A Survey on Fault Tolerance in Wireless Sensor Networks

Nancy Alrajei, Huirong Fu
Department of Computer Science and Engineering
Oakland University
Rochester MI 48309-4498
Email: nmalraje@oakland.edu, fu@oakland.edu

Abstract— Wireless Sensor Networks (WSN) are useful for monitoring of physical conditions and passing the data gathered to the required location. They are like any other system prone to failure due to limitation of resources. In this paper we address fault tolerance concept in general, give analysis to the definition of fault tolerance and terms related to it based on the system requirements. We explored redundancy and touched upon fault tolerance in Wireless Sensor Networks. Topics covered include redundancy in hardware, N- Modules Redundancy (NMR), and software including N-version programming and check pointing. We also cover fault tolerance in Wireless sensor networks including connectivity and coverage and multipath.

Keywords*—fault tolerance, fault prevention, redundancy, dependability; connectivity, coverage; multiple path*

1 INTRODUCTION

All human-made systems are fault-prone. With each technology emerges, we cannot predict if it will continue to work happily ever after or not. These technologies are vulnerable to failure and it is hard to predict when a failure will happen. Human-made systems are designed so as to minimize the occurrence of faults, yet it is understood that faults are bound to occur caused by wear and tear or by some overlooked conditions, design errors, or usage errors.

For most systems, the handling of faults in terms of diagnostic, repair, and maintenance occurs offline. For example with the system out of operation and extrinsically, handled outside of the system and not considered to be an intrinsic part of its functionality. This is the case for most home appliances, and entertainment systems. A failure of such systems is an annoyance, but is generally neither life threatening nor safety critical, and the cost of their being down or behaving erroneous is not prohibitive. On the other hand there are many systems and applications where down-time is too prohibitive and incorrect behavior carrying serious consequences.

With the increasing reliance on technology, the percentage of such systems is also on the rise. For which it is critical to design them to be fault-tolerant, i.e., to be intrinsically equipped to recognize the occurrence of faults, and react to them so that their behavior remains if not correct, at least safe.

This failure varies from mild failure that will not affect the functionality of the system to a sever failure that could damage the being of the system. The existence of this failure has inspired the researches to look for ways to survive. One of these ways is fault tolerance. *Fault tolerance* for any system means enabling that system to continue working in the presence of these faults, which requires understanding of possible faults, their sources, their likelihood, and their consequences and cost if they are not mitigated. It also requires an understanding of what constitutes a safe behavior in the context of the system at hand as well as knowledge of mechanisms by which such safety can be provided.

There are two primary assumptions for any system: first, it is not fault free, faults will happen sooner or later in the system, especially hardware, which will be worn out after years of use. Second, not every fault results in a server failure. Given its importance, fault-tolerance has been studied extensively in a variety of systems from those that are primarily mechanical, electrical, electronic, or software, to the increasingly complex and hybrid systems that have all aspects combined.

While some principles developed early on in the history of this field remained invariant, other characteristics, methods, and approaches are more closely related to the application domain and the technologies used and thus are still a highly researched topic.

This paper gives an understanding of fault tolerance from the system's requirements view of point. It also puts fault tolerance in clear point in the map of achieving dependable system. We discussed fault and failures so that as threats to the dependable system and offered fault tolerance as a mean to achieve it.

Moreover, we are interested in fault tolerance as it applies to sensor networks, and possibly to sensor networks under malicious attack. We surveyed the field of fault tolerance generally and fault tolerance in WSN specifically, we found that the key to fault tolerance is redundancy. It can be achieved by adding copies of certain components or alternatives that are not identical so that in the case of failure in one of them the system will switch to the redundant component

This paper is divided as follows: In Section 2, we discuss fault tolerance to achieve dependability. In Section 3, we explore redundancy to achieve fault tolerance. In Section 4, we talk about redundancy measures in WSN, coverage and connectivity, multiple routing. In section 5, we touched upon fault prevention in WSN. We conclude our work in Section 6.

1.2 Motivation of Fault Tolerance

Humans are not perfect, unexpected behavior could arise in any human-made system in the future. If the probability of a fault occurrence in the system is zero, then there should be no fault tolerance consideration. On the other hand we don't want to reach that point of when the system fails, but our goal is not to have them propagate into errors then finally failures. It is important to comprehend the concept of fault tolerance before the designing process of any system. It is essential to know where and when we can let go of faults, or if the need of fault tolerance is needed.

1.3 History of Fault Tolerance

The concept of fault tolerance emerged in the beginning of the 50's, the first scientist to design a fault tolerant computer SAPO was Antoion Svoboda, he designed it using relays and a magnetic drum memory. The processor used triplication and voting (TMR), and the memory implemented error detection with automatic retries when an error was detected. The same team developed a second machine (EPOS) also contained comprehensive fault-tolerance features. These features of these machines were motivated by the local unavailability of reliable components.

Over the past 50 years, a number of fault-tolerant systems have been developed and eventually, they separated into three distinct categories: 1) long-life, these machines last long with no maintenance such as NASA space probes. 2) Ultra-dependable, real-time computers, they require constant monitoring such as systems used to monitor nuclear plant. 3) High-availability computers with a high amount of runtime which would be under heavy use such as super computers used by insurance companies.

