Autonomous Middleware Framework for Sensor Networks

Kiran Kumar Modukuri, Salim Hariri
Dept of ECE, University of Arizona
{kiranm, hariri}@ece.arizona.edu

Nader V. Chalfoun
School of Arch., University of Arizona
chalfoun@u.arizona.edu

Abstract

Programming Sensor Networks currently is a subtle task not because of enormous amount of code but due to inherent limitations of embedded hardware like the power, memory, network bandwidth and clock speed. In addition, there are very few programming abstractions and standards available which lead to close coupling between the application code and the embedded OS requiring understanding of low-level primitives during implementation. A Middleware can provide glue code between the applications and the heterogeneity of devices by providing optimized set of services for autonomously managing the resources and functionality of wireless nodes in a distributed wireless sensor network. This paper presents an autonomous middleware framework for low power distributed a wireless sensor network that supports adaptive sensor functionality, context aware communications, clustering, quality of service and fault-tolerance. Finally an application on the Envelope System Research Apparatus (ESRA) is also illustrated.

Index terms: middleware, context-aware, fault-tolerance, network reprogramming

1. Introduction

Next generation wireless networks are likely to be much smaller, cheaper yet powerful and robust and are likely to be deployed in large number for applications ranging from habitat monitoring, medical applications, industrial automation to mission critical military applications. Networked Sensor networks can improve sensing resolution by providing a weighted average of sensed data from a large number of spatially distributed sensors as compared to a conventional monolithic sensor. Common examples include Seismic data, acoustic data and high-resolution images. Unlike wired network applications, sensor applications require adaptation to dynamically changing environment. Self-configuring and cluster formation are currently the active research areas. Novel architectures [3][4][5][6], protocols [14][15][16][17][18][19] have been proposed in these fields focusing on power constraints and ad hoc nature of the sensor applications.

Currently, no standards and design rules have been followed in designing sensor architectures because of the wide range of applications that lead to a wide range of hardware capabilities and sensor data. There are certain common services that need to be provided in a distributed sensor network such as ad hoc routing, clustering, data fusion, resource management, security and fault tolerance. We believe that these services should not be static, rather they should be optimized and adapted for a certain type of applications based on the context state of the environment, current sensor state and the changing application requirements like power and latency. Managing this dynamic behavior and providing all the required functionalities under constrained resources is a challenging research problem.

In Section 2, we present the requirements that should be supported by the next generation of wireless Sensor Networks (WSN). In Section 3, we present our approach to implement an autonomous middleware Framework that supports the requirements identified in Section 2 such as adaptive functionality, resource management and Fault tolerance for wireless sensor networks. Section 4 discusses our implementation of a proof-of-concept prototype of the framework where we discuss its implementation to the control and management of a portable building architecture (ESRA).

2. Related Work

Many Middleware approaches have been proposed for sensor networks focusing on clusterization, energy consumption and data aggregation and collection mechanisms. Sensor middleware is categorized under two main categories, which is based on how the context is viewed.

In database approaches, the network is viewed as distributed database where the context depends only on changing data. Requests are in the form of SQL queries see Fig 3, which are flooded across the network following a spanning tree algorithm. Each sensor monitors the sensor values and informs its parent when the condition is satisfied. This process repeats until the node in the highest level of hierarchy gets the reply, which is the node that initiated the SQL query. TinyDB [3], Cougar [4] are some
examples of database approaches in sensor networks. The database approach is a good approach for getting filtered data periodically from a group of nodes. However, this approach does not provide an efficient way for intermediate nodes to take decision depending on the events based on network messages, time and changing resources.

