A Fault Tolerance Management Framework for Wireless Sensor Networks

Iman Saleh[†] Mohamed Eltoweissy[†] Adnan Agbaria[§] Hesham El-Sayed[‡]
[†]Bradley Department of Electrical and Computer Engineering, Virginia Tech
Email: {iman.saleh,toweissy}@vt.edu
[§]IBM Haifa Research Laboratory, Haifa
Email:adnan@il.ibm.com
[‡]College of Information Technology, United Arab Emirates University
Email: helsayed@uaeu.ac.ae

***Abstract*— Wireless Sensor Networks (WSNs) have the potential of significantly enhancing our ability to monitor and interact with our physical environment. Realizing a fault-tolerant operation is critical to the success of WSNs. The main challenge is providing fault tolerance (FT) while conserving the limited resources of the network. Many schemes have been proposed in this area. Our main contribution in this paper is to propose a general framework for fault tolerance in WSNs. The proposed framework can be used to guide the design and development of FT solutions and to evaluate existing ones. We present a comparative study of the existing schemes and identify potential enhancements. A primary module of the framework is the learning and refinement module which enables a FT solution to be adaptive and self-configurable based on changes in the network conditions. We view this as vital to the resource-constrained and highly dynamic WSNs. Up to our knowledge, we are the first to propose the implementation of such module in FT solutions for WSNs.**

***Index Terms*— Networked sensor systems, Fault tolerance, Checkpoint/Restart**

I. Introduction

Wireless Sensor Networks (WSNs) provide a bridge between the digital world and the physical world. Typically, a WSN is comprised of numerous tiny sensor nodes (or sensors for short) deployed in an environment for monitoring and tracking purposes. Sensed data are aggregated and, at times, stored "in-network" at sink nodes which may themselves be sensors or other nodes richer in capabilities and resources. Data are then communicated to the end users either periodically or on-demand through the sinks or a higher order node; the base station. Clearly, WSNs find numerous applications ranging from healthcare to crisis management and warfare. Many of these applications require a continual stream of dependable data with specific quality of service (QoS) requirements such as bounded delay or minimal packet drop.

This paper is based on "A Fault Tolerance Management Framework for Wireless Sensor Networks," by I. Saleh, H. El-Sayed, and M. Eltoweissy, which appeared in the Proceedings of the 3rd IEEE Conference on Innovation in Information Technology (IIT06), Dubai, UAE, November 2006. © 2006 IEEE.

4300 Wilson Blvd., Suite 750.Arlington, VA 22203.Tel: (703) 528-5500, Fax:(703) 528-5543

While WSNs inherit most of the dependability and QoS provisioning issues of wireless networks [3], [8], [17], their characteristics pose unique challenges that make present-day dependability and service differentiation schemes unsuitable. Sensors are battery operated and possess limited computing and communication capabilities. In addition, when deployed, these sensors are likely to operate unattended, closely interacting with their physical environment that may be hostile (for example, enemy territory or hazardous terrain). These limitations render WSNs more prone to failure than other wireless networks and mandate numerous tradeoffs, for example between safe mobility and performance, buffer space/bandwidth and data redundancy, Energy consumption and Quality of Data (QoD). We define QoD as the number of readings received by the user divided by total number of readings generated by the network during an observation period.

Realizing a fault-tolerant operation is critical to the success of a WSN. In addition to resource preservation and achieving high QoD, we identify the following as key requirements for FT in WSNs:

1) Awareness of the network main operation and the status of the network resources.
2) Adaptability to the frequent changes in WSNs conditions.

While several schemes satisfy the first requirement, only a few address the second requirement and to a limited extend. In addition, There is no common framework for a comprehensible comparison of FT schemes. The main contribution of this paper is proposing a general framework for fault tolerance (FT) in WSNs. The framework achieves the following:

- Provide general guidelines for the design and development of solutions for FT in WSNs.
- Present a unified approach for FT which can be used to identify main modules and compare and contrast different solutions.

To provide adequate adaptability to the network changes, a learning and refinement mechanism is needed. However, existing solutions were found to lack such mechanism. We present CRAFT (Checkpoint/Recovery-bAsed scheme

for Fault Tolerance), a fault-tolerant scheme for data collection and dissemination in WSNs. CRAFT applies in-network data checkpointing and recovery in order to achieve high Quality of Data (QoD) with minimum energy overhead. CRAFT integrates the FT tasks with the network main operation. The scheme is also adaptive to the changes in the network conditions, specially the energy level of the sensors. Based on the proposed framework, we present a comparative study of representative schemes including CRAFT. To the best of our knowledge, our work is the first to provide such handling of FT in WSNs.

The remainder of the paper is organized as follows. In section II, we describe the WSN model and basic definitions. In Section III, we describe the fault tolerance framework. In Section IV we overview the CRAFT scheme. In Section V, we discuss and compare related work. Finally, we conclude this work in section VI with future directions.

II. System Model and Definitions

Sensors are scattered in a *sensor field* as presented in Figure 1. We consider a homogeneous network in terms of node capabilities. The role of aggregating and forwarding data to the end user can be assumed by any sensor in the field with sufficient energy. Such a node is termed *sink*. Initially, sensors are *operational* having the capabilities to collect data and route data back to the sink. However, due to lack of energy, a sensor may die and become *non-operational* and cannot participate in any activity in the system.

We assume a homogeneous network; the sink functionality could be located in any sensor in the sensor field and hence the sink has the same limited capability as other sensors in the field. Assuming sinks with limited resources allows us to study more basic WSNs.

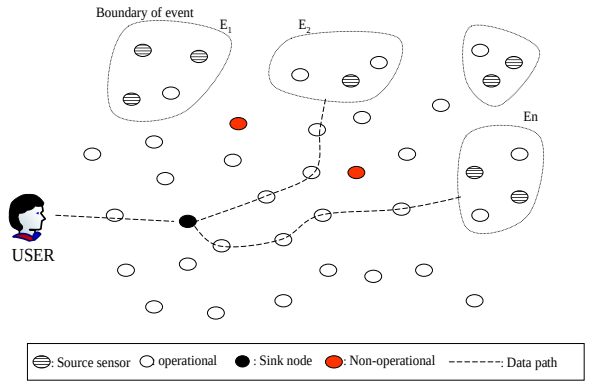

Figure 1. An example of a WSN

The sensors communicate unreliably via a multihop wireless connections. Usually, to reduce the energy consumption and improve network throughput, not all the sensors send data to the sink. A sensor that sends data to the sink is called a *source* sensor. In addition, as presented in Figure 1, WSNs may have different locations of sensors for the same desired event. Each location has its boundaries where there is at least one source sensor that sends *source packets* to the sink [16]. We denote the source sensors by $E_1, \cdots, E_n$. We define $U_{i,j}$ to be the jth source packet from a source sensor in E_i.

