been to solve interoperability problems and to enhance performances in noisy media; however, they are also ideally suited for realization of a variety of fault tolerance techniques at the physical layer. For example, if some components of the software radio are not functional, one can switch to modulation and encoding schemes that can be realized with the still operational hardware resources. Adaptation to noise characteristics can also be considered from the fault tolerance point of view.

36.5.2 Hardware

At the hardware level, components can be divided into two groups. The first consists of a computation engine, storage subsystem, and power supply infrastructure that are very reliable. Exceptionally reliable systems are available that incorporate sophisticated fault tolerance techniques; even off-the-shelf microprocessors and DSP processors and controllers are very reliable devices with very low rates of malfunctioning.

At least three main reasons indicate why this does not necessarily imply that computational subsystems of sensor nodes will be exceptionally reliable. The first is that sensor nodes are very cost sensitive and therefore will not always be able to design using the highest quality components. The second is that strict energy constraints imply that repeated computations are often not realistic options. The third is that these systems are often deployed in much harsher environments than those in which today's computers function. Although programmability and flexibility are of high importance in sensor networks, strict energy constraints will result in extensive use of application-specific designs that can have up to two orders of magnitude less energy consumption for the same functionality. For these subsystems, heterogeneous BISR fault-tolerant schemes will simultaneously provide the targeted level of fault tolerance and low energy consumption [48].

With respect to storage components, several options exist. Memories can be divided into volatile and nonvolatile. Volatile memories are used for short-term storage; the best known and most widely used are SRAM and DRAM. Due to their relatively simple and regular structure, both types are very reliable, in particular DRAM. Starting with 4-Mb components, BISR has been often used to enhance the yield of DRAM memories. Note that memories are designed using the standard semiconductor processes. Nonvolatile memories such as flash and MRAM are also very reliable. Flash has the restriction that one cannot write too many times at the same location, so it will be used mainly for storage of system programs. Therefore, it can be concluded that storage systems are generally fault resilient and will very rarely incur a fault tolerance bottleneck.

With respect to energy supply, two parts of the energy supply subsystem can be distinguished. Traditional energy sources are rechargeable batteries and fault tolerance is achieved by providing a back-up battery. The first commercial fuel cells and subsystems that leverage on energy scavenging have recently been demonstrated. Although it appears that fuel cells will be very reliable, some energy-scavenging subsystems, such as the ones converting light into energy, can have very volatile performance. For energy distribution, the standard solution to enhance fault tolerance is to deploy multiple distribution networks.

Another component that greatly depends on the surrounding environment is the wireless radio. The standard way to enhance the performance of radios is to use aggressive error correction schemes and retransmission. These two schemes are examples of time redundancy. In addition, several schemes have been proposed in which two or more radios are used. Although the primary goal of these approaches is to save energy, they can also be used to enhance fault tolerance. In addition to the

schemes that operate on the physical and link layer, it is important to mention techniques that operate at the network layer. However, these schemes are more naturally considered and implemented at the system software level.

As mentioned previously, sensors and actuators are the subsystems most prone to malfunctioning. In the case of sensors, three types of faults can be distinguished: (1) calibration systematic error; (2) random noise error; and (3) complete malfunctioning. The first two can be addressed through time redundancy; however, the last one is enhanced using hardware redundancy. Depending on the type of data and information processing, an effective fault tolerance scheme for sensors can be accomplished using heterogeneous BISR techniques [26]. Currently, no scheme other than hardware redundancy is envisioned for actuators.

36.5.3 System Software

System software consists of the operating system (OS) and utility programs. Fault tolerance at the system software level can be addressed in several ways with respect to the computational subsystem [4–6]. Probably the most promising is through software diversity: each program is implemented in n different versions in hope that different versions will not have identical bugs [2, 3]. The subsystem that can most benefit from fault tolerance realized at the system software level is the communication unit. For example, one can reroute messages using different paths in the multihop network. With respect to sensors and actuators, the most important piece of system software is the one related to calibration. Recently, a number of schemes have been proposed for this task [12, 43]. A very important component of system software is the one that supports distributed and simultaneous execution of localized algorithms. For example, in the case of energy minimization under functionality constraint requirements, several protocols have been developed for the coordination of distributed actions [14, 28]. It is important to note that when communication protocols are considered, there is a clear trade-off between complexity and effectiveness.

