Dependability Considerations in Wireless Sensor Networks Applications

Amirhosein Taherkordi, Majid Alkaee Taleghan and Mohsen Sharifi
Computer Engineering Department, Iran University of Science and Technology, Tehran, Iran
Email: {taherkordi, alkaee}@comp.iust.ac.ir, msharifi@iust.ac.ir

***Abstract*—Recently, the use of wireless sensor networks has spread to applications areas that are not viable or cost-efficient to be run on other types of networks. Due to some critical tasks done in these types of networks, the majority of sensor networks applications should be dependable and should be run continuously and reliably without interruption. Hence, the two more significant dependability factors that should be nowadays taken into account in developing wireless sensor networks applications are '*availability*' and '*reliability*' of application services. The specific characteristics and constraints of wireless sensor networks require a different interpretation of these two factors when developing applications for such networks. In this paper, we propose a middleware layer mechanism for satisfying these two factors as more important dependability issues in sensor networks applications. We propose an event-based middleware service that is specifically designed for wireless sensor networks in which a group of sensor nodes forms a cluster and a replicated service is run on each cluster head. The communication model among cluster members and cluster head is based on the publish/subscribe scheme. We show how the replicated services and communication model in cluster nodes satisfy dependability issues and increase the availability and reliability of applications running under the proposed middleware.**

***Index Terms*—wireless sensor networks, dependability, middleware, publish/subscribe, event-based**

I. Introduction

Recent advances in wireless communications and miniaturization of hardware components have enabled the development of low-cost, low-power, multifunctional and intelligent sensor nodes. These devices are small in size and communicate in short distances over an RF (radio frequency) channel. These tiny nodes, which consist of sensing, data processing, and communicating components, realize the objectives of sensor networks.

A Wireless Sensor Network (WSN) is composed of a large number of integrated sensor nodes that are densely deployed either inside the phenomenon or very close to it, and collaborate through a wireless network in collecting environmental information or reacting to specific events [2, 5].

Wireless Sensor Networks are amongst the cutting edge technologies alongside Ubiquitous and Grid Computing. Like other such innovative technologies, they have their own unique constraints, capabilities, complexities, features and specific operational environments. These include limited and nonrenewable energy and resources, small size, low cost, operation in large numbers in physical environments with dynamic and fault prone conditions, and data-centric nature of communication. These specific properties differentiate WSNs from other distributed data networks, e.g. Ad Hoc networks, but by their very nature, possess several characteristics of distributed systems such as fault tolerance, real-time, security, safety, reliability, and availability [24].

WSNs applications are used to perform many critical tasks, including aerospace, automation, weather prediction, medical monitoring, natural event monitoring, object tracking, monitoring product quality, combat field reconnaissance, and military command and control [9, 17]. Properties that such applications must have include availability, reliability, security and etc. The notion of *dependability* captures these concerns within a single conceptual framework, making it possible to approach the different requirements of a critical system in a unified way. The unique characteristics of WSNs applications make dependability satisfaction in these applications more and more significant.

Similar to other computing areas, sensor network computing systems are characterized by four fundamental properties: functionality, performance, cost and dependability. Dependability of a system is the ability to deliver services that can justifiably be trusted. The notion of dependability is broken down into six fundamental properties: (1) reliability, (2) availability, (3) safety, (4) confidentiality, (5) integrity and (6) maintainability. Informally, it is expected that a dependable system will be operational when needed (availability), that the system will keep operating correctly while being used (reliability), that there will be no unauthorized disclosure (confidentiality) or modification of information that the system is using (integrity) and that operation of the system will not be dangerous (safety) [20].

What we discuss in this paper is about achieving two primary factors of dependability in WSNs applications, namely *availability* and *reliability*. In the classical definition, a system is highly available if the fraction of its downtime is very small, either because failures are rare, or because it can restart very quickly after a failure [19]. In WSNs context, availability means that in a long sensing duration, how much the network services are up and continue to sense, send and deliver data to base station. Reliability is defined as the probability that the system functions properly and continuously in the interval $[0,\tau]$, assuming that it was operational at time 0 [18]. Based on this definition, in WSNs applications, we define functions as a set of processes sending interested data to base station during application execution. Therefore, less loss of interested data leads us to higher reliability of system. To explain how we can achieve these quality factors in WSNs applications, let us first define the software levels of typical WSNs.