The way fault tolerance developed over the years was possible to create specific hardware and software solutions from the ground up, but now chips contain complex, highly integrated functions, and hardware and software must be crafted to meet a variety of standards to be economically viable. This field is improving as new technologies are developed and new applications arise, new fault-tolerance approaches are also needed.

1.4 System Requirements

When designing any system in general, there is no doubt that ultimate goal for this system is to maintain working according to its requirements all the time. Before designing any system, the system should go through different phases that are called the system lifecycle. The first step is defining the requirements of the system. The requirements are what the system should be doing, and what constrains it should operate under.

Requirements branch into two parts: functional requirements and non-functional requirements. Functional requirements mean what the system should be doing, such as the input and the output of the system, all the functionalities related to the system, and how the service should be delivered. On the other hand, the non-functional requirements represent the properties of the system, such as reliability, availability, fault tolerance, etc. The non-functional requirements also cover the constraints on the system. Constrains on the product, organizational constrains and other external constraints.

Constraints on the product would be derived from the user needs, such as the acceptable failure rate, how fast it is, etc. Organizational constraints have to do with organization policy and procedures like what language or design method to use. Finally, there are constraints that come from external sources, such as ethical requirements: Is this product acceptable to the public or not?. Legislative requirements: is the system within the law boundaries such as privacy or safety requirements. These non-functional requirements are very critical and application dependent. In some applications like medical related applications, safety is considered awfully important. In real time applications, reliability is important. Other application related to aircrafts and space shuttles, fault tolerant is highly required.

Fault tolerance is considered a non-functional requirement, and any non-functional requirement should be verifiable. Instead of saying the system should be fault tolerant, we say the system should be able to tolerate this number of failures, or this amount of time between failures is acceptable. We should not confuse the non-functional requirement with the overall goal of the system. There is no doubt that the ultimate goal when designing any system is to maintain working according to its functional and non-functional requirements. When it meets these requirements it is called a dependable system.

2. A DEPENDABLE SYSTEM

The definition for a dependable system is different from one application to another. So what is known to be a dependable system in a bank system is different than airspace shuttle. Let’s take few examples:

In a bank system, it is important to the customer that the bank does not share the user’s information with third parties. And that any information shared with the bank system is not modified without the permission of that customer. So a dependable system in this case should satisfy the condition of confidentiality and integrity. Availability is also as important as the other two, some people cannot stand any delay in their bank transactions, but it does not cause a catastrophic result in the end.

In monitoring systems in general, such as weather or traffic monitoring systems, it is important to have correct information, so confidentiality and availability is not an issue here, but integrity is important. Another example, when designing any medical tool, such as robotic medical equipments all the focus is on the patient’s life. The system must not harm or threat their lives. So safety is the main non-functional requirement.

The questions after these examples ar: what is a dependable system? And what is dependability? And how is it related to fault tolerance? Over the past fifty years the means to achieve dependable computing and communicating services have been developed. The dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable to the user(s) [11].

What is acceptable to each user and what is not vary, but all users agree on some attributes of a dependable system. Considering different cases we can say that a dependable system should satisfy one or more form the following attributes depends on the type of the application and the environment this application works in:

- Availability: service should be there when it is needed.
- Reliability: continue to provide correct service.

- Confidentiality: only authorized people should access the information.
- Integrity: cannot be modified without authorization.
- Safety: system contains no threats to its users.
- Maintainability: the ability to be easily repaired, airplanes, cars, are expensive so it not realistic every time a car breaks down should be replaced by another one.

2.1 Threats to Dependability: Faults, Error, Failure

A threat will cause a violation to one or more of its required attribute. These threats could be noticed. Others could be noticed without affecting the functionality of the system. Other threats could be noticed and negatively affect not only the functionality of the system but also violates the requirements of the system. There are three types of threats, fault, error and failure. To comprehend the concept of fault tolerance it is important to explain the difference between these three threats.

A threat starts out as fault, which is a physical defect that takes place in some parts of the system. It could be a physical defect (hardware), or a software defect. It could be accidentally made or purposefully forced into the system. No matter what the category of this fault is, we need to contain these faults and not to have them propagate into something bigger and even more dangerous.

Faults could be active or passive. An active fault could be noticed, such as dead battery in a sensor. A passive fault couldn't be noticed. A mistake or a bug in the code is a passive fault. If a fault has emerged and was not taken care of, it might prorogate and affects other parts of the system, so it is no longer a defect, it will become an error. Error is a noticed phase that leads the system into a state of not doing things the right way. So a fault is active when it causes an error, otherwise it is dormant. If errors propagate they lead to a failure.

Fault→ Error → Failure

A failure is the deviation from correct service may assume different forms that are called service failure modes. It is important to note that many errors do not reach the system's external state and cause a failure. So since a service is a sequence of the system's external states, a service failure means that at least one external state of the system deviates from the correct service state. The deviation is called an error. An example of an error is lost connection between two nodes due to the dead battery in one of them. If an error propagates it leads to a failure. That is when the system starts to behave in a way that not what it should be doing. Follow our example of the dead sensor node, the failure here is not being able deliver the data from that dead node.

