BriMon: Design and Implementation of Railway Bridge Monitoring Application

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Hemanth Haridas

under the guidance of

Dr. Bhaskaran Raman

and

Dr.Kameswari Chebrolu

to the

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur.

May 2006

Abstract

In this thesis we design a *multi hop wireless sensor network* based solution for the problem of structural bridge monitoring (of railway bridges). The problem presents several novel challenges. The real challenge here is to collect vibration data of a railway bridge, enabling on-demand analysis of the bridge. We not only provide an in-depth analysis of these challenges but also design a solution using multiple wireless technologies (*IEEE 802.11 and IEEE 802.15.4*). We take an application driven approach in providing a solution to the above problem. The implemented solution shows how the design choices dictated solely by the application are different from the general solutions that exist in the sensor network domain.

We design and implement protocols (*routing and transport*) on motes which are driven by the application. The transport protocol designed by us is NACK based and is used for reliable data transfer from all nodes to the base node. The routing algorithm determines the path that data follows. We further integrate our work with a fellow colleagues' synchronization mechanism and Wake on WLan mechanism. We use synchronization mechanisms to synchronize readings of accelerometers placed at different points and for synchronized sleep and wakeup mechanisms and a novel Wake on WLan mechanism for detection of arrival of a train. The *transfer of data from the base node to the moving train is achieved using WiFi.* We use off the shelf hardware to implement most of the solution, but where inexpensive off the shelf hardware is not available we build our own hardware. The solution has been implemented on *Moteiv's Tmote-sky and the Soekris* (single board computer). TinyOS and nesC has been used for programming the motes.

The challenges we have tackled in this work are one of a kind in this domain. There have been various works on monitoring of bridges and structures in India and abroad but most of them are wired solutions but a few are single hop wireless sensor network solutions which severely limit the size of the bridge being monitored. The architecture of the solution we have designed and implemented is unique to the best

of our knowledge. The transfer of data to a moving train and detection of the arrival of the train are a new experiment in their own right.

Acknowledgment

I would like to thank Dr Bhaskaran Raman and Dr Kameswari Chebrolu for their continuous guidance, support and mentoring. It is their support at the time of crisis that has made this thesis a reality. I was fun working with them. I am honored to have interacted with such wonderful people. I would like to thank Dr C V R Murty for his critical inputs. Dr K K Bajpai and Abhishek Singhal provided us with valuable domain knowledge which shaped the application design and their contribution was invaluable. Nilesh Mishra's help and coordination while integrating both of our components was indeed remarkable. I would like to thank the department for such a healthy, conducive and entertaining environment that made my thesis work so much fun even in times of frustration. All the staff at Media Labs Asia (DGP Project) were very helpful all along the thesis, a special mention should be made of Sukanto Kole who along with lending a helping hand was also a constant companion. I take this opportunity to thank V M D Jagannath, all my collegues and all my batch mates and my juniors for their support and encouragement. This thesis would not have been possible without the love and affection of my parents and my sister. I dedicate my thesis to my parents and all my teachers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

India has the second largest rail network in the world, transporting over four billion people annually. An essential part of this network are 120,000-plus steel bridges [17], a lot of which are aging, weak, distressed and accident-prone. Actual numbers indicate that 43% are over 100 years old and 57% are over 80 years old. There is currently no system in place to continuously monitor these bridges. It is extremely dangerous to operate trains carrying hundreds of passengers on a railway network, of which these bridges are a part. It is mandatory to either repair these bridges or construct new ones in place of them. To replace all the old bridges at once is not a feasible solution because it would involve prohibitive costs. Moreover with no data to support the claim that these bridges are indeed old and need urgent attention, it would be rather difficult to have a strong case for the replacement of these bridges. A more feasible solution is to continuously monitor these bridges and have data on the status of these bridges which could be easily interpreted by structural engineers. Meaningful conclusions could be drawn about the health of the structure in question [14]. A comprehensive data set can also help in the advancement of structural engineering research.

Currently structural engineers use wired systems to monitor these bridges. The setup requires accelerometers, shake-tables and expensive data acquisition systems.

The equipment is expensive and bulky, so the analysis can be carried out only at few points of the structure and it also limits the data set available for structural engineers for analysis. The setup takes anywhere between a few days to few weeks [30] and requires manual expertise. Due to the kind of costs involved in terms of money, time and expertise, such an exercise is performed infrequently and that too not on all the bridges that require monitoring. On the other hand an inexpensive, light weight, multi hop wireless automated setup would require less time to setup and can be relatively widely deployed. This would result in data set being available in digital form for detailed off line analysis using various scientific software on back end computers.

1.1 Problem Statement

In this section we articulate the problem statement and describe the problem in detail and thus motivate the solution for the application. The problem can be stated as follows **The structural response (readings of the accelerometer) of the Railway bridge have to be reliably delivered to a central location for further processing.**

A further analysis of the problem yields the following requirements

1. Vibration readings has to be obtained at every pier, in fact at 2 points on a single pier
2. There should not be significant loss of vibration data .
3. Data should be recovered even in case of system failure.
4. Vibration data needs to be collected in the following scenarios
    - Vibrations induced when train is passing on the bridge (Forced Vibrations).
    - Vibrations induced after the train has passed over the bridge (Free Vibrations)

- Vibrations induced by wind (Ambient vibrations)

The above requirements have to be satisfied with the following constraints

1. All equipment has to be operated by battery power, calls for intelligent sleep and wakeup mechanisms.
2. Currently no energy efficient way available to tell, when train passes over the bridge .
3. Hostile environments both for equipment and wireless transfers.
4. System may fail due voltage variations, physical damage etc.

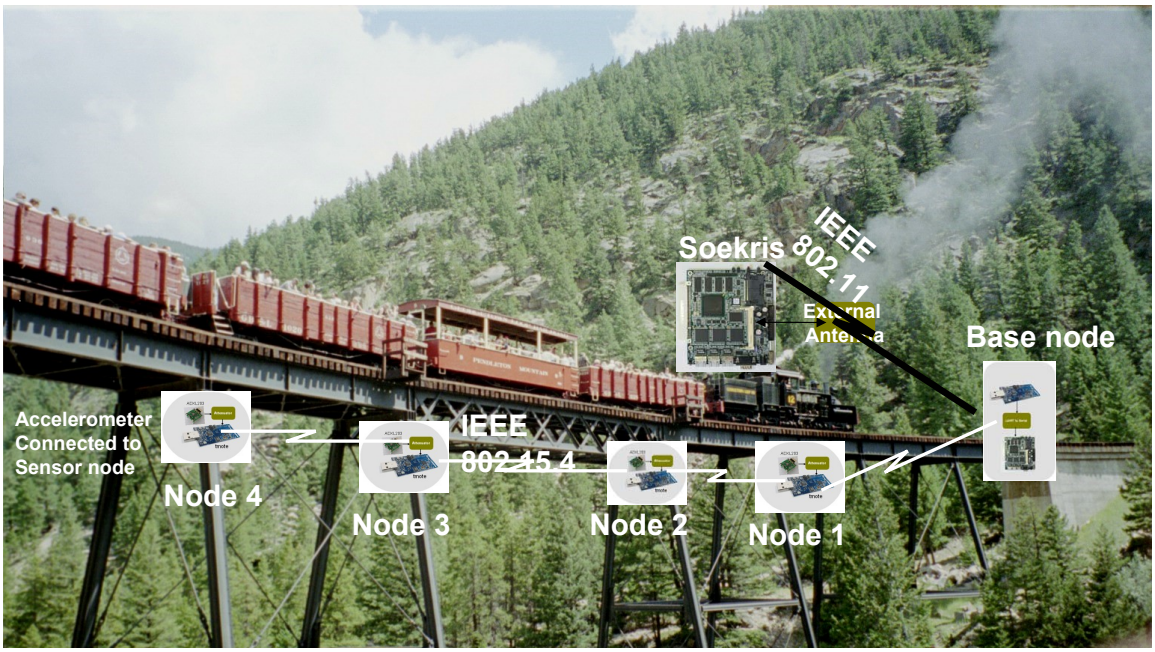

Figure 1.1: Schematic description of our solution at work along with the problem setting

The Figure 1.1 gives a peek into the kind of problem we are trying to solve and the entire solution we have proposed.

1.2 Overview of the Application Design and Implementation

1.2.1 Overview of Application Design

In this work we take a bottom up approach to provide a wireless sensor network solution to the problem under consideration as opposed to a top down approach. We design and implement algorithms and protocols specific to our application, which may not necessarily be optimal in other application settings. We optimize our solution to give the best solution in the considered application setting as opposed to designing generalized solutions which will work for all situations. In taking an application driven approach we seek to separate actual problems from potential ones and differentiating relevant issues from irrelevant ones.

We use Tmote-sky's sensor node as nodes of a sensor network and soekris single board computer as an aggregator and gateway at the base node.

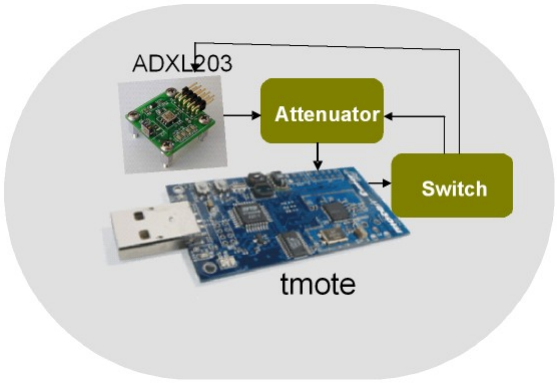

Figure 1.2: Components of a single sensor node

Figure 1.2 shows the block diagram of a single node to be deployed at each of the piers of a bridge. The diagram shows an accelerometer connected to the tmote through an attenuator. The attenuator just matches the voltage level of the accelerometer to that of the tmote. Both the attenuator and the accelerometer are controlled by the switch circuit connected to the tmote. The switch circuit is used to power on and

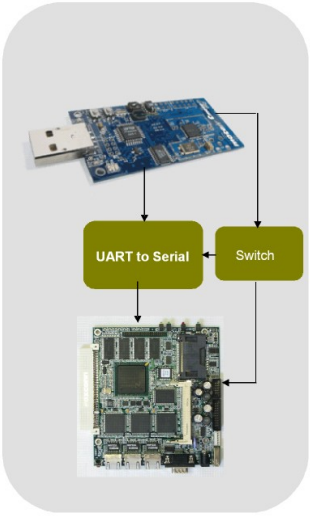

Figure 1.3: Components of the base node

power off both the circuitry.

Figure 1.3 shows the block diagram of the base node deployed at the edge of the bridge responsible for collecting data. The diagram shows a tmote connected to a soekris through a UART to serial converter. This setup is used to transfer data from the tmote to the soekris for further data transfer (using WiFi) to another soekris sitting inside the train. Both the soekris and the UART to serial converter are controlled by the switch circuit connected to the tmote. The switch circuit is used to power on and power off both the circuitry.

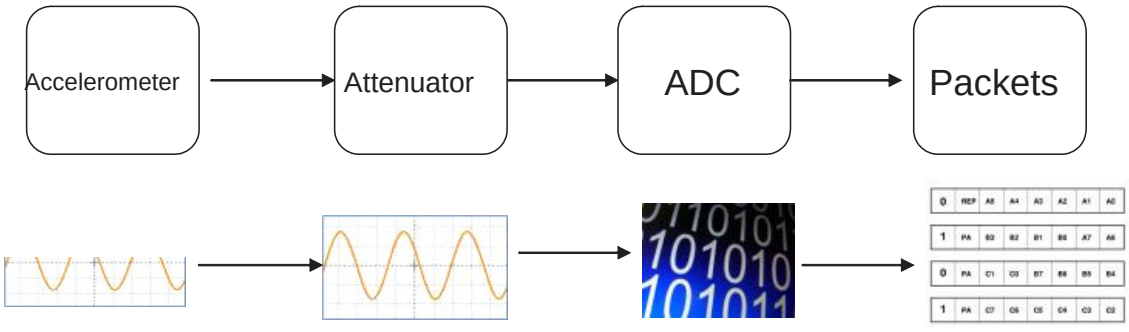

Figure 1.4: Application Design of a Single Node

We use vibration analysis to monitor the health of the railway bridges. Vibration analysis [6] involves analysis of time and frequency domain waveforms, identify-

ing changes in amplitude and frequency suggesting repair or/and replacement. The technology is based on the assumption that frequencies of the collected data can be mapped to specific components of the test subject.

A tri-axial accelerometer is used to get vibration data of the Railway bridge, by placing them one each on the piers of the bridges. An accelerometer is an electro mechanical device that produces electrical signal proportional to the applied acceleration. A tri-axial accelerometer measures acceleration in the X, Y and Z axis. The vibration data is fed into a wireless sensor node through the external ADC of the mote. The attenuator placed between the ADC of the mote and the accelerometer makes sure that the signal level of the accelerometer is within the operating range of the mote's ADC.

The sensor node (mote) collects the data from the accelerometer and stores the collected data in its flash. Each of the motes is equipped with flash (write once read many times) of size 1MB. The mote upon reception of a command to transfer data transmits the data reliably to a designated node called as the base node. The base node is responsible for issuing commands , aggregating data and collecting information about the status of the network periodically. The application design at a single node is depicted in the Figure 1.4.

1.2.2 Overview of Implementation

In this section we describe the relevant features of off the shelf hardware and software that impacts our implementation.

- The tmote-sky motes , hereafter referred to as the tmotes is one of the latest offering from Mote-IV [11] corporation in the wireless sensor network domain. The tmote-sky features a CC2420 radio from Chipcon [4]. The CC2420 radio provides two metrics RSSI (Receive Signal Strength Indicator) which can be measured at any time and LQI (Link Quality Indicator) which is calculated by measuring the first 8 chips of each packet. The CC2420 chip is controlled by the MSP430 micro controller from TI [22] through the SPI it shares with the flash

as well as the external expansion connector. This means that there is a need for careful bus arbitration while using any of them simultaneously. The tmote features an on-board MSP430F1611 8Mhz micro controller from TI with 48KB ROM and 10KB RAM. The 10KB ram, which incidentally is the largest on-chip RAM means that a large amount of data can be kept in the RAM without any writes or reads needing to the flash. As we demonstrate later this capability has influenced our design decisions. There are 8 external ADC ports provided which can be used to collect external signals. A 1MB STM25P80 Flash [21] on the tmote is another extremely useful feature. This can be conveniently used to store data, code and other information permanently on the tmote until the flash gets formatted.

- TinyOS [23] is an open source operating system designed for networked embedded systems or sensor networks. The platform we are using (tmotes) is comparatively new and all the software has been written for mica motes. Even though most of the libraries work for the tmotes too, some of the applications are still not available on the tmote platform. But development is catching up.

1.2.3 Steps in Data Collection

The parent child relationship between the nodes is established by a routing algorithm initiated by the base node once the network is setup. The mote after it receives transmit command, transmits the collected data first to its parent, the parent to its parent and all the way to the base node. This parent selection procedure may have to be repeated a number of times during the lifetime of the network as individual node failures/reboots may disrupt the network topology. As the nodes themselves are stationary the topology is not expected to change much.

All the data is transmitted from all the nodes to the base node using a reliable NACK based transport protocol. A node is not only responsible for its own data but also the data that it has received from its children and children's children (descendants), essentially data of all the nodes that are below it in the network topology. The leaves in the network (nodes which do not have children) are the first to start trans-

mitting to their parent. A node starts transmitting data only after it has received data from all its children.

As soon as data is received at the base node the base node forwards this data to the Soekris (single board computer) connected to it by the SPI interface. As Soekris has much more memory than the data collected by all the motes combined (in case of a small to medium deployment of 10s of nodes), it can store all this data in its memory. Soekris actually runs a scaled down version of the linux kernel and has a PCMCIA (Personal Computer Memory Card International Association) card slot to which a WiFi (Wireless Fidelity, 802.11) card can be attached, thus enabling WiFi based communication. The application design of the entire network is depicted in the Figure 1.5. Soekris then transmits this data to a WiFi device on the moving train. The WiFi device on the train has been put to more use than merely collecting data. It has also been used to wake up the entire setup wakeup using Wake-on-WLan [9]. Intelligent sleep and wakeup schemes allow the wireless sensor network (WSN) deployed on the railway bridge to run for months together on battery power. The WSN wakes up using the Wake-on-Wlan scheme and subsequently collects data and transmits data to the base node.

