CHAPTER 3
DESIGN AND IMPLEMENTATION

In this chapter we have mentioned the reprogramming system design and implementation. The starting sections mentioned about the requirements and properties of the code distribution.

3.1 WIRELESS CODE DISTRIBUTION

Sensor network users need remote programmability in order to add new functionality to the nodes. The administrator sends the code images over the air to the remote nodes using base station/sender nodes.

3.1.1 REQUIREMENTS AND PROPERTIES OF CODE DISTRIBUTION

A code distribution mechanism should be designed to fulfill the following:

1. The code distribution scheme should be able to update all code on the sensor node, including itself and the operating system. Also, it should not restrict the size of the programmes that can run on the nodes significantly, compared to when they are manually uploaded.
2. The scheme should be resilient to losing some packets during the process since nodes may operate in noisy conditions, have very simple radios, and cannot afford expensive transmission schemes.
3. Since communication is expensive in terms of energy, the code distribution scheme must limit communication as much as possible. Often changes to the code will be small. In those cases little communication should be required to distribute the new code.
4. Resources on the sensor nodes are scarce. The part of the distribution scheme running on the nodes should not require excessive processing, and should use little memory.
5. Finally, when updating code, we want the application to be stopped for only a short period of time.

Reliably disseminating a piece of data to every node is a fundamental primitive for wireless sensor networks.

3.1.2 RESOURCE UTILIZATION

Sensor nodes are generally lack of power and memory. Our target platform is Mica2 motes, which are severely resource constraint devices, so careful plan is needed to meet the requirements of the code distribution. The most limited resource on a mote is energy. All operations require it and there is only finite amount available. It is not always possible to replace their batteries.

The most energy-intensive operation on the mote is radio usage and in particular, packet transmission. The CC1000 radio consumes 12 mA on transmit mode and 4 mA on receive mode [21]. Another significant energy consumer is stable storage (EEPROM) access. A Write( ) operation needs on average approximately one eighth the amount of energy required for transmitting the same number of bytes. Reads( ) are significantly cheaper than Write( )s, since most FLASH EEPROM's are optimized for Read( ) operations. However, due to the nature of code distribution, every code segment has to be stored in EEPROM; therefore, the number of Write( )s can be a significant factor in the overall energy consumption. Among computing, communication, and sensing functions, communication consumes a large portion of the energy, as shown in Table 3.1 [3].

Operations	Power consumption (nAh)
Read a data block from EEPROM	1.261
Write a data block to EEPROM	85.449
Send one packet	20.000
Receive one packet	8.000
Idle listen for 1 ms	1.250

Table 3.1: Typical Power Consumption of Mica2 Motes [3]

3.2 CODE DISTRIBUTION PROTOCOL

Our distribution protocol is based on the hierarchy in the network and it is divided into two cases

1. Code distribution protocol with no hierarchy
2. Code distribution protocol with hierarchy

First case has no hierarchy in the topology (i.e. one sender, many receivers) and second case has hierarchy (i.e. super nodes, normal nodes). In the coming sections we will clearly see about these two cases.

3.2.1 CODE DISTRIBUTION PROTOCOL WITH NO HIERARCHY

One sender (i.e. base station), and assume that all the sensor nodes are in the coverage range of the sender. Our protocol assumes that all nodes in the network (except sender/base station) running the same application and are interested in the same code updates. So here there is no problem of sender redundancy. There are two basic schemes in code dissemination as we discussed in section 2.2, of chapter 2 of this thesis.

In our protocol the sender/base station initiates for the dissemination. So it is **sender-initiated** paradigm for distributing the code. In **receiver-initiated** paradigm, the sender broadcast the piece of code images when a receiver requests it. The code which is to be broadcasted is divided in to pages, which are further divided into communication packets.

The protocol consists of the following four phases.

1. **Advertise phase:** Sender advertise the new version of the code it has, and all interested nodes request for code. Depending on the requests it receives, sender decides whether it should start forwarding code or go to sleep.