As presented in the pseudo code of Figure 2, the main functionality of the sink is to collect source packets and route the packets back to the *end user*. An end user could be a human, satellite, or stationary computer [2]. The sink can be invoked to send source packets to the user either periodically, e.g., every T seconds, or on demand when the user sends a query to the sink asking for an update.

As presented in Figure 2, U is a data structure that maintains the source packets in the sink. In addition, we assume that U keeps the packets history up to h packets for each source sensor s. For n different source sensors, U will be a $h \times n$ matrix for maintaining these packets.

```
U = ∅
do   //An infinite loop
settime(T)   //Install a timeout
    upon recv(U_{i,j}, s)
        add(U, U_{i,j})
    upon (recv(REQ, USER) OR timeout(T))
        Send (U, USER)
Until ("The sensing is done")
```

Figure 2. The functionality of the sink

In our WSN model, we assume three possible failures. The first failure is a fail-stop failure in sensors. Due to a lack of power in the battery, a sensor becomes non-operational. The second failure is due to losing data during communication. We assume that each sensor knows the percentage of the remaining power of its battery. To help us predict the fail-stop failure in sensors, we define a percentage threshold τ_1 such that if the percentage of the remaining power reaches τ_1, then the sensor cannot be an operational sensor that participates in any communication or sensing activity. In addition, we define another percentage threshold τ_2, $\tau_1 < \tau_2$, such that if the percentage of the remaining power reaches τ_2, the sensor cannot be a sink. We assume that the sink consumes more power than an operational sensor. The third type of failure that we consider is the sudden hardware failure. Hardware failure is unpredictable by the sensor.

III. The Fault Tolerance Framework

The FT process in WSNs consists of two main modules; namely the FT System Management and the FT System Operations. The process is depicted in Figure 3 and is described as follows.

The FT System Management takes as input the *Fault Specification*, which includes information provided by the system engineer (or their proxy) on the expected faults and their estimated frequencies. It also includes information about the valuable resources that need to be fault-tolerant as no FT system will tolerate *all* faults. For example, a FT system may be designed to tolerate data loss due to unreliable communication. Also, this

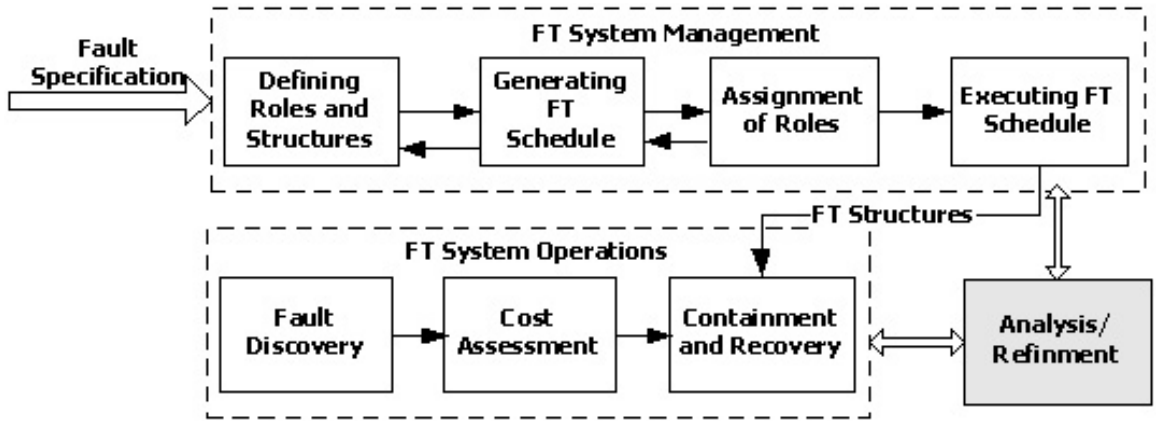

Figure 3. The Architecture of the fault tolerance Framework for WSNs

specification includes a description of the system's normal operation. A rule of the form "the network delivers 98% of the collected readings" can be used to define the system's normal operation in the absence of faulty components. Next, we describe in more detail, the functionalities that should be supported by a FT process.

The FT System Management module consists of the following:

- Defining Roles and Structures: A role defines a set of tasks and a task may be executed by one or more role. We refer to roles sharing one or more task as *cooperative roles*. *Data Replicator*, *Resource Monitorer*, *Fault Discoverer*, *Damage Assessor* are examples of roles. A FT structure is any special setting used for FT operations. For example, this could be a replicated set of data, or a special clustering structure used to provide hardware redundancy. The definition of such tasks, roles and structures differ from one scheme to another and depends on the fault specification provided by the system engineer. It should be noted that the tasks, roles and structures definition can be done once at system initialization, or they can adaptively change based on the network conditions or feedback from the FT operations as will be detailed shortly.

- Generating FT Schedule: The tasks within a role are scheduled based on network conditions and performance considerations. Dependence between schedules should be exploited to achieve high performance while preserving resources. For example, a schedule for data replication is set based on the frequency at which new data is acquired by the network and the user update schedule.

- Assignment of Roles: The assignment of roles defines the mapping between the roles and the network components. We classify FT schemes into *in-network* or *off-network*. An *in-network* scheme assigns the FT operations to one or more sensors within the network, the scheme is hence aware of the network main operations and adapts to the changes in the network conditions. Some of the network resources are consumed by the FT tasks. An *off-network* scheme assigns the FT operation to an entity outside the network, a powerful station for example. The scheme is hence oblivious to the network main operation. However, some information may be needed to be sent from the network to the outside entity to maintain the FT environment, which again consumes some of the network resources. A scheme may be *hybrid*; applying both in-network and off-network assignment of roles.

- Executing FT Schedule: refers to applying tasks needed to maintain a fault-tolerant environment. These tasks can be done once, for example, through hardware redundancy by initially deploying extra sensors. Or, it can be done periodically, for example, a periodical data replication.

The FT System Operations consist of the following:

- Fault Discovery: refers to the detection of faulty behaviors and the identification of faulty components. A faulty behavior is detected when the actual system behavior is different than the normal expected behavior.

- Cost Assessment: refers to the damage estimation and the evaluation of different recovery alternatives. The goal of this module is to deduce a *recovery plan* that optimizes the use of resources to recover from fault.

- Fault Containment and Recovery: Once a *recovery plan* is set, the plan is applied to contain the fault, eliminate its effect and restore the network to its normal operation. This module uses the maintained FT structures to achieve these goals. For example, a replicated set data may be used to recover from data loss due to a sensor failure.