1.3 Related Work

In this section we describe previous available literature in this domain and the current research going on similar problems elsewhere. We start with a survey of various sensor network applications available in the literature. We compare those with our application and design. We chose 3 applications among the many present to do the comparison. Then we go on to survey the various transport protocols available and reason why they are not a best fit for our application. We do the same with the available routing protocols.

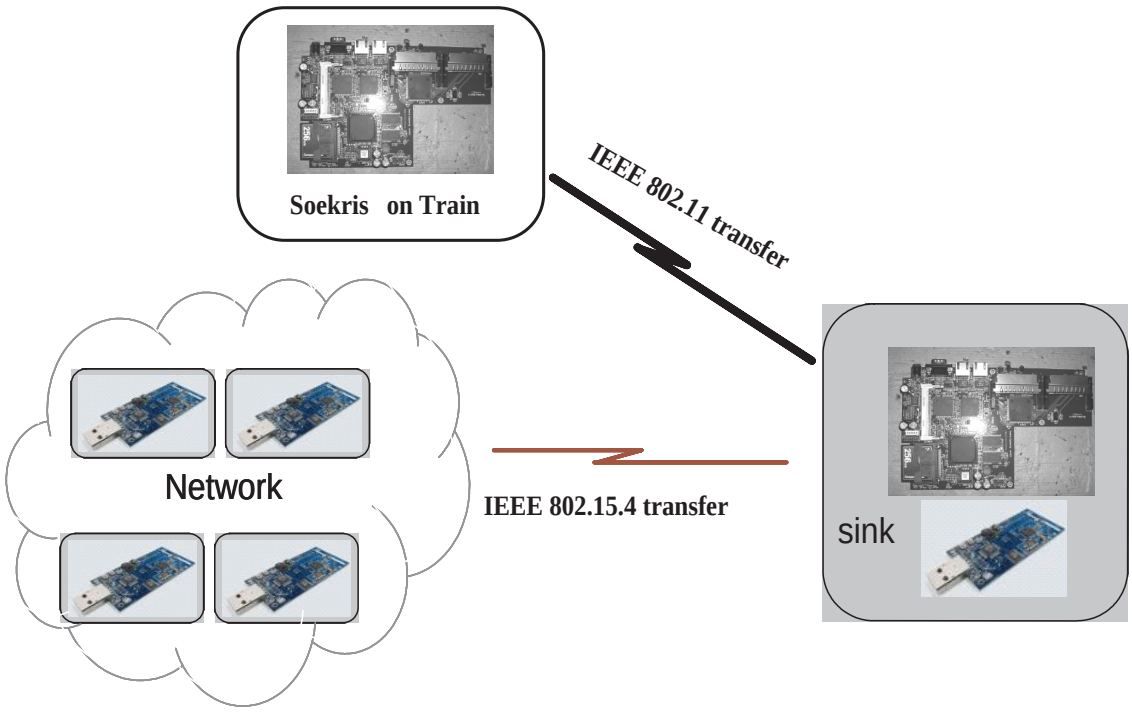

Figure 1.5: Application Design of the Entire Network

1.3.1 Application Design

The application requires to collect vibration data from the accelerometer placed at the piers of the railway bridge. The data collected should be retrieved and available for processing at a central server. The deployment is in a isolated location where there is neither network connection nor power.

Habitat Monitoring [7] is one of the first sensor network deployments. The application aims to monitor the habitat of seabird nesting environments at GREAT DUCK ISLAND MAINE. The current deployment is of 32 mica nodes. The architecture of the application is tiered, the lowest level consists of sensor nodes, which transmit their data to the gateway, which transmits sensor data to the base station through the transit network. The sensor nodes senses temperature, pressure humidity and other phenomena. The data collection is periodic. The deployment is a long term deployment, ran for more than 6 months. Compression techniques have been employed to reduce the amount of data transfer.

WISDEN [30] is multi-hop wireless data acquisition system for structural health monitoring used to monitor large structures. This work aims to replace existing wired systems to collect vibration data from different parts of the structure. It uses reliable data delivery and light weight synchronization mechanisms. As the data requirements surpass the radio bandwidths intelligent compression techniques are employed. The deployment is short term consisting of 14 micaz nodes. The architecture is flat, sensor nodes feeding data to the PC or laptop through the base node. The sensors used are accelerometers and the sampling rate is 200-250Hz.

The deployment of industrial sensor networks at semiconductor plant and north sea [6] aims to use sensor network technology to do predictive equipment maintenance. Further the work aims to produce a cost effective solution. The deployment focuses on vibration analysis for predictive maintenance and uses piezo electric accelerometer to measure vibrations. The work uses PSFQ [25] for reliable data delivery both in single hop and multi hop setting. It uses a hierarchical architecture where sensor nodes feed data to a stargate gateway which feeds to the root stargate which in turn feeds to the internet and to the enterprise server.

The comparison of all the applications explained in Table 1.1

	Habitat Monitoring	WISDEN	Industrial sensor network	BriMon
Duration of deployment	Long term (6 months)	short term	few months	long term
Hardware	Mica2	Mica2 and Micaz	Micaz and Imotes	Tmote-sky
Existing system to be replaced	manual	wired	expensive wireless systems	single hop
Data collection model	periodic	periodic	periodic	periodic and event based
Architecture	Tiered	Flat	Tiered	Tiered
Sensors used	temperature pressure etc	accelerometer	accelerometer	accelerometer
Compression	YES	YES	NO	NO

Table 1.1: Comparison of existing deployments with BriMon

1.3.2 Transport Protocol

We need a transport protocol to reliably transmit large amount of data from each of the nodes in the sensor network to the base station. As the amount of data we are handling is higher than the amount of RAM available in each of the nodes the protocol should tightly integrate with the tmote's flash and handle it intelligently. The fact that the tmote's flash and radio share the same bus, increases both the difficulty and the relevance of this intelligent arbitration of radio and flash. Interleaving both the operations is thus a challenge. Though most of the arbitration is done by lower layers we need to do some careful programming along with exercising caution.

There are a plethora of transport protocols available in the literature. PSFQ [26], RMST [18] , ESRT[2] , CODA [27] , GARUDA to name a few. Our initial analysis rules out a lot of protocols as unsuitable. CODA does not provide reliability, and is thus unsuitable for us. Most of the transport protocols in the sensor network domain were designed for hardware with limited radio capacity, but in the recent times better radio chips have improved the capability leaps and bounds. This call for new and improved protocols which are optimized for the new radio. ESRT is a event to sink reliable transport and provides only event reliability hence is ruled out.

We are left with PSFQ and RMST which provide reliability. RMST was designed for upstream transfers i.e. from each node to the base node, the kind of transfers that happen in our application, but it does not integrate with the flash and no proper implementation and support is available. PSFQ is a protocol basically designed for down stream transfers i.e. from base node to each of the nodes. It does integrate well with the flash but for mica hardware where the flash and the handling with respect to the radio (arbitration) is different. PSFQ has been used in upstream transfers and known to perform non-optimally [6]. Still there is an effort to port PSFQ to tmote and adapt it to our application [16]. So we design a reliable transport mechanism optimal for our application and hardware borrowing ideas from existing literature. A comparison of all protocols can be found here [24]

1.3.3 Routing Protocol

The BriMon application is to be deployed in isolated environments for long durations of time. This warrants a routing protocol to find the parent of each node in order to transmit data to the base node. Like transport protocols there are a lot of routing protocols available in the literature. Many of these protocols are generic protocols which aim to find the route from all nodes to all nodes. But our requirement is to just find the route from each of the nodes to the base nodes.

We consider protocols MultiHopLQI [12], MintRoute [28] and single destination DSDV [13] suitable for our application after initial evaluation. Both MultiHopLQI and MintRoute these protocols use LQI and number of hops as a metric to determine the parent of each node. Single destination DSDV prevents loops by numbering each routing entry, though was not designed for this domain shows some promise with respect to use in our application, but as the case in MultiHopLQI and MintRoute even this protocol does not have a separate routing phase. In all of these protocols all the nodes start transmitting the beacons simultaneously and the routing phase continues through out the network lifetime, which is unacceptable for our application design. We want a separate routing phase, followed by a data transmission phase. The reason for having a separate routing phase is to help us save power in an efficient manner and to avoid interference between data transmission and routing phases. We can however modify these protocols to suit our requirements but as we found in our experiments in section 6.3, LQI is not a good measure of packet loss and packet loss itself is a good measure of itself, we change the metric to packet loss from LQI and create a new routing phase in our protocol.

1.4 Thesis Contributions

We would like to mention that we use the synchronization mechanism and a few other hardware components from the master's thesis of our colleague [8]. Our major Thesis contributions can be detailed as follows.

1. Analysis of the Bridge Monitoring application (BriMon) and designing an optimal solution to the problem. We employ a application specific approach, which helps us to separate out actual concerns from probable concerns.

2. Design and implementation of an application specific Routing protocol for the BriMon application.

3. Design and implementation of an application specific reliable transport protocol for the BriMon application. We design and implement a NACK based reliable transport protocol to transfer large amounts of data (100s of KB).

4. Integration and testing of the application, routing and transport protocol for large amounts of data transfers and topologies probable in the application setting. The application is supposed to work without any glitches for months together, hence this type of testing plays an important role in the success of the application.

5. Fabrication of attenuator circuit to make sure that the output voltage level of the accelerometer is within the range of the tmote.

1.5 Thesis Organization

The rest of the report is organized as follows. Chapter 2 gives the application design of the solution. Chapter 3 describes all the hardware components we had to design. Chapter 4 describes the reliable transport mechanism and Chapter 5 explains the routing protocol. In Chapter 6 we detail the various experiments conducted and their results. In Chapter 7 we conclude and describe the future work needed in brief.

Chapter 2

Application Design

In this chapter we explain the application Stack, the various components and their interfaces we have designed. Here we give a bird's eye view of the entire application before proceeding to give a much more detailed description. The series of events leading to collection of vibration data are as follows (Figure 1.1 in previous chapter).

1. The train (engine) contains a powerful radio (IEEE 802.11), and transmits beacons continuously as it arrives near the bridge. The sensor mote (mote A) which is kept at a distance of d2 from the bridge in the direction of the arrival of the train wakes up on sensing the channel on which the train is transmitting beacons using Wake-on-WLan technology [9]. This mote does a duty cycling between sleep and wake-up. This means that its sleep and wake up time have to be such that it positively detects the arrival of the train [9, 8]. Now mote A wakes up mote B (base node) by transmitting 802.15.4 packets, which in turn wakes up other sensor nodes similarly by transmitting wakeup packets. The wake-up mechanism is depicted in Figure 2.1.

2. The sensor mote connected to the soekris (base node) is responsible for powering on the soekris board as well as the other nodes deployed on the piers of the bridge.

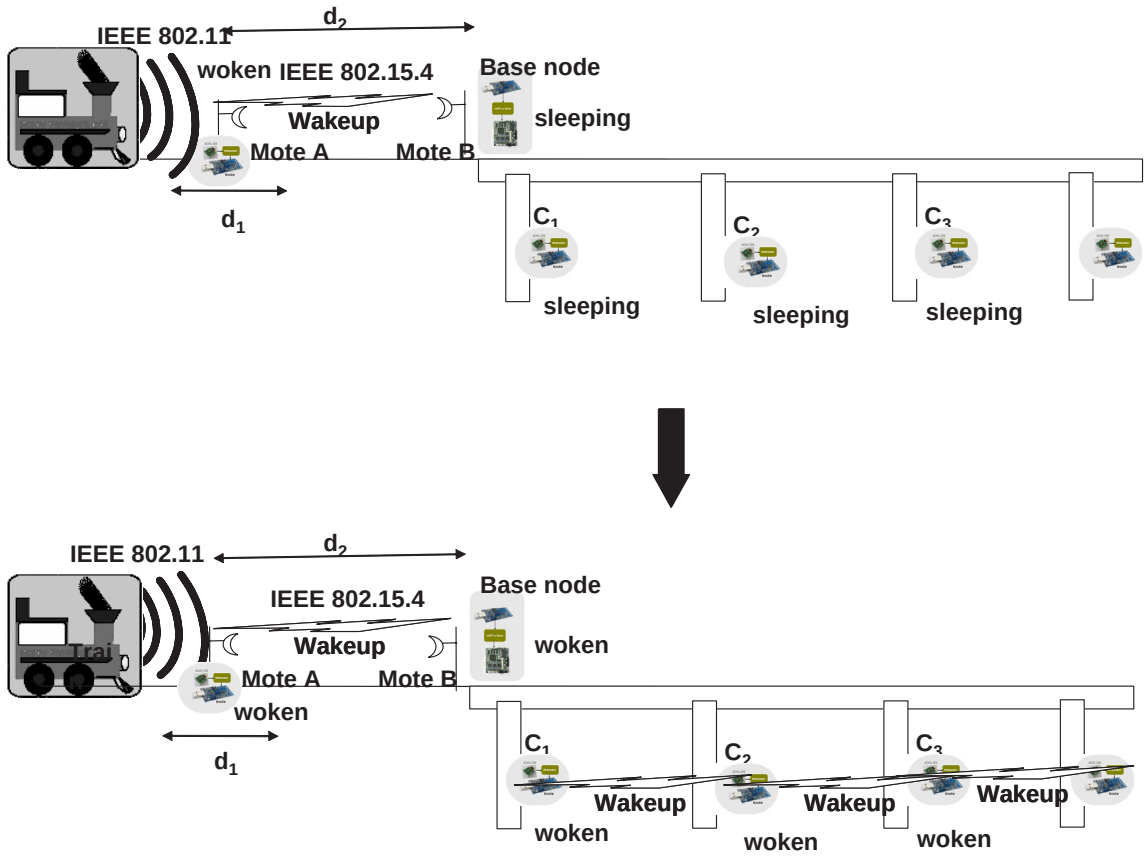

Figure 2.1: Wake up mechanism for the network

3. These sensor nodes collect vibration data from the tri-axial accelerometer and transmit it to the base node through the multi-hop wireless sensor network.

4. The base node (sensor node) transfers this data to the soekris. The soekris transmits data pertaining to previous passage of the train to soekris sitting on the train.

5. The data collected at this point of time is transmitted to the next train passing by. The train which has collected the data from the sensor network, ultimately feeds the backbone at any of the stopping stations. The data is either available online or made available for further processing.

The organization of the rest of the chapter is as follows. We start by introducing how data collected by each sensor node is organized and the way it gets transmitted.

Then we go on to explain the time line of the network followed by the explanation of interaction between various components with special emphasis on the layering of the components that so much enhances the design.

2.1 Data Organization

The solution employs a Wireless Sensor Network (WSN), each of the node in the WSN connected with a tri-axial accelerometer. Every node collects vibration data (accelerometer readings) as the train passes over the bridge. The setup (accelerometer and sensor network) is responsible for measuring 3 kinds of vibrations as already stated.

- Ambient Vibrations (Induced by wind)
- Forced Vibrations (Induced by Train passing on the bridge)
- Free Vibrations (After the train has passed the bridge)

It is important to know the exact amount of data that each node will collect.

	Symbol	Units
Let the speed of the train passing over the bridge be	V	Kmph
Let the length of bridge be	L	meter
Let the length of the train be	T	meter
Let the length between each pier of the bridge be	S	meter
Number of piers in the bridge	n	-
Number of samples per second (Sampling rate)	r	per second
Number of Axes	a	-
Size of each sample	b	bytes

Table 2.1: Constants for calculation of amount of data collected at each node

- The amount of time the train remains on the bridge

$$t = \frac{(L+T)}{(V*1000)/3600} seconds \tag{2.1}$$

- The total amount of data

$$D = (r * a * b * t) \tag{2.2}$$

Now we shall plug in the values of the various parameters to calculate the actual amount of data generated at each node. Refer Table tab:datacal

Symbol	Value	Units
V	60	Kmph
L	600	meters
T	500	meters
S	80	meters
n	7	-
r	100	per second
a	3	-
b	2	bytes

Table 2.2: calculation of amount of data collected at each node

Here we provide a justification for the values in Table 2.2. In India, trains run at an average speed of 80-100 kmph (fastest trains), but their speed is lesser when they pass on some bridges, so we have taken the value 60kmph. The length longest railway bridge in India is 3.06km, but a lot of bridges are in the range of 0.5 to 2km. The length of a train is around 500m (30m per coach and 20 coaches). We need to sample at a rate of 100 samples for second to get useful data about the vibrations and all 3 axes are necessary.