In agent-based approach, the network is a reactive system, which responds to changing context based on events like network messages and timeouts. An agent is a special code that moves among the nodes to execute the code locally based on the events it monitors, gathers state information and then migrates to other nodes. This approach scales as the network grows compared to previous approaches but has more computational overhead of interpreting the agent code at every node. Sensor Ware [5] uses agent-based approach. In SensorWare, a Tcl interpreter runs on each node, which interprets any new script spawned onto the device. With this approach, new applications can be developed by writing a new script without the need to program the devices. Another middleware approach is Dfuse [6], which is similar to sensor ware views the application as coarse-grained data flow graph designed for data-fusion applications with emphasis on energy conservation. It supports role assignment based on the available resources in the network. Currently SensorWare and Dfuse are implemented on high end embedded devices like PDAs equipped with location, light, and temperature sensors.

Mate [7] is based on virtual machine approach implemented specifically for the Berkeley mica mote hardware architecture. Mate supports Simple assembly instructions to monitor and control the sensors in form of byte codes, which can be transmitted from one node to other. Reactive Sensor Network (RSN) [8] is similar to applet approach where all remote functions and remote data are specified by URLs and supports mobility by calling remote functions and downloading remote data. Sensor Querying and Tasking Language (SQTL) [9] uses combination of SQL queries and Agent based approach, which is an improvement, but the tasking is only to support dynamic queries.

All the above approaches either support homogeneous architecture with more or less similar capabilities. In real world applications, the nodes in a network contain varying capabilities ranging from simple RFID tags to complex signal processing devices. Because of the huge number of sensor nodes and varying complexities, programming each individual node is highly difficult. New programming paradigms need to be developed to specify functionality at the application level rather than low-level specifications like how the communication between individual nodes should be, which parameters need to be monitored, what should be the monitoring rate and what protocols to be used for a given application. Efficient Middleware solutions can hide the complexity involved in configuring individual nodes based on their capabilities and hardware architecture.

3.1. Programmability

Existing wired networks assume nodes to be stationary, reliable and having good bandwidth and no power constraints. In this situation, the communications among the nodes follow standard communication protocols (TCP, UDP, etc.). However, in mobile sensor networks, the nodes are interconnected by an ad hoc network, and the sensor nodes have very limited and constrained processing and communication capabilities. Furthermore, wireless sensor networks frequently experience disconnected operations, low bandwidth communications, and battery outages. Consequently, sensor networks need to support programmable functionality, asynchronous operations and component-based architecture. Event based programming languages like NesC [10] and galsC [11] support disconnected operations by using split phase operations; thereby allowing operations to be carried out asynchronously. NesC used component-based model similar to hardware description languages to provide a hierarchical architecture, so that applications can reuse existing software code.

Sensor Networks are likely to be deployed in large numbers ranging from simple RFID tags to complex signal processing devices. Because of the huge number of sensor nodes and varying complexities, programming each individual node is highly difficult. New programming paradigms need to be developed to specify functionality at the application level rather than low-level specifications like how the communication between individual nodes should be, which parameters need to be monitored, what should be the monitoring rate and what protocols to be used for a given application. Efficient Middleware solutions can hide the complexity involved in configuring individual nodes based on their capabilities and hardware architecture.

3.2. Adaptability

Many sensor applications need to change their behavior depending upon the environment state they monitor. Consider an example of a traffic application that monitors the number of vehicles passing a given point. During normal day operation, the video cameras can take snapshots and process the data based on normal imaging techniques but during night, the device can use an infrared camera for localization, fault tolerance, load balancing are dynamically downloaded to the devices.

3. Sensor network requirements

In this section, we identify the main services that must be provided by the next generation of sensor networks. These services will then be used to develop our autonomous sensor framework.
monitoring the vehicles. If the device does not have memory space required to fit all the functionality then we have two options to either send the data to a remote server for decision-making or provide a mechanism to download the appropriate algorithms into the device at sunset. Clearly programming the device with new functionality is a better solution compared to processing the data remotely. Therefore, sensor applications need to have remote reprogramming capability. XNP [12] provides simple remote programming mechanisms for remote programming the devices. SensorWare supports adaptability at much higher level by providing scripts to be deployed and spawned dynamically.