36.5.4 Middleware

At the system software level, in addition to the OS of the individual nodes, networking (communication) plays the most dominant role. Starting with the middleware level, emphasis is shifted toward data aggregation, data filtering, and sensor fusion. These are tasks mainly related to sensor readings. Because it is difficult to provide fault tolerance in an economic way at the level of a single sensor, numerous fault tolerant approaches for this task will appear at the middleware level. Although currently the majority of applications are very simple, in order to address real-life applications, it is necessary to develop much more complex middleware. In order to combat software faults, n-versioning is one of the options [2, 3].

In particular, heterogeneous approaches that can substitute the readings of one type of sensor with the readings of another type are very important because of their low overhead. Another important issue solely related to middleware is how many sensors of each type should be placed on which positions on a particular node. If error resiliency of communication is much higher than the error resiliency of sensors, solutions in which sensors of the same node are placed on the same node will be favored.

36.5.5 Application

Finally, fault tolerance can be addressed at the application level. For example, to identify a particular person, one can use sensors try to measure a variety of biometric features of that person. Each feature and possibly a combination of features will be sufficient to identify that person. Addressing fault tolerance at the application level may be very efficient; unfortunately, any given application will require a customized way in which to address the issue. On the other hand, an additional advantage of application-level fault tolerance is that it can be used to address faults in essentially any type of resource.

36.6 Case Studies

36.6.1 Heterogeneous Fault Detection

Before this case study is described, key facts and assumptions about fault models, fault detection, and embedded sensor networks will be briefly outlined.

Each sensor node has five components: computation; communication; storage; sensors; and, often, actuators. Widely accepted fault and error models for processors; FPGA-based components; SRAM and DRAM; nonvolatile memory and disks; and communication systems are readily available. However, the situation for actuators and sensors is very different. Both types of resources are conceptually more complex and intrinsically more diverse to allow for simple, yet realistic and widely applicable fault and error models.

Koushanfar et al. [26] restricted attention on faults in sensors. They adopted two fault models, the first of which is related to sensors that produce binary outputs. In this case, obviously, one can envision a number of applicable fault models. For example, one model can capture probability or statistics of erroneous reported results. Nevertheless, it appears that the most logical model with potentially largest applicability range is the permanent fault model in which the only possible outcomes are that the sensor is functional or not. For this fault model, the fault detection procedure is often straightforward — usually, just observing the output of the sensors.

The second fault model is related to sensors with continuous (analog) or multilevel digital outputs. The fault models for this type of sensor are even additionally more complex and diverse. They propose to measure the level of discrepancy of the output of individual sensors with the multimodal model used for fusion as the indication of the level of error in that sensor.

The approach has two key advantages. The first is that fault tolerance approaches are such that the developed technique is applicable to a great variety of fault models. This approach is particularly well suited for addressing transient errors and errors in measurements. The second advantage is that the approach simultaneously addresses fault detection and correction. Overall, Koushanfar and colleagues made only mild assumptions; the main one was that the majority of sensors were functioning correctly.

The sensor resource assignment (SRA) problem can be formulated in the following way:

INSTANCE: Set A_1 of points p_i (x_{i1}, …, x_{im}) in m-dimensional space where $1 \leq i \leq n$, a positive integer J_1, set H that consists of $m(n-1)$ $[m-1]$-dimensional hyperplanes that are perpendicular to one of the m axes, such that each hyperplane is separating two points p_i and p_j that have the closest coordinates along the axis to which the hyperplane is perpendicular.

QUESTION: Find a subset of selected hyperplanes H, such that any two points p_i and p_j are separated by at least one of the selected hyperplanes and also the cardinality of H is, at most, J_1.

CLAIM: Sensor resource allocation (SRA) is NP-complete.

Next, these authors present their approach and algorithms for fault-tolerant sensor assignment. It is easy to envision a monolithic solution that simultaneously considers fault tolerance requirements and sensor allocation and assignment problems; however, following principles of separation of concerns and orthogonality, they designed a fully modular system with separate optimization mechanisms for the subtask: sensor assignment, sensor allocation, and fault tolerance. These three steps are addressed in the following way.

Koushanfar and coworkers employed two different algorithmic engines to the SRA problem: ILP based and simulated annealing based. The rationale behind the integer linear programming (ILP) approach is that although ILP solvers are often not fast, they are attractive because they guarantee an optimal solution. In addition, many smaller instances of practical importance can be solved using this approach. The point is that they must find the solution to the SRA problem before the deployment so that it is a one-time expense in computational time on the workstation and may be acceptable. In cases when ILP is not applicable, they provide the option of using simulated annealing as the optimization mechanism.