- The amount of time the train remains on the bridge (Substituting in equation 2.1)

$$t = \frac{(600 + 500)}{(60 * 1000)/3600} = 66 seconds \tag{2.3}$$

- The total amount of data (Substituting in equation 2.2)

$$D = (100 * 3 * 2 * 66) = 39.6 KB \tag{2.4}$$

The data collected in each node is around 39.6 KB, but the tmote has only 10KB RAM. But the RAM is also used for storing other local variables and arrays,

so the effective RAM available for actual data storage will be around 6-7KB (in our present implementation). So how do we store 39.6 KB of data in 6-7KB. The answer lies in the huge flash that the tmote features. So we create an abstraction called a `File`.

A `File` is an abstraction created to handle the amount of data generated at a single node, which is greater than the amount of RAM available. The size of the File is fixed in a network. All data generated at a node is divided into fixed number of files. The number of files at all nodes is the same because the amount of data generated for an event at every node is the same. The File has the following attributes.

- The node id of the node which originally collected the data.
- The file no within that node.
- Start address of the file in the flash.
- Size of the file (for convenience we keep the size of the file constant, this does not affect the performance).

We divide all the data node collects into files, the size of the files is small enough that files can be easily stored in the RAM. But these files need to be transferred to the flash before the arrival of data for the next file. The files are transfered one file at a time to the next node (parent node) in the network. The transfered file is stored in the flash before the next file is transfered. So in this way we store and transfer data in the form of files. But we store all the properties of the File in the RAM.

Once the data is stored on the flash as files , data is also read as files and transmitted to the lower layer (transport) as a buffer to be reliably transmitted to the next node in the network as packets, where the packets are then reassembled and reconstructed as a buffer in the transport layer, which is then passed as a file to the application layer, where it is again stored as a file in the flash. This node is now responsible for transmitting not only its file but also the file that it has received from its children. Consider a 7 node network arranged as a temple topology with the base node and nodes numbered 1-6. Consider that the entire data at each node is divided

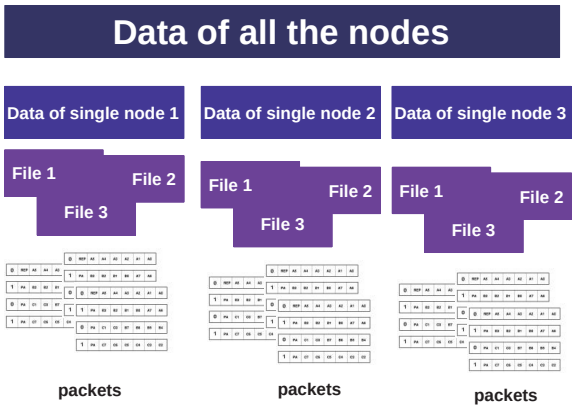

Figure 2.2: Organization of data in the network

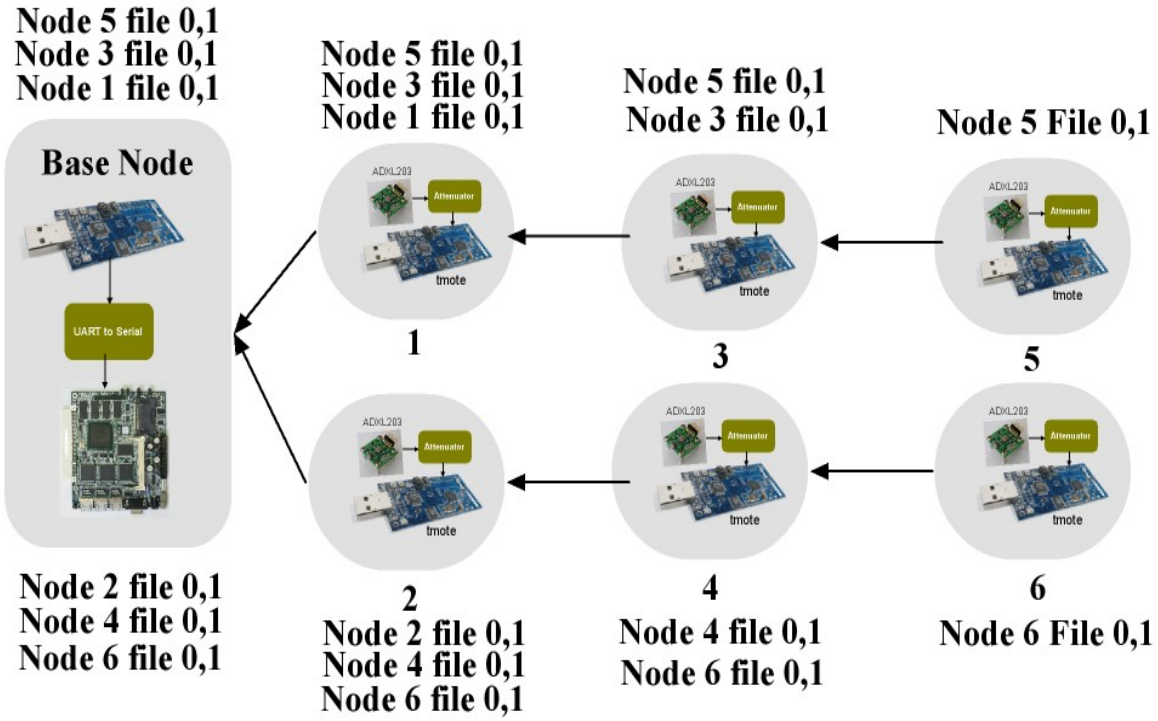

Figure 2.3: Data transfer within the sensor network

into 2 files File 0 and 1 of equal size. The organization of data is depicted in the Figure 2.2. Figure 2.3 shows which nodes are in possession of what data at the end of the entire transfer.

2.2 Time line Diagram of the Network

We have described how the data is organized within the network and how this data gets transmitted to the base station and ultimately to the central server for processing. In this section we describe the time line of the entire network describing the status of the network at various time instants.

All the nodes in the network run the same code except the base node. The reason the base node runs a different code is because we experienced arbitrary packet errors and packet truncation while running the same code as that run on other nodes. So we had to do a lot of testing and implementation changes to work around the problem. So when data was sent at high rates to the base node we experienced arbitrary packet errors and packet truncation. We could not exactly pinpoint the error to a particular component but the problem was solved by using the radio component one layer below the one used in previous implementations. All nodes except the base node runs layered code where as the base node runs monolithic code due to the reason stated above. The same problem has been faced by few others, who are working in this domain [15, 20].

Figure 2.4 depicts the timeline of all nodes in the network except the base node. All nodes when they boot up, undergo some initialization (initialization of states , variables and formatting the flash) and then they duty cycle between sleep and wakeup.

The nodes do a duty cycling between sleep (t_s) and wakeup (t_w). At the receipt of a command from the Base station all the nodes switch on their respective sensing circuitry and start collecting data (t_c). The circuitry consists of an triaxial accelerometer and the corresponding attenuator circuit. The details of the attenuator circuit

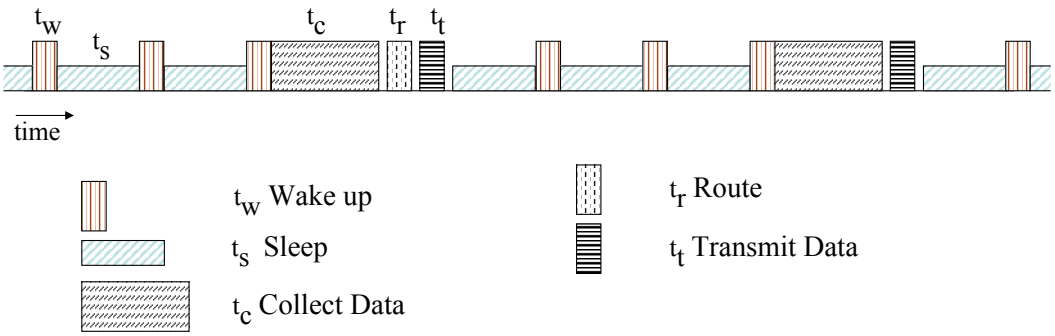

Figure 2.4: Timeline diagram of the wireless sensor network

are described in section 3.3. The switching circuitry is responsible for powering on and off the accelerometer and the attenuator. We need a switching circuit because unlike the tmote the rest of the circuit cannot be powered off by a command and the rest of the circuitry draws as much, if not more current. After the motes have collected data , they transmit the data during the transmit phase (t_t) . The routing phase can happen at anytime of the lifetime of the network but before the data transmission phase (t_r). The Routing phase has been shown to happen just after the data collection phase in the diagram for convenience. After the node has transmitted its data to its parent it can go back to its duty cycling until it receives any other command from the base station. The gaps between (t_t) and (t_r) indicate that the transmit phase and routing phase may have a small amount of time lag. Similarly there routing phase may not start immediately after data collection phase and the nodes may wait for some time before going to sleep after transmission is completed.

2.3 Component Interaction

In this section we describe the various software components and the interaction taking place between them for the application to work. We also explain the layering in the stack. Instead of showing all the components and their interactions in a single diagram we make use of multiple diagrams for reasons of better clarity.

Here we would like to mention that we cannot have more than one transfers to the same destination at the same time. Such transfers according to our experiments would lead to unacceptable packet losses for both the transfers. So if more than one node is requesting transfer to a particular destination, one of the nodes is asked to wait while the other transfers the data. The waiting node resumes its transfer later. This mechanism is achieved through Wait To Send (WTS) messages, described in chapter 4 (Transport Protocol).

2.3.1 Layering

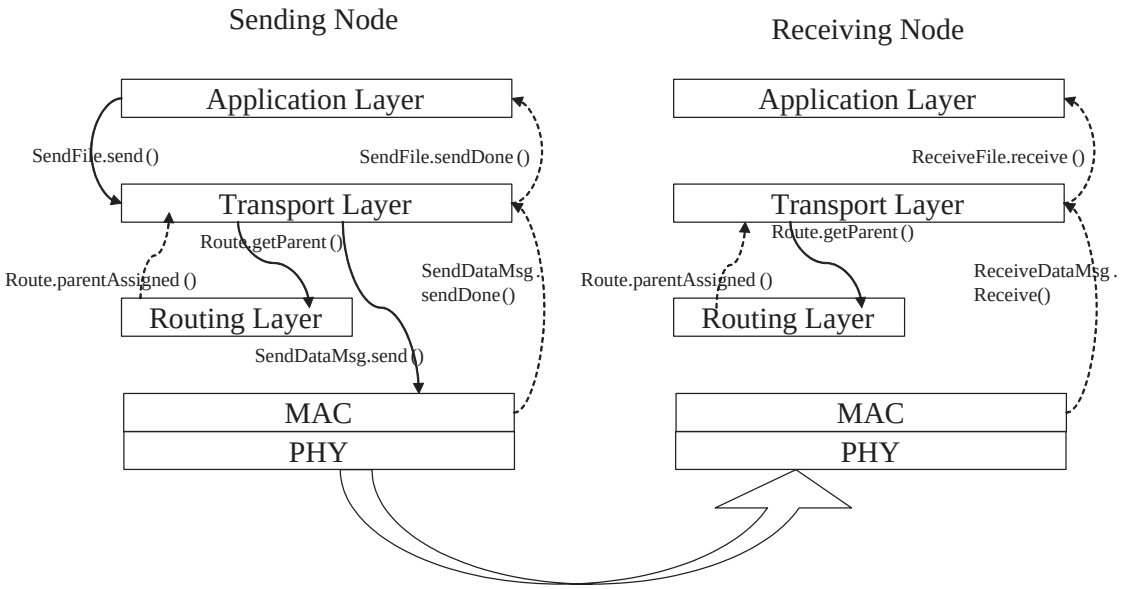

Figure 2.5: Application stack of a node sending a file from one node to another

The entire application (Figure 2.5) is developed as layers (or components) interacting with one another. The diagram shows the interaction of application, transport and routing Layers to transmit a single file from one node to the other node. Implementing functionality in the form of layers is a natural choice for protocol development in the networking domain. But we have an additional reason to use layering. TinyOS and NesC embodies the concept of implementation and interfaces. It not only provides extensive support for definition and implementation of interfaces, it also widely encourages the use of it. In a layered stack like ours the lower layer implements

and provides a well defined interface and the upper layers use these interfaces to implement other functionality. For example the transport layer provides the interface `SendFile.send ()`, which the application layer uses to split the entire data set to files and transmit it reliably through the transport layer. Similarly the routing layer handles the selection of next hop for data transfer which the transport layer uses to transmit a file to the base node. The MAC layer handles the sending of individual packets which again the transport layer uses.

The application layer issues a `SendFile.send()` command to the transport layer when it gets a transmit message from the base node through the other nodes. Note that only the leaves in the network start transmission, while the other wait to receive transmission from their children and then they start their transmission after they get data from their children. The transport layer breaks down the file into packets and transmits the packets to the receiving node. The transport layer employs a reliable NACK based mechanism, where NACKs are sent for the entire file and not as soon as packet loss is detected. The transport layer gets the parent address to which it has to transmit the packets from the routing layer by issuing a `Route.getParent()` command. As soon as the receiving node receives all the packets (after reliable NACK based mechanism), the transport layer signal the application layer using `ReceiveFile.Receive()`, indicating that the receiving node has received a file from the sending node (Figure 2.3). Now the receiving node is responsible for sending this file and all the files that it has received from its child (sending node), along with the files with respect to the data collected from its own accelerometer forward along the network to its parent. Then that parent is responsible for sending the data to its parent and so on till the data reaches the base node.

At every node the parent is queried from the routing layer by issuing the command, `Route.getParent()`. For a node to query its parent from its Routing layer, the precondition is that the node should have the knowledge of its parent. To make sure that a node does not get an invalid parent , the transport issues a command `Route.isActive()` to find out whether the route is active or inactive. The other mechanism that helps is that the Routing layer itself signals the completion of route assignment by sending `Route.parentAssigned(isLeaf)`, telling the transport and subsequently the application that whether the corresponding node is a leaf or not. It

is important for a node to know whether it is leaf or not because it is the leaves in the network that are responsible to start the transmission of data, the other nodes start transmitting data once they get data from their children. The mechanism has been shown in Figure 2.5.

2.3.2 Components

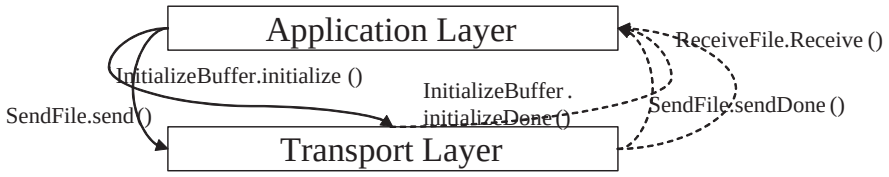

Figure 2.6: Interaction between application and transport layer

- Figure 2.6 describes the interaction between the application and the transport layer (component). The application layer is responsible for collecting data, storing it in the flash , reading it from the flash , dividing it into files and sending it to the transport protocol for reliable transmission. The application layer uses the command `InitializeBuffer.initialize()` to initialize the buffer at the transport layer to its buffer so that it can seamlessly pass data to the Transport layer and also to pass other initialization data. The transport layer sends a signal `InitializeBuffer.initializeDone()` to trigger the success of initialization. The Application layer uses the command `SendFile.send()` to send a file to the Transport layer which in turn signals a `SendFile.sendDone()` to notify the success of send. Whenever the transport layer receives a file it notifies the application layer by signaling a `ReceiveFile.receive()`

- Figure 2.7 describes the interaction between the application and the flash-HAL (Flash Hardware Abstraction Layer) and the application layer and the AccelerometerHAL (Accelerometer Hardware Abstraction Layer). The Flash-HAL is basically a library provided by the TinyOS distribution to write and read to the flash. The write to the flash is accomplished using the command `BlockWrite.write()`, passing data, length and addr in the flash to write to,

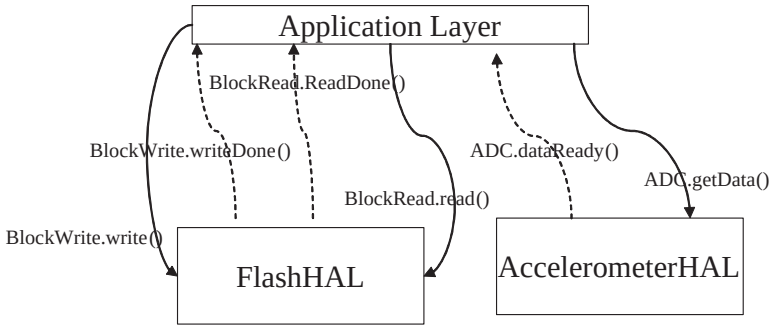

Figure 2.7: Interaction between application Flash and the Accelerometer

which in turn triggers a `BlockWrite.writeDone()`. Read is similarly done using `read()` and `readDone()` . The AccelerometerHAL abstracts the accelerometer Hardware. To get data the application uses the command `ADC.getData()`, the AccelerometerHAL in turn signals a `ADC.dataReady()` giving the application layer with ADC data two bytes at a time (because it is a 12 bit ADC).