3.3. Expandability

Sensor networks have serious resource constraints in terms of memory, power, CPU and network bandwidth. Middleware for traditional wired networks is not application specific and hardware specific, which makes it inefficient and too complex. In wireless networks the middleware should be adaptive to the device being programmed depending upon the hardware capabilities and application needs. This calls for component-based middleware. Depending on the device (component) capabilities, the appropriate functionality can be added to the device such that it can perform in-network processing and thus saving energy; this reduces significantly the amount of data transmission.

3.4. Scalability

Most sensor network implementations take advantage of the spatial locality. For example, in case of object tracking and detection, the information from closely located nodes is evaluated and decision is made. Because of this data localization, the sensors can automatically form clusters based on their location and radio connectivity. In addition, as the sensor devices become cheaper sensor networks are expected to contain thousands of these devices scattered over a large geographical area thus making the node to administrator ratio very high. This calls for scalable solutions like self-configuration, self-maintenance and in-network data processing. Methodologies for forming clusters, electing cluster heads among the nodes based on energy and radio connectivity and reprogramming of the new nodes should be done transparently at runtime.

3.5. Inherent Dynamism

In Mobile sensor networks, network topologies, nodes, and services are inherently dynamic because of node mobility and device failures. Nodes can be added and removed from a cluster frequently thereby creating a change in the network topology. Traditional middleware do not support this frequent dynamism in the network. Therefore, we should provide mechanisms for the application to dynamically detect nodes and supported services. The naming schemes to be used should be “data centric” which will be discussed in the next section.

3.6. Level of Transparency

In Traditional middleware the goal is to hide most of the system functionality by providing a common view of the system but in case of sensor middleware, the applications need to be aware of the system state during runtime. This can be implemented in two ways either by providing context information directly to the application or by allowing runtime policies to dictate the middleware operations to be supported; this is also known as reflection mechanisms.

3.7. Real time Priorities

Sensor Networks are normally used to monitor real time phenomenon rather than stored information. This leads to the question of how to assign priorities during runtime and how to maintain in real-time the event order; which event to be delivered first. Real time CORBA [1] [2] (TAO and dynamic TAO) provides real time priorities for the task by mapping native system priorities to Cobra’s priority but the priorities need to be explicitly specified. In sensor networks, the priority of a message depends upon the context of the system. Hence, priorities to messages have to be assigned at runtime by the middleware and should be based on the context. Two types of priorities have been identified depending upon the requirement QOD [13] (quality of data) and QOS (quality of service), which are discussed in section 5.3.

3.8. Resource Management Features

Traditional Middleware relies heavily on the services provided by the operating system such as memory allocation policies, and IO management, which are highly sophisticated. However, in sensor network applications operating systems are very primitive and support application specific functionality. The middleware should complement operating system features like memory and IO management, multi threading for efficient utilization of resources.

4. System Architecture

The system architecture shown in Figure 1 provides a high level description of various subsystems present in the framework. Application Management Editor analyzes the application specifications and requirements at higher level and intelligently makes decision based on the available
active sensor nodes and their capabilities, environment and the network model of the application. For example, if the sensor network consists of few nodes the interference due to synchronized transmitters is less and simple CSMA/CD protocol can be used. If the network consists of many sensors spaced, closed to each other the number of collisions will be high and consequently, SMAC [20] can be used to minimize the collisions. In addition, device capabilities can dictate the appropriate communication protocol to be used. For example, in STEM [21] the sensors have two different transmitters a low power radio to hear and send control messages and high power radio for data transmission. This can improve the performance in dense networks and minimize energy used for idle reception. The device capabilities, network models, environment models and message protocols are stored and updated in the Application Knowledge database. These decisions are run through a policy engine that extracts low-level network models, clustering models and sensor models specific to the device model for high level events and requests presented by the application. These models are compiled for the specific device model and are handed over to the Application Deployment Manager (ADM). Finally ADM reprograms the devices remotely with new functionality.