Faults could occur during the phase of creating the system, they are called development faults, but operational faults that occur during service delivery of the use phase. Faults are classified into eight categories:

- The location of the faults with respect to the system boundary, *internal* faults that originate inside the system boundary, but *external* faults that originate outside the system boundary and then propagate errors into the system by interaction or interference.
- The cause of the faults either *natural* faults that are caused by natural phenomena without human intervention. Or *human-made* faults that result from human actions.
- The dimension in which the faults originate either *hardware* (physical) faults that originate in, or affect, hardware. Or *software* faults that affect software, i.e., programs or data.
- The objective of introducing the faults could be *malicious* faults that are introduced by a human with the malicious objective of causing harm to the system. Or it could be the opposite, *non-malicious* faults that are introduced without a malicious objective.
- The intention of the human who caused the faults, *deliberate* faults that are the result of a harmful decision or *non-deliberate* faults that are introduced without awareness.
- The capacity of the human who introduced the faults *accidental* faults that are introduced inadvertently or incompetence faults that result from lack of professional competence by the authorized human, or from *inadequacy* of the development organization.
- The temporal persistence of the fault, either *permanent* fault whose existence is assumed to be continuous in time. Or *transient* faults whose presence is bounded in time.

So, the existence of faults might or might not lead to a failure in the end. We should have these faults under control and not have them end up in a system failure when repair is harder or impossible. Failure starts off not meeting a functional requirement that results in failing in one or more of the non-functional requirements. That leads us to definition of *fault tolerance* is that to maintain the system working according to its non- functional requirements even in the presence of functional faults.

2.2 Means to achieve dependability

The dependability in the system, eliminate the occurrences of the threats in the system by predicting all the possible faults that are likely to happen. If we have faults under control that means we can prevent system failures. The more faults we prevent the less failures might occur. This step is called fault prevention. Fault prevention is to prevent the occurrence or introduction of faults.

Another mean, is fault tolerance, which is to avoid service failures in the presence of faults. It has things in common with fault prevention in the sense that they both should know (predict) before hand what faults the system might face and provide some mechanism to avoid them. Fault tolerance concern is not only preventing faults from happening but it also it assumes that these faults will actually happen and find solutions on how the system will survive and continue working according to its goals without any deteriorating in its performance of services it provides.

What comes after fault tolerance? Should the system continue working in the presence of faults? Where is this fault going? The next step that comes after fault tolerance is fault masking which means hiding faults and prevent them from resulting in errors. Finally, fault isolation; this move the fault from system either manually or automatically. Every mean of these is a topic of research on its own.

Redundancy is used to achieve dependability in the system at different levels, hardware, software, and time redundancy:

2.2.1 Hardware Redundancy (NMR N Modular Redundancy)

Hardware redundancy is duplicated elements (hardware) that can work in parallel usually this part is considered a critical one. Such as TMR Triple Modular Redundancy, that has 3 duplications of a certain module then it uses a voting mechanism to decide the output of the system.

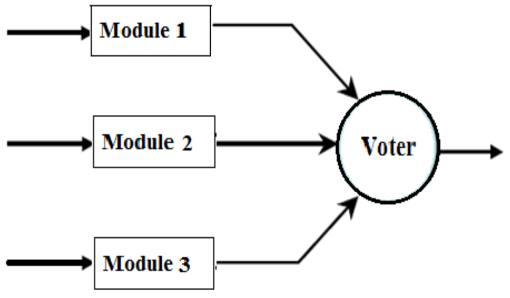

Figure 1. Triple modular redundancy

In Figure 1, if one of these modules has a fault, the other two will have the same output that should be considered the faulty one will be ignored. Triple is the minimum number of modules to be used, it also has to be an odd number. The more number of duplications indicates the more faults tolerant the system is, when the number of duplicated modules increase, the reliability will also increase. In some fault sensitive system, such as space shuttles, not only they duplicate certain modules but they also duplicate the voter. They assume that the voter itself might have faults. In this case, the ending outputs should match from all voters.

2.2.2 Software Redundancy

There are two types of redundancy in software, static software redundancy, by using *N* versions of the software to execute one task. The other type is dynamic software redundancy, which is through the creating of checkpoints that the system goes back to after the state of fault. The difference between them is that in the static approach we don't need any error detection mechanism, whereas in dynamic we need error detection mechanism.

a) Static Software

That is called *N*- Version programming, so a single task will be executed many times by *N* number of programs where $N \geq 2$. The idea here is to give the specifications for that program to different *N* developers that will result in N versions. A majority voting is performed on the result which makes it similar to hardware redundancy NMR. This technique has two main drawbacks. First, it is expensive because each version is developed by different developers. Second, *N*- version is expensive because each version is developed by a different programming team independently, if the specifications that they were given were flawed, the system will deal the *N* independently errors, which makes its maintenance hard as well.

b) Dynamic Software redundancy

All approaches in dynamic redundancy depend on detecting the error first, then recover from it and then move on. It is hard for software to redo any error that was detected, so the solution is to create checkpoints at the beginning of a transaction. A transaction is a collection of operations on state of an application. Two examples of dynamic software redundancy, check-pointing is to take a snapshot of the software situation before it starts the following transaction if and only if the previous transaction was completed without any faults. This state is saved in a block called check pointing block. The system tries a different alternative block of processing then tests again, if fails go through another alterative processing block and so on. Once an error is detected the system moves back to this saved stated in the check pointing block, this is called rollback, see Figure 2. Backward error recovery, is the basic recovery strategy, the system rollback refers to the state of recovering from a failure after repair.