- The application design is such that there can be only one transfer happening at a time to a node. This is because having multiple transfers to a single node creates enough interference that the resulting packet loss becomes unbearable for the application. As a result of having only one transfer at a time the latency of transfer increases, which is acceptable as far as our application is considered as long as only the nodes which are doing the data transfer are awake and the rest of the nodes are in low power mode, in turn saving power and increasing the lifetime of the network.

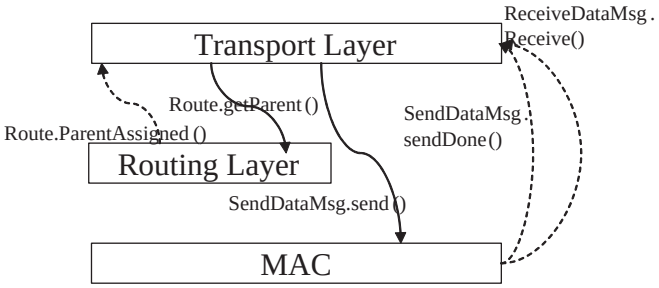

Figure 2.8: Interaction between transport layer, Routing and MAC

- The interaction between the transport, routing and MAC layers are depicted in Figure 2.8. The transport Layer's responsibility is to reliably deliver the file

passed to it by the application layer to the Transport Layer of the node's parent. Routing layer handles the determination of the route for every node to the base node, simply put it determines every node's parent. The interaction between the routing, transport and MAC have already been sufficiently described in the previous sub-section .

We have explained various aspects of the application design, a bit of hardware design, component interaction, layering and the timeline diagram of the network. In the coming chapters we would explain each of these in detail.

Chapter 3

Hardware Design

In this chapter we describe the design and fabrication of hardware components as required by our application. We have built most of our hardware on PCB's. Our motive was not to create reliable and sound hardware but to develop a prototype as quickly as possible. We would like to mention that though we explain all the hardware components here for the sake of completeness, some of the hardware components have been designed as part of a colleague's thesis [8]. The organization of the chapter is as follows. First we explain the various parameters that influenced our choice of an accelerometer chip then we go on explain the fabrication of the application circuit based on the chip. The design of the attenuator circuit is explained in the next section followed by the design of the UART to SPI circuit. The design of the switch circuit and the power circuit follows next.

The prototype of the base node and the ordinary nodes are shown in Figure 3.1 and 3.2 respectively. The photograph of the base node shows a tmote connected to a soekris through a UART to serial converter. This setup is used to transfer data from the tmote to the soekris for further data transfer (using WiFi) to another soekris sitting inside the train. Both the soekris and the UART to serial converter are controlled by the switch circuit connected to the tmote. This setup is required because the data rates of the tmotes is limited and we use WiFi capability of soekris for

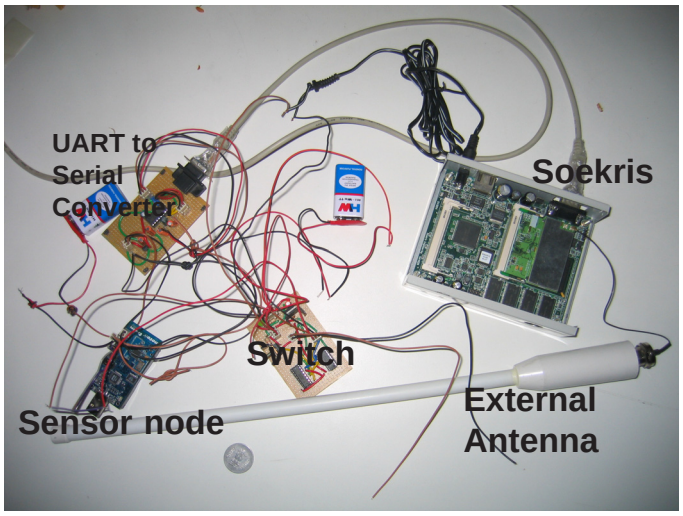

Figure 3.1: Circuit of Base Node developed by us

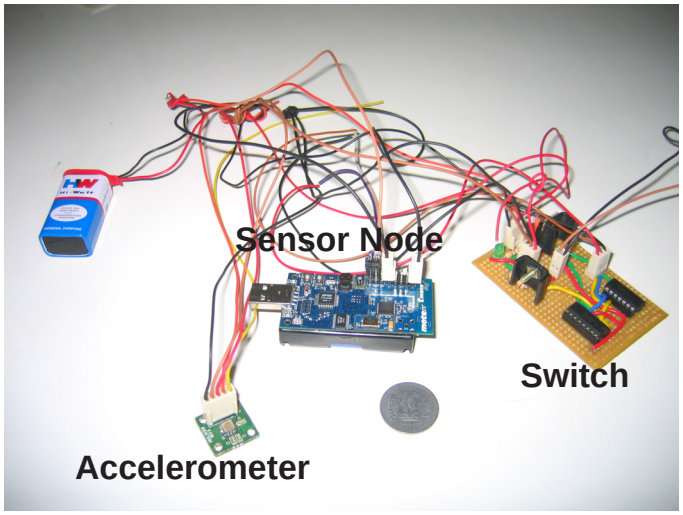

Figure 3.2: Circuit of Node developed by us

transferring aggregated data of the sensor network. In the photograph of an ordinary node an accelerometer connected to the tmote can be seen. The accelerometer is controlled by the switch circuit connected to the tmote. The switch circuit is used to power on and power off both the circuitry to operate in low power mode. The switch circuit is used to power on and power off both the circuitry.

3.1 Choice of Accelerometer Chip

The choice of the accelerometer is crucial to the success of the application. Accelerometers are used for various applications ranging from mobile phone handsets, hard drive protection systems, safety critical automobile systems to monitoring health of structures. The most important parameters of the accelerometer that impact our application are as follows.

1. The number of axes along which the accelerometer can measure the acceleration is important. Accelerometers are available which can measure single axis, dual axis and triaxial accelerations. We would ideally prefer a triaxial accelerometer which can monitor all the axes namely X, Y and Z for detailed analysis of the vibrations of the bridge.

2. The resolution of the accelerometer defines the lowest unit of acceleration that can be measured by the accelerometer. The lower the resolution the better. A 1mg resolution accelerometer can measure accelerations as low as 1/1000th of a g (1g=9.8m/s^2).

3. The sensitivity of an accelerometer is the magnitude of the output voltage per g of acceleration. A sensitivity of 800mV/g means that that the output of the accelerometer increases by 800mV with every 1 g increase in acceleration.

4. The noise performance of an accelerometer is extremely critical. It determines the amount of noise that get added per $\sqrt{Hz}$. Noise of up to 350μ g/$\sqrt{Hz}$ rms is bearable for our application [14].

5. The range of acceleration is the maximum and minimum acceleration that an accelerometer can measure. We need an accelerometer that can measure from -2g to +2g. The bandwidth of the accelerometer defines the range of accelerations that an accelerometer can measure. The application of monitoring of huge structures like bridges requires the accelerometer to be able to measure very low frequencies (high time periods)[14].

6. The supply voltage of the accelerometer should be low as we are using it for low power application where battery power is a major constraint. Furthermore the current consumption should also be low.

7. The output voltage of the accelerometer at 0g determines whether we require an additional circuitry to adapt this voltage to that of the mote's. The tolerable voltage of the mote is 0-3V so the sum of voltage at 0g and the additional voltage at 1g should neither be greater than 3v nor lower than 0V.

Parameter	ADXL203	MMA7260Q
Company	Analog Devices	FreeScale Semiconductors
Package	8-Terminal Ceramic LCC	16-pin QFN
No of axis	2	3
Resolution	1mg	-
Sensitivity	1000mV/g	800mV/g
Noise performance	$110\mu g/\sqrt{Hz}$	$350\mu g/\sqrt{Hz}$
Bandwidth	0.5-2500Hz	XY-350Hz Z-150Hz
Acceleration range	$\pm$1.7g	$\pm$1.5g, $\pm$2g, $\pm$4g, $\pm$6g
Supply voltage	5V	2.2V-3.6V
Output voltage at 0g	2.4V-2.6V	1.485V-1.815V
Current consumption	700μA	500μA

Table 3.1: Comparison of accelerometers chosen by us : ADXL203 and MMA7260Q

We finalized on two accelerometer chips based on the above said parameters, ADXL203 [1] and MMA7260Q [10]. The comparison between these chips is given in Table 3.1.

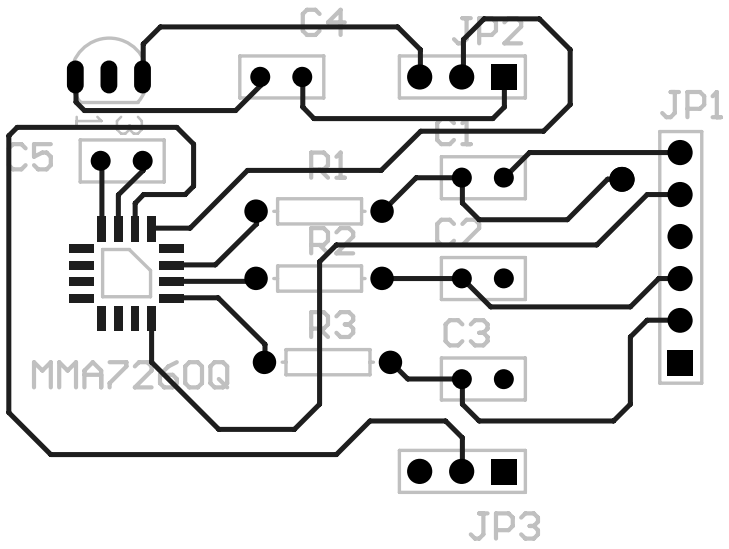

Figure 3.3: Application circuit for MMA7260Q accelerometer chip

3.2 Design of Application Circuit for Accelerometer

We have used an evaluation board for ADXL203 chip while we have tried to fabricate the application circuit for the MMA7260Q. The design of the circuit for MMA7260Q was done using protel trial version. One of the difficulties we faced was getting the proper package (16-pin QFN) for the chip so we ended up creating the package itself before using it in the application circuit. A package basically consists of the schematic and the footprint of the chip on the circuit board. The application circuit is given in Figure 3.3. We have designed the application circuit but the fabrication of the circuit is pending.

The physical layout of the evaluation board we are using is reproduced (Figure 3.4) as it is given in the data sheet [1].

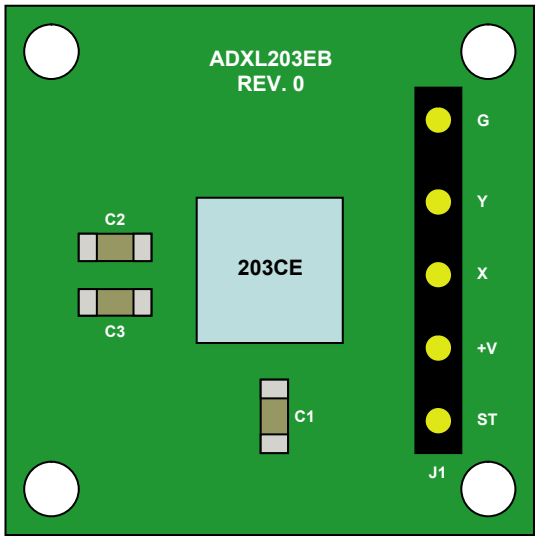

Figure 3.4: Physical layout of the ADXL203 Evaluation Board

3.3 Design of Attenuator Circuit

The response from the accelerometer is in the range of 1.5V to 3.5V. The output voltage at 0g acceleration of the accelerometer is 2.5V and there is an additional 1V voltage increase for every 1g of acceleration. So if we expect the structure to have maximum acceleration of at least +/- 1g, the output range of the accelerometer will be 1.5 to 3.5 V. But the tmote ADC (analog to digital converter) is capable of withstanding voltage from 0 to 3V only, so we use an attenuator circuit to reduce the output voltage of 1.5 to 3.5 V to 0.5 to 2.5 V which is a safe voltage level for the tmote ADC. The application circuit is available at [3](section 3 differential amplifier). $V_{output} = R_2/R_1(V_2 - V_1)$. Here V_2 = signal from accelerometer, V_1 = 1V, R_2 = 1ohm, R_1 = 1ohm. Since signal ranges from 1.5V-3.5V thus using formula we get V_{output} in the range 0.5V-2.5V.

3.4 Design of Circuit to Interface Mote with Soekris

Our application design demands that we use a soekris single board computer along with the tmote, both talking to each other. The tmote has a easily usable USB

interface, unfortunately the soekris does not have a USB port but only has a serial port which comes close to what we want. We even tried to use a USB to serial converter. But somehow it did not even work on a windows machine and we needed it to run on a scaled down version of a linux running on soekris. The reason as we later hypothesized may have to do with the right driver not getting installed or the FDTI chip on the mote which is responsible for talking to the USB interface not getting powered on as the USB to serial converter does not give any power at the USB end. So we had no choice but to use the UART of the tmote and convert it into serial port interface. We could not use the RS232 chip because the tmote UART works at 3.3 V voltage level so we at last chose the ST3232 to interface the tmote with the soekris. Details can be found in [8].

This setup is used to transfer data from the tmote to the soekris for further data transfer (using WiFi) to another soekris sitting inside the train. Both the soekris and the UART to serial converter are controlled by the switch circuit connected to the tmote. This setup is required because the data rates of the tmotes is limited and we use WiFi capability of soekris for transferring aggregated data of the sensor network.

3.5 Design of Switch circuit

The entire setup the sensor network consisting of wireless motes and the soekris board is supposed to be deployed on a battery where no continuous power source is unavailable. The sensor nodes (tmotes) can be switched to low power mode to save battery power but the rest of the circuitry (accelerometer, soekris , attenuator) have to switched off using an additional circuitry. We use chips 4069 and 4027 to achieve the required functionality. Details can be found in [8]

3.6 Design of Power Circuit

The various components described above run on battery power. Each of these components run on different operating voltages. But we cannot have more than one battery on a deployment, so there is a need to design a power circuit which takes input from a single battery source and feeds to different circuits handling their voltage and current ratings. The Table 3.2 lists the different hardware components and their respective voltage ratings. Details can be found in [8].

Hardware Component	Voltage Rating	Chip Used
Soekris	5-15 V	-
Tmote-sky	2.7-3.3V	LP2950
Accelerometer ADXL203	5V	7805
Switching circuit	5V	7805
UART to Serial converter	5V	7805

Table 3.2: Voltage Rating of various components

Chapter 4

Transport Protocol

4.1 Overview

In this chapter we describe the reliable transport protocol we have designed and implemented to be used for transmitting data from each node to the base node. We use a store, wait, and forward technique for data transfer. It is quite different from store and forward in the sense that a node after it receives data from a child node stores the data and waits from time T_w to hear transmissions from any other child node which may have been delayed. We make use of the RAM as well as the flash to store data in the intermediate nodes. As described in the earlier chapter the data collected by a single node (mote) is much larger than can be stored in the RAM of the mote hence the data is divided into files, small enough that a single file can be easily stored in the RAM. Each of these files are then transmitted one by one to the parent and then they are stored in the flash, until it gets forwarded to the base node, where it gets forwarded to the Soekris board though the SPI interface shown in Figure 1.3 .