4.1. Event server

For large-scale distributed sensor networks, event driven architecture is efficient compared to client server architecture because of the asynchronous operations between the service provider and users. The system consists of Event server that is responsible for listening to all incoming events from the wireless network. The Wireless networks can vary from simple sensor devices that have very less or no signal processing capabilities to highly sophisticated sensors with in-network data aggregation and fusion capabilities. The Event server is responsible for listening to all events generated from the network, applying application specific mappings like raw value conversions and filtering unwanted events.

4.2. Sensor Fusion Manager

Applications involved in object tracking and identification use a number of different sensors to identify a particular object. The Sensor Fusion Manager supports fusing of finer low-level events (raw temperature and light levels) to coarse high-level events like (object detection event, explosion event, trip wire detection event) using different fusion algorithms. Sensor Fusion Manager is responsible for selecting appropriate role assignment models and allocating roles to the nodes in the network.

4.3. Context Manager

Sensor applications require adaptations to the changes in their behavior and adapt to the context in which they are working. Context can be defined as a chain of state changes that lead to a particular state of the network. Consider a sensor network situation near an Oil field, which monitors oil levels, pressure and temperatures in the pipes and tanks. In addition, there are water tanks near the pipes and hydrocarbon storage tanks equipped with oil sheen detectors that are installed in nearby water tanks. When there is an oil leakage in the pipe the pressure level drops than normal level and if there is a sudden change in temperature the variable expansion in pipes can cause cracks. In addition, oil seeps into nearby water sources, which can be seen as a thin film over the water layer. For the sensors fitted in the pipeline, the context could be defined as show below.

```
Context1 <Leak Detect, Pipeline> =
(Pressure change > 200 psi) &&
(Temp change > 50 deg Fahrenheit) &&
(Delta Time < 2hrs)

Context event for sensors in pipeline

Context2 <Leak Detect, water Tank> =
(Oil sheen detected == true)
```
Context event for sensor in water tanks

<Actions> <close valves>,”call Field Operator”.

Action on Context1 & Context2

4.4. Resources and Network Status Monitor

Wireless Networks are characterized generally by constrained resources, dynamic network connectivity, and unreliable power sources. The Resource and Network status monitor forms the core of the Middleware runtime system, which monitors the heartbeats, sent by wireless devices in the network periodically informing the critical status of the devices. These messages are stored in a database for history analysis, which can provide vital information for decision-making and management of the network. Fault tolerance, load balancing and energy management. The system keeps records of various health parameters of sensor devices like the battery level, the CPU speed, task queue length and radio cost. In clustering algorithms, the root or head node selection can depend on the conditions such as the one with lower network costs or the one with highest battery power. The system can start a new root node selection whenever the head node’s battery power falls down below a certain threshold. These simple mechanisms can change catastrophic failure to a gradual degradation in the performance of the network. Also, consider a load-balancing scenario in which there are more than one sink nodes (base station) that connects to the conventional wired network. The load here is the number of messages to be forwarded to the network in a given amount of time. The middleware can balance the load by changing the sink node for some of the worker nodes in network. Fault tolerance can be achieved by adding redundant tasks or moving existing tasks for nodes having lesser life to healthy nodes.

5. Middleware Runtime Services

5.1. Sensor Management services

These Services provide various sensor-monitoring services using data centric and address centric solutions. For example, at the application level the data -centric monitoring services can be in form of SQL queries as shown in Listing 2 or as parametric functions as shown in Listing 1 below

SELECT AVG (volume), room FROM Sensors WHERE floor =6 GROUP BY room HAVING AVG (volume)>threshold EPOCH DUATION 30s

Listing 2. SQL query to monitor sensor information [3]

Address–centric sensor monitoring services can be used for conventional data-acquisition systems and behavior analysis for monitoring health status of an individual node and simple data acquisition applications. Also address centric services are needed for remote reprogramming of nodes in the network. Listing 3 below shows a function using address centric form of requests.