The ILP formulation for the SRA problem can be stated in the following way.

INPUTS: set of n, m-dimensional points $p_i(x_{i1}, x_{i2}, \ldots, x_{im})$, $1 \le i \le n$; set of all possible tests T, with elements t_k $(1 \le t_k \le m(n-1))$, where the $(l(n-1)+1)$ to $(l+1)(n-1)$ tests are in dimension l, $1 \le l \le m$, each separating two closest points in that dimension. The cost of each test t_k is c_k.

Variable X_k: $X_k = 1$ if test t_k is selected and $X_k = 0$ otherwise.

Objective function: to minimize the total cost of all of the selected tests. In other words:

OF:

$$\sum_{k=1}^{m(n-1)} x_k \cdot c_k$$

The constraint of the problem is that for each pair of points p_i and p_j, at least one test has a different outcome when applied to these two points. These authors define an auxiliary matrix $A[n \times k(m-1)]$, with constant elements a_{ik}, as $a_{ik} = 1$ if the test t_k produces 1 on point p_i and $a_{ik} = 0$ otherwise.

Using the matrix A and the variables, Koushanfar et al. find a linear expression that produces zero, if a test produces similar results on the two points p_i and p_j, and one otherwise. One such expression is $X_k \times (a_{ik} + a_{jk}) \times (1 - a_{ik} \times a_{jk})$, which has the required property. Therefore, to have a different test result on each set of two points p_i and p_j, they write the following constraints.

CONSTRAINTS: For each pair of points p_i and p_j,

$$\sum_{k=1}^{m(n-1)} x_k \cdot (a_{ik} + a_{jk}).(1 - a_{ik}.a_{jk}) \ge 2$$

A standard simulated annealing code was used. The four components of simulated annealing (moves — neighborhood structure; objective function; cooling schedule; and stopping criteria) are defined in the following way. Move is the replacement of one sensor with another sensor of the same type; the goal is to maximize objective function. The standard geometric cooling schedule was used. Finally, as stopping criteria, the user-specified number of steps in which the improvement did not occur were used.

The resource allocation is conducted in the following way. The number of sensors that is lower bound on the potential solution is proposed as the initial solution. The bound is calculated assuming that all dimensions have the same number of sensors and each n-dimensional compartment will eventually contain one point. After that, the simulated annealing RSA algorithm is run. During this running process, the move is modified so that one type of sensor can be replaced with another type. Statistics about which type of sensor helps the most to improve objective function after each move are accumulated and this information is used to decide which type of sensor to add or remove.

For fault tolerance, one can envision three different mechanisms:

- The first is to specify in the ILP formulation or in the simulated annealing code that each two points must be separated by at least r hyperplanes. Because this approach essentially doubles the redundancy, this alternative was not accepted.
- The second alternative is to add exactly one extra sensor of each type to the solution generated by the sensor resource allocation problem. When a large number of sensor nodes of each type is used, the overhead is relatively low. Also, in this case, the need for storing or communicating more than one resource assignment solution is eliminated. Therefore, if moderate levels of fault tolerance are needed, this can be an attractive alternative.
- The final and most attractive alternative in terms of overhead is to leverage on heterogeneous back-up of sensors of different modality. Here, allocation is generated in the following way. First, the cost of an overall solution is calculated for each type of sensor for all allocations k, from 1 to smaller than the number allocated in the best resource allocation solutions. Then the cost of all these solutions is plotted on the y-axis on the graph, where the x-axis is the number of allocated sensors of analyzed type. In such a way, m graphs, where m is the number of sensors of different

modality, are obtained. Obviously, now it is necessary to use the RSA algorithm to analyze only allocations worse in terms of cost than the optimal solution and better than the solution from the second alternative. This analysis is conducted in the order dictated by increasing cost of the proposed solution.

The applicability of the preceding technique can be generalized, and therefore enhanced, in a number of ways. One possibility is to characterize objects using statistical data and to build a statistical model for decision making using data from sensors. Another, equally important and with equally large application, domain option is to conduct multimodal sensor fusion in order to support the decision process. As a matter of fact, multimodal, multilevel sensor fusion has emerged as one of the canonical problems in sensor networks. Informally, it can be defined in the following way: a number of sensors, some of them with different modalities, are given; the goal is to extract the information requested by a user as accurately as possible from noisy measurements.