The typical topology in the BriMon application is shown in the Figure 4.1. We implement robust hop by hop reliable mechanisms and do away with end to end reliable mechanisms for performance reasons. A end to end reliability adds additional

overhead to the protocol and decreases performance. Consider a network with 3 nodes, node0, node1 and node2 where node2 is trying to transmit data reliably to node0. Consider that the medium is very lossy and packet loss rates of 50% are not uncommon. If we had a end-to-end reliability mechanism node0 would request node2 for its packets and the transfer would happen again through node1. So all three nodes are awake, which reduces the network lifetime. But in case of hop-by-hop reliability node1 makes sure it has all packets of node2 before beginning transmission to node0. So only 2 nodes are awake at a time. The case is severe in case of long linear topologies, which are common in our application.

A hop-by-hop reliable mechanism implies that every node is responsible for the reliable delivery of not only its own data but also the data of its children and its children's children and so on till the leaf is reached. So leaf nodes are responsible for transmitting its data to its parent only. In this case nodes 6 and 7 transmit their data (only their data) to nodes 4 and 5 respectively. But node 4 has to transmit not only its data but also the data that it has received from node 6 to its parent i.e. node 3. A similar procedure applies for node 5 also.

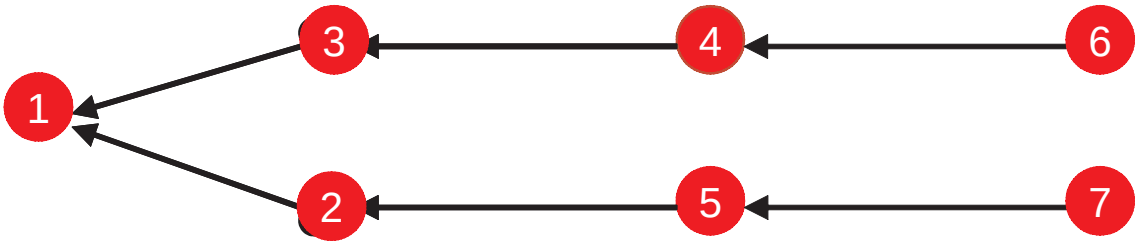

Figure 4.1: Temple Topology : Typical Topology specific to the application

4.2 Protocol Description

In this section we describe the details of the transport protocol, explaining exactly how the transmission happens. We emphasize on the reliability of transfer detailing the actions taken by the protocol from various errors and packet losses. Let us assume for the rest of our discussion that nodeS is transmitting a file to nodeD. We explain

the scenario during the transmission of a single file between 2 nodes, most of what is explained here applies to all transmissions that happen.

The Header and Tail messages act as delimiters to the file and data message carries the actual payload. The ACK message is sent by the receiving node to the sending node to notify the receipt of the entire file. The receiving node sends NACK messages in case of packet loss, upon receipt of NACK messages, the sending node sends RETR messages (retransmissions). The WTS Message is used to stop a node from transmitting when the receiving node is busy receiving data from another node.

We use a NACK based reliable delivery mechanism. The reason for using a NACK based mechanism instead of ACK based mechanism is that, NACK based mechanisms perform better in environments where packet losses are high. This means the number of packet transfers that need to happen to get data to the destination is less in the case of NACK based mechanisms compared to ACK based mechanisms. This is because there is an extra packet transmission for every packet received in ACK based mechanisms but in the case of NACK mechanism a packet may be transmitted for n number of packet losses. If for instance ACKs are bundled the number of transmissions will be higher because the entire bundle has to be retransmitted even if one packet gets lost, which is not the case in NACK based mechanisms where individual retransmissions are possible.

Here we explain the need for the WTS message. Consider the scenario when 2 nodes node2 and node3 are trying to transmit to nodeD simultaneously. If both are allowed to transmit simultaneously packet losses would be unbearable. So nodeD arbitrates and asks one node (node3) to stop transmitting through the WTS message which is a Boolean indicating whether a node should transmit or not. Once node2 is done with its transmission it asks node2 to resume transmission again through WTS message. Node3 can go to sleep all this while and be on a sleep wakeup duty cycle.

Consider nodeS (source) is sending data to nodeD (destination) and nodeS has partitioned that data into n number of files, all data (files is initially stored in the flash). Here we go through the basic mechanism before delving into the details. Now nodeS first reads the first file from the flash making it available in the RAM and then sends the file (as packets of course) to nodeD. If nodeD does not receive all the packets

it sends NACKs for all the packets not received to nodeS. Each NACK packet carries NACK for NACK_MSG_LEN packets. NodeS after receiving NACKs retransmits the corresponding packets. This process can continue till the maximum number of retransmissions are done and then the nodeS gives up and sends the tail message indicating that the file transfer is complete. This maximum number of retransmissions is a configurable parameter. On the other hand if nodeD receives all packets sent by nodeS , nodeD sends an ACK to nodeS. Upon receipt of the ACK nodeS, reads the next (second) file from the flash and prepares to send the same to nodeD.

Now we start explaining the protocol in detail. NodeS first sends nodeD the header message notifying nodeD of the details of the file transmission. The header message is basically a request for transmission. The header message is sent by nodeS to nodeD to inform nodeD that a transmission of a file is going to begun. NodeS informs nodeD about its identity (from_addr), apart from the details of the data (file) it is going to send. The details of the file such as the node to which the data of the file belongs, the sequence number of the file, the number of packets, the starting Sequence no of the packets and the number of bytes of data it is going to transmit is also transmitted. If nodeD is ready to accept the transmission, then nodeD sends a header ack message which is exactly same as the header message it received except that it set the is_ack field. After nodeS receives Header ack message from nodeD it starts transmitting data packets at a rate of one packet every T_d seconds. Actually T_d can be varied or we can also transmit at the maximum speed (in TinyOS terms sending a packet as soon as `sendDone` is called) possible without considering any specific interval. Actually the this maximum speed is around 10ms so we use T_d = 10ms in our implementation.

After all data messages are transmitted nodeD sends an acknowledgment message to nodeS. Now nodeS sends Tail message to nodeD. After nodeS receives Tail ack message from nodeD , nodeS repeats this process for other files if it has any other file to transmit. If nodeS does not have any more file to transmit, nodeS notifies this to nodeD by setting the `is_transmitDone` field in the Tail message. The reason we need a tail message is not very obvious, it seems the header message can very much replace the tail message, but the case is not so. Consider the scenario when nodeS is transmitting the last file to nodeD and it is facing heavy packet loss and is done with

maximum number of retransmissions, so nodeS needs to tell nodeD that it is done with its retransmissions even though nodeD does not have all the packets. Now it cant send a header message as it does not have any other file to transmit, so we use a tail message for this functionality.

These sequence of events occur when everything goes right and there are no packet losses but the wireless medium is lossy and packet losses are bound to occur. So we shall examine various cases when some of the packets get lost. This scenario is depicted in Figure 4.2.

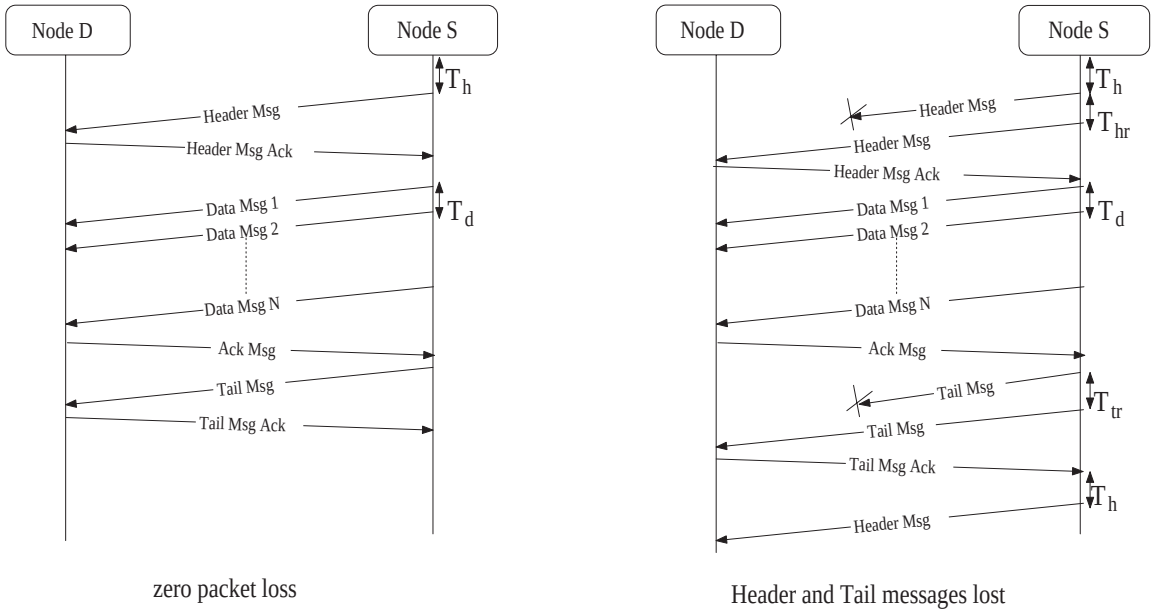

Figure 4.2: Description of Transport Protocol

Let us consider that in case the header message sent by nodeS gets lost and does not reach nodeD. Now nodeS starts a timer T_{hr} just after it has sent a header message, if it does not get a header ack before the T_{hr} timer expires, it triggers a retransmission of header message. The timer T_{hr} has to be greater than 1 round trip time of a packet. In case a Tail message gets lost the sequence of events that leads to a recovery is exactly similar to that described. This scenario is depicted in Figure 4.2.

If a data packet gets lost, the receiving node nodeD in this case stores the seqnos of the packets that it has not received, and once it receives the last packet, nodeD

starts transmitting NACK messages with the seqnos of packets it has not received as payload. Each NACK packet carries sequence numbers of NACK_MSG_LEN packets. After nodeS receives the final NACK packet it starts retransmission of lost packets at the rate of 1 packet every T_{retr} milliseconds. The final NACK packet is determined by a boolean field `fin` in the NACK packet. Usually T_{retr} is the same as T_d. After nodeD receives all packets it sends an ACK to nodeS. This scenario is depicted in Figure 4.3.

If the last packet gets lost then nodeD does not transmit neither a NACK nor a ACK. So the source nodeS starts a watchdog timer as soon as it transmits the last packet T_{lst}. If T_{lst} expires and it has received neither a NACK or an ACK it retransmits the last packet.

If the final NACK packet gets lost, the source does not know when to begin retransmission of packets, so the source nodeS runs a watchdog timer T_{NACK}, upon the expiration of which the source starts retransmission. The timer T_{NACK} gets reset every time a NACK packet is received.

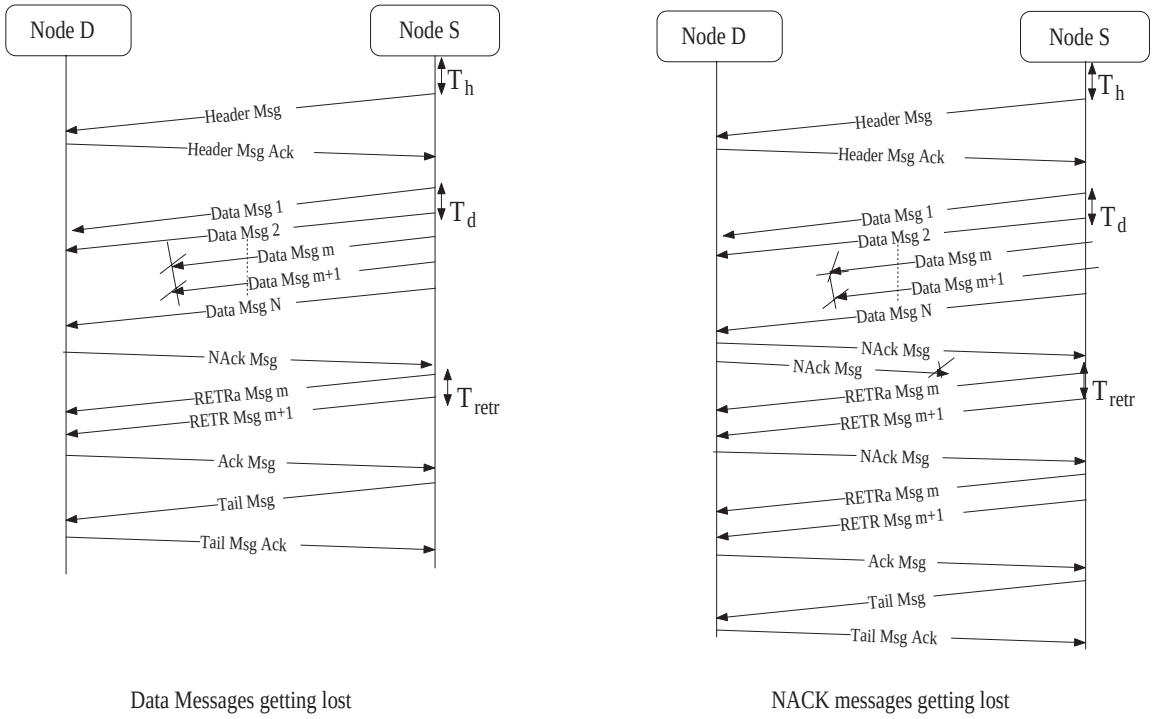

Figure 4.3: Description of Transport Protocol

If even after retransmissions nodeD has not got all the packets, it again sends NACK packets and the same procedure repeats. After nodeD receives all packets it sends an ACK to nodeS. In case a NACK message gets lost, it is treated as if , RETR packets with seqnos being carried in the NACK packet were lost. So nodeD (receiving node) takes no extra care to ensure reliable delivery of packets, the onus is on nodeS (sending node) to make sure to transmit all packets or give up after maximum no of retransmissions. This scenario is depicted in Figure 4.3.

There are always other unforeseen errors that can lead to failure, in that case both nodeS and nodeD run a watchdog timer and just restarts if its in a middle of a data transfer for too long with no activity. The reasons for failure can be sudden voltage increases leading to restart of nodes, some improper handling of memory by the code etc. And then it starts transmitting the same file again. If the source nodeS restarts it asks nodeD to discard the previous unfinished transfer and start afresh. If the destination restarts it requests a new transfer from the source nodeS.

4.3 Message Format

In this section we describe the different types of messages used to reliably transmit a file from one node to its parent. We describe the different types of active messages (AM_) we use in the implementation of the protocol. The TOSMsg packet explained in appendix 1 includes the MAC header and the MPDU (MAC Protocol Data Unit). All the active messages are accommodated within the data field of the TOSMsg packet which is the MPDU.

All the message types have been in Table 4.1. For all future explanations let us consider the scenario where nodeS is trying to transmit a file to nodeD.

- The structure of the Header Message is shown in Figure 4.4. The header message contains the destination the node requesting transmission, node to which file belongs, the file number, number of packets and number of bytes. The need for first 3 parameters are quite obvious whereas the latter 2 parameters war-

Message Type	Description
AM_DATA_MSG	Data packets
AM_HEADER	Header Message
AM_TAIL	Tail Message
AM_ACK	Acknowledgment Message
AM_NACK	Nack Message
AM_RETR	Retransmission Message
AM_WTS	Wait To Send Message

Table 4.1: Message Formats

rants some explanation. The start sequence no of the data packets are required because, the previous node may have experienced heavy packet losses and may not have data for the first N packets, so it can start sending data from N+1 packets, effectively saving bandwidth. The no_bytes field is required because the last packet may not contain data in its entire payload.