Listing 3. Address centric requests

5.2. Network Management Services

Every Application requires the nodes in a network to form ad-hoc networks based on energy constraints or latency constraints. Network Management services enables applications to tune the wireless network to the Application specific requirements. It will not be uncommon for sensor devices to support multiple radio technology on the same devices. For example in an activity monitoring application a sensor device can have a low power radio to listen for wake up messages from guard nodes to save the power that would be otherwise expended by using a high power radio. Many radio technologies are currently in use for wireless sensor networks like the ISM band RF, bluetooth, piconets and zigbee. There are several protocols proposed for different application scenarios to save power consumption at different layers of the radio-communication stack. These services override the default network algorithm selection by the middleware for example providing secure transmission protocol during authentication and data exchange.

5.3. QOS/QOD/ cost services

Conventional networks achieve higher QOS qualities by utilizing more network bandwidth but in wireless network, the QOS is dependent on the energy constraints imposed on the network. Three types of metrics have been identified for protocols in sensor networks namely QOD (quality of data) and QOS (quality of service) and cost.

QOD determines the freshness of the parameter being monitored or the accuracy of the event. Depending on the application specifications, the number of nodes required to monitor the parameter can be adjusted and the ADC
sampling can be adjusted in favor of energy conservation for the latest data.

QoS on other hand is related to the latency between the event generations at source to observation at the monitoring station. QoS is a function of resolution of data, number of hops, priority of the event, and the communication protocol being used. Multiple hops can be used to reduce the transmission range of the radio thereby saving energy consumption.

Sensor Networks are normally used to monitor real time phenomenon rather than stored information. This leads to the question of how to assign priorities to generated events and messages during runtime and how to maintain which event or message is to be delivered first. In sensor network the priority of a message depends upon the context of the system. Hence, priorities to messages have to be assigned at runtime by the middleware based on the context. Event and message priorities in the wireless network determine the QOS for an application.

The protocol involved in the data transmission can increase the reliability of the message by adding security, parity bits but this increases the processing overhead and latency. Cost is defined as the amount of resource spent for transmitting data for a given set of nodes. The cost could be defined as the length of time, amount of energy used or a mix of both time and energy. A simple QOS/QOD/cost specification could be as shown in listing 4 below.

\[
\text{(Latency < (2ms, 10ms)), Freshness<=5s, Cost: minimum energy)}
\]

**Listing 4 QOS/QOD/cost specifications**

5.4. Administration and control Services

Our System architecture supports multiple applications to run on a grid of wireless nodes. These applications are short-lived applications like querying for certain data by field operators. For example, in a parking lot application a query for next free parking slots. The middleware reserves the slot during the transaction and claims unused resources from other applications, which would otherwise lead to inconsistent information of the system. The middleware provides sensor administration services like controlling the devices within the sensor network like raising alarms and turning on and off actuators. We require certain administrative privileges to restrict the resource usage and control of devices in a network. The middleware authenticates users before providing control or access to devices in a network.

6. Experimental Results: ESRA project

In this section, we discuss about Envelope System Research Apparatus (ESRA) project, An Application based on our autonomous sensor middleware framework. We discuss briefly about ESRA project followed by the implementation.

6.1 Envelope System Research Apparatus (ESRA)

The Envelope Systems Research Apparatus (ESRA) is an ongoing research project at the College of Architecture of The University of Arizona to develop a sustainable, adaptable and mobile laboratory [23]. The goal of ESRA is to promote the learning of energy efficiency while exploring the unique aspects of the thermal properties of advanced building envelope systems and passive solar designs and technologies. The project also demonstrates innovative structure deployment systems capable to adapt to a variety of sites and conditions. An important link between the learning of energy efficiency and the place is the structure itself which had to be a testimony to the purpose of the project. Viewers and visitors of ESRA can understand first hand through testing and demonstration what a sustainable, flexible, and mobile structure can achieve.