In general, sensor networks software can be layered into three levels [7]: *sensor* software, *node software* and *sensor network software.* The sensor software contains the process by which events in the real world are sampled and converted into machine-readable signals. The generated digital signals play the role of input data for the higher level. Therefore, sensor software has full access to the sensor hardware and need not to access the network. The output and functions of sensor software is used by sensor node software. This level includes system software for network maintenance and application specific software. At this level, a collection of common services for application development, called *middleware,* reside over the operating system. Application programs use this middleware according to their own specific requirements; these programs often access the individual node resources and local services, and do not need to access the network level capabilities. Finally, the sensor network software specifies the main tasks and required services of the entire network without assigning either any specific tasks or services to individual nodes. The levels of sensor network software are shown in Figure 1.

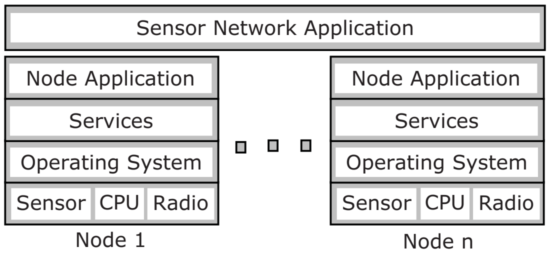

Figure 1. Software levels in Wireless Sensor Networks

In this paper, we define a new *Event-Driven Approach* for WSNs applications that provides its support services in the middleware, node application and network levels. Our main objective is to achieve reliability and availability in WSNs applications through applying event-driven approach. We use *publish/subscribe scheme* [6] as a core architectural model for providing the essential event-driven services as middleware and adapt this core to satisfy the requirements and constraints of WSNs [23]. WSNs are assumed heterogeneous, and the publish/subscribe scheme is applied to WSNs applications by assigning tasks according to individual node features. Furthermore, the cluster-based organization of sensor nodes is chosen for data dissipation.

The rest of paper is organized as follows. Section 2 discusses related works on WSNs application architecture and middleware. A detailed description of the proposed solution is presented in section 3 as an event-driven architectural model to achieve reliability and availability. Evaluation results are shown in section 4. Finally, section 5 presents conclusions and the issues that remain open for future works.

II. Related Works

Several research activities have been carried out in addressing common quality factors and exclusive factors of WSNs applications by providing some architectural models. But, most of research activities focus on exclusive factors such as power efficiency and traffic reduction. In most reported works, satisfaction of common factors is usually derived from satisfaction of exclusive factors.

In the context of distributed systems, solutions such as Java RMI (Remote Method Invocation), EJB (Enterprise Java Beans) and CORBA (Common Object Request Broker Architecture) are reported in [13, 10]. In [14], attempts are made to leverage the idea of distributed system architecture to embedded systems, which is called GAIA. It features coordination of software units and heterogeneous networks. Although CORBA and EJB provide a confident infrastructure for satisfying quality factors of applications, but using CORBA, XML, SQL and JAVA is not an efficient choice for sensor networks because they are normally heavy weighted in terms of memory and computation.

Another work [7] has applied Service-Oriented Architecture (SOA) to sensor networks applications. The main objective of this work is the connection of consumers and service providers in a loosely-coupled way in order to improve flexibility and extensibility. It uses a simple and clear interface to bind all participating software components and provides service reuse.

Another solution with an artificial intelligence flavor is the Agent-Oriented Architecture (AOA) reported in [12]. It proposes an infrastructure that applies active agent technology to sensor networks. The idea of this solution is that, on the one hand, sensor networks must be dynamically configurable and adaptive in order to response actively to events, and on the other hand, security must be built into sensor networks at the very initial design stages to prevent any potential threats. The infrastructure which is provided by this model may be appropriate for establishing security and increasing the adaptation of sensor networks.

In event-based area, most of the initial related researches have concentrated on leveraging event-driven mechanism in a fixed network. But the emergence of mobile systems in recent years, with properties such as client mobility, wireless communications and resource limitations, has opened up new research topics on how to efficiently adapt the publish/subscribe model for mobile environments. Combination of unique characteristics in publish/subscribe model makes it advantageous in a mobile or wireless environment. A few numbers of researches have been carried out in this area that focus on different aspects of mobile computing. Due to the commonalities between mobile environments and sensor networks, the results of these works provide useful hints in finding appropriate models and methods for WSNs applications.

A more recent work based on event-driven model has been carried out specifically in the WSNs area by Mires [16]. In Mires, the publish/subscribe scheme in the middleware layer of WSNs is provided to facilitate the development of event-based applications. Perhaps, the only contribution of Mires is in introducing an aggregation service into every sensor node to aggregate data gathered from individual nodes, and to send the result to another node. The use of this technique has reduced both the number of message transmissions and power consumption. Simply said, Mires has merely introduced a primitive middleware for even-based communication across sensor networks. Mires do not discuss about quality factors of proposed middleware.