- Analysis and Refinement: This module is critical to the operation of the WSN as it enables the network to self-configure based on changes in its conditions or based on a feedback from the FT operations. For example, based on the fault frequency, a rescheduling may be triggered. Roles Assignment may also be re-applied to cope with a change in the sensors' energy levels. FT schemes in WSNs can be classified into *static* or *dynamic* solutions based on whether the Analysis/Refinment module is implemented. The refinement module renders the solution self-configurable and adaptive to the changes in the network conditions or the changes in the FT operations. To the best of our knowledge, all solutions proposed so far are static, this opens up a new direction for enhancement.

IV. CRAFT: A FAULT-TOLERANT SCHEME FOR WSNS

In this section, we apply our framework to CRAFT, our FT scheme for data collection and dissemination in WSNs. CRAFT is based on Checkpoint/Recovery to realize high QoD in the presence of faults. A more detailed description of the scheme and its performance study can be found in [18]. The scheme assumes a homogeneous network where a sink is hosted on any sensor with enough energy. The main idea of CRAFT is to tolerate failure of the sink by applying periodical data checkpointing. The checkpointed data is used to recover lost data at sensor failure.

A. CRAFT Explained

Figure 4 presents the new behavior of the sink according to our FT scheme. As presented in the figure, the sink inserts the new source packet $U_{i,j}$ to U_m. In addition, it has a $h \times n$ matrix D_m that contains the most updated source packets since the latest checkpoint. The message CK represents the checkpointed data with the following fields: (1) the message type, which is CHKPT in this case, (2) the most updated source packets since that last checkpoint, and (3) the percentage of the remaining battery power.

As presented in Figure 4, upon a checkpoint, the sink sends D_m, but not U_m to the checkpoint sensor. This is because D_m contains the new source packets that have not been sent yet since the previous checkpoint. By sending D_m, actually, we implement here the *incremental* checkpointing approach rather than a full checkpointing as described in [12]. As a result, we reduce the power consumption in the sink and reduce the number of transmitted packets in the network as well.

By our FT scheme, the main new modification in the sink functionality is to send the CHKPT and TAU2 messages to the checkpoint sensor. The CHKPT message contains the checkpointed data. Notice here that the checkpointed data is saved in the memory of another sensor. This is because in many WSNs, secondary storage may not be available in the sensor field. Moreover, by saving the checkpointed data in more than one node, we make the checkpointed data more persistent and available. Regarding the TAU2 message, the sink sends this message to the checkpoint sensor if it is not able to continue running the sink task due to lack of battery power.

The sensor takes the checkpoint periodically every T_m seconds. The time between two consecutive checkpoints is called the *checkpoint interval*. In any checkpointing approach, the length of the checkpoint interval affects the tradeoff between the reliability and the performance of the system [1], [15]. Similarly, in our approach, the value of the checkpoint interval affects the system reliability and performance. For example, if we set small values for T_m, then we gain frequent saving of the most updated U that cause better reliability for the sink. However, due to the frequent checkpoints, the sensors involved in checkpointing consume more energy and overhead. In order to gain maximum benefits of the checkpoints to provide FT in the sink, the checkpoint interval T_m should be less than the interval time of the user update (T), but more than the interval time of sensing update, which is the rate of sending the source packets. As we mentioned before, we consider a fail-stop failure, due to a loss of battery power, in which a faulty sensor stops its operations. In our scheme we are mostly interested in monitoring the sensors along the checkpoint path. In general, the sensor S_i monitors its successor sensor S_{i+1} for all i, $1 \leq i < m$.

The main indication of failure detection that we use is by letting S_i to send its percentage of the remaining battery power to its predecessor sensor in the checkpoint path. In addition, the checkpoint sensor (S_{m-1}) uses the periodic checkpoints from the sink to monitor the sink. By the first indication, once the battery of the sink S_m reaches the threshold τ_2, S_m sends a *control* message to its predecessor indicating that S_m would not be able to run the sink. Potentially, as part of the recovery mechanism, the checkpoint sensor S_{m-1} will take over and run the sink instead. In order that S_{m-1} runs the sink, the percentage power of its battery should be more than τ_2.

Similarly, we use the threshold τ_1 to detect failures in all the sensors in the checkpoint path. Once the battery of the sensor S_i, $1 \leq i < m$, reaches the threshold τ_1, S_i sends a control message to its predecessor S_{i-1} indicating that S_i would not be able to be in the checkpoint path, the control message also includes reference to S_{i+1}. Consequently, the checkpoint path shrinks itself to exclude S_i. Therefore, S_{i-1} informs S_{i+1} that S_i is excluded from the path and S_{i-1} is its new predecessor. Notice here that we mostly concern about the checkpoint sensor S_{m-1} which will be the sink if S_m fails.

The second indication of failure detection is by letting the checkpoint sensor S_i trace the checkpoint interval in S_{i+1}. Since every T_m, S_{i+1} saves its checkpointed data in S_i, S_i can estimate the time of arriving the checkpointed data of S_{i+1}. Therefore, if the communication delay between S_{i+1} and S_i is d_{i+1}, then S_i can set a timeout that is great or equal of $d_{i+1} + T_m$. Upon an expiration of the timeout, S_i suspects that S_{i+1} is faulty. Upon a suspicion, S_i performs a recovery as mentioned above. Notice here that since the checkpoint message contains the percentage power of the battery, S_i can predict in a high probability if the missing checkpoint is due to a message loss in the wireless network or a failure that has occurred in S_{i+1}.

Figure 5 presents a pseudo code of the behavior of a checkpoint sensor S_i on the checkpoint path. S_i uses the timeout t for monitoring the incoming messages from the successor sensor S_{i+1}. An expiration of the timeout is due to an expected message from S_{i+1} that did not arrive in time. By our system model, a message may not arrive because of either S_{i+1} is faulty or the message is lost in the unreliable wireless network. In this case, S_i

```
U_m = ∅; D_m = ∅
settime(T)   //For user update
settime(T_m)   //T_m is the checkpoint interval
do   //An infinite loop
   α_m = myPerPower() // Percentage power of battery
   upon recv(U_{i,j}, s)
         add(U_m, U_{i,j}); add(D_m, U_{i,j})
   upon (recv(REQ, USER) OR timeout(T)) //Send update to user
         Send (U_m, USER); U_m = ∅
   upon timeout(T_m) // Do checkpoint
         CK = ⟨CHKPT, D_m, α_m⟩; Send (CK, S_{m-1})
         D_m = ∅
   if (α_m < τ_2)
      Send (⟨TAU2⟩, S_{m-1}) // inform about my energy
Until ("The sensing is done")
```