In figures 3.1, 3.2, the black circle represents the base station/sender and the white circle represents the normal sensor node. The Figure 3.2 shows the advertisement phase of our protocol. In the Figure 3.3, we can see that the nodes

requesting to the base station for new code. Based on the number of requests from the remote nodes we can predict that how many nodes are alive and their details (i.e. Node ID). So the system administrator comes to know about the failed nodes.

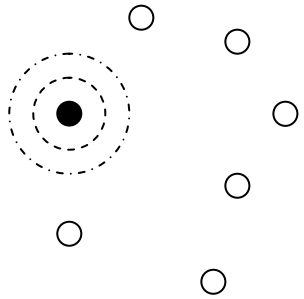

Figure 3.1: Advertise Phase

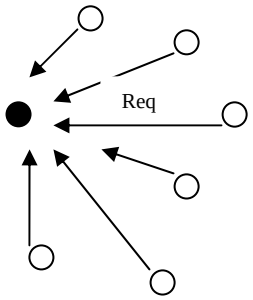

Figure 3.2: Requests from the Remote Nodes

2. **Download phase:** Once a source node decides to forward code, it broadcasts a "StartDownload" message to inform all the receivers to get prepared for the arrival of new code. The program image is divided into several pages and those are further divided into packets. The sender broadcasts the program code packet by packet to the receivers. During this process the receivers keep silent and store the received packets in EEPROM.

3. **Query phase:** Once the new program has been transmitted, the sender broadcasts "query" message to its receivers, which respond by requesting the packets they are missing. The missing packets are re-transmitted by source node using broadcast. This process continues until there are no more requests for lost packets from receivers. At this point, the sender assumes that every receiver has correctly received the program code, and it goes back to "sleep" state.

4. **Reboot phase:** During reboot period, new program code is transferred from EEPROM to program memory, new program ID is written to a fixed address space in EEPROM. After reboot, the node starts running new program.

3.2.2 DATA MANAGEMENT HIERARCHY

In this section we clearly see about the data management hierarchy. The Figure 3.3 shows the division of the code object into pages. In the Figure 3.3, the S_{obj} says the size of the program image; S_{page} says the size of the each page. Each page is further divided into N number of packets. The total code which is to be broadcasted is initially divided into certain number of pages, say P.

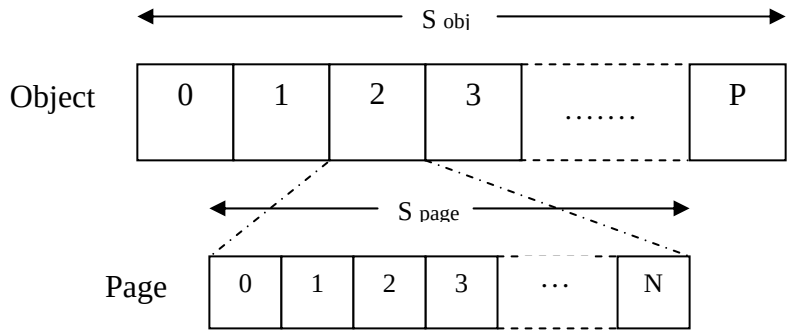

Figure 3.3: Data Management Hierarchy

Again each page is divided into certain number of packets, say N. The P and N values depends upon the code size which is to be broadcasted by the base station/sender. To ensure the receipt of all packets, the node must keep track of which packets needed are needed to complete the object.

However, because the packet size is generally much smaller than the object, simply maintaining a bit vector consumes an unacceptably large amount of RAM. Instead, our protocol fragments the data object into P pages each of size

$$S\ page = N * S\ pkt$$

Where N is a fixed number of packets, as shown in Figure 3.1. By requiring a node to dedicate itself to receiving a single page at a time, the bit-vector need only be N bits in length.

Both packets and pages include 16-bit cyclic redundancy checks (CRCs). If a packet or a page fails the CRC, all data represented by the CRC is discarded and must be received again. Redundant data integrity checks at both the packet and page level help ensure that data is correctly received by all nodes.