In this paper, we present an integrated collection of services as a middleware in WSNs that satisfies both reliability and availability. In this way, we define our middleware services according to our reasonable assumption about heterogeneity of sensor networks. To provide support for availability and reliability of our middleware, replicated services are run in more powerful nodes, as is reported in [8].

III. The Proposed Approach

To introduce the services of our proposed approach and their relations, let us first present an overall view of the publish/subscribe paradigm. Our assumptions, challenges in finding suitable components of the infrastructure and the required components themselves together with their interactions, are detailed immediately afterwards [22].

The event-driven mechanism is based on the publish/subscribe paradigm. According to this paradigm, a user expresses his/her interest in receiving certain types of events by submitting a predicate, called the user's subscription [6]. When a new event is generated and published to the system, the publish/subscribe infrastructure is responsible for checking the event against all current subscriptions and delivering it efficiently and reliably to all users whose subscriptions match the event. If the consumer is unavailable, the infrastructure can store the event and try to forward it later. Specifically, the key elements in the publish/subscribe paradigm are the Notification Service, Subscription Matching Service and the Subscriptions Data Store.

Subscription Matching Service is responsible to check the published event against the subscriptions issued by subscriber to forward the event towards interested ones. In the earlier versions of publish/subscribe systems, this service was subject-based (group-based) so that each event is dispatched to interested objects based on predefined subjects or groups.

The inflexibility of subject-based approach had forced researchers to introduce another approach called content-based [1, 8]. In content-based method, subscriptions are defined based on events' contents. For each event, the content of event is checked against the content of subscriptions reported from interested objects and registered in Data Store (where the subscriptions are preserved and messages are queued before they are passed to subscribers). If the result of matching becomes true, then the event is forwarded to interested nodes via Notification Service.

A. Assumptions

To define our architecture, let us make some assumptions about the overall sensor network structure.

Firstly, each node may have different capabilities and execute different functions. In other words, the network consists of heterogeneous nodes. Therefore, some nodes may have larger battery capacities and more powerful processing capabilities, and other nodes may only execute the sensing functions and lightweight processing operations.

Secondly, cluster-based mechanism [9,11] is adopted for node communication and routing. In a cluster-based system, sensor nodes form clusters; a cluster head for each cluster is selected according to some negotiated rules. Sensor nodes only transmit their data to their immediate local cluster head and the cluster head conveys the data towards the sink node. Consequently, more powerful nodes in the topology play the role of cluster heads and other nodes are responsible for sensing data and forwarding them to cluster nodes.

We assume that events have a record-based structure. In record-based approach, notifications are defined as sets of typed fields characterized by a name and a value. For instance,

```
Struct NewSensedData{
    String environmentParam = "Temperature"
    Int paramValue = 23
    Time sensingTime = "14:23"
}
```

is a record-based notification composed of three typed fields. So, subscriptions are defined based on the record-based structure of events. For instance,

```
{environmentParam == "Temperature" ∧
(paramValue > 22 ∧ paramValue <34)}
```

is a sample subscription in which every temperature event that has a value between 22 and 34 can subscribe to it.

Contrasting our chosen structure to EDA, we can infer that sensor nodes play the role of event sources (ES) in event-driven mechanism, and the task of subscribing and notification is assigned to cluster head; thus we name this node Event Broker (EB). As shown in Figure 2, we assume that the network may have more than one sink node. Thus, our environment consists of several ESs, some EBs, and one or more sink nodes.

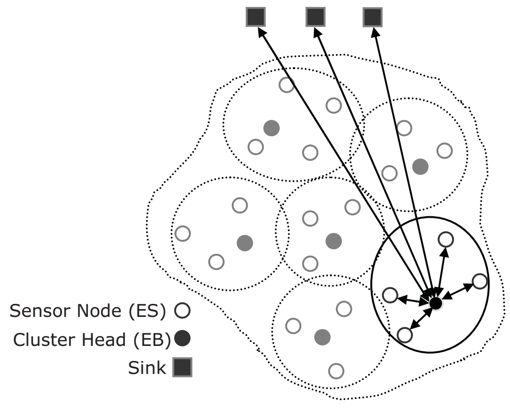

Figure 2. Sensor nodes classification and communications in a cluster-based organization

An ES generates events in response to changes in an environment variable that it monitors, such as the location of an object. Events are published to the EB that matches them against a set of subscriptions and dissipates through wireless connection to interested sink nodes. For example, a sink node subscribes to all events reporting the location of any object within a certain range.