Figure 4. The new behavior of the sink

suspects S_{i+1}. Upon a suspicion, S_i invokes the function **suspect**(). This function uses the value of α_{i+1}, that is obtained from the latest checkpoint, to predict the cause of the timeout expiration.

```
U_i = ∅
settime(t = d_{i+1} + T_m) // For detecting lost messages
do   // An infinite loop
   α_i = myPerPower()
   upon recv(msg, S_{i+1})
         case (msg.type)
               CHKPT:   //This is a checkpointed data
                     add(U_i, msg.D)
                     α_{i+1} = msg.α
               TAU1:   // Sent by a checkpoint node
                     successor(S_{i+2})
                     settime(t = d_{i+2} + T_m)
               TAU2:   // Sent by the sink node
                     if (α_i > τ_2)
                        recoverSink(S_i)
                     else
                           S_{i+1} = newSink()
                           notify(S_{i+1})
   upon timeout(t)
         suspect(S_{i+1}, α_{i+1})
   if (α_i < τ_1) // inform about my energy
      Send (⟨TAU1, S_{i+1}⟩, S_{i-1})
Until ("The sensing is done")
```

Figure 5. The behavior of S_i

By Figure 5, S_i may receive three different types of messages from its successor S_{i+1}. The first type is `CHKPT`. This message indicates the receiving of the checkpointed data from S_{i+1}. Upon receiving a `CHKPT` message, S_i saves this data in its data structure U_i. The second type of messages is `TAU2`. This message is sent by the sink (in this case $i = m - 1$), when the percentage power of its battery is less that the threshold τ_2 (see Figure 4). Upon receiving `TAU2`, S_i becomes the sink if it has enough power in its battery. In this case, S_i invokes **recoverSink**(S_i) before running the sink task. In this function, S_i informs the source sensors and the user about the new sink location. Notice here that by recovering the sink in S_i, we minimize the period time in which the system does not have a sink. However, if S_i cannot run the sink, it selects a new sensor to act as a the new sink (see Figure 5). We are currently exploring alternatives for sink selection. The third type of messages is `TAU1`. The checkpoint node S_{i+1} sends the control message `TAU1` if the percentage power of its battery is less that the threshold τ_1. Upon receiving `TAU1`, S_i excludes the checkpoint node S_{i+1} from the checkpoint path and indicates that S_{i+2} is its successor in the path. Then, S_i updates the value of t to be $d_{i+2} + T_m$.

We can see here the main two benefits of our FT scheme due to Checkpoint/Recovery. The first benefit is reducing the loss in collected source packets upon a sink failure. Due to checkpoints, some source packets that have not sent yet to the user are saved in the checkpoint sensor. Therefore, these packets are recovered from the checkpoint at sink failure. The second benefit is to reduce the time in which the NSS runs without sink. Since the checkpoint sensor monitors the sink, it is ready to recover the sink immediately upon any failure detection. In Section IV-E we quantify the benefits of our FT scheme in the quality of data arriving at the user. We also study the energy consumption.

B. CRAFT within the FT Framework

- Defining Roles and Structures: CRAFT defines the following roles; *Data Replicator*: replicates the data possessed by the sink. The replicated data is used to recover from a sink failure. *Damage Assessor*: evaluates the damage by inspecting the replicated data and deciding whether some data packets have to be recovered from the source nodes. *Fault Discoverer*: discovers a sensor fault by monitoring the energy level or detecting a response timeout. The *checkpointed data* is a structure used by the scheme to recover from a fault. It consists of the set of data packets collected by the sink since the latest user update. It also contains the sink energy level which is used to predict a power failure.

- Generating FT Schedule: A static schedule is defined for the periodical update of the checkpointed data structure. The schedule is set based on the data generation rate and the user update frequency. A possible

enhancement on the current version of CRAFT is to adapt the schedule to the changes in the network conditions.

- Assignment of Roles: All roles defined before are assigned to a checkpoint sensor, which is a previous sink that has reached a defined energy threshold. Also, by monitoring its energy level, a sink predicts its own failure.

- Executing FT Schedule: The checkpointed data is periodically updated by including new data packets, and the latest energy level of the sink.

- Fault Discovery: A checkpoint sensor detects a failure when a response from the sink is timed out. Also, a sink may predict its own failure when its energy level reaches a certain threshold.

- Cost Assessment: The current version of CRAFT recovers from a fault regardless of the cost.

- Fault Containment and Recovery: Once a failure is detected by a checkpoint sensor, the replicated data is used for recovery, packets may be also recovered from the source nodes.

- Analysis and Refinement: This module is not currently implemented by CRAFT.

We compared our proposed CRAFT scheme with a NOFT scheme where there is no data checkpointing. In the NOFT case, when the sink fails, the user selects a new sensor to act as a sink. Data lost due to sink failure cannot be recovered in this case. In both schemes, we measured the QoD achieved and the corresponding energy consumption.

C. The SAN Model

Our evaluation of the CRAFT and NOFT schemes was carried through model-based simulation. We modeled both CRAFT and NOFT as a Stochastic Active Networks (SANs) using the Möbius modeling tool [4]. SANs are a convenient and high-level language for capturing the stochastic behavior of a system. Möbius is a tool dedicated to creating and simulating (or solving) SAN-based models. A SAN has the following components: *places* (denoted by circles), which contain tokens; *tokens*, which indicate the "value" or "state" of a place; *activities* (denoted by vertical ovals), which change the number of tokens in places; *input arcs*, which connect places to transitions; *output arcs*, which connect transitions to places; *input gates* (denoted by triangles pointing left), which are used to define complex enabling predicates and completion functions; *output gates* (denoted by triangles pointing to the right), which are used to define complex completion functions; and *instantaneous activities* (denoted by vertical lines), which are used to specify zero-timed events. An activity is enabled if for every connected input gate, the enabling predicate contained in it is true, and for each input arc, there is at least one token in the connected place. When an activity completes, one token is added to each place connected by an output arc, and functions contained in connected output gates and input gates are executed. The output gate and input gate functions are usually expressed using pseudo-C code. The times between enabling and firing of activities can be distributed according to a variety of probability distributions, and the parameters of the distribution can be a function of the state.

The sensor and the sink operations are modeled using the two subnets shown in Figure 6 and 7, respectively. The source sensor subnet is replicated as many times as the number of source sensors in the network. The experiments settings are listed in TableI. We will refer to the number of tokens at a place *p* as *mark(p)*.