3.2.3 CODE DISTRIBUTION PROTOCOL WITH HIERARCHY

A general protocol of code disseminating in WSNs is classic flooding. A base station broadcasts the new code to its neighbors. Upon receiving the code, each node stores and then broadcasts to its neighbors. In section 3.2.1 we have seen the code distribution protocol with no hierarchy, that protocol is useful if all the nodes are in the coverage range of the base station. In many sensor networks it is not possible to cover all the nodes with single base station. Some sensor networks like flood detection networks [19], smart dust [20] have thousands of nodes. So to reach all the nodes, we have to hop the data. Sending the new code using flooding will not work if the network is too big. With flooding there are some problems like collision of the data, data redundancy, and contention. Broadcasting the code too quickly can easily overload the network, causing the “broadcast storm” problem [11], and is mainly caused by two insufficiencies:

• Data redundancy [16]: A sender may send out unnecessary (e.g., already received) data to its neighbors. To reduce data redundancy, a sender should be aware of what data has already been received by its receivers.

• Sender redundancy: Some senders are redundant to cover a desired area. These nodes cannot offer additional coverage (i.e., nodes that have not been covered by other broadcasts).

To overcome the above problems, we propose a technique which uses the combination of routing and broadcasts to rapidly deliver a piece of data to every node in a network. In this case we first send code to the super nodes and then super nodes reprogram other nodes in their local areas. Super nodes can be cluster head nodes, or a set of connected dominating nodes that are sufficient to cover the whole network.

This technique has two phases. First phase is routing phase and broadcast phase. When sender/base station decides to disseminate data, it first routes data to super nodes in the network. Nodes along these routes store the data as if they had received it as well. Once all super nodes receives the data, then each super node uses a broadcast –based local dissemination protocol as mentioned in the section 3.2.1 of this chapter.

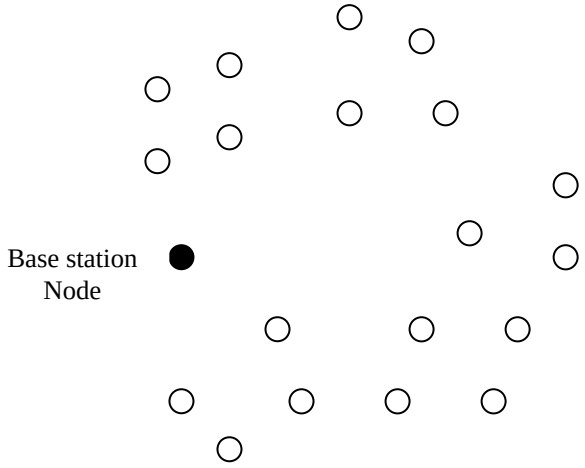

Figure 3.4: Sensor Network Topology

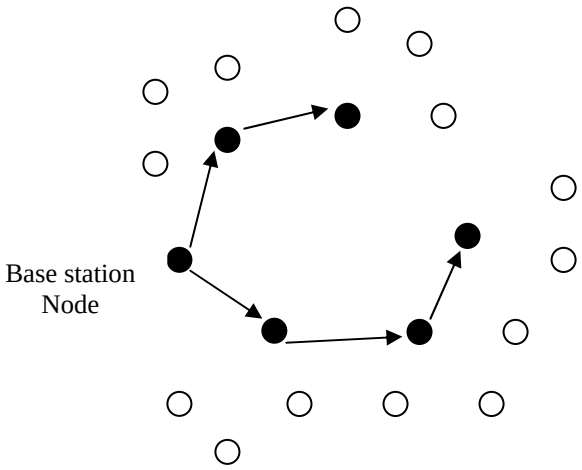

Figure 3.5: Routing Code to Super Nodes

The Figure 3.4, 3.5, 3.6, 3.7 shows the behavior of the protocol. In Figure 3.4 we can see the topology of the network; the black node is a base station node and white nodes are normal sensor nodes. In Figure 3.5 we can see that the base station node routes the code to super nodes. In Figure 3.6; the super nodes start broadcasting code to the normal nodes.