It should be noted that EB, the server of publish/subscribe, could be implemented as a single server (one EB), multiple distributed ones working together or replicated servers. In this paper, we have assumed that multiple EBs are replicated to increase availability and reliability. In a replicated mode, a subscription is monitored by multiple EBs, independently.

Since EBs are the server of ESs, it is crucial that EBs be available and reliable in the duration of sensing. In the rest of paper, we propose the middleware components in ES and EB and concentrate in achieving reliability and availability of services provided in ES and EB.

B. Services in Event Source

As mentioned before, ES senses data as an event from environment and publishes the generated data to cluster head, namely EB. At this point, we don't say anything about the type of ES and assume that ES is connected to EB continuously. New issues arise when we consider ES as a mobile sensor node. Specially, we face two critical challenges: energy constraint and mobility. Sensor nodes can be frequently disconnected from the cluster head because they may be off when the battery runs out, or they might not be accessible because of transient wireless communication problems or moving into an area outside radio reception. In fact, all of these factors can influence the dependability of WSNs application. So, we should address above challenges toward satisfying application dependability issues [22].

At first, we should take into consideration the power constraint problem in sensor nodes. The bottleneck in energy consumption of sensor nodes is the process of transmitting data via radio. In other words, the radio transmission consumes more energy than processing tasks.

As an example, Table 1 shows the current draw needed for MICA node [25] in different states of radio and CPU. It can be illustrated that the required draw for radio transmission node is considerably more than the value needed in active mode of CPU. So, we can save more battery energy when the number of radio transmissions is decreased.

To decrease the number of radio transmissions, two alternatives exist: (1) discarding some data in ES and refusing to send them to EB and (2) aggregating a set of sensed data based on fusion algorithms and send the result to EB. Each one has some drawbacks and benefits which are explained next.

TABLE I. CURRENT DRAW IN DIFFERENT MODES OF CPU AND RADIO IN MICA NODE

	Mode	Current
CPU	Active	2.9 mA
	Idle	2.9 mA
	Sleep	1.9 mA
	Off	1 µA
Radio	Transmit	12 mA
	Receiver	1.8 mA
	Sleep	5 µA

Events should be discarded on the basis of some knowledge about the content of events. In other words, if the content of a sensed event is not of interest to any sink node, we can discard it. To use this approach, in the network setup phase, we disseminate the sink node interest to all other nodes. So, in addition to the process of checking subscriptions in EB, it is possible that similar processes may be running in ES. But, due to limited energy in ES, we cannot do exactly the same as in EB. Therefore, the energy efficient component should be run in ES which requires significantly lower energy than the one running in EB. This mechanism is nicknamed Quenching in the literature [8, 26].

In Quenching, a "combined active subscription expression" $match_{all}$ is given to ES [26]. The $match_{all}$ only says whether any sink node is interested in the sensed data; it does not exactly identify the interested sink nodes. Since the quenching service is very lightweight, there is no need to duplicate all the work that is being done in EB. When a new event e is generated, ES checks it against $match_{all}$. If $match_{all}(e)$ = false, it means that no subscription will match e in EB. Hence the event is discarded (quenched) at the source. If e matches the call, then at least one subscription will match, and the event is forwarded to EB as usual. Thus, the mechanism refuses to send an uninterested event to cluster head, and the number of radio transmissions is decreased. Quenching has proved to be particularly effective in reducing network traffic and the load of the EB, if a significant portion of the events generated do not match any subscriptions.

Now let's see what can be done for mobility of ES and wireless communication problems, which may cause the frequent connection and disconnection of ES from network. One option is to make the ES queue all events that are generated when it is disconnected. Running quenching on queue during disconnection time is not reliable because it is quite possible that during this time a new interest is dissipated from a sink node, and the sink node cannot contact the ES about newly added subscriptions. The queuing of all events in disconnection time may not be feasible too, because ES is often a low capability storage device. Moreover, the rate of event accumulation can be very high in ES. Consequently, ES will have to discard some events once its buffer is filled up. We may explain the approach as follows [8]:

> *"At the beginning of a disconnection, ES saves all events in the buffer. If the queue buffer overflows during disconnection, the first incoming event after disconnection is checked for quenching criteria that it has. If it doesn't match the criteria then it is de-queued and a new event is en-queued. Otherwise, the removing process runs on the second one until the quenching candidate is found. Upon reconnection, ES searches for new interests and updates its quenching service data. It then filters the queued events by quenching and if matched send to EB".*

The following sample pseudo code shows the recursive function for finding and removing a candidate event when queue is filled up.

```
Overflow Function in Quenching:

 function handleOverflow(){

   SensorEvent nextEvent =
 Queue.nextElement();
   if (nextEvent == null){
       Queue.dequeue(Queue.headElement());
       return;
   }

  boolean isMatch  =

 Quenching.testLightMatch(nextEvent);
   if (isMatch){
       handleOverflow();
   }else{
       Quenching.dequeue(nextEvent);
       return;
   }
 }
```

Another way to decrease the number of queued events is to apply the aggregation function on a certain number of queued data. Consequently, instead of saving all events, we can only store the aggregated value of them [16]. To realize this solution, several research activities have been reported in literature with titles such as *data fusion* and *aggregation*. But this subject is out of the scope of this paper. So, in our implementation, the output of aggregation service is just the simple average result of some input data.