The SAN model is constructed as follows:

- The $sensing\_act$ activity is fired every Ts simulating a new sensing event, *mark(newSensing)* is set to 1 indicating that a new packet is generated and *mark(sensingPacket)* is incremented to keep track of the current packet id. *Mark(src)* is updated to simulate the energy consumed by a source node for sensing.
- This marking fires $frwdSink\_act$, if *mark(enPLost)* is equal to 1, *mark(PLost)*is incremented indicating that a packet is lost, otherwise, the packet id is added to *msgQ* and *mark(inTransit)* is incremented to indicate the number of packets ready to be collected by the sink.
- If *mark(inTransit)* is greater than 1, and *mark(sinkRecovery)* is equal to 0, the $sinkRec\_act$ is fired, a packet is consumed from the *msgQ* and added to the sink, and *mark(inTransit)* is decremented.
- $user\_act$ is fired deterministically every T, and the packets collected at *sink* are moved to the *user* place. *mark(sink)* is updated accordingly to reflect the energy consumption.
- $chkpt\_act$ is fired every T_m, and the data held by the *sink* place is copied into the *chkpt* place to simulate a checkpoint event.
- When *mark(sink)* reaches τ_2, fd_act is fired, and *mark(sinkRecovery)* is set to 1 indicating that the sink is being recovered, the delay to recover a sink is simulated within the $recovery\_act$ activity.
- The fh_act simulates a sink hardware failure. When fired, *mark(sinkRecovery)* is set to 2 indicating that the sink failed. After a sink recovery delay, simulated within the $recovery\_act$ activity, *mark(sinkRecovery)* is reset to 0 indicating that the sink is alive again. in case of the FT scheme, the data recovery is simulated by copying the data from *chkpt* to the *sink*. In case of NOFT, *sink* is initialized with empty data. *Mark(sink)* and *mark(chkpt)* are updated to reflect the energy consumed in the recovery process. A similar recovery process is modeled for the *chkpt* according to the scheme described before.

D. Reward Assignment

The QoD is measured by comparing the number of packets received in the place *user*, to the number of packets generated. The total number of generated packets is equal to the number of times the $sensing\_act$ activity

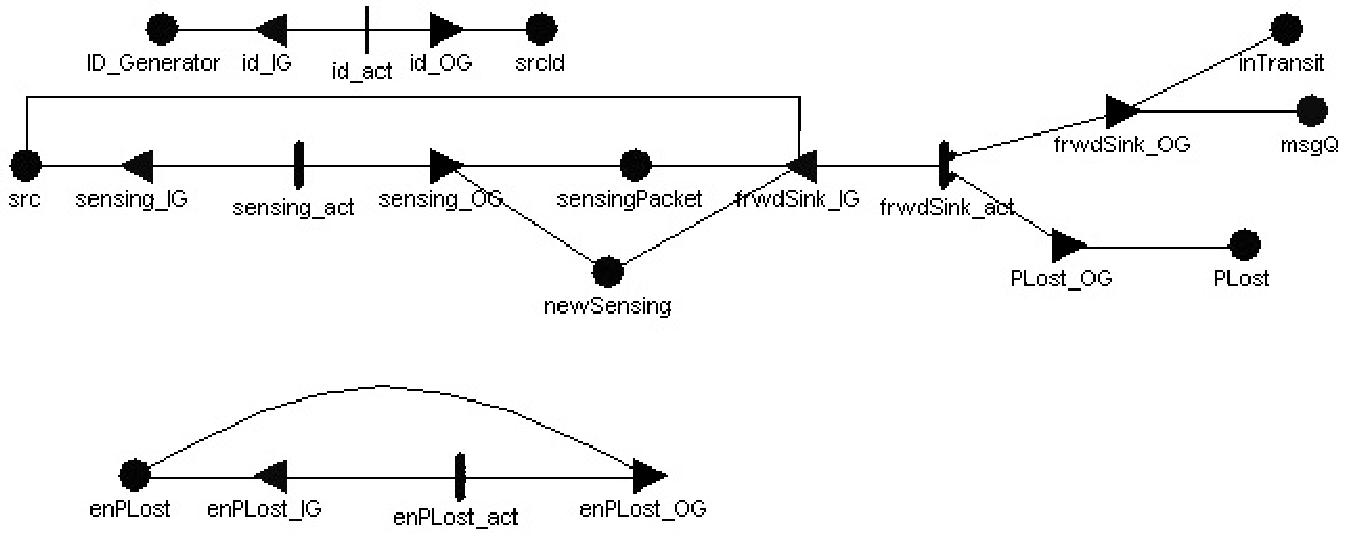

Figure 6. The SAN model simulating the operation of a source sensor

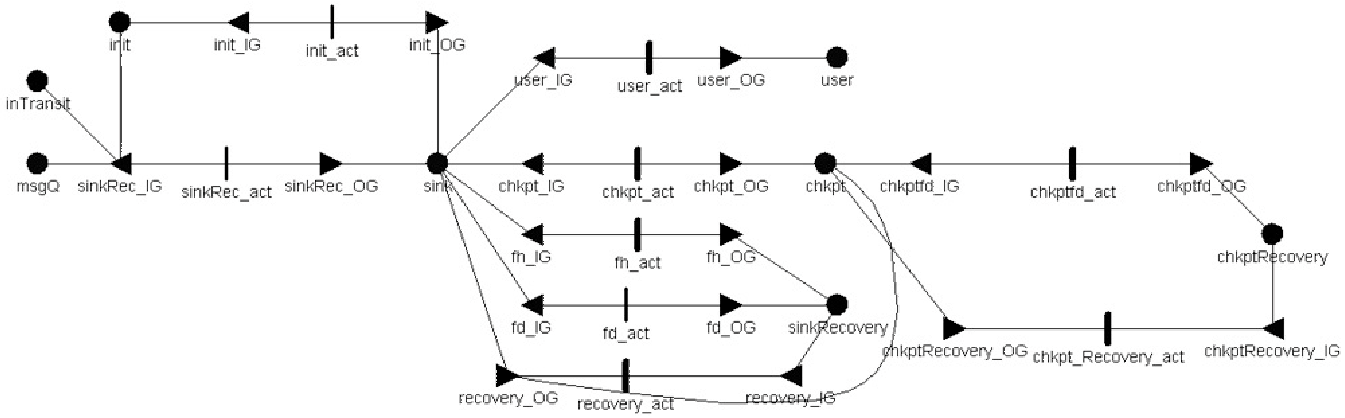

Figure 7. The SAN model simulating the operation of a sink and checkpoint sensors

is fired. The energy consumption is measured by keeping track of the energy marking of the places *src, sink and chkpt*.

E. Experiments and Results

We assume that a sensor consumes one energy unit when sensing a packet, sending a packet consumes three units and receiving a packet consumes one packet. The sink incurs 30 energy units consumption per packet sent to the user as the user is assumed to be far away from the sensors' field. Table I summarizes the environment settings of our experiments. First, we study the effect of the checkpoint path length on the performance of the FT scheme compared to the NOFT scheme for different failure rates.We consider the case where the checkpoint path consists of three nodes, we call this setting FT_CPP3. The path consists of the sensors denoted S_3, S_2 and S_1. As described before, S_3 acts as a sink and periodically checkpoints its state to S_2 which in turn checkpoints its state to S_1. When S_3 fails, S_2 takes over acting as the new sink, and again extends the checkpoint path by adding new sensor. Similarly, when S_2 fails, S_1 takes over. When S_1 fails, its failure is detected by S_2, which recover by selecting a new sensor. We compare this setting by the case where we have only two sensors on the checkpoint path, we denote it by FT_CPP2. In this case, the path consists of the sink S_3 and a checkpoint sensor S_2. When the sink fails, the failure is detected by S_2 which acts then as sink and selects a new sensor to act as checkpoint. When S_2 fails, its failure is detected by the sink which recover it by selecting a new checkpoint sensor and sending the latest checkpointed states to this new sensor. The results of simulating the two schemes

Number of Source Nodes	5
Node Initial Energy	10000 units
Packet drop rate	0.0001
τ_1	0.125
τ_2	0.25
Ts	30 sec
T_m	90 sec

TABLE I.
SIMULATION ENVIRONMENT SETTINGS

and the NOFT are shown in Figure 8 and 9.