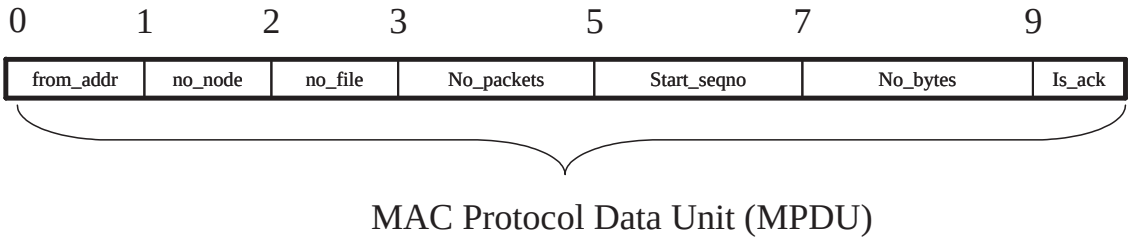

Figure 4.4: Structure of Header Message

- The data message carries the actual data of the files as an array 2 byte data fields. The `seqNo` is the sequence no of the packet and helps in determining packet loss. The data message carries no extra information than what is required and is optimized for increasing throughput. The structure of the data message is depicted in Figure 4.5.

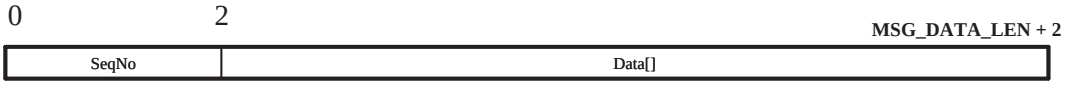

Figure 4.5: Structure of Data Message

- The structure of the Tail Message is shown in Figure 4.6. The structure of

the tail message is very much similar to that of the header message. The only difference is the `transmitDone` boolean field which notifies nodeD that nodeS is done with its transmission of data and that nodeD can either start receiving data from other nodes or it can start transmitting data.

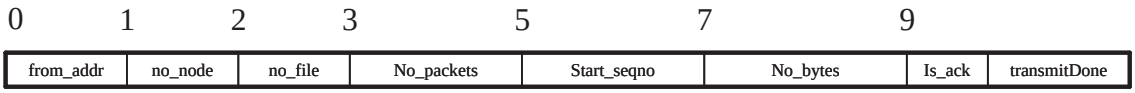

Figure 4.6: Structure of Tail Message

- The ACK message (shown in Figure 4.7) is relatively small in size and is a notification from nodeD (receiving node) to nodeS (sending node) that it has received the file completely. It contains the id of the node to which the data belongs and the id of the file that it has received along with the number of packets received.

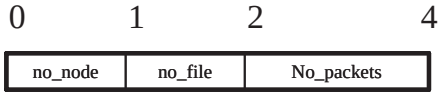

Figure 4.7: Structure of Acknowledgment Message

- The NACK message is used by the receiving node to notify packet loss to the sending node so that it begins retransmissions of lost packets. Each NACK packet has information about the node id of the data , file id , total packet loss experienced and the seqnos of the packets not received. Note that there may be multiple NACK messages as there may be more packets lost than can be accommodated in a single NACK packet, so the Boolean field `is_final` notifies the final packet among many packets being transmitted. The NACK message structure is shown in the Figure 4.8.

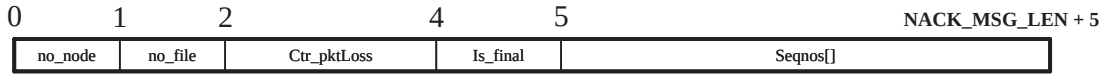

Figure 4.8: Structure of NACK Message

- The RETR message has exactly the same structure as the Data message, but it is used when retransmitting packets. It is more for programming convenience that this message type is created. Data message would done just fine
- The structure can be seen in Figure 4.9. The WTS message consists of just a boolean field `sendData`.

Figure 4.9: Structure of Wait To Send Message

4.4 Implementation Details

In this section we describe a few implementation details which influenced a lot of our design decisions. The tight integration of the transport protocol with handling of flash and the fact that tmote's flash and radio use the same UART are the most important of those details. Our application design requires the application layer to interleave writing and reading data to the flash with sending messages over the radio. But as the radio and flash share the same UART for controller functionality, the application layer has to do some intelligent arbitration. Though most of this arbitration is done by a layer down the stack, we had to do some careful programming and exercise caution. Having appropriate timers at required places, designing a layered architecture for handling flash and radio at different layers and ensuring their proper interaction were challenging tasks. Careful experimentation after implementation allowed us to reduce the value of timers at various places, and thus helping us to optimize the protocol to a great extent.

Chapter 5

Routing Protocol

5.1 Overview

In this chapter we describe the routing protocol we have designed and implemented to determine the route to transmit data from all nodes to the base node. It is interesting to note that the data transmission is from all nodes to a single specified node called the base node and not from all nodes to all nodes. This can lead to many optimizations in the route determination. Another fact that favors the problem of route determination is that nodes are static and the position of nodes hardly change after the initial deployment. This would lead to the argument that the routing protocol is not at all needed and the route can be actually hard coded in each of the nodes. But we also need to consider the following scenarios.

- If any of the nodes fail and is unavailable for a small amount of time the entire network will fail, on the other hand if we determine routes dynamically, there is a great chance that an alternate route gets formed. But hard coding all the possible routes is a crude way and may be feasible in certain conditions.
- Having a routing protocol will also enable structural scientists deploy nodes in

ad-hoc manner and collect data. This kind of data collection may dictate the nature of deployments.

5.2 Protocol Description

In this section we describe the routing protocol in detail. Let us consider the temple topology with 7 nodes of the BriMon application for the rest of our discussion. Node1 is the root or the base node. Following is the set of assumption made by the protocol.

- The routing protocol assumes that all the nodes have a unique id, which is not an unreasonable assumption to make.
- In our current implementation we assume that all the nodes are awake all the time. But our design has provision for the nodes to duty cycle between sleep and wake up.
- The routing protocol assumes that during a transmission of packets from node1 to node2, at least one packet transmitted by node2 is heard by node1. This basically means that it assumes limited symmetry in transmission.

The Figure 5.1 describes the conventions used in describing the routing protocol.

The routing process sets in prior to data transmission within the network. The base node starts the routing process by switching its state to NODE_TRANSMIT and It starts sending beacons (Figure 5.2). The state of all other nodes is NODE_LISTEN. And none of the nodes have their parent decided and their parent address is set to -1. The beacons contain the identity of the originating node and the no of hops it is to the root and the sequence number of the beacon. The nodes send beacons with increasing sequence numbers and the sequence number space is reset at the start of the routing phase.

For the base node the originating node is 1 and number of hops is 0. Lets say that nodes 2 and 3 are able to hear the beacons transmitted by node 1. Node1

Convention for Routing Protocol		
Symbol	**Name**	**Description**
	NODE_LISTEN	The node is listening, its parent not yet determined
	NODE_TRANSMIT	The node is transmitting beacons, its parent has been determined
	NODE_IDLE	Both its parent and its children have been determined, it can go back to duty cycling
	NODE_LEAF	The node is leaf in the network (no children) and is in NODE_IDLE state
	NODE_ORPHAN	The node is orphan but got connected to a parent but with a poor link
	LINK	Used to represent the link from a node to its parent
	LINK_POOR	Used to represent the poor link from a node to its parent, orphan to its parent

Figure 5.1: Routing Protocol : key

Figure 5.2: Routing Protocol : Stage 1

transmits beacons for a maximum time period T_{maxt}. The reason node1 transmits beacon for T_{maxt} is because, if it does not receive any beacons from its "'child"' nodes within T_{maxt} then node1 assumes that there are no nodes to form the route and can go back to sleep to wake up later to repeat the same procedure. Nodes listening to node1 wait for a time period T_l before transmitting beacons. The reason the nodes 2 and 3 wait for T_l time is that node1 need not be the only prospective parent (node transmitting beacons). In effect a node in NODE_LISTEN state waits for T_l amount of time to hear to all its prospective parents. The other functionality of T_l is to calculate packet loss during that duration. As the rate of transmission of beacons is known to all nodes (fixed initially) the node listening for T_l will be able to figure out how many packets it has received as against how many packets it is supposed to receive.

After node 2 and 3 have waited for T_l they have enough information to decide on their parent. The parent decision process is simple Just choose the parent from which you have experienced the least packet loss. This condition is too simple, and has some other additions. No node is allowed to choose a parent if the packet loss is greater than MAX_ADMISSABLE_PKTLOSS, until and unless it has no other choice. Now in this case node 2 and node 3 have heard beacons from only nodeD and lets assume that the packet loss has been less than MAX_ADMISSABLE_PKTLOSS. So node 2 and 3 choose node 1 as their parent. So as node 2 and node 3 have chosen their parent their state changes to NODE_TRANSMIT and they in turn start transmitting beacons.

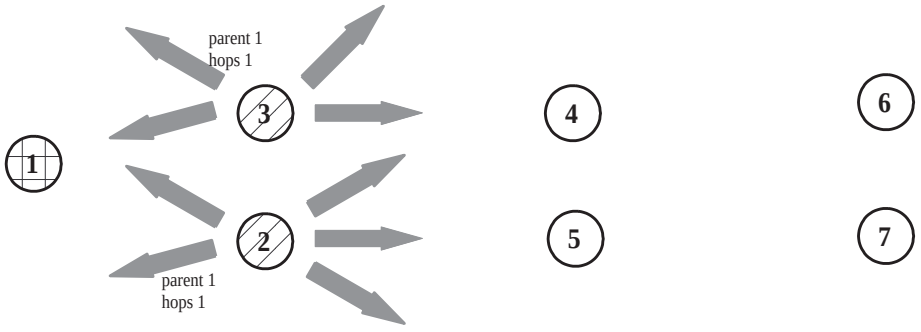

Figure 5.3: Routing Protocol : Stage 2

Now node 2 and node 3 are in NODE_TRANSMIT state and are transmitting beacons with originating node as node 2 or 3 respectively and no of hops as 1 (Figure 5.3). Node 1 can also hear this and with the help of these beacons updates its list of children and stops transmitting after receiving beacons for time T_l (It continues to receive beacons for time period Tc after the receipt of its first beacon from child). The reason for this is similar to the reason why node2 and 3 waited for time T_l. Node1 actually waits to get information about all its children . Once node1 gets information for time period T_c node1 stops transmitting beacons and can go back to sleep. The value of T_c should be such that it should be able to get information about its children before T_c timer expires. The relation between T_{maxt}, T_l and T_c is given below.

$$T_{maxt} >> T_l \tag{5.1}$$

$$T_{maxt} >> T_c \tag{5.2}$$

$$T_{maxt} \geq T_l + T_c \tag{5.3}$$

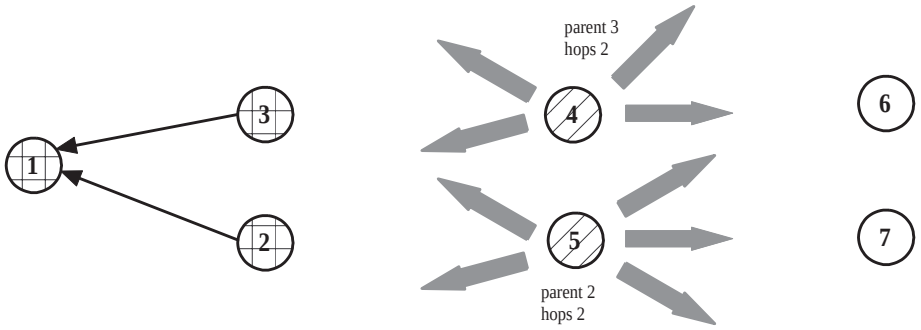

Figure 5.4: Routing Protocol : Stage 3

Node4 and node5 hear to the beacons transmitted by node2 and node 3. Lets suppose that node4 experiences packet loss greater than MAX_ADMISSABLE_PKTLOSS from node 2 and from node3, it experiences tolerable packet loss , so node 4 designates node 3 as its parent (Figure 5.4). In case of node 5 lets suppose that it experiences tolerable packet loss from both node 2 and node3. So node5 has node2 and node3 as prospective candidates as parents. It chooses one of them as its parent (lets say it chooses node 2) and stores the rest (prospective parent candidates from which this node has experienced packet loss which is within limits). After choosing their

parent and waiting for time T_l (just like node 2 and 3) they change their state to NODE_TRANSMIT and start transmitting beacons. Node 2 and node 3 hear to these beacons and node3 adds node4 to its children list while node2 adds node5 to its children list.

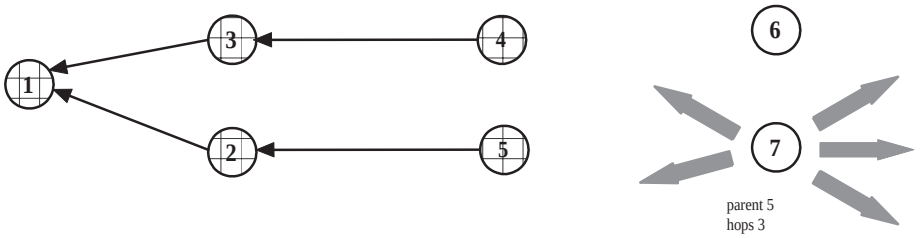

Figure 5.5: Routing Protocol : Stage 4

Now node6 and node7 hear to the beacons of node4 and node5 (Figure 5.5). Lets say that node6 experiences packet loss greater than MAX_ADMISSABLE_PKTLOSS from both nodes 4 and 5. So node6 designates both these nodes node4 and node5 as reserve parents and its state change to NODE_ORPHAN (it becomes an orphan for now) and it starts a timer T_{or}.

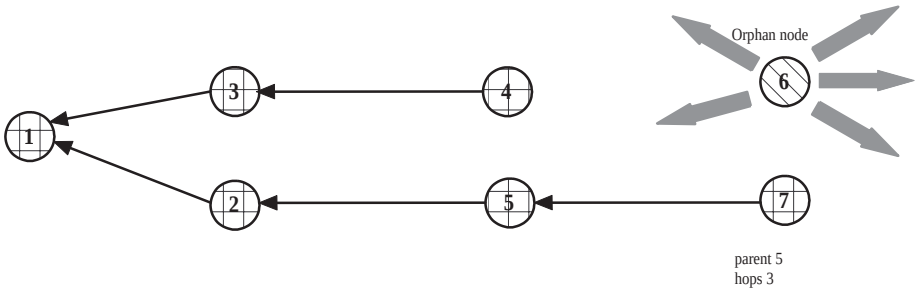

Figure 5.6: Routing Protocol : Stage 5

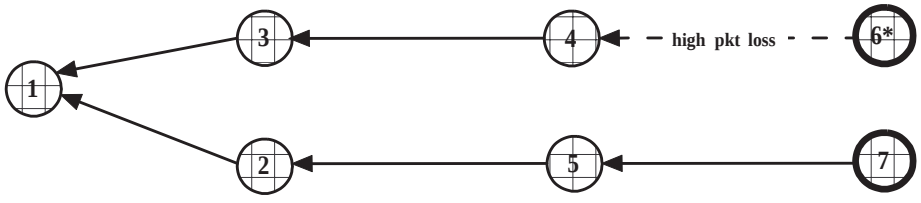

Figure 5.7: Routing Protocol : Stage 6

If the orphan node does not get any beacons (prospective parents with lesser packet loss than its current reserve parents), it is forced to choose one of the reserve

parents (one with lesser packet loss). But node7 has got beacons from node4 and node5 and the packet losses are within the limits. So as usual it designates one of them as its parent (say node5) and puts the other in the ancestors list. Now node7 starts transmitting beacons after it changes its state to NODE_TRANSMIT. Nodes 4 and 5 hear to the beacons of node7 and they put node7 in their children list. In the meantime the T_{or} timer of node6 which is in NODE_ORPHAN expires and lets say it has not got any other prospective parents so it ends up choosing node4 as its parent (Figure 5.7, 5.6). Node7 does not hear to any of the beacons from its child node (nodes with 1 greater hop count) so after T_{maxt} it changes its state to NODE_LEAF. After node6 chooses its parent, its state also changes to NODE_LEAF in a similar manner to that of node7. But note that beacons transmitted by node6 are ignored by nodes whose parents have already been selected. Now we have two leaf nodes in the network, the leaf nodes start transmitting AM_ROUTEENTRY messages to its parent after populating its parent entry in the AM_ROUTEENTRY message, its parent in turn populates its parent entry and sends it to its parent and so on till it reaches the base node. So with AM_ROUTEENTRY messages the base node becomes aware of the topology of the network.