Recovery Blocks is similar to *N*- version programming, but *N* versions of processing. The difference is that here we don't run them at the same time.

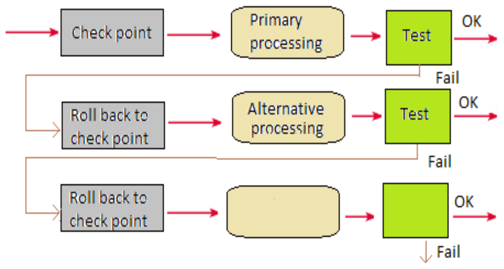

Figure 2 Recovery blocks

works like this for a certain transaction, there is a check-pointing block that saves the state of the system before it process this transaction, then a test to check for errors, if an error was detected it rolls back to the saved state in the checkpoint block.

2.2.3 Time Redundancy

It refers to the repetition of a given computation a number of times and a comparison of the results to determine if a discrepancy exist [18]. The discrepancy between the multiple computations indicates a transient fault as in Figure 3. So a computation is done more than one time with time difference without any voting here, but there a comparison between the results to determine if they are consistent. If there is inconstancy in the result, that indicates a fault existence. Usually this scheme is useful if the fault is not permanent if it is permanent then there is no need to continue computing and comparing.

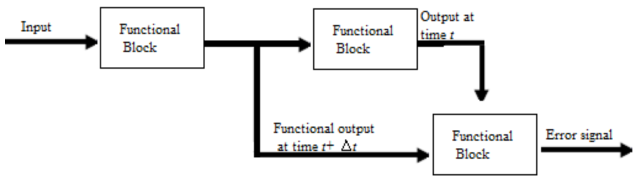

Figure 3. time redundancy basic scheme

According to Figure 3 computation is first performed at time τ using the input data in the functional block, then the result is saved in a register, the same data is used again to repeat the computation in the functional block at time $\tau + \Delta\tau$. the comparator will give an error if the two results mismatch.

Another scheme in Figure 4 shows the two sets of resources represent space redundancy and the sequential computations represent time redundancy. In the figure, the top processing resources shows permanent fault, time redundancy is not capable of tolerating it, but is adequate to tolerate the transient fault in the lower resource. This scheme is appreciated for transient faults not for permanent ones.

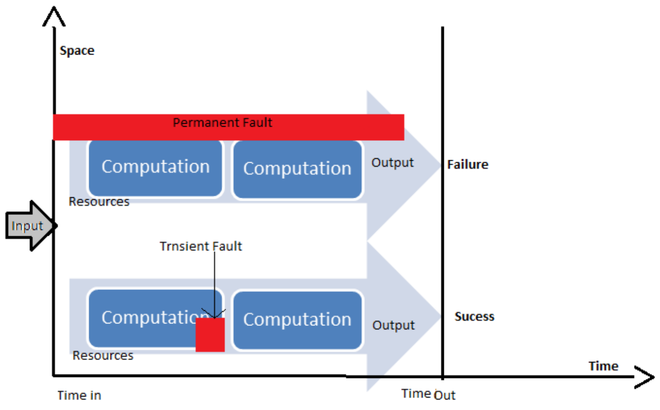

Figure 4. Time and Space Redundacy

3 Fault tolerance in wireless sensors Networks

Sensor network are small devices that are deployed over the field of interest, each one is consists of sensor batteries, processing unit and memory. These sensor nodes can be used in different applications such as environmental monitoring, military applications and health care applications and others. The resources for these sensor nodes are limited, the most critical component is the node’s battery, once the battery dies the node dies, and eventually the network dies. These nodes sense data and either communicate with other nodes or directly to the base stations.

Wireless sensor networks are fault prone because of the limited resources, and because nodes are deployed in harsh environment they are vulnerable to different threats. Faults are possible to happen and lead to life treating failure. For example physical damage to the node itself, or delay in sending receiving the data, or the data itself is could be wrong. To understand the type of faults that could happen, we categorized the faults in each of the system layers as in Figure 5.

3.1 Classifications of fault in WSN

Each node consists of different components: Hardware level, software level, links for communications, and others. Sensor nodes are cost sensitive and will not always be designed using the highest quality components, they are also deployed in harsh environments that make them prone to failure. To understand the type of faults that could happen, we categorized the faults in each of the system layers as in Figure5

3.1.1 Node level

Each node consists of hardware and software components. The hardware is: sensors, CPU, memory, and battery. And the software: routing protocol, Mac and other things. Hardware failure will generally lead to software failure in the node. Low power level in a node can give incorrect readings. The node as whole might fail if that node was in a critical position such as if that nodes lies on the routing path to the sink.

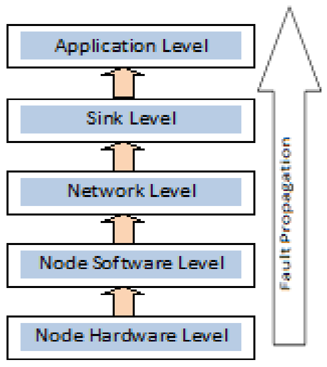

Figure 5 Fault Classification

3.1.2 Network level

The radio communication between nodes is also vulnerable. When these nodes communicate wirelessly, two nodes might send their data at the same time which will end up in losing or colliding of data. The main reason of collision is congestion, when allowing sensing nodes to transfer as many packets as possible at same time, so packet loss is the result of this congestion.