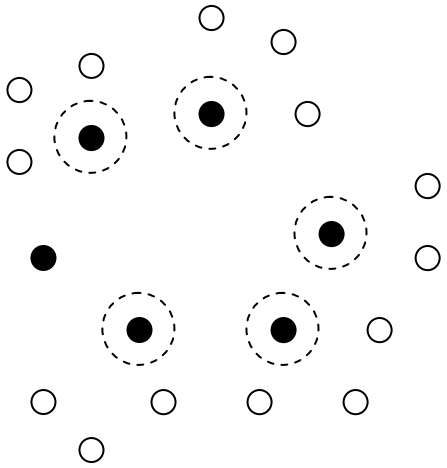

Figure 3.6: Broadcasting the Code by Super Nodes

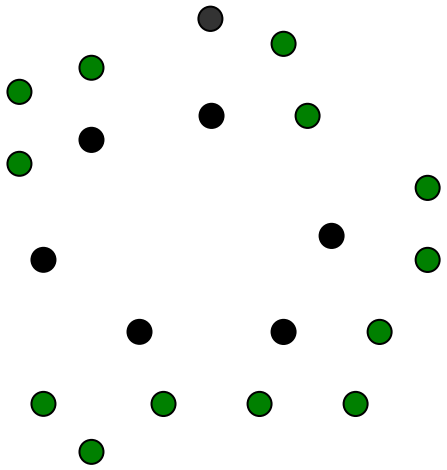

Figure 3.7: Completion of Code Disseminations

To maximize the energy efficiency, we have divided the dissemination protocol in to two phases. Broadcast based dissemination begins a few seconds after routing.

This minimizes the time for electing the nodes for broadcasting and maximizes the number of nodes that receive broadcast messages. By creating the hierarchy we send the code to all the nodes with out contention. The administrator should find out the set of super nodes by observing the topology. There is no fixed topology in sensor networks, so one has to define the super nodes, base station according to the topology.

The route for the super nodes is initially fixed by the network administrator, so the super nodes can go for pipelining (i.e. super node start sending the page to the other super node, once it is received a particular page successfully).

3.3 IMPLEMENTATION

Based on the design techniques mentioned in the above sections, we have implemented the protocol (Code Distribution Protocol with no hierarchy). The modules are written in NesC for TinyOS operating system. In our lab we have got Crossbow sensor kit. This contains four Mica2 Motes. Out of four one Mica2 mote is a base station and remaining are normal sensor nodes. We have implemented the protocol “Code Distribution Protocol with no hierarchy” by creating test bed based on Berkely motes (Mica2 motes) and TinyOS operating system. We will see the results in the next chapter. We were unable to see the working of the protocol “Code Distribution Protocol with hierarchy” on Mica2 Motes due to lack of hardware. We have the following modules in our implementation.

1) Advertise Module
2) Broadcast Module
3) Query Module
4) Reboot Module
5) Routing module (used if the hierarchy is there in the network)

We have modified the Surge protocol [22], which comes with TinyOS installation for creating Broadcast module. Surge is a protocol that forms a spanning tree that covers every node in the network and the root of the tree is at the base station. Base station periodically floods the network with beacon messages. Surge periodically measure the link quality among nodes according to the beacon message reception. Each node selects its parent for data forwarding based on the link quality

measurements. CSMA MAC is in the TinyOS release, our reprogramming system uses CSMA.

Radio communication in TinyOS follows the Active Message (AM) model, in which each packet on the network specifies a *handler ID* that will be invoked on recipient nodes. Think of the handler ID as an integer or "port number" that is carried in the header of the message. When a message is received, the receive event associated with that handler ID is signaled. Different motes can associate different receive events with the same handler ID.