For sustainability, ESRA will be designed to a high standard of energy efficiency levels while it will be capable of testing many other types of green and energy efficient building materials. For flexibility, it will experiment with an adaptable structure that will be deployed with ease, expanding the areas of its indoor spaces, while transforming the adjacent outdoor spaces to thermally comfortable places capable of accommodating different extended activities. The envelope itself will mediate the microclimate through passive design strategies such as ventilation, evaporative cooling, solar heating and light. The mobility aspect of ESRA allows it to travel to diverse or remote destinations while the structure is demonstrating how such transportable environment can serve for living, working, learning, servicing and assisting a variety of functions.

ESRA dimensions is envisioned to be a 32 feet length comprised of four 8 feet width by 8 feet height by 8 feet length modules capable for testing pre-manufactured 4X8 feet flat (or tilted) roofs or 4X8 feet vertical walls exposed to different orientations. The facility will also be capable of testing windows technologies and other different types of envelop systems. The apparatus will be equipped with advanced wireless data acquisition instruments for measuring temperatures, moisture conditions, radiant fields, air movement, and solar radiation and light. These parameters will be compiled through an interface that will be programmed to predict the energy efficiency of the indoor spaces as well as the thermal comfort conditions of the outdoor spaces.
6.2 Implementation Details

This project aims to create an autonomous system, which is able to monitor and control remotely and take informed decisions based on various environmental factors like temperature, humidity, light, pressure and wind within and outside the Mobile space and to monitor structural integrity of the mobile unit.

ESRA as discussed in previous section requires an easy to deploy system with very less infrastructure operated on batteries and no field personnel to setup the system. Therefore, this system requires services like self-configuring, fault-tolerance and load balancing and capability to monitor and controlled remotely.

In this paper, we focus on Remote Control and Management, Fault Tolerance and load balancing services of the middleware. We are currently testing on a test bed of ten processing boards comprising of four Mica2 radio processor boards (MPR410), six mica2dot radio processor boards (MPR510). We have nine sensor boards comprising of three mica2 compatible sensor boards (MTS 310) and six mica2dot compatible boards (MTS 510) and base station (MIB 510). A sensor board cascaded over a compatible radio processor board is called “Mote”.

Table 1 shows capabilities of Mica2 and Mica2dot devices.

<table>
<thead>
<tr>
<th>Mote Type</th>
<th>Mica2</th>
<th>Mica2Dot</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroController</td>
<td>ATMEGA128L</td>
<td>ATMEGA128L</td>
</tr>
<tr>
<td>CPU clock</td>
<td>7.3827 Mhz</td>
<td>4 Mhz</td>
</tr>
<tr>
<td>RAM &amp; Code Memory</td>
<td>4kb &amp; 128 kb</td>
<td>4kb &amp; 128 kb</td>
</tr>
<tr>
<td>Photo sensor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Temp. Sensor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Magnetometer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Accelerometer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MIC sensor</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Sounder</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Radio chip</td>
<td>ChipconCC1000</td>
<td></td>
</tr>
</tbody>
</table>

Motes have TinyOS [22] operating system that follows event-based architecture where events call higher level events or post tasks. The TinyOS puts the device in low power mode when there are no active tasks to be performed in task queue. Communication between different motes is established by a simple three-layer network stack.