3.1.3 Sink level

The sink or the base station is responsible for collecting all the data from the sensors. This component could fail due to many reasons. It could be the wrong of the sink, it could be the presence of bugs or other problems in the hardware or the software of the sink. Or it could be that it is not possible to supply power to the sink permanently.

3.1.4 Application level

The malfunction nodes in the wireless sensor network, it does not perform the intended operation according to user requirements, or it performs other operations like sending incorrect information to the base station. Hence the faulty sensor node must be identified and separated from being a communication node.

3.2 Classifications of Failures in WSN

These failures are results of the faults that happened in one on the above levels.
These failures could be either one of the followings:

- *Crash failure*: occurs when the service doesn't respond to the request. It could be because of message loss. Or physical damage that broke the network apart into different disjoint components that the sink can longer communicate with.
- *Time failure*: if the application has a strict time interval, the node responds to a request with possibly the correct value but response was received either too early or too late.
- *Incorrect value failure:* generating an inaccurate value, by either a software malfunction, corrupt message, or a malicious. This failure results in lack of accuracy in the aggregated final value.

4. MEASURES OF REDUNDANCY IN WIRELESS SENSORS NETWORKS

We talked in general about redundancy as the main technique for fault tolerance. WSN consist of one major component the node itself. It can also be provided with redundancy, what kind of redundancy?

Each node has two ranges as in Figure 6:

- *Sensing range Rs*: a node can sense any point in the field that is located within this range. That represents the sensing *coverage.*
- *Communication Range Rc:* A node can communicate with or any node that is located within this range. Neighbors are nodes that are allocated within this communication disk. A node is connected with its neighbors.

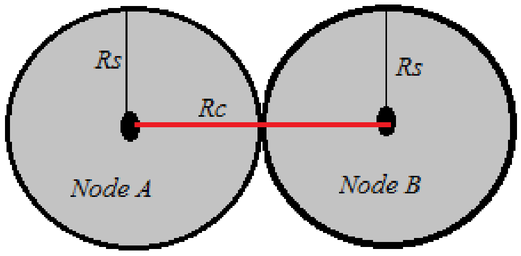

Figure 6 Sensing Range Rs, and Communication Range Rc

Fault tolerance in WSNs is tightly dependent on node redundancy to maintain coverage and connectivity. To explain redundancy in WSNs sees Figure 7 it is a snapshot of the same area, one that captures the sensing disks, the other captures the network connectivity.

4.1 K-Connectivity (Neighboring Redundancy)

Connectivity refers to the fact that between any pair of nodes *(n, n')* there is a sequence $n_0=n, n_1, n_2, \ldots n_k=n'$ such that the distance between any n_i, n_{i+1} is $\leq R_c$. In other words the network is said to be *K*-connected if there are *K* number of disjoint paths from node *u* to node *v*. Every node along this path can communicate with the next node if their Euclidian distance less than its communication range.

We understand that having a *K*-connected network where $K \geq 2$, if any node dies or any link breaks between 2 nodes the network is still functioning because there is an alternative path to the sink. The higher the *K* the more it provides routing redundancy, hence network resilience.

4.2 K-Coverage (Node redundancy)

Sensor node redundancy provides a degree of sensing coverage same area. We say a region is *K* covered if any point in that region is covered by at least *K* different sensing disks. If any node dies, then for each spot there are *K-1* other nodes will take its place and sense that data instead of the dead node. When we have good coverage, $K \geq 2$, we guarantee more accuracy and better quality of service (QoS).

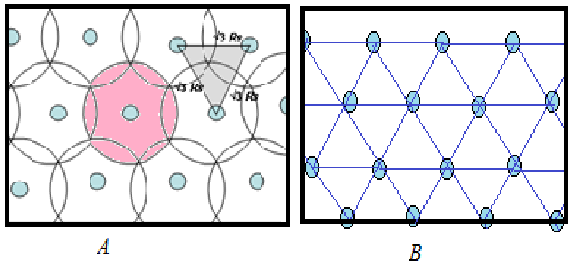

Figure 7 (A) Coverage, (B) Connectivity

If it is covered it should be also connected to guarantee that the sensed data is received by the base station, otherwise what is the point of coverage if the nodes are not able to connect. Many algorithms were proposed to obtain coverage by the predefined replacement of the nodes in the field. See Figure 7 (A), optimal pattern to obtain coverage by organizing the nodes in a lattice of equilateral triangles of side $\sqrt{3}\, R_{s}$..

The areas of coverage and connectivity are closely related. Much work has been done to discuss the relationship between them. Each is necessary conditions for a functional wireless sensor network. There are attempts have been made by researches to combine the two into a single algorithm. If connectivity and coverage are both to achieve at same time then, the optimal configuration depends on the relationship between R_s and R_c. If the communication range is equal to or larger than $\sqrt{3}R_s$, the optimal configuration for coverage more than satisfies connectivity, as in Figure 7. An important principal to consider is that if the communication range of the sensors is at least twice that of the sensing range then coverage of an area implies connectivity, hence the network is covered and connected. On the other hand, if the region is covered and $Rc < 2Rs$; then it is not connected, there are protocols out there to achieve connectivity in this case.