5.3 Message Formats

The routing protocol uses 2 message types for its operation. AM_BEACON is used for propagating the route and AM_ROUTEENTRY is used to inform the route formed to the base node. The AM_BEACON messages are broadcast messages, propagate from the base node while AM_ROUTEENTRY messages are unicast messages start propagating from the leaves and go all the way to the base node.

Message Type	Description
AM_BEACON	Beacon messages
AM_ROUTEENRY	Route report messages

Table 5.1: Routing Layer Message Formats

The beacon contains the identity of the originating node, number of hops to the base node and the sequence number. The route entry message contains the

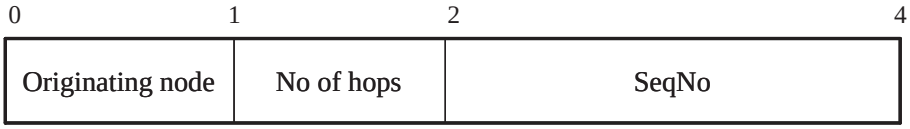

Figure 5.8: Message structure of a beacon

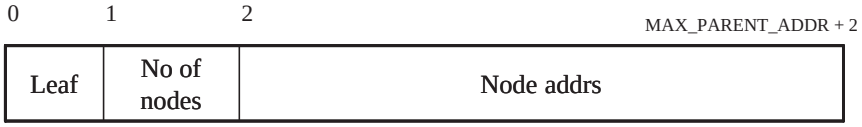

Figure 5.9: Message structure of a route entry message

leaf of the route, the number of nodes in the route and the actual route up to MAX_PARENT_ADDR nodes. The route is ordered such a way that nodes with greater number of hops to the base are before ones with lesser hops, thus node with just 1 hop is at end.

Chapter 6

Experimental Results

In this chapter we explain the initial experiments conducted by us, which influenced our design decisions along with the evaluation of our protocol by varying the parameters. Before proceeding to evaluate our protocols, we would like to mention that our aim was not to compare our protocols with existing protocols in the domain. Though we have built our own protocols, they are optimized for our application. Our main aim is to show that our protocols do well in the our application setting.

6.1 Range of the Tmote

We have modeled our solution around wireless sensor networks. One of the first experiments that we conducted was to test the transmission range of the sensor node hardware (moteIV's tmote-sky). Our solution involves placing these motes on piers of a bridge where the inter pier distance can be anywhere between 30-90 m. Our experiment on the range of motes aims to find if we can have successful data transfer between motes when we have a mote placed on every pier, without having a forwarding mote in between.

The experimental setup involved 2 nodes, one being the base node connected to the laptop (node0) while the other (node1) sending packets continuously at the rate of one packet per second. The experiment was carried out by configuring both the nodes to maximum transmit power (default). While having both nodes switched on we progressively increase the distance of node1 in steps of 10m and then after 70m in steps of 5m till we stop getting packets. We repeated these experiments for different environments. The results are in given in Table 6.1. We define the range of the mote as the largest distance at which the packet loss is less than 10%.

Environment	Range
On a road with trees	70 meters
Indoors in office corridor	30-40 meters
On a air strip	120 meters

Table 6.1: Transmission Range of Tmote-Sky

The airstrip is a large open space with no obstruction, place where small aircrafts and gliders can land and take off. The important observations of the experiment are as follows.

1. The range of the mote is highest on the airstrip because there are no physical obstructions and no RF interference.

2. In the case of a road with trees, the range reduces to 70m because of the obstructions created by trees, vehicular traffic and movement of personnel

3. The range is the least in an office environment because of obstructions in the form of furniture and people. The involvement of RF interference also cannot be ruled out.

4. The environment that matches the environment in our application is the airstrip environment, so we can expect the motes to operate at a range of the order of 100m in the setting.

The other important observations are as follows

1. The range did not improve with an external antenna in all cases suggesting that the internal antenna of the tmote is quite good. We had to do some hardware tweaking (shifting a SMD capacitor) followed by soldering the SMA connector to connect the external antenna.

2. The range significantly decreased if an old battery which has been used for quite some time is used. However we have not quantified this.

3. During our experiment on the airstrip we observed that the range of the mote significantly improves if there is an obstruction (human obstruction) just behind the mote (not in the line of transmission).

4. In all our experiments the motes had to be placed on raised platform, failing to do so would significantly reduce the range.

We draw the following conclusion from these experiments.

1. On bridges with inter-pier distance of the order of 100m, there is no need for a forwarding mote between two motes placed on a pier, but for greater distances we likely need a forwarding mote.

2. However only extensive experiments conducted on the actual site will reveal the actual scenario. But the worst scenario is that the range of the mote may decrease or we may face higher packet loss. If the range of the mote decreases we could use a forwarding mote in between, and any further packet losses are taken care of by our robust transport protocol.

6.2 Significance of LQI and SS

In this section and the next we explain the experiments we have done to evaluate parameters for our routing protocol. In this section we measure LQI and SS vs distance. We evaluate LQI (link quality indicator) and SS (received signal strength) as a possible parameter for our routing protocol. These parameters are used by the

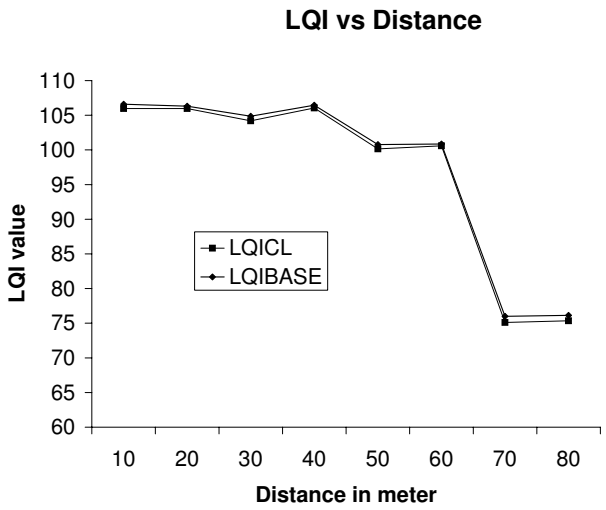

Figure 6.1: Plot of LQI vs Distance

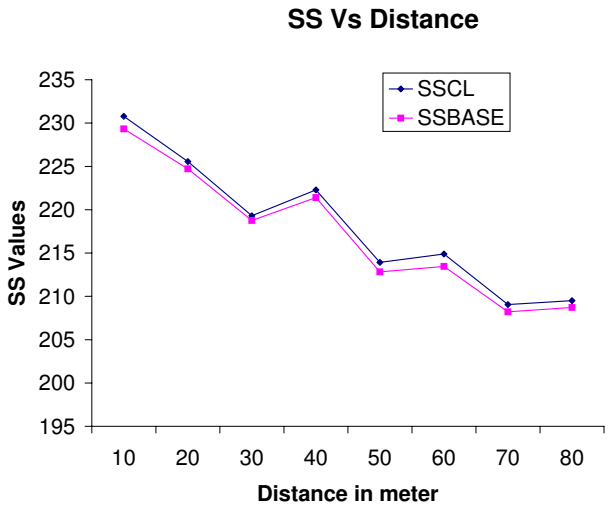

Figure 6.2: Plot of SS vs Distance

routing protocol to select the best parent for a node in order to transmit data to the base node.

Both LQI and SS are register values and have no units. The radio chip CC2420 provides SS (Received Signal Strength Indicator) that may be read any time. It also provides LQI upon reception of each packet by sampling the first 8 chirps of the packet. Actually both LQI and SS are part of the TOS_Msg packet but never transmitted. They can be accessed by `Msg->lqi` and `Msg->strength` (In TinyOs terms).

The correlation of LQI (link quality indicator), and SS (received signal strength) with the distance between the motes is very important for our work. With the help of this experiment we would like to see how the LQI and SS vary with distance and how their values correlate at either direction of the bidirectional link to establish symmetry of these values at both the sides of the link.

The experimental set up is quite similar to that of the previous experiment. It consisted of 2 nodes S (node0 and node1), one being the base node (node0) connected to the laptop. The base node (node0) sends packets continuously to node1, node1 upon reception of these packets, populates the packets with LQI and SS values at its side (client side) and returns it to the base node. The base node once it gets these packets populates these packets with LQI and SS values and passes it on to the laptop. So we have 4 values in total LQI at node1, SS at node1, LQI at node0 and SS at node0. Then we plot the the average of these values take over a period of time (60s) at each distance. The experiment was conducted at the airstrip.

Figure 6.1 shows how LQI at both the client (node1 , LQICL) and base (node0, LQIBase) varies with distance. We experienced consistent values as far as LQI is concerned. By consistent we mean that for a particular distance between two motes the variance between all values was small. The LQI values also exhibited symmetry which means that the LQI value at the transmitting node and the receiving node was almost the same (not only the average but also the individual values were almost the same) As can be seen from the graph LQI remains almost constant (varies between 100-110) for distances from 10 to 60 m and then suddenly dips to around 75 at 70m. At 70m we experienced significant packet loss (around 10-20%), but the packet loss

significantly increased (hardly few packets got by) at 80m. The range has significantly decreased from greater than 100m to a mere 70m because of the fact that we were using batteries which were not new for the motes in this particular experiment. The results are affected only to the extent that the range may increase but the shape of the curve would remain the same.

This experiment initially proves that LQI is in fact a very good measure to determine packet loss, thus a good candidate for the parameter for our routing protocol. The reason we want to use LQI is because, calculating LQI is quite straight forward where as calculating packet loss is quite complicated in our protocol setting. With the choice of LQI as the suitable parameter we can use a lot of existing routing protocols with the required modifications.

Figure 6.2 shows how SS at both the client (node1 , SSCL) and base (node0, SSBase) varies with distance. We experienced inconsistent values as far as SS is concerned. This means that even for the same distance the SS values varied widely. The SS values exhibited symmetry which means that the SS value at either side of the link was almost the same (not only the average but also the individual values were almost the same). As can be seen from the graph the SS values decreased with distance, but discrepancies can be noticed at distances 40, 60 and 80 m. We repeated the experiment and found similar inconsistencies in the repetition too. So we rule out SS as a parameter for our routing protocol because of the inconsistencies observed.

6.3 Correlation of LQI and Packet Loss

As explained in the previous section SS is not a consistent measure of distance, though LQI shows some promise. Here we investigate whether LQI is good enough as a measure for measuring packet loss, which can be use as a metric in our routing protocol. The results were verified in the work [19].

The experimental set up is quite similar to that of the previous experiment. It consisted of 2 nodes (node0 and node1), one being the base node (node0) connected

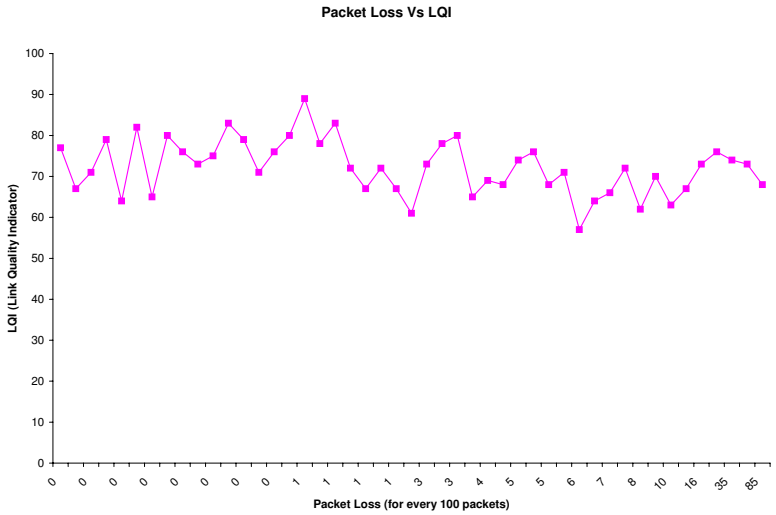

Figure 6.3: Plot of LQI vs packet Loss

to the laptop. Node1 sends packets continuously to node0 at the rate of 1 packet every 25ms. The number of packets lost for every 100 packets sent is calculated along with minimum and maximum LQI values for all the received packets. Multiple such readings are taken at every distance. Readings are taken at different distances increasing at 10m initially, and then increasing in smaller granularity at greater distances (3-5m). Experiments were carried out in a long hallway.

Figure 6.3 shows the plot of LQI vs Packet Loss. The X-axis shows number of packets lost for 100 packets transmitted. The Y-axis shows the corresponding average LQI value. Ideally LQI values should decrease as packet losses increase but what is seen is quite inconsistent. We are able to see instances at certain distances (Here distances are not important so they are not shown) where despite having low LQI have low packet loss and vice versa. The distance is not important because we are trying to show that LQI is not a good measure of packet loss at some instances and so we have to use packet loss instead to LQI.

The implication of this measurement is that we use packet loss as a metric in

our routing protocol instead of using LQI as in other conventional protocols.

6.4 Analysis of Transport Protocol

In this section we analyze the transport protocol for reliability and scalability (stability with increasing data) . We analyze the protocol on various parameters in a single hop setting. We have also successfully tested our protocol in a multihop setting but have not done elaborate experiments. The aim of this exercise is to quantify the scalability and reliability of our protocol.

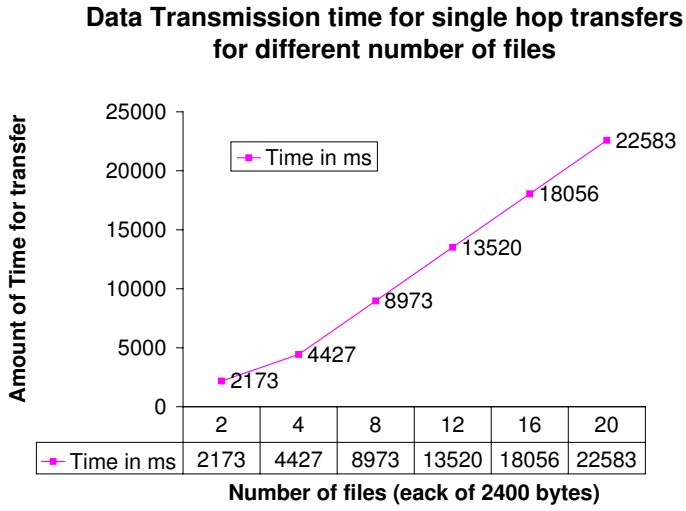

Figure 6.4: Plot of data transfer duration vs number of files

One of the metrics of the transport protocol is the amount of data that it can transfer. Our application requirements demand that the protocol should be able to transmit around 40KB of data. The experimental setup involves 2 nodes node0 and node1 where node0 is the base node, runs the base node code and connected to the laptop. Node1 is loaded with the normal code for data collection (that all the nodes

in the network are loaded with). We issue a command to node1 to start transfer of X amount of data through the laptop and node0. The amount of time required for the transfer of X bytes is recorded for different values of X. The time recorded is from receipt of the first header message to the receipt of the last tail message. The experimented was conducted indoors with a distance of few meter between the nodes.

Figure 6.4 shows the plot of the amount of time taken in ms for different file sizes. The plot is almost linear which indicates that our protocol scales linearly for increasing amount of data. We shall take one of the readings and try to account for the time taken for the transfer with respect to our protocol. We choose for simplicity the case when we transfer 2 files each of 2400 bytes (so total of 4.8KB of data). We have resorted to approximations in a lot of these calculation for lack of accurate data, however we show that our protocol does quite well even if take on a conservative explanation.

As mentioned earlier the time calculated is from the receipt of the first header message to the receipt of the last tail message. This means the time required for reading the first file from the flash (only the first file) is not included in the transfer. With this this we try to account for the 2173ms taken for 2 file transfers (4.8KB) below.

- The time taken for sending acknowledgment to header (H1) is 15ms. Actually the total round trip time as we calculated by experiments i.e from the time to send a message and then its receipt of acknowledgment with no other overhead at all is around 25ms. The reason for this is that it takes around 10ms for a packet to get sent at a node (in fact something less than 10ms), we have 2 packets to be sent and we have round trip time (RTT) and some processing overhead to account for. So the time for sending acknowledgment after receipt of acknowledgment is around 15ms.

$$H_1 = 15ms(approx) \tag{6.1}$$

- After receipt of header messages are sent at an interval of 10ms. The total amount of packets to be sent in this case is 100 (2400 bytes/24bytes per packet).