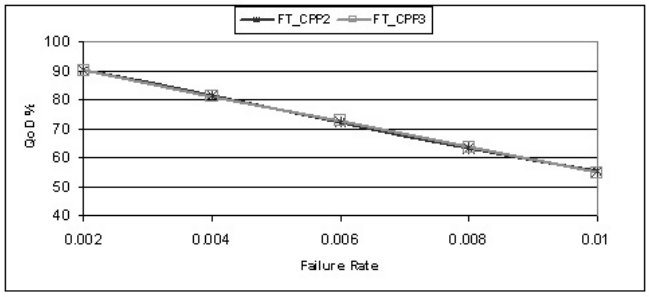

Figure 8. The effect of the checkpoint path length on the QoD achieved by the FT scheme

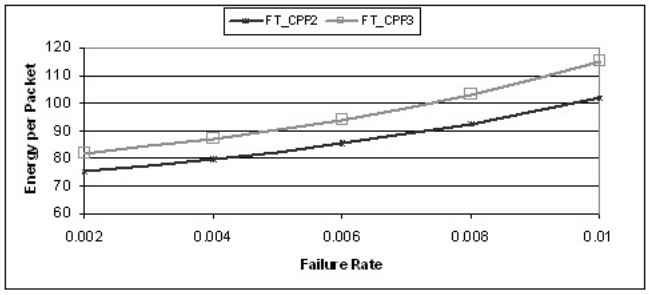

Figure 9. The effect of the checkpoint path length on the energy consumption of the FT scheme

As shown by the results, the QoD achieved by both FT_CPP2 and the FT_CPP3 are approximately the same as they both recover from checkpoint failure. However, FT_CPP3 increases the energy overhead by about 14.5% compared to FT_CPP2. This increase is due to the checkpointing activity from S_2 to S_1. Hence, the recovery mechanism applied by FT_CPP2 is more efficient than FT_CPP3. In the following experiments, we will implement the FT scheme using a sink path of length two.

Next, we will study the effect of the checkpointing on the QoD and the energy consumption per data packet, we have compared the FT (i.e. the application of CRAFT) and NOFT schemes under different values of the sensors' failure rate. Figure 10 and Figure 11 show the QoD and the corresponding energy consumption, for both FT and NOFT, at user update interval T of 180. We have plotted the difference on performance (FT-NOFT) at different values of T in Figure 12 and Figure 13. The results show that the FT scheme is less sensitive to the failure rate and it always achieves a better QoD than the NOFT. From Figure 12, the QoD achieved by the FT is higher than the NOFT and the difference becomes more significant as T increases and at higher failure rates. As T increases, more packets are lost by the NOFT scheme, and unlike the FT scheme, these packets cannot be recovered. For higher probability of sink failure, the number of lost packets is further increased. At high failure rates, the number of lost packets increases which decreases the QoD of both schemes but still the FT performs better. Due to the checkpointing, FT is expected to be more energy consuming than the NOFT scheme. However, the energy overhead of the FT decreases compared to the NOFT as the failure rate increases. This is due to the fact that, at high failure rate, the NOFT scheme is sending smaller percentage of the generated packets to the user, which increases its energy overhead per packet. Consequently, for very large values of failure rates, the FT scheme is actually consuming less energy per packet than the NOFT scheme while achieving higher QoD at the same time. For example, at failure rate of 0.008 and T of 180 sec, the FT scheme achieved about 14.4% enhancement in the QoD and saved about 2.16% of the energy consumed per packet.

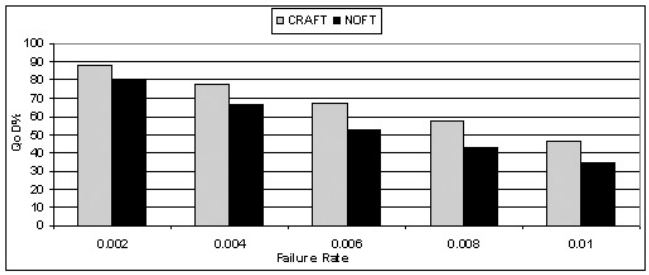

Figure 10. The QoD achieved by the FT and the NOFT schemes at T=180 under different failure rates

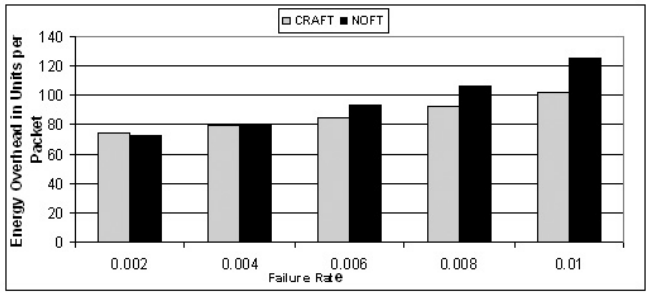

Figure 11. The energy consumption of by the FT and the NOFT schemes at T=180 under different failure rates

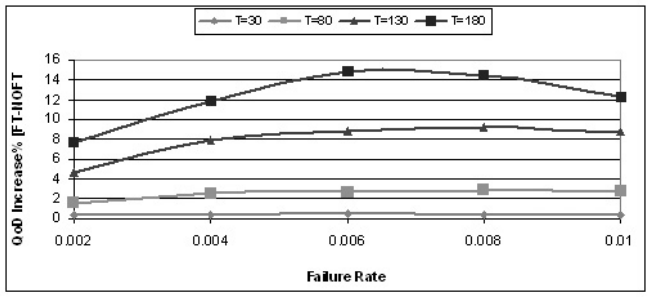

Figure 12. The increase in the QoD achieved by the FT scheme compared to the NOFT scheme for different T and failure rates

In our next set of experiments, we study the effect of the checkpointing interval T_m on the QoD and the energy consumption of both schemes; the FT and NOFT. We have set T to 130, and the failure rate to 0.004. As shown in

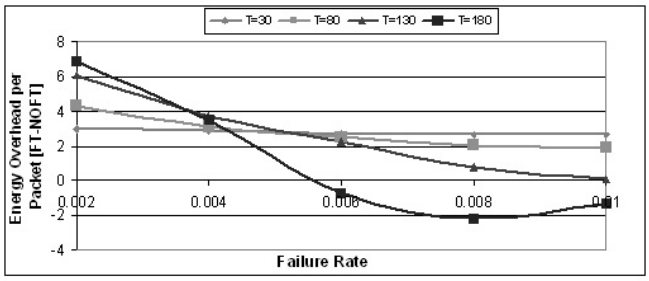

Figure 13. The energy overhead of the FT scheme compared to the NOFT scheme for different T and failure rates

Figure 14, as T_m increases, the percentage of data packets that are lost before being sent to the user increases, which decreases the QoD achieved by the FT scheme. Also, as T_m increases, the energy consumed by the FT decreases as the chekpointing is done less frequently (refer to Figure 15). However, for large values of T_m, the number of packets sent to the user decreases which increases the energy consumed per packet. Hence, for a certain settings of failure rate and user update interval, an optimal value of T_m can be found where FT achieves the maximum increase in the QoD with minimum energy consumption overhead.