4.3 Multiple Routing

Multiple paths have been used to provide load balance to the network, and to add rout redundancy which improve the reliability of data delivery. Three different approaches are discussed that utilize multipath routing.

4.3.1 Meshed multipath

Meshed multipath such as GRAB[10] sensor node forwards a data packet to several neighbors as long as the total power consumed by that packet is still within a certain given budget. The number of neighbors to which an under-budget packet is forwarded is static. Mesh is more reliable, even it may cause some delay in data delivery, and it works well in highly dense networks.

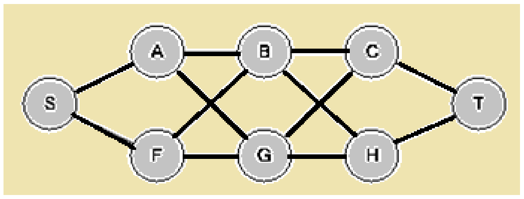

Figure 8 Meshed Multipath

4.3.2 Disjoined multipath

Disjoint paths share the same source and sink but none of the intermediate nodes, one primary path and one or more alternative paths. It is best used when the topology is known in the network; it has less delayed than mesh, but less reliable. Because any node along this path is important if it fails we lose the whole line of connection.

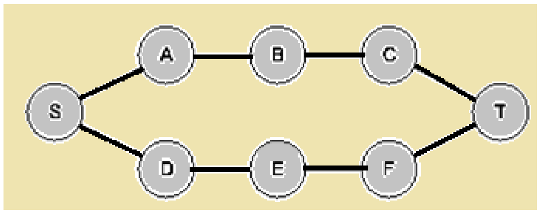

Figure 9 Disjointed Multipath

4.3.3 Braided multipath

As the name suggests it looks braided. It is partially disjoined paths then they joined again, for each node on the primary path find an alternate path does not including that node. There is not a lot of overhead and latency will be added to the primary path. It works well when only one node or few nodes fails, but when the nodes along the whole primary path fail, a new path discovery should be found.

In ad-hoc networks, multipath routing is used to rapidly find alternate paths between source and destination, and so guarantee a better robustness of the network. From the application's perspective, a desirable goal of multipath routing is to deliver data along the primary path. To recover from failure of this primary path, without flooding the network for rediscovery, multipath routing constructs and maintains a small number of alternative paths.

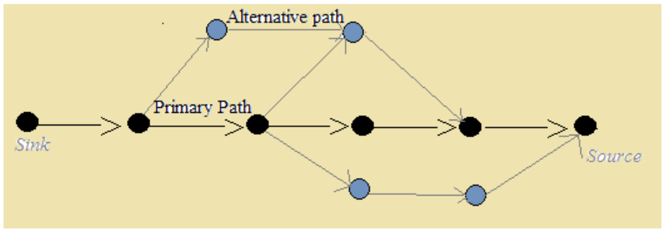

Figure 10 Braided Multipath

Which multipath is more reliable? The disjoint network in Figure 9 has two mini-paths (s; a; b; c; t). The mesh network in Figure 8 has eight mini-paths (s; a; b; c), (s; a; b; h; t),(s; a; g; h; t),(s; a; g; c; t) In general by adding an operational minipath to a graph, the graph reliability increases.

Reliability can be calculated using this equation:

$\mathrm{Rel(G)} = \sum_{i=1}^{h} Prob[D_i]$ .. (1)

let P_1….. P_h be the enumeration of min-paths and let the event E_i be the event that path P_i is operational. The Boolean formulation of reliability uses the events $D_1 = E_1$and

$D_i = E_1 \cap E_2 \cap E_1 \dots \cap E_{i-1} \cap E_i$ (2)

adding a minpath never decreases the reliability hence:

Rel (meshed)> Rel (disjoined)

Each routing has its own benefits, the disjoin paths can achieve traffic balancing and bandwidth increase with intelligent path selection; the braided multipath can gain reliability and robustness

Disjoint multipath traffic can be many times faster than braided multipath routing. Thus, disjoint multipath can send *r* copies of the same packet in sequence.

In [10] they compared all three; they found that Packet error probability of a single path (no redundancy) is the highest among all. Followed by disjoint multipath, and then braided multipath has the least packet function of channel/link error rate.

5. Fault Prevention techniques in WSN

The following section is not about fault tolerance techniques, rather more of fault prevention techniques. People confuse the two concepts with each other. Fault tolerance techniques use redundancy in different forms. But we need techniques to prevent faults from happenings.

Monitoring is important to have eyes open on the area where failure might happen. Monitoring the network either by the base station, or self node collaborated monitoring is important to capture any abnormal behavior in the network and probably deal with it before it causes more damage. It could be by sending constantly reports about the behavior of the network, or reporting could be only when something suspicious happens. What is it that we need to monitor?