In applications that all interested events need to be sent to sink node in turn, the aggregation service does not suffice. We have thus added a new component (called *StateChecker*) that checks the residual energy in a sensor node in specific time intervals. If the energy value is lower than a threshold, then the component prevents the publisher to send all of matched data and instead, it aggregates a specific number of events through aggregation service and sends the output of aggregation service to EB via publisher. The idea of including this component in our middleware has come from the work reported by Lonescu and Marsin in [27].

Figure 3 shows the middleware components and their interactions in ES. As mentioned in previous paragraphs, in time of connecting ES to network, each new event is checked in Quenching service against existing interests in a lightweight manner and if the $match_{all}$ yields true, then it will be inserted into built-in queue of Publisher service for sending data to EB (cluster head). If the aggregation result is sufficient in the context of application, quenching calls the Aggregation service to get the aggregation result of certain number of event data and quenching gives the aggregation result to publisher. During disconnection, all of sensed data is queued in Queue service and our approach is applied for handling the situation. Also StateCkecker component is added to publisher to check the residual energy of node in specific intervals. If the energy value is lower than a predefined threshold, then the publisher is forced to send the aggregation result of some data instead of sending all events.

So far, our effort was on how to cope with power supply constraints and mobility of sensor nodes. Based on solution proposed the availability of ES has been increased. But, the more important side of our solution is how to satisfy dependability issues in EB because EB is the heart of a cluster in WSN and if factors such as availability and reliability are not guaranteed the overall network may not be dependable.

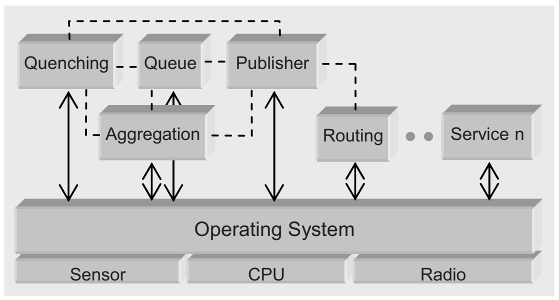

Figure 3. Components of Event Source (ES) and their interactions

C. Services in Event Broker

We assume that after applying the appropriate mechanism in ES, the primary event or aggregated result has reached EB (publish/subscribe server) through a specific cluster-based routing algorithm. As mentioned before, EB has more powerful resources than ES. In this node, the three important parts are: Notification Service, Subscription Checker and Subscriptions Storage. These components and their relationships are shown in Figure 4.

As soon as an event reaches EB, the Subscription Checker checks the content of the event against the interests stored in Subscriptions Storage. Because of the higher flexibility of content-based method for checking subscriptions, we have preferred to implement the subscription checker service based on this approach.

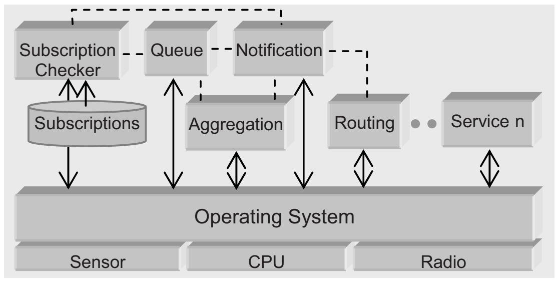

Figure 4. Components of Event Broker (EB) and their interactions

The output of subscription checker is a list of sink nodes that are interested in receiving the checked event. After finding the interested sinks, the content of event (along with the event handler [15]) is forwarded to sink nodes by Notification Service in turn.

The mobility of EB raises some problems [4]. In EB, when a node is disconnected from the sink, the subscription checker queues all events for the interested sink(s) in Queue Service. It should be noted that in EB, the capacity of queue could be lower than the one in ES. This is because only the interested events are saved in EB. So, it is less likely that the queue in EB is filled up. Also, in reconnecting the EB to sink, we face a large number of events that should be forwarded to sink node(s) which lead to high traffic in network. So, it is a trade off between the sending of all saved events and the aggregated result which is returned by Aggregation Service.