Figure 3 Shows block diagram representation of a prototype designed for the ESRA application. The wireless sensors network provides temperature, light, tone detection, battery levels from mica2dot, and mica2 sensors that are sent at regular time intervals to base station via the Mica2 sensors. The Mica2 nodes act as forwarding nodes for mica2dot nodes. The base station collects the data from mica2 nodes and forwards these network messages to the server via the RS232 serial interface. “Serial Forwarder" collects sensor device data and forwards it to “MySql” database and to a “9001” TCP/IP port which provides a gateway for remote clients. Figure 4 shows a web page containing java applets showing real-time data from three sensors. The “Online Monitor” shows data in real time in form of tables and graphs, and provides interface to control each wireless sensor node at fine granularity. Currently we provide control of sampling interval, turning the Sensors, LEDs, MIC and device on and off. The sensor data is time stamped and stored in a database which is used by “Offline Monitor” for behavior analysis over a given time range. The amount of data stored in database is controlled by specifying simple policies in form of parameter tolerances for different information. Figure 5 shows the data logger, offline monitor and collected data in form of tables and graphs.
6.3 Fault tolerance service for Sensors

The system implements fault-tolerance by providing backup nodes for each sensor node. Listing 5 shows algorithm for a simple fault-tolerance for the application. Initially during cold boot up every node registers with the server informing about its device capabilities. Every node informs the system about its health (battery level and task queue length) periodically to the monitoring system [refer to Listing 5.1 Lines 1,2]. We call these messages heartbeats. Depending on the application specification the system adds friend node to the device to which the device should send alive messages. In the algorithm, the server checks for the available nodes with same capabilities and the “cost” which determines the nearest available friend [refer to Listing 5.1 Lines 3 –7]. The server informs each node with friend ID [refer to Listing 5.1 Lines 8–13].

When the registered friend node receives a “friend” message from the server, it adds the friend node id and monitors for “friend alive” messages from its assigned friend node. Listing 5.2 show the above functionality. When the node receives the “friend alive” message, it gets information about the node functionality like the events to be monitored and sampling interval. When the node receives, a message it resets the “friend live” counter to a predefined constant MAXTIME. Listing 5.3 shows the algorithm for receiving “friend alive” messages.

The Node periodically sends information about its status to the friend node and decrements the wait counter “friend live”. If the counter is zero then the node assumes that the assigned friend is dead and adds the friend tasks to its task list. Listing 5.4 shows the above functionality for sending alive messages and detecting node failures.

Figures 6, 7, and 8 show a simple scenario as the implementation of the fault tolerance algorithm. Initially all the nodes are working and mote10 monitors both light and temperature, mote11 monitors temperature and mote12 only light as shown in figure 6. Now mote10 and 12 are assigned friends in this case. Mote10 is turned down remotely from the control panel as shown in figure 7. Now the Mote 12 waits for a specified amount of time to account for lost messages. When the timer expires, the Mote12 adds temperature-monitoring task to the task list until the mote10 is alive again as shown in figure 8.

### Listing 5.1. Server side algorithm

<SERVER>

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Wait for NEW_NODE Messages</td>
</tr>
<tr>
<td>2.</td>
<td>GET NODE_CAPABILITIES</td>
</tr>
<tr>
<td>3.</td>
<td>IF (NODE_LIST has nodes)</td>
</tr>
<tr>
<td>4.</td>
<td>For each FREENODE in NODE_LIST</td>
</tr>
<tr>
<td>5.</td>
<td>Check if (FRIEND_ID.Type = FREENODE.Type)</td>
</tr>
<tr>
<td>6.</td>
<td>Cost &lt;= W1* (Num tasks) + W3 * (Battery)</td>
</tr>
<tr>
<td>7.</td>
<td>IF FREENODE has Cost &lt; limit</td>
</tr>
<tr>
<td>8.</td>
<td>Send FriendMSG NEWNODE with FreeNodeID as Friend</td>
</tr>
<tr>
<td>9.</td>
<td>Send FriendMSG FREENODE with NewNodeID as Friend</td>
</tr>
<tr>
<td>10.</td>
<td>Remove FREENODE from NODE_LIST</td>
</tr>
<tr>
<td>11.</td>
<td>ELSE Add NEW_NODE to the NODE_LIST</td>
</tr>
<tr>
<td>12.</td>
<td>ENDIF</td>
</tr>
<tr>
<td>13.</td>
<td>Go to step 1</td>
</tr>
</tbody>
</table>