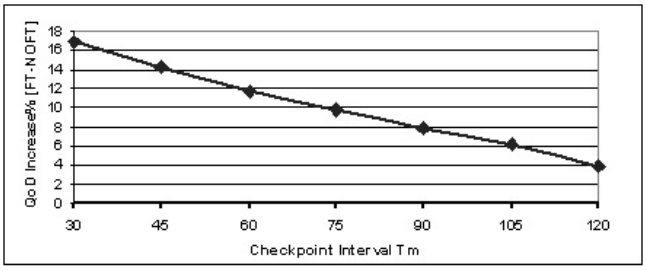

Figure 14. The effect of the checkpointing interval T_m on the QoD achieved by the FT scheme compared to the NOFT scheme [FT-NOFT]

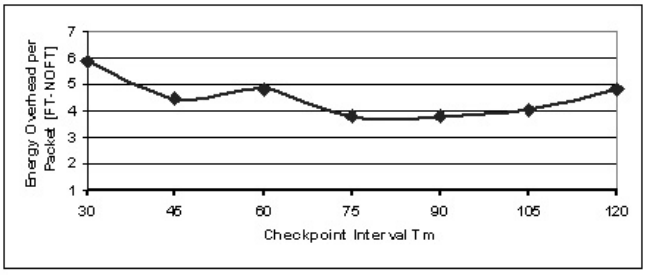

Figure 15. The effect of the checkpointing interval T_m on the energy consumed per packet by the FT scheme compared to the NOFT scheme [FT-NOFT]

V. Related Work

Fault tolerance in measurements by a group of sensors, was first studied by Marzullo [11] who proposed a model that tolerates individual sensor failures. In this model, a processor receives inputs from several sensors whose outputs are connected intervals and gives a fault tolerant algorithm that takes these intervals as inputs and gives the output as a connected interval representing the sensor values. The work in [13] and [9] extend Marzullo's model by reducing the output interval estimate and relaxing the assumption on the number of faulty nodes, respectively. Unlike Marzullo's work, we focus on providing fault tolerance in WSNs focusing on performance, but not the sensor values.

In [6] an algorithm is developed that guarantees reliable and fairly accurate output from a number of different types of sensors when at most k out of n sensors are faulty. The results of the scheme are applicable only to certain individual sensor faults and traditional networks. They are not generalizable to the reliability needs in complex network levels and most importantly; they do not address the reliability issues that are induced by the ad-hoc nature of the wireless sensor networks.

A fault tolerance technique is proposed in [7] where a single type of resource backs up different types of resources. For example, if communication bandwidth is reduced and all of the computation power is available, the system can compress data using more computationally intensive compression schemes.

The Geographic Hash Table (GHT) [14] for data dissemination uses data replication for tolerating faults in WSNs. GHT defines a *home node* and a *home perimeter* of an event. The *home node* stores all data readings related to that event. These readings are replicated on a set of neighboring nodes that constitute the *home perimeter*. Data on the home perimeter is periodically refreshed and when a home node fails, a node on the perimeter is selected as the new home. If we consider our source nodes as home nodes, then this solution and ours may be viewed as complementary. Alternatively, home nodes may be viewed as stationary sinks. In this latter case, we present a different approach that relies on a fewer number of mobile sinks.

RideSharing [19] is a fault-tolerant data aggregation scheme. In RideSharing, when a packet sent from a child node to its parent is lost, the packet is recovered by a backup parent. A backup parent is assumed to be a node in the child transmission range and hence it overhears the packets sent by the child. Figure 16 highlights the main differences between the different schemes based on our proposed framework.

VI. Conclusion

WSNs enable a microscopic view of our surroundings. Dependable and efficient data collection and dissemination in WSNs are important for numerous applications. Realizing a fault-tolerant operation is therefore critical to the success of a WSN. Many FT solution for WSNs have been proposed with diverse approaches. In this paper, we identified key FT requirements and presented a new framework for FT in WSNs. We demonstrated a partial application of our framework to CRAFT; a Checkpoint/Recovery scheme that we proposed in [18]. By employing this framework, we highlighted similarities

Scheme	Fault Specification	Roles	Schedule	Roles Assignment	Maintenance	Discovery	Assessment	Recovery
CRAFT	- Random node failure - Energy exhaustion - Lost packets	- Replicator - Damage Assessor - Fault Discoverer	- Static Checkpointing schedule set based on data update frequency and the sensors' buffer size.	All roles are assigned to the Checkpoint sensor	Refreshing data by periodically replicating the sink buffer contents	Based on energy level and communicati on timeout	- Based on last recovered data sequence ID, data can be further recovered from the sources	- Checkpoint node recovers a dead sink - Data recovered from checkpoint - Data recovered from sources
GHT	- Random node failure - Energy exhaustion	- Replicator - Fault Discoverer	Static schedule set for data refreshment based on how often new data is acquired	Roles are assigned to node on the home perimeter of an event.	Refreshing data on the routing path	Based on the routing protocol, a node detects the death of the sink	None	A refresh packet echoed at a node indicates that it is the new candidate to be sink
RideSharing	Lost packets	- Parent - Backup Parent	None	Parent and Backup Parents are selected based on a node transmission range	None	Using flags added to a packet that indicate whether a child value is received or not	A parent checks the flag bits in a packet and decide whether a missing value can be recovered	Backup parents aggregate the missing value into theirs
Marzullo	Random failure with bounded effect on data accuracy	"Averager"; an external module that recover the missing values	None	Role is assigned to an external entity	None	Based on previous knowledge of the system's expected correct values	None	By applying an averaging algorithm
Note: None of these schemes is implementing the Analysis/Refinement functionalities								

Figure 16. Different schemes compared based on FT Framework

and differences among existing schemes and identified directions for enhancements.