D. Replicated Services in EBs

As mentioned before, three forms of EB is envisaged: centralized, distributed and replicated. We select the third one for increasing the availability and reliability of our proposed middleware. The one level cluster-based organization of sensor nodes makes replication of services a well-suited mechanism for increasing the aforementioned quality factors.

It should be noted that what we named as replication in the context of this paper, is different from conventional ones. In conventional replicated systems, an individual request from a client is simultaneously dispatched to all replicated servers in the system. But, in our proposed model, the cluster members deliver a request only to the associated cluster head as one of replicated servers.

If we use a single centralized EB, the overload on the middleware may cause node failure and consequently all interested events may be lost. But, in a replicated form, if one of EBs fails, then a limited number of interested events may be lost. This may be because after reconfiguration phase of sensor nodes, the remaining EBs are responsible for monitoring ESs that lost their EB. So, the replicated services in EBs make it possible for every ES to select any EB at any time and send an event to the associated EB without worrying about subscription storage and checking, or whether the event has arrived in the sink or not. This is because, all EBs are the same and have the same subscription content and checking strategies.

In general, three desirable properties for a replicated publish/subscribe are envisaged: orderedness, consistency and completeness [8]. Orderedness indicates that events from the same ES are delivered to the user in the order they are generated at the ES. Consistency indicates that the set of events delivered to a sink over time must be a set that can possibly be generated by a non-replicated system. For example, a replicated system that delivers duplicates to the sink is trivially not consistent. Finally, completeness guarantees that the publish/subscribe middleware delivers all notifications of a client eventually.

As mentioned, according to the proposed replication mechanism, it is not necessary to worry about orderedness and consistency properties of our middleware. However, the completeness may be slightly unsatisfied; especially during reconfiguration time that happens when an ES cannot route an event to a specific EB until reconfiguration finishes successfully.

Application of power efficient matching algorithm in EB and replicating EBs based on the cluster-based organization of sensor nodes is advantageous in that if an EB fails then after reconfiguration phase of sensor nodes, the remaining EBs can take care of monitoring ESs that had lost their EB. So, we will not worry about unavailability of EBs. Also, the power efficient services in EBs enhance the target architecture with reliable components which receive large amount of sensed data from ESs, process them and then forward them to sink nodes.

IV. Evaluation

Evaluation of the proposed approach is done according to our availability and reliability objectives. We used the JIST simulator [21] to simulate a 20m length by 20m width field with 20 similar sensor nodes randomly deployed. As shown in Figure 5, two sink nodes interact with 6 cluster heads which were more powerful than the above 20 nodes. Simulation was run for 200 seconds and in each second, every sensor node sensed the environment 4 times and generated events if needed.

For measuring the availability of EB, we run the mobilizer module in EBs. Due to the mobile nature of EB, EB was in some occasions disconnected from the base station. Thus, based on the proposed approach, we measured the amount of data which was lost during the failure of a specific EB. In our approach, EB failure starts the reconfiguration phase of sensor nodes, and the remaining EBs are responsible for getting data from nodes that had lost their cluster head (orphan nodes) and routing them to base station. During simulation approximately 16000 events were generated on a normal distribution basis in which 3500 base station interested events should had ideally been received by base stations, but only nearly 3450 events were received. So we defined the availability of system (our simulated field) as follows:

$$availability(0,\tau) = availability_{cM,cH} \times availability_{cH,Sink} \Rightarrow$$
$$availability_{cM,cH} = \lambda$$
$$availability(0,\tau) = \lambda \times \frac{DeliveredEvents_{cH,Sink}}{InterestedEvents}$$
$$\tau = 200\sec ond$$
$$InterestedEvents \approx 3500$$
$$DeliveredEvents \approx 3450$$
$$availability(0,200) = \lambda \times \frac{3450}{3500} = \lambda \times 98.57\%$$

$availability_{cM,cH}$ is the measurement of availability of a cluster within cluster members and cluster head, and $availability_{cH,Sink}$ is the value of availability of connection within cluster head and the base station.

The simulation result showed that 1.5% of events, i.e. events generated in reconfiguration times, were lost. So, system availability was nearly 98.5%.