### Listing 5.2. Node receiving messages from server

<RECEIVE FRIENDMSG>

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Wait for FRIEND_MSG FROM SERVER</td>
</tr>
<tr>
<td>2.</td>
<td>ADD FRIEND_MSG.FRIEND as FRIENDID</td>
</tr>
<tr>
<td>3.</td>
<td>FRIENDLIVE = MAXTIME</td>
</tr>
<tr>
<td>4.</td>
<td>HASFRIEND=true</td>
</tr>
<tr>
<td>5.</td>
<td>FRIEND_TASK ={}</td>
</tr>
<tr>
<td>6.</td>
<td>GOTO STEP 1</td>
</tr>
</tbody>
</table>

### Listing 5.3. Node receiving alive messages from assigned friend

<RECEIVE FRIEND_ALIVE_MSG as MSG>

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>WAIT FOR FRIEND_ALIVE_MSG</td>
</tr>
<tr>
<td>2.</td>
<td>IF MSG.ID = FRIENDID</td>
</tr>
<tr>
<td>3.</td>
<td>HASFRIEND = true</td>
</tr>
<tr>
<td>4.</td>
<td>INITIALISE FRIENDLIVE = 3</td>
</tr>
<tr>
<td>5.</td>
<td>FRIEND_TASK = MSG.TASK</td>
</tr>
<tr>
<td>6.</td>
<td>GOTO step 1</td>
</tr>
</tbody>
</table>

### Listing 5.4. Node sending status periodically to assigned friend and checking timeouts

<SEND FRIEND_ALIVE_MSG >

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Wait Until HASFRIEND = true</td>
</tr>
<tr>
<td>2.</td>
<td>START WAIT TIMER</td>
</tr>
<tr>
<td>3.</td>
<td>IF TIMER_EXPIRE = true</td>
</tr>
<tr>
<td>4.</td>
<td>IF FRIENDLIVE &lt;= FRIENDLIVE - 1</td>
</tr>
<tr>
<td>5.</td>
<td>ENDIF</td>
</tr>
<tr>
<td>6.</td>
<td>IF FRIEND LIVE = 0</td>
</tr>
<tr>
<td>7.</td>
<td>ADD FRIEND_TASK to TASKLIST</td>
</tr>
<tr>
<td>8.</td>
<td>HASFRIEND = false GOTO step 1</td>
</tr>
<tr>
<td>9.</td>
<td>ELSE</td>
</tr>
<tr>
<td>10.</td>
<td>CREATE NEW FRIEND MSG WITH ID</td>
</tr>
<tr>
<td>11.</td>
<td>ADD TASKLIST to FRIEND_MSG.TASK</td>
</tr>
<tr>
<td>12.</td>
<td>SEND FRIEND_ ALIVE_MSG to FRIENDID</td>
</tr>
<tr>
<td>13.</td>
<td>ENDIF</td>
</tr>
<tr>
<td>14.</td>
<td>GOTO step 2</td>
</tr>
</tbody>
</table>
Figure 4. Showing remote data monitoring over Internet

Figure 5. Showing data storage for offline monitoring and behavior Analysis.

Figure 6. Showing initial Monitoring condition

Figure 7. Showing Mote 10 failure condition

Figure 8. Showing restored state of Mote 10.

7. Conclusions and Future Work

Distributed wireless sensor networks provide a promising future in areas of pervasive computing. In this paper we focused on implications of distributed low power wireless sensor networks and discussed essential services for a middleware to dynamically monitor, analyze and adapt the wireless sensor network with changing application context.

We believe that by providing a autonomous middleware to control the wireless network we not only add automation but intelligence to the network further enabling mission critical applications to run reliably and efficiently by effective utilization of resources.
8. References