Although the problem seems too general to be solved efficiently using a single approach, it can be addressed in a systematic way. It is necessary to develop, or even better to find, some already developed analytic models related to the measured quantities. Once the equations of an analytical model are assembled, the intriguing and important question is to try to figure out which measurements are faulty or have a high degree of noise. One way to answer this question is to try to find a subset of measurements that produces a consistent set of analytic models. Using this set of equations, the value for all quantities of interest can be calculated. Therefore, the key to providing fault-tolerant multimodal sensor fusion is to generate a model of the physical world that is rich enough to ensure that the system is solvable even when some of the equations are not used. The main difficulty is that the systems of equations are often nonlinear and therefore it is very difficult to say in advance when the system is well defined in a sense that it can be uniquely solved.

Probably the best way to clarify the introduced approach is to take a closer look at an example. For this purpose the scenario illustrated in Figure 36.2 will be used. An object O moving along its trajectory, which includes points p_i in an embedded sensor network, consists of a number of nodes, each represented by a shaded circle n_i. Four types of sensors — RSSI-based distance discovery; speedometer; accelerometer; and compass — are used to measure the angle in two-dimensional physical space. Three RSSI-based measurements can be used to locate the object O in any particular moment. Euclidian space, Newton mechanics, and trigonometry laws can be used to establish relationships between measurements.

Specifically, Equation 36.1 through Equation 36.9 are trilateration equations; Equation 36.10 through Equation 36.13 are Newton law equations and Equation 36.14 and Equation 36.15 are trigonometry laws. The key observation is that more equations (15) than variables (12) may have errors. Thus, if one sensor is not functioning, it can be calculated from the established system of equations. Also, for each variable, one can find how much it must be altered in order to make the whole system of equations maximally consistent; variables that must be altered most are most likely measured by faulty sensors. Therefore, one

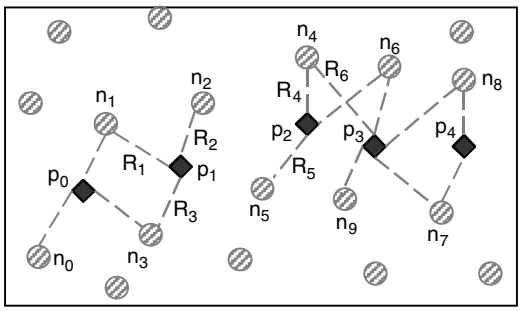

FIGURE 36.2 Sensors tracking an object. (From Koushanfar, F. et al., *IEEE Sensors*, 2003. With permission.)

way to identify and correct sensor measurements is to try all scenarios in which exactly one type of sensor measurement is not taken into account and compare the maximal error in the system. Another very important observation is that, by sampling all operational sensors more often, one can compensate for faulty sensors.

$$(x_1 - s_1)^2 + (y_1 - t_1)^2 = R_1^2 \tag{36.1}$$

$$(x_1 - s_2)^2 + (y_1 - t_2)^2 = R_2^2 \tag{36.2}$$

$$(x_1 - s_3)^2 + (y_1 - t_3)^2 = R_3^2 \tag{36.3}$$

$$(x_2 - s_4)^2 + (y_2 - t_4)^2 = R_4^2 \tag{36.4}$$

$$(x_2 - s_5)^2 + (y_2 - t_5)^2 = R_5^2 \tag{36.5}$$

$$(x_2 - s_6)^2 + (y_2 - t_6)^2 = R_6^2 \tag{36.6}$$

$$(x_3 - s_7)^2 + (y_3 - t_7)^2 = R_7^2 \tag{36.7}$$

$$(x_3 - s_8)^2 + (y_3 - t_8)^2 = R_8^2 \tag{36.8}$$

$$(x_3 - s_9)^2 + (y_3 - t_9)^2 = R_9^2 \tag{36.9}$$

$$\sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2} = \frac{1}{2} \cdot a_1 (\Delta t)^2 + v_1 (\Delta t) \tag{36.10}$$

$$\sqrt{(x_3 - x_2)^2 - (y_3 - y_2)^2} = \frac{1}{2} \cdot a_2 (\Delta t)^2 + v_2 (\Delta t) \tag{36.11}$$

$$a_1 . \Delta t = v_1 - v_0 \tag{36.12}$$

$$a_2 . \Delta t = v_2 - v_1 \tag{36.13}$$

$$\alpha_1 = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1} \right) \tag{36.14}$$

$$\alpha_2 = \tan^{-1}\left(\frac{y_3 - y_2}{x_3 - x_2} \right) \tag{36.15}$$

36.6.2 Discrepancy-Based Fault Detection and Correction

The work by Koushanfar et al. [27] introduces a cross-validation-based technique for online detection of sensor faults — an approach that can be applied to a broad set of fault models. These authors define a fault as an arbitrary type of inconsistent measurement by a sensor that cannot be compensated systematically. In particular, they consider faults associated with incorrect measurements that cannot be corrected using calibration techniques. The approach is based on two ideas:

- Comparing the results of multisensor fusion with and without each of the sensors involved
- Using nonparametric statistical techniques to identify measurements that are not correctable, regardless of the mapping function used between the measured and accepted values

Sensor measurements are inevitably subject to errors of two types: (1) random fluctuations in data due to a noise in a sensor or in a sensed phenomenon; or (2) gross errors — faults. A practical method to distinguish a random noise is to run maximum likelihood or Bayesian approach on the multisensor fusion measurements. A random noise would exist if running these procedures improves the accuracy of final results of multisensor fusion. Although several efforts have attempted to minimize random errors, very little has been done for fault detection. In multisensor fusion, measurements from different sensors are combined in a model for consistent mapping of the sensed phenomena. Although the new fault detection technique is generic and can be applied to an arbitrary system of sensors that uses an arbitrary type of data fusion, for the sake of brevity and clarity they focus on equation-based sensor fusion [26].

Assume a set of sensors s_i $(0 \le I \le n)$, each measuring a value x_i at a time t. The multimodal sensor fusion model equations are $f_1, \ldots, f_p$ and are typically nonlinear functions with the following forms: $f_j(x_1, x_2, \ldots, x_n) = 0$, $(0 \le j \le p)$. The system of equations is overconstraint and solves the system $n + 1$ times. First, they solve with all the equations in the original format; then they ignore each variable and solve a least-constrained system with $n - 1$ variables (n times). They compare the values for each variable x_n in all $n + 1$ scenarios. In order to improve accuracy of fault detection, the system can be solved for m measurements by each sensor. At last, they conduct statistical analysis on the data for each sensor. If the obtained values for a sensor are not consistent within a confidence interval calculated by the percentile method [20], that sensor is considered faulty.

36.7 Future Research Directions

It is well known that it is very difficult to predict anything; nevertheless, certain directions will inevitably attract a higher level of research interest because of their intrinsic importance. Future research directions can be classified into four groups. The first two are related to fault models and testing. The third is related to resiliency mechanisms and the last to analogies between fault tolerance and other domains such as power minimization and security.

The development of theoretically attractable and realistic fault models is one of the key prerequisites for development of sound and real-life relevant fault tolerance techniques for sensor networks. Apart from fault tolerance models for components such as computation, storage and, to serious extent, communication are available; however very little has been published in terms of fault models for sensors and actuators. At the same time, these components are the most important for overall system fault tolerance. The development of fault models for sensors will be particularly difficult due to the great variety of their types, environments in which they will be deployed, and requirements in terms of fault tolerance of various applications. For example, it is clear that electromagnetic and mechanically based sensors will have fault characteristics very different from those of biological and chemical sensors.

Also, sensors deployed in harsh environments, such as nuclear plants, will have very different characteristics from those of sensors deployed in friendly environments, such as offices and residential areas. An intrinsic trade-off takes place between more complex fault models and relatively simple ones. In the VLSI domain, only the simplest fault models, such as those stuck at one and those stuck at zero, have been extensively used. However, in addition to these models, more complex models will be required for sensors. In particular, it will be interesting to see the kinds of fault models developed for the sensors common in a number of biological and chemical sensors that can be used only once for reading. In the VLSI domain, testing received significantly greater attention and application range than fault tolerance did. Testing needs to be addressed not only on component and individual node levels, but also at network and distributed system levels.

Closely related to testing is calibration, which can be defined as the process of mapping row sensor data to a new set of data that is, according to some statistical measure, more accurate than the initial

readings. Calibration can be done offline and online. In the former case, the emphasis will be on the accuracy and strict interval of confidence; in the latter, the focus will be on the localized mode of operation. Also, calibration — not only of sensor readings but also of other parameters relevant to operation of sensor networks, including timing and the available energy level at each node — should be conducted.