In any messaging layer, there are 5 aspects involved in successful communication:

1. Specifying the message data to send;
2. Specifying which node is to receive the message;
3. Determining when the memory associated with the outgoing message can be reused;
4. Buffering the incoming message; and,
5. Processing the message on reception

To participate in the network a mote must have the following network level information.

1) A unique 16 bit NODE ADDR
2) An application level GROUP ID
3) A network level Radio frequency

Sensor nodes may not have global identification.

We use the following four types of Active Messages (AM).

1) Beacon (messages for making connection with base station)
2) Data (messages for data transfer)
3) ACK (messages for the acknowledgements for successful reception of each packet)
4) Cmd (Command messages like reboot, prepare…etc)

NesC applications consist of one or more components linked together to form an executable. A component provides and uses interfaces. Interfaces declare a set of functions called commands that the provider of the interface must implement and

another set of functions called events that the user of the interface must implement. NesC has two types of components: modules and configurations. Modules implement one or more interfaces. Configurations are used to assemble other components together, connecting interfaces to their implementation.

3.3.1 PROGRAMMING ENVIRONMENT

The TinyOS tools are available for Linux/Unix and Windows (under Cygwin) and contain various tools:

- NesC compiler,
- AVR compiler and utilities
- Java SDK and Java COMM

TinyOS provides tools to convert TinyOS messages to Java classes, so the messages can be used in Java programs. This way Java can be used to create programs to communicate with the nodes, using a serial connection via the programmer and a dedicated node on the programmer. The TinyOS tools also come with a Java program called the "Serial Forwarder", which can provide communication with the sensor network (using a programmer and node) via a TCP/IP network. This tool can be used to link any program to the sensor network.

3.3.2 ENCODING

Our reprogramming system compile whole program image and produces the machine dependent binary. The size of the binary depends on the application that we are sending over the air. The binary size is generally in the order of kilo bytes.

3.3.3 UPDATE INSTALLATION

The reprogramming system code is not loaded into the base station mote. After compilation, we get the binary image. We upload only that binary into the base station mote. There are two types of binaries, one is running on the remote nodes and other is running on the base station.

Each mote has a 512 KB external non-volatile flash divided into 4 slots. These slots have a default size of 128 KB. Slot 0 is reserved for the over the air program image. Slots 1, 2 and 3 can be used for user-specified firmware. During the reprogramming process base station sends a command to remote mote to reboot into slot 0. A user-specified firmware image is broken up into fragments and transmitted to the mote and stored into Slot 1, 2 or 3. The base station can send a message to transfer the newly uploaded firmware into the program flash and reboot the mote. For remote reprogramming the mote, it needs to have pre-configured with a bootloader in the program flash.

Bootloader: A piece of code that is guaranteed to execute after each reset independent of the TinyOS application. The bootloader is responsible for programming the microcontroller and recovers from programming errors by loading the Golden image.

Bootloader is a special Mote program

- Fetches image (CPU instructions) from external flash
- Writes image directly into processor's program memory: self-programming
- Repeats until new code image is completely loaded
- Reboots the processor: new program starts execution

Golden image: A program image with minimal support for network programming is stored in a safe location on external flash. Always have a piece of code that will allow for recovery.

3.4 MAINTAINABILITY OF SENSOR NODES

Reprogramming is one of the maintainability issues of sensor networks. For efficient operation of the sensor network we have to go for node level maintainability. Apart from reprogramming maintainability also includes integration of new nodes into the network, setting new threshold values at the sensors. We monitored the node health, by observing battery voltage and Received Signal Strength.

Health Data: Health packets are sent every several minutes, the health information includes data on how well the node is performing in the network with regards to the radio traffic, battery voltage and parents node Radio Signal Strength Indicator (RSSI). Sending the health packet is similar to the data packet except that it uses end to end acknowledgement.

The following parameters of the remote mote can be set using the base station mote.

1) Data rate
2) Node id
3) Radio power

Sensor networks can be easily adaptable to the changes by achieving the maintainability.