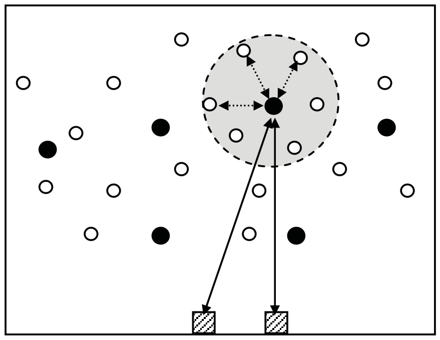

Figure 5. Simulation field and nodes' scattering

We modeled the reliability of system by comparing the generated data in ESs and the received data in base station. If any data in ES is equally received by the base station we could claim that our system is highly reliable. As mentioned before, by the provision of replicated services in EBs, if one of EBs fails, another EB will handle in reconfiguration phase the orphan nodes that had generated events. Since the subscription data in all EBs are the same, so it can be concluded that the interested data had reached the base station.

Another important parameter studied in the simulation was the number of interested events delivered to sink node compared to the events detected in sensor nodes. Two mobility issues, namely, EB failure and ES movement, were taken into account simultaneously. In Our Approach, EB failure starts the reconfiguration phase of sensor nodes, and ES movement outside radio reception results in the queuing of generated events in the queue service within ES. The queue capacity inside ES was 100 entries. Similar to the previous simulation, this simulation was also run for 200 seconds. Our desired mobility cases (EB failure and ES movement) were activated at times between 50 up to 100.

As shown in Figure 6, the number of delivered interested events is slightly less than the actual number of events that must be delivered. The reason for this slight difference is that a few interested events are lost during reconfiguration. The loss rate in Mires Approach is shown to be higher. This is because mobility in this approach is not supported.

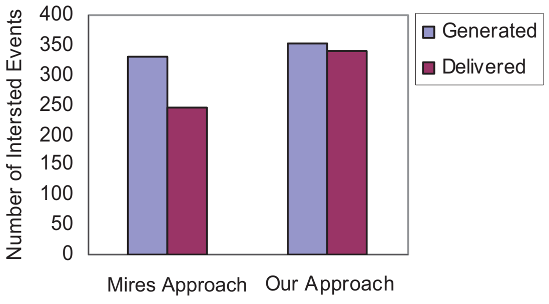

Figure 6. Event delivery in proposed middleware

V. Conclusion

WSNs' applications have placed new challenges to application developers due to the low availability of resources and mobile nature of nodes. We demonstrated that the publish/subscribe scheme can be successfully deployed to satisfy the quality factors of WSNs. A target middleware was designed which provided a power efficient, asynchronous and fault-tolerant mechanism for the development of applications to be run over WSNs.

We replicated middleware services on more powerful nodes called cluster heads. Replicated services in EBs made the services reliable and available in sensing duration. Although, we had addressed some of the main issues arising in the development of WSNs applications by applying a publish/subscribe middleware in WSNs computing domain, certain issues remain open for future work. Enhancing the middleware with some standard services such as caching and resource management and some quality factors such as security and safety is one of our future objectives. Achieving availability and reliability on nodes in a cluster is another research topic that will be taken into consideration.

References

[1] M.K. Aguilera, R.E. Strom, D.C. Sturman, M. Astley, T.D. Chandra, "Matching Events in a Content-Based Subscription System", In Proceedings of the 18th Annual ACM Symposium on Principles of Distributed Computing, pp. 53–61, May 1999.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "A Survey on Sensor Networks", IEEE Communications Magazine, pp. 102–114, August 2002.

[3] J. Bloomberg, "Events vs. Services: The Real Story", ZapThink, LLC, October 2004.

[4] G. Cugola, E. D. Nitto, “Using a Publish-Subscribe Middleware to Support Mobile Computing”, In Proceedings of the Works-hop on Middleware for Mobile Computing, November 2001.

[5] D. Culler, D. Estrin, M. Srivastava, “Overview of Sensor Networks”, IEEE Computer magazine, pp. 41–49, August 2004.

[6] P. T. Eugster, P. A. Felber, R. Guerraoui, A. M. Kermarrec, “The Many Faces of Publish/Subscribe”, ACM Computing Surveys 35(2), pp. 114–131, May 2003.

[7] F. Golatowski, J. Blumenthal, M. Handy, M. Haase, H. Burchardt, D. Timmermann, “Service-Oriented Software Architecture for Sensor Networks”, International Workshop on Mobile Computing, IMC’03, Jun 2003.

[8] Y. Huang, H. Garacia-Molina, “Publish/Subscribe in a Mobile Environment”, In Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'01), pp. 27–34, May 2001.

[9] M.Ilyas, I. Mahgoub, “Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems”, CRC Press, 2004.

[10] Java 2 Platform, Enterprise Edition (J2EE). http://java.sun.com/j2ee/

[11] Q. li, J. Aslam, D. Rus, “Hierarchical Power-Aware Routing in Sensor Networks”, In Proceedings of the DIMACS Work-shop Pervasive Networking, May 2001.