At the application level, fault resiliency mechanisms required for common applications such as sensor fusion, data filtering, and data aggregation will be of primary importance. It is important to observe that, for each of these applications, a significant variety of approaches will be used. For example, in addition to equation-based sensor fusion, sensor fusion based on graphs, statistics, and stochastics will be possible. Each of these techniques has a number of unique peculiarities. Although specific fault resiliency techniques will not be developed for each of them, the primary emphasis will be on fault tolerance techniques that can be applied to multiple classes of approaches.

An interesting relationship exists between fault tolerance and several other fields. One of the main constraints in the deployment and operation of wireless sensor networks is energy. The most effective way to prolong the lifetime of the network is to place a subset of nodes in sleep mode. For example, power consumption of the software radio in three operational modes (transmission, receiving, and idle) rarely differs for more than a factor of two. At the same time, energy consumption in sleep mode is most often lower by two orders of magnitude (sometimes even more). A simple and powerful observation is that a node in sleeping mode can be treated as faulty and vice versa. It will be possible to retarget theoretical results and algorithms and even software for one objective to the other relatively easily.

Security and privacy are a major concern. For example, a key question concerns the extent to which one can trust results obtained using sensor fusion or data aggregation in a particular scenario in a particular sensor network, assuming that one or more nodes are compromised. In order to study this question, it will be necessary to develop a threat model and models of attacks in one or more nodes in a sensor network. These attacks can be modeled as worst-case fault models. Another interesting and important observation is that any technique that is resilient against nonintentional faults could also be retargeted to intentional faults.

36.8 Conclusion

Because of potential deployment in uncontrolled and harsh environments and due to the complex arch, wireless sensor networks are and will be prone to a variety of malfunctioning. The goal in this chapter was to identify the most important types of faults, as well as techniques for their detection and diagnosis, and to summarize the first techniques for ensuring efficiency of fault resiliency mechanisms. In addition to a comprehensive overview of fault tolerance techniques in general, and in particular in sensor networks, techniques were discussed that ensure fault resiliency during sensor fusion as well as the approach for heterogeneous built-in self-repair fault tolerance. The chapter concluded by outlining potential future research directions along several dimensions.

Acknowledgment

This material is based upon work supported in part by the National Science Foundation under Grant No. ANI-0085773 and NSF CENS Grant.

References

1. I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cyirci, Wireless sensor networks: a survey. *Computer Networks*, 38(4), 393–422, 2002.
2. A. Avizienis and J.P.J. Kelly, Fault tolerance by design diversity: concepts and experiments, *IEEE Computer*, 17(8), 67–80, August 1984.