5.1 Monitoring Node Status

The status of the nodes could be the physical status of the node, so it will be looking for damaged or crashed nodes [5]. Or it could be watching for the most scare resource, residual power. Network reliability and forecasting network behavior is one king of energy management and forecasting. The dynamic network behavior can be captured effectively by the energy management and forecasting scheme. This can be useful for predicting network failures and taking preventive actions accordingly. This way, the reliability of the network can be maintained within dynamic sensor network environment. Monitoring the residual power could be done using many techniques:

An eScan in[6] depicts an aggregated picture of the remaining energy levels for different regions in a sensor field. Instead of the detailed information of residual energy at individual sensors, this scan provides an abstracted view of energy resource distribution. An eScan can help users to decide where new sensor nodes be deployed to avoid energy depletion.

Forecasting-based Monitoring and Tomography (FMT) in [7] is energy monitoring mechanism for resource constrained sensor networks. Instead of collecting the raw available energy information from individual sensor nodes periodically, it applies energy forecasting and network aggregation mechanisms to capture the network energy tomography map with minimum energy consumption to continuously update the energy information with minimum energy waste. Each sensor node sends its available energy and its forecasted energy dissipation rate to the monitoring node. This way, energy information is transmitted to the monitoring node only when there is a variation in the network behavior. Instead of collecting the raw energy information from individual nodes like eScan [6], FMT apply energy forecasting and network aggregation mechanisms together in order to further reduce the monitoring costs in the network.

5.2 Monitoring Link Quality

Link quality estimation in WSN is more challenging than any other network, because of the use of low power radios are more prone to failures. There are different estimators resented and evaluated in[8] some of them are :

***PRR*(Packer Reception Rate):** measure the average of successfully at the receiver side using this :

$$PRR = \frac{\text{Number of received packets}}{\text{Number of sent packets}} \qquad (3)$$

Some statistical data is collected to determine the lost packet such as the sequence number for each packet.

***RNP*(Required Number of Packets Transmitted)**: counts the average number of packets transmission/ retransmission. The sender determines the successful received packets as the number of acknowledged packets

$$RNP = \frac{\text{Number of transmitted and retransmitted packets}}{\text{Number of successfully received packets}} - 1 \qquad (4)$$

***ETX* (Expected transmission count):** an approximate number of transmitted and retransmitted packets required before a successful reception (RNP):

$$ETX = \frac{1}{\text{PRR forward} * \text{PRR backward}} \qquad (5)$$

where PRR forward uplink quality from sender to receiver, and PRR backward is downlink quality from receiver to sender. The combination of both gives an estimation of the bidirectional link quality**.**

***Four-bit*:** a hybrids estimator uses passive and active monitoring initiated at the sender side.

5.3 Monitoring congestion

Network congestion occurs when a link or node is carrying so much data that its quality of service deteriorates. That results in delay, packet loss or the blocking of new connections. To avoided congestion from happening there are different congestion control mechanisms such:

CODA (Congestion Detection and Avoidance) in Sensor Networks [9] is an energy efficient scheme that involves three mechanisms, Congestion detection, Open-loop hop-by-hop backpressure and Closed-loop multi-source regulation. It is based on sampling scheme of local channel loading at appropriate rate and form accurate estimation of congestion, if congestion detected nodes send signals to the neighbors.

5. CONCLUSION

In this paper we present a clear overview on fault tolerance techniques in general, and then we focused on WSN in specific. Some systems are critical such as health related systems which require high resiliency against failures. Fault tolerance is all about Redundancy in the system component, when one module fails we switch over to the redundant one. In general fault tolerance techniques are categorized as hardware such as *TMR* or software such as *N*-version programming.

In Wireless sensor networks, it is important of the network to continue functioning and provide the service needed. The need for fault tolerance is important because maintenance is not an option. It could be achieved by using *K*- connectivity, *K* coverage, and multipath. These techniques make the network more resilient. We also touched upon fault prevention in WSN, by monitoring the health of the node, links, power level, and congestion.

Finally, fault tolerance is strongly related to security, it deals with all threats that might attack the system including intruders from humans, who attack the system exploiting any weak hole in it. But fault tolerance does not look at who caused the fault, it focuses on how continue working with fault with minimum casualties.

REFERENCES

[1] L Paradis, Q Han, "A Survey of fault management in WSN," Journal of Network and Systems Management, 2007

[2] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli, "Fault tolerance in wireless sensor networks," in Handbook of Sensor Networks, M. Ilyas and I. ahgoub, Eds. CRC Press, 2005, ch. VIII.

[3] M Yu, H Mokhtar, M Merabti, "Fault Management in WSN," IEEE Wireless Communications, 2007

[4] Guoliang Xing , Xiaorui Wang , Yuanfang Zhang , Chenyang Lu , Robert Pless , Christopher Gill, "Integrated coverage and connectivity configuration for energy conservation in sensor networks," ACM Transactions on Sensor Networks (TOSN), v.1 n.1, p.36-72, August 2005.

[5] S Chessa, P Santi, "Crash faults identification in wireless sensor networks," Computer Communications, 2002

[6] Y Zhao, R Govindan, D Estrin, "Residual Energy Scan for Monitoring Sensor Networks," In Proceedings of the IEEE 2002

[7] VC Gungor, "Efficient available energy monitoring in wireless sensor networks," International Journal of Sensor Network, 2008

[8] N. Baccour, A. Koubaa, M. Ben Jamaa, H. Youssef, M. Zuniga, and M. Alves "A comparative simulation study of link quality estimators in wireless sensor networks," in MASCOTS, 2009.