[12] Z. Liu, Y. Wang, “A Secure Agent Architecture for Sensor Networks”, In Proceedings of the International Conference on Artificial Intelligence, IC-AI '03, Las Vegas, Nevada, June 2003.

[13] Object Management Group. CORBAservices: Common Object Service Specification. Technical Report, Object Management Group, July 1998.

[14] M. Roman, R. H. Campbell, “Gaia: Enabling Active Spaces”, Department of Computer Science, University of Illinois at Urbana-Champaig, 2002.

[15] D. Schmidt, M. Stal, H. Rohnert, F. Buschmann, “Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects”, John Wiley & Sons, 2000.

[16] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, “A Message-Oriented Middleware for Sensor Networks”, In Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing, pp. 127–134, New York, USA, 2004.

[17] N. Xu, “A Survey of Sensor Network Applications”, Computer Science Department, University of Southern California, 2004.

[18] A. Heddaya, A. Helal, “Reliability, Availability, Dependability and Performability: A User-Centered View”, Boston University, Computer Science Department, Tech. Rep. BU-CS-97-011, 1996.

[19] J. C. Knight, “An Introduction To Computing System Dependability”, In Proceedings of the 26th International Conference on Software Engineering (ICSE’04), Scotland, UK, May 2004.

[20] A. Avizienis, J. Laprie, B. Randell, “Fundamental Concepts of Computer System Dependability”, IARP/IEEE-RAS Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments – Seoul, Korea, May, 2001.

[21] R. Barr, Z. J. Haas, JiST/SWANS Web site, http://www.cs.cornell.edu/barr/repository/jist/, 2004.

[22] A. Taherkordi, M. Alkaee Taleghan, M. Sharifi, “Achieving Availability and Reliability in Wireless Sensor Networks Applications”, In Proceedings of the First International Conference on Availability, Reliability and Security(ARES'06), in conjunction with the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA'06), Vienna, Austria, April 20-22, 2006.

[23] M. Alkaee Taleghan, A. Taherkordi, M. Sharifi, "A Publish-Subscribe Middleware for Real-Time Wireless Sensor Networks", Lecture Notes in Computer Science (LNCS), No. 3991, Springer-Verlag, pp. 981 – 984, May 28-31, 2006.

[24] M. Sharifi, M. Alkaee Taleghan, A. Taherkordi, “A Middleware Layer Mechanism for QoS Support in Wireless Sensor Networks”, In Proceedings of the 4th IEEE International Conference on Networking (ICN'06), Mauritius, April 23-29, 2006.

[25] Crossbow Technology, Inc., Wireless Sensor Networks (product data sheet), http://www.xbow.com/ Products/Wireless_Sensor_Networks.htm, 2003.

[26] B. Segall, D. Arnold, “Elvin has left the building: A publish/subscribe notification service with quenching”, In Proceedings of the 1997 Australian UNIX Users Group Technical Conference, pp. 243–255, 1997.

[27] M. Ionescu, I. Marsic, “Stateful Publish-Subscribe for Mobile Environments”, In Proceedings of the 2nd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMash 2004), pp. 21-28, Philadelphia, PA, October 2004.

Amirhosein Taherkordi obtained his B. Sc. degree in software engineering from Sharif University of Technology, Iran (2001). He received his M. Sc. in Information Technology (Software Development) from Iran University of Science and Technology, Iran (2005). He is currently a research assistant in Computer Engineering Department at Iran University of Science and Technology and works on middleware, dependability issues, and application architectures for wireless sensor networks. His research interests include software architecture, distributed computing, wireless networks, sensor networks and web engineering. Mr. Taherkordi has been a member of Computer Society of Iran (CSI) from 2005.

Majid Alkaee Taleghan obtained his B. Sc. degree in software engineering from Sharif University of Technology, Iran (2002). He received his M. Sc. in Software Engineering from Iran University of Science and Technology, Iran (2005). Hi currently pursue his research on challenges in wireless sensor networks such as middleware, quality of service and fault tolerance in Computer Engineering Department at Iran University of Science and Technology. His research interests include distributed computing, wireless networks, sensor networks, quality of service, fault tolerance, and web engineering. Mr. Taleghan has been a member of Computer Society of Iran (CSI) from 2005.

Mohsen Sharifi is a B.Sc., M.Sc. and Ph.D. graduate of Victoria University of Manchester, U.K. He is an associate professor of software engineering and the head of Computer Engineering Department of Iran University of Science and Technology, working on distributed system software in fields such as distributed operating systems, sensor networks and web.