In a sequel to this paper, we will implement the learning and refinement module in CRAFT. This will render the solution adaptive to the changes in the networks conditions and self-configurable based on feedback from the FT operations.

REFERENCES

[1] A. Agbaria, A. Freund, and R. Friedman. Evaluating Distributed Checkpointing Protocols. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS'03)*, pages 266–273, Rhode Island, May 2003.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38(4):393–422, 2002.

[3] S. Chen and K. Nahrstedt. Distributed Quality-of-Service Routing in Ad-Hoc Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1488–1505, 1999.

[4] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The möbius tool. In *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, pages 241–250, Germany, September 2001.

[5] E. N. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. *ACM Computing Surveys*, 34(3):375–408, September 2002.

[6] D. N. Jayasimha. Fault tolerance in multi-sensor networks. In *IEEE Transactions on Reliability*, June 1996.

[7] F. Koushanfar, M. Otkonjak, and A. Sangiovanni-Vincentelli. Fault Tolerance in Wireless Ad-Hoc Sensor Networks. In *IEEE Sensors*, pages 1491–1496, June 2002.

[8] C. R. Lin. On Demand QoS Routing in Multihop Mobile Networks. In *Proceedings of IEEE INFOCOM01*, Alaska, April 2001.

[9] H-W Liu and C-D Mu. An Efficient Algorithm for Fault Tolerance in Multisensor Networks. In *International Conference on Machine Learning and Cybernetic*, volume 2, pages 1258–1262, August 2004.

[10] M. Eltoweissy M. Younis, K. Akayya and A. Wadaa. On Handling QoS Traffic in Wireless Sensor Networks. In *Proceedings of HAWAII International Conference on System Sciences (HICSS-37)*, Hawaii, January 2004.

[11] K. Marzullo. Tolerating Failures of Continuous Valued Sensors. In *ACM Transactions on Computer Systems*, volume 8, pages 284–304, November 1990.

[12] J. S. Plank. An Overview of Checkpointing in Uniprocessor and Distributed Systems, Focusing on Implementation and Performance. Technical Report UT-CS-97-372, Department of Computer Science, University of Tennessee, July 1997.

[13] L. Prasad, S. S. Iyengar, R. L. Kashyap, and R. N. Madan. Functional Characterization of Fault Tolerant Integration in Distributed Sensor Networks. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 21, pages 1082–1087, Sep/Oct 1991.

[14] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table. *Mob. Netw. Appl.*, 8(4):427–442, August 2003.

[15] N. Vaidya. On Checkpoint Latency. In *Proceedings of*

the Pacific Rim International Symposium on Fault-Tolerant Systems, Newport Beach, December 1995.

[16] M. Younis, A. Lalani, and M. Eltoweissy. Safe Base-Station Repositioning in Wireless Sensor Networks. In *Proceedings of the IEEE Workshop on Information Assurance (in conjunction with IPCCC2006)*, Arizona, April 2006.

[17] C. Zhu and M. S. Corson. QoS Routing for Mobile Ad Hoc Networks. In *Proceedings of IEEE INFOCOM02*, New York, June 2002.

[18] I. Saleh, A. Agbaria, and M. Eltoweissy. In-Network Fault Tolerance in Networked Sensor Systems. In *2nd ACM/SIGMOBILE Workshop on Dependability Issues in Wireless Ad Hoc Networks and Sensor Networks DIWANS06*, California, September 2006.

[19] S. Gobriel, S. Khattab, D. Moss, J. Brustoloni and R. Melhem. RideSharing: Fault Tolerant Aggregation in Sensor Networks Using Corrective Actions. In *Proceedings of Third Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks SECON06*, Virginia, September 2006.

Iman Saleh received her M.S. and B.S. in Computer Science and Automatic Control from Alexandria University, Egypt in 2002 and 2005, respectively. Since spring 2006, she has been pursuing her Ph.D. degree in the Bradley Department of Electrical and Computer Engineering at Virginia Tech, where she is a Graduate Research Assistant in the Information Assurance Lab. Her research interests include sensor networks, network security, and ubiquitous computing. Iman is a student member of IEEE, ACM, and SWE.

Mohamed Eltoweissy is an associate professor in The Bradley Department of Electrical and Computer Engineering at Virginia Tech. He also holds a courtesy appointment in the Department of Computer Science. His research interests are in the areas of information assurance and trust, networking and security in large-scale, ubiquitous, unstructured and resource-constrained environments, service-oriented architectures, and group communications. Eltoweissy received his Ph.D. in Computer Science from Old Dominion University in 1993 and his M.S. and B.S. in Computer Science and Automatic Control from Alexandria University, Egypt in 1989 and 1986, respectively. He has over 100 publications in archival journals and respected books and conference proceedings. Eltoweissy is a senior member of IEEE and a member of ACM, ACM SIGBED, and ACM SIGSAC. In 2003, Eltoweissy was nominated for the Virginia SCHEV outstanding faculty awards, the highest honor for faculty in Virginia.

Adnan Agbaria received his Master degree (on "Communication-Processor Tradeoffs in Limited Resources Parallel RAM") and his Ph.D (on "Reliability in High Performance Distributed Computing Systems") in computer science from The University of Haifa in 1997 and in computer science from the Technion in 2002, correspondingly. Dr. Adnan Agbaria is currently a Research Staff Member at IBM Haifa Research Laboratory and adjunct professor at the CS department at Haifa University. Before joining IBM Research, Dr. Agbaria spent a year as a computer scientist at the ISI of the University of Southern California and about three years as a Postdoctoral Fellow in the CSL of the University of Illinois at Urbana-Champaign. Dr. Agbaria's research interests include, but not limited to, fault and intrusion tolerance, wireless networking and computing, parallel and distributed systems, and model-based validation. Dr. Agbaria served as a program committee member on the International Conference on Dependable Systems and networks (DSN) and the International Conference on Distributed Computing Systems (ICDCS). He is the Co-organizer of the international Workshop on Applied Software Reliability (WASR).

Hesham El-Sayed is currently an Assistant Professor at the College of Information Technology, United Arab Emirates University. He received his Ph.D. in Systems and Computer Engineering from Carleton University, Canada, in 1999. His broad research interests span the areas of Mobile and Wireless Networks, IP Networking, Data Mining in Telecommunications, and Software Performance Engineering. Before joining the United Arab Emirates University in 2003, Dr. El-Sayed held several industrial positions at Nortel Networks, Wind River Systems, and Paragon Networks (was acquired by Carrier Access Corporation). During his tenure in industry, Dr. El-Sayed was honored with several awards, including Nortel Networks President's Award of Excellence in the Category Leadership. Nortel Networks Vice President's of Advanced Software and Networking Technology Award of Excellence, AND Paragon Networks CEO's Award of Excellence. Dr. El-Sayed is a member of IEEE and has been a regular reviewer of many international conferences and journals.