3. A. Avizienis, The *n*-version approach to fault tolerant software, *IEEE Trans. Soft. Eng.*, SE-11(12), 1491–1501, 1985.
4. K. Birman. Replication and fault-tolerance in the ISIS system. In *Proc. 10th ACM Symp. Operating Syst. Principles*, 79–86, 1985.
5. K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Trans. Computer Syst.*, 5, 47—76, February 1987.
6. K. Birman and R.V. Renesse, *Reliable Distributed Computing with the ISIS Toolkit,* IEEE Computer Society Press, 1994.
7. D. Bitton and J. Gray. Disk shadowing. *VLDB*, 331–338, 1988.
8. A. Brown and D.A. Patterson. Embracing failure: a case for recovery-oriented computing (ROC). *2001 High Performance Trans. Process. Symp.*, Asilomar, CA, October 2001.
9. A. Brown and D.A. Patterson. To err is human. *Proc. 1st Workshop Evaluating Architecting Syst. Dependability (EASY '01)*, Göteborg, Sweden, July 2001.
10. A. Brown and D.A. Patterson. Rewind, repair, replay: three *r*'s to dependability. *10th ACM SIGOPS Eur. Workshop*, Saint–Emilion, France, September 2002.
11. R.R. Brooks and S.S. Iyengar. Robust distributed computing and sensing algorithm. *IEEE Computer*, 25(6), 53–60, June 1996.
12. V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak. Colibration: a collaborative approach to in-place sensor calibration. *2nd Int. Workshop Inf. Process. Sensor Networks (IPSN'03*), 301–316, April 2003.
13. G. Candea, J. Cutler, A. Fox, R. Doshi, P. Garg, and R. Gowda. Reducing recovery time in a small recursively restartable system. *Proc. Int. Conf. Dependable Syst. Networks* (DSN-2002), Washington, D.C., June 2002.
14. B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad-hoc wireless networks. In *ACM MobiCom*, July 2001.
15. S. Chessa and P. Santi, Crash faults identification in wireless sensor networks, *Computer Commun.*, 25(14), 1273–1282, Sept. 2002.
16. R.W. Downing, J.S. Nowak, and L.S. Tuomenoksa, No 1 ESS maintenance plan. *Bell Sys. Tech. J.* 43(5), pt. 1, 1961–2019, September 1964.
17. D. Eckhardt and P. Steenkiste, Measurement and analysis of the error characteristics of an in building wireless networks. In *Proc. SIGCOMM*, 243–254, 1996.
18. D. Eckhardt and P. Steenkiste, A trace-based evaluation of adaptive error correction for a wireless local area network. *J. Special Topics Mobile Networking Applications (MONET)*, 1998.
19. D. Eckhardt and P. Steenkiste, Improving wireless LAN performance via adaptive local error control. *INCP*, 1998.
20. B. Efron, *The Jackknife, The Bootstrap, and Other Resampling Plans*. S.I.A.M., Philadelphia, 1982.
21. C. Fragouli, P. Lettieri, and M. Srivastava, Low power error control for wireless links. *ACM/IEEE MobiCom*, 139–150, 1997.
22. J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, California, 1993.
23. L. Guerra, M. Potkonjak, and J.M. Rabaey, High level synthesis techniques for efficient built-in-self-repair. *Int. Workshop DFT VLSI Syst.*, 41–48, 1993.
24. C. Intanagonwiwat, R. Govindan, and D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks. *ACM/IEEE MobiCom*, 56–67, 2000.
25. P. Jalote. *Fault Tolerance in Distributed Systems*. P.T.R Prentice Hall, Englewood Cliffs, NJ, 1994.
26. F. Koushanfar, M. Potkonjak, and A. Sangiovanni–Vincentelli, Fault tolerance in wireless ad-hoc sensor networks. *IEEE Sensors*, 2, 1491–1496, June 2002.
27. F. Koushanfar, M. Potkonjak, and A. Sangiovanni–Vincentelli, Online fault detection in wireless sensor networks. *IEEE Sensors*, 2003, to appear.
28. F. Koushanfar, A. Davare, D.T. Nguyen, M. Potkonjak, and A. Sangiovanni–Vincentelli, Low power coordination in wireless ad-hoc networks. ISLPED, Aug 2003.

29. R. Lackey and D. Upmal, Speakeasy: the military software radio. *IEEE Commun. Mag*., 33(5), 56–61, May 1995.
30. L. Lamport, R. Shostak, and M. Pease. The Byzantine general's problem. *ACM Trans. Programming Languages Syst*., 4(3), 382–401, July 1982.
31. L. Lamport, The weak Byzantine generals' problem, J. ACM, 30, 668—676, July 1983.
32. P.A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*, 2nd ed., Springer–Verlag, Heidelberg,1990.
33. N. Mandayam, A software radio architecture for linear multiuser detection. *IEEE JSAC*, 17(5), 814–823, May 1999.
34. J. Mitola, The software radio architecture. *IEEE Commun. Mag*., May 1995.
35. J. Mitola, Technical challenges in the globalization of software radio. *IEEE Commun. Mag*., February 1999.
36. E.F. Moore and C.E. Shannon, Reliable circuits using less reliable relays. J. Franklin Institute, 262, 191–208, September 1956.
37. V.D. Park and M.S. Caron, A highly adaptive distributed routing algorithm for mobile wireless networks. *Infocom* 1997.
38. M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2), 228–234, 1980.
39. L. Prasad, S.S. Iyengar, R.L. Kashayap, and R.N. Madan, Functional characterization of fault tolerant interaction in distributed sensor network. *Phys. Rev. E*, 49(2), April 1994.
40. F.B. Schneider, Byzantine generals in action: implementing fail-stop processors, *ACM Trans. Computer Syst*., 2(2), 145–154, May 1984.
41. W. H.W. Tuttlebee, Software-defined radio: facets of a developing technology, *IEEE Personal Commun*. Mag., 6(2), 38–44, April 1999.
42. J. von Neumann, Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C.E. Shannon and J. McCarthy, Eds. *Automata Studies*, Princeton University Press, 1956, 43–98.
43. K. Whitehouse and D. Culler, Calibration as parameter estimation in sensor networks. ACM WSNA, 2002.