[9] Wan, Chieh-Yih, Shane B. Eisenman, and Andrew T. Campbell. "CODA: congestion detection and avoidance in sensor networks." Proceedings of the 1st international conference on Embedded networked sensor systems. ACM, 2003.

[10] Soon Y. Oh, Mario Gerla, and Abhishek TiwariRobust, "MANET routing using adaptive path redundancy and coding," first international conference on communication systems and network India, 2009

[11] Ammari, H. M., Das, S. K. "Fault tolerance for large scale WSN," ACM Transactions on Autonomous and Adaptive Systems, Volume 4 , Issue 1 , Article No.: 2 Year of Publication: January 2009, ISSN:1556-4665

[12] J. Thelen, D. Goense, and K. Langendoen, "Radio wave propagation in potato fields," In First workshop on Wireless Network Measurements (co-located with WiOpt 2005), Rivadel Garda, Italy, Apr. 2005.

[13] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a sensor network expedition," In Proceedings of the First European Workshop on Sensor Networks (EWSN), Jan. 2004.

[14] S. Harte and A. Rahman, "Fault Tolerance in Sensor Networks Using Self-Diagnosing Sensor Nodes," In The IEE International Workshop on intelligent Environment, pages 7–12, June 2005.

[15] Daler Rakhamtov , Sarma Vrudhula, "Time-to-failure estimation for batteries in portable electronic systems," Proceedings of the 2001 international symposium on Low power electronics and design, p.88-91, August 2001, Huntington Beach, California, United States

[16] Ann T. Tai, Kam S. Tso and William H. Sanders, “Cluster-based failure detection service for large-scale ad hoc wireless network applications,” in: Proc. of the 2004 International Conference on Dependable Systems and Networks (DSN'04) (Florence, Italy, June 2004) pp. 361–370.
[17] S. Rost et al. Memento, “A Health Monitoring System for Wireless Sensor Networks,” in SECON 2006.
[18] Abd-El-Barr, Mostafa, “Design and analysis of reliable and fault-tolerant computer systems,” World Scientific, 2006.
[19] Chen, Liming, and Algirdas Avizienis. “N-version programming: A fault-tolerance approach to reliability of software operation,” Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8). 1978.
[20] Bidgoli, Hossein. “Handbook of Information Securit Information Warfare”, Social, Legal, and International Issues and Security Foundations” Vol. 2. John Wiley & Sons, 2006.
[21] David Kalinsky, “Design of high availability embedded By, Embedded Systems Programming,” March, 03,URL: **http://www.eetimes.com/story/OEG20020729S0030**
[21] Goutam Kumar Saha, “Software based fault tolerance: a survey”, Ubiquity, v.7 n.25, p.1-1, July 5, 2006 - July 10, 2006
[22] A. Avizienis, J.-C. Laprie, and B. Randell, “Dependability and its threats: a taxonomy,” In Proceedings of the 18th IFIP World Computer Congress. Building the Information Society, volume 12, pages 91
[23] Florio, Vincenzo De, and Chris Blondia. "A survey of linguistic structures for application-level fault tolerance," ACM Computing Surveys (CSUR) 40.2 (2008): 6
[24]Felix Gärtner, Felix C. "Fundamentals of fault-tolerant distributed computing in asynchronous environments." ACM Computing Surveys (CSUR) 31.1 (1999): 1-26.
[25] Pleisch, Stefan, and André Schiper. "Approaches to fault-tolerant and transactional mobile agent execution---an algorithmic view." ACM Computing Surveys (CSUR) 36.3 (2004): 219-262.
[26] Cheatham, Jason A., John M. Emmert, and Stan Baumgart. "A survey of fault tolerant methodologies for FPGAs." ACM Transactions on Design Automation of Electronic Systems (TODAES) 11.2 (2006): 501-533.
[27]Cheatham, Jason A., John M. Emmert, and Stan Baumgart. "A survey of fault tolerant methodologies for FPGAs." ACM Transactions on Design Automation of Electronic Systems (TODAES) 11.2 (2006): 501-533.
[28] Mohapatra, Prasant. "Wormhole routing techniques for directly connected multicomputer systems." ACM Computing Surveys (CSUR) 30.3 (1998): 374-410.
[29] Medeiros, Raissa, et al. "Faults in Grids: why are they so bad and what can be done about it?." Grid Computing, 2003. Proceedings. Fourth International Workshop on. IEEE, 2003.
[30] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). “A survey on sensor networks. Communications magazine,” IEEE, 40(8), 102-114.
[31] Krishnamachari, B., & Iyengar, S. (2004). “Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks.” Computers, IEEE Transactions on, 53(3), 241-250.
[32] Gupta, G., & Younis, M. (2003, March). “Fault-tolerant clustering of wireless sensor networks.” In Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE (Vol. 3, pp. 1579-1584).
[33] Luo, X., Dong, M., & Huang, Y. (2006). “On distributed fault-tolerant detection in wireless sensor networks,” Computers, IEEE Transactions on, 55(1), 58-70.
[34] Chen, J., Kher, S., & Somani, A. (2006, September). “Distributed fault detection of wireless sensor networks.” In Proceedings of the 2006 workshop on Dependability issues in wireless ad hoc networks and sensor networks (pp. 65-72). ACM.