So the amount of time for data transfer is 1000ms.

$$D_1 = 1000ms \tag{6.2}$$

- After receipt of last packet acknowledgment message gets sent. Interval between sending of last packet and receipt of acknowledgment is 25ms due to reasons explained above.

$$A_1 = 25ms(approx) \tag{6.3}$$

- Interval of time between receipt of acknowledgment and sending of tail message is 10ms.

$$T_{start1} = 10ms \tag{6.4}$$

- Interval between sending of tail message and receipt of acknowledgment is 25ms.

$$T_1 = 25ms(approx) \tag{6.5}$$

- Interval of time between receipt of acknowledgment for tail and resumption of next file transfer is δ. This means that the interval is negligible for all practical purposes

$$In_1 = \delta \tag{6.6}$$

- Amount of time to read the second file from flash is α.

$$F_2 = \alpha \tag{6.7}$$

- Interval of time to start sending header message is 10ms.

$$H_{start2} = 10ms \tag{6.8}$$

- Interval between sending of header message and receipt of acknowledgment is 25ms (approx).

$$H_2 = 25ms \tag{6.9}$$

- Now the file transfer for second file has started and we can plug in values for variables similarly.

$$D_2 = 1000ms, A2 = 25ms, T_{start2} = 10ms \tag{6.10}$$

- Amount of time to receive the tail message once it is sent is 15ms

$$T_2 = 10ms \tag{6.11}$$

By summing all the equation we have the total time take to be

$$TotalTime = 2155 + \delta + \alpha \tag{6.12}$$

This almost accounts for the 2173ms time taken for 48KB of data transfer. In the worst case we may not be able to account for 2-5% of the total time of transfer (considering approximations also).

The same reasoning can be used to explain all the numbers in the graph. We conclude by saying that our protocol scales linearly for increasing amount of data transfers.

In the next experiment we seek to observe the trade off between small and large file sizes. The experimental setup is same as that of the previous experiment except that the amount of data transfered is the same for all transfers but the file sizes are different for every transfer. With this experiment we aim to show that larger file sizes are better than smaller file sizes.

The experimental setup is the same as that of the previous experiment except

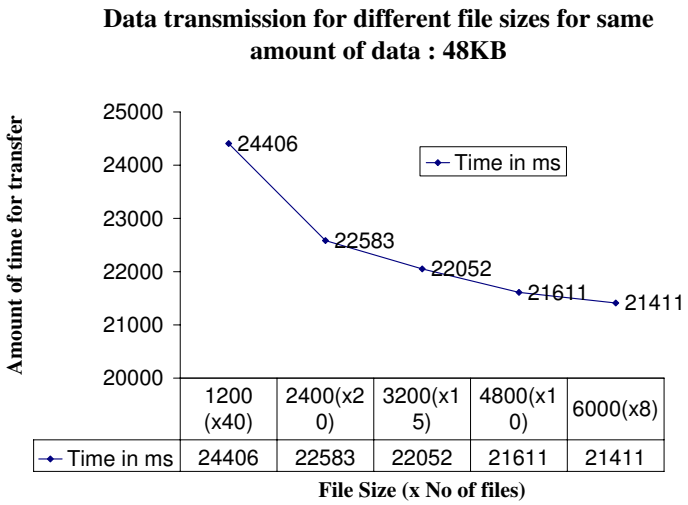

Figure 6.5: Plot of data transfer duration for same amount of data transfer vs number of files and different file sizes

for the differing file sizes and variable number of files being transfered.

The experimental results are shown in Figure 6.5. The total time for transfer readings were taken for the same amount of file transfer, 48 KB but the number of files and hence the file sizes were varied. The results show a plot of amount of time taken in ms for different number of files. We can observer a sudden dip in the graph because the number of files decreases from 40 to 20 at that instant but for the rest of the readings the change is much less. In this experiment and the previous one we have observed that the amount of time for each file transfer from the receipt of header for that file to the receipt of tail for that transfer is directly proportional to the amount of data being transfered. But the inter-file interval (consisting of round trip times of header, tail and ack and file read) is almost constant irrespective of the file size (around 30ms). So when the number of file increases (thus file size decreases) the time for file transfer decreases individually but the number of inter-file intervals increase proportional to the increase in number of files hence the transfer time increases.

$$TransferTime = n * T + (n-1) * \beta \tag{6.13}$$

Where n is the number of files and n-1 is the number of inter-file intervals. T is time required for transfer of a single file. β is the inter-file interval, which is almost constant. So as n increases (file size decreases) T decreases hence the the term $n * T$ remains constant. But as n increases so does n-1 but β is a constant hence the term $(n-1) * \beta$ increases hence total transfer time increases as number of files increase even with total data transfer remains a constant.

The application layer throughput can be calculated using the formula

$$Throughput = Data/Time \tag{6.14}$$

Where Data is the amount of data transfered while Time is the amount of time taken for the transfer. Taking values from our experiment we have throughput = 48000/21.411 = 17.9kbps. But the MAC level throughput calculated separately is (about 100-110 packets, each of 36 bytes per second) 31.6kbps. The MAC level throughput is calculated by transmitting packets as fast as possible and calculating the throughput by taking into account the MAC header. But the CC2420 radio claims to have 256kbps throughput but the achievable is only around 30-40 kbps []. In the rest of the section we account for the loss in throughput.

- The through put at the application layer is 17.9kbps

$$Throughput_{application} = 48000/21.411 = 17.9kbps \tag{6.15}$$

- Accounting the MAC header MAC header we get an additional 8.8kbps (10 bytes per packet and 100 packets per second).

$$Throughput_{MACheader} = 10 * 100 * 8 = 8.8kbps \tag{6.16}$$

- Taking into account the sequence number field will account for additional 2 * 100 * 8 = 1.6kbps.

$$Throughput_{seqno} = 2 * 100 * 8 = 1.6kbps \tag{6.17}$$

- So we have accounted for 28.3kbps in total.

$$Throughput_{accounted} = 28.3kbps \tag{6.18}$$

The header and tail messages and the corresponding delays due to round trip times account for the additional 3.3kbps, hence we are able to account for almost all the throughput loss. We conclude by saying that our protocol is efficient and add very little overhead (around 10%-20%) for reliable data transfer.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

India has the second largest rail network in the world, transporting over four billion people annually. An essential part of this network are 120,000-plus steel bridges [1], a lot of which are aging, weak, distressed, accident-prone. Actual numbers indicate that 43% are over 100 years old and 57% are over 80 years old. There is currently no system in place to continuously monitor these bridges. It is extremely dangerous to operate trains carrying hundreds of passengers on a railway network, of which these bridges are a part. To replace all the old bridges at once is not a feasible solution because it would involve prohibitive costs. Moreover with no data to support the claim that these bridges are indeed old and need urgent attention, it would be rather difficult to have a strong case for the replacement of these bridges. Our work aims at providing sufficient data for analysis so that structural engineers can make a strong case for repair or replacement.

We conclude by saying that we have successfully developed all the various software and hardware components required for the application, successfully integrated

[1] http://www.rediff.com/news/2001/jun/25spec.htm

most of the components. Data collection from the wireless sensor network and further aggregation at the central server has been successfully achieved. The application has exhibited desired levels of reliability and performance as per our testing so far, however only a actual deployment on a railway bridge will expose problems with the application if any. The major problems we have faced both during development and testing of our protocol was unreliable nature of the wireless medium and compatibility problems with with various software and hardware components. Other than this unexpected failure of hardware and software required additional reliable mechanisms and certain redundancy in design put in place. We plan to do the actual deployment on one of the railway bridges in summer of 2006.

Though our work was specifically designed and implemented for the BriMon application, we found that the protocols developed finds almost direct use, without much modification to other applications. *WiBeam* is a parallel work done by one of our colleagues which is a wireless bearing monitoring system which aims to automate the collection of vibration data from various motors in a ship. Our routing and transport protocol found direct use with few modifications in this application. Apart from this a lot of code developed for the BriMon project was used in most of the course projects of the *CS725 (Topics in Networking)*course most directly and few indirectly. Due to wide usability of our code and protocols we also plan to package our code and release it to the open source community to further research in this domain.

7.2 Future Work

Deployment of the system developed till now is the top priority. Initial deployment would involve simple short lived sensor network collecting data and storing the same on a computer placed on the same bridge. A further step would be to transfer the data to a mobile vehicle, if possible a moving train. Deploying and testing a long lived network with detection of train arrival, wake up of the network and data transfer to the train would be the final step of deployment planned. A detailed analysis of the mentioned deployments would follow. As a further extension of the work we would also like to evaluate the suitability of our protocols to other sensing applications like

the WiBeam project.

Appendix A

Description of Hardware and Software

A.1 Tmotes

The tmote-sky motes , hereafter referred to as the tmotes is one of the latest offering from Mote-IV [11] corporation in the wireless sensor network domain. The tmote-connect and tmote-invent are newer offerings but the basic hardware remains much the same. The tmote-connect adds an ethernet port and the tmote-invent is just a cosmetic makeup of the older tmote-sky. The other hardware platform which is similar to the tmote is the micaz, here after referred to as mica motes which is an offering from CrossBow Technologies [29]. We choose tmote-sky as our hardware platform. The tmote-sky features a 250kbps CC2420 radio controlled by a 8MHz MSP430 micro controller, which provides ADC ports and other peripherals. It is programmable through the USB port of the pc, has a 1MB non volatile flash for data storage, provides 3 leds as user interface and a slot to attach a external antenna. The key features of this platform are detailed below.

- The tmote-sky features a CC2420 radio from chipcon [4]. It is a 802.15.4 compliant high data rate radio with 256kbps data rate, provides PHY and some MAC layer functions. The radio can be configured to operate in different power levels from 1 to 31, 1 being the lowest power and 31 being the highest power. The corresponding current consumptions, output power are (8.5mA, -25dbm) and (17.4, 0dbm) respectively. However making the radio operate in lower power significantly lowers the rage of operation. The CC2420 radio provides two metrics RSSI (Receive Signal Strength Indicator) which can be measured at any time and LQI (Link Quality Indicator) which is calculated by measuring the first 8 chips of each packet. The CC2420 chip is controlled by the MSP430 micro controller by TI [22] through the SPI it shares with the flash as well as the external expansion connector. This means that there is a need for careful bus arbitration while using any of them simultaneously. The radio also provides for clear channel assessment based on the measured RSSI value and a programmable threshold. This feature is used to implement CSMA-CA. The different CCA modes are as follows

  1. Reserved
  2. Clear channel when received energy is below threshold.
  3. Clear channel when not receiving valid IEEE 802.15.4 data.
  4. Clear channel when energy is below threshold and not receiving valid IEEE 802.15.4 data.

  The CCA feature has enabled the Wake-on-WLAN scheme which we have used in our application[9].

- The tmote features an on-board MSP430F1611 8Mhz micro controller from TI with 48KB ROM and 10KB RAM. The 10KB ram, which incidentally is the largest on-chip RAM means that a large amount of data can be kept in the RAM without any writes or reads needing to the flash. As we demonstrate later this capability has influenced our design decisions. A 48KB ROM means that the maximum size of the code that can fit into tmote is 48KB. This is very less compared to the 128KB offered by the micaz from Crossbow technologies [29]. The 16 bit RISC based micro controller features extremely low sleep and active current consumptions which allow the tmotes to run for months together

on battery power. It has a digitally controlled oscillator (DCO) which may be operated up to 8MHz, and can be turned on from sleep mode in 292 ns at room temperature, which for seamless switching between sleep and wake up modes. Apart from DCO there are 8 external ADC ports and 8 internal ports. The external ADC ports can be used to collect external signals while the internal ports can be used to measure battery voltage, internal thermistor. Apart from the ADC ports, a variety of peripheral like SPI (Serial port Interface) , UART (Universal Asynchronous Receiver/Transmitter), I2C (Inter IC) are also available. The external ADC ports and peripheral are provided through 2 expansion connectors one of 10 pin and the other of 6 pin.

- The tmote uses the USB (Universal Serial Bus) controller from FTDI (Future Technology Devices International Ltd) [5] to communicate with the host pc. We can also program the tmote using the USB interface, we do not need a separate programming board unlike the micaz which need an additional programming board to program the motes. This feature is extremely useful especially when there are more than one group working on the same hardware platform, as we do not need to purchase additional programming boards, just USB connectors would do.

- A 1MB STM25P80 Flash [21] on the tmote is another extremely useful feature. This can be conveniently used to store data, code and other information permanently on the tmote until the flash gets formatted. The flash shares SPI communication lines with the CC2420 radio and the external SPI pins. So Flash read and write has to be carefully interleaved with radio communication. The 1MB or 1024 KB of storage is divided into 16 segments each of size 64KB.

- The internal antenna of the tmote has indoor range of around 50m and an external range of 125m at full battery power. There is also an option of attaching an external antenna by first soldering SMA connector to the slot provided and later attaching the antenna to that slot. Additionally one of the capacitors from its place. This seems to be quite a cumbersome procedure compared with the mica motes which come with an external antenna. The use of internal antenna with the tmotes requires that the tmote be placed on a raised platform for testing with higher ranges (10s of meters).

A.2 TinyOS and NesC

TinyOS [23] is an open source operating system designed for networked embedded systems or sensor networks. Its component architecture enables rapid innovation and development. TinyOS system, libraries and applications are written in a programming language called NesC. NesC has syntax similar to C but supports the concurrency model of TinyOS as well as various other mechanisms that allow application designers to build components and easily integrate them into their applications.

The platform we are using (tmotes) is comparatively new and all the software has been written for mica motes. Even though most of the libraries work for the tmotes too, some of the applications are still not available on the tmote platform. But development is catching up. TinyOS executes a single program but has 2 threads of control - Tasks and Hardware event handlers. Tasks once started , run to completion , don't preempt one another and can be deferred. On the Hardware event handler are executed as a response to hardware events , preempt tasks and other hardware event handlers, and once started run to completion. As tasks and hardware event handlers can be preempted certain race condition may occur which can be avoided either by accessing all data exclusively or by using atomic statements inside tasks.NesC apps are built out of components which have well defined bidirectional interfaces. It also defines a concurrency model based on tasks and hardware event handlers to detect data races at compile time.Every NesC app consists of several components wired together to form an executable. A component provides a interfaces and may use several interfaces. These interfaces are the only point of access to these components. A interface declares commands which the interface provider has to implement and a set of events that the interface user has to implement. So if you want to use a command provided by a interface you have to first implement all the events specified by that interface. Modules are used to implement interfaces and configurations are used to wire other components together.

A.3 Accelerometers

An accelerometer is a electro mechanical device that produces electrical signal proportional to the applied acceleration. Accelerometers are characterized by several performance parameters: sensitivity, which denotes the smallest measurable acceleration and is expressed in gs (gravitational acceleration); dynamic range, which denotes the range of accelerations that the device is capable of measuring and is also expressed in gs; and noise, which is measured either as an RMS value, or is expressed as a function of the frequency of vibration. the output of an accelerometer is a time series of sensor readings with a specified resolution and a specified sampling rate. Strictly speaking, these are parameters associated with the analog-to-digital circuitry attached to an accelerometer, but they nevertheless constrain the performance of the accelerometer. The resolution constrains the sensitivity of an accelerometer; a 10-bit accelerometer whose dynamic range is 1g cannot have a sensitivity less than 1mg. The sampling rate, on the other hand, governs the frequencies that can by measured by the accelerometers.

A.4 TOSMsg Packet Format

In this section we describe the different types of active messages (AM_) we use in the implementation of the protocol. Before that we describe the structure of the TOSMsg packet as shown in the Figure A.1 which includes the MAC header and the MPDU (MAC Protocol Data Unit). All the active messages are accommodated within the data field of the TOSMsg packet which is the MPDU. The size of the MPDU is 28 bytes by default but can be easily changed by changing the TOSH_DATA_LENGTH field in the AM.h file. The other relevant field is the length field which holds the length of the MPDU, in default case it is set to 28. The addr field is used to specify the address of the destination node. If address field is populated with TOS_BCAST_ADDR the message is a broadcast message, other wise the message is a unicast message meant for the "addr" node. The type field specifies the message type and allows us to define as many as 255 active messages. These active messages can be used to handle

different message types in a customized manner easily. The group field specifies the TOS_AM_GROUP. For nodes to transmit and receive each other's messages they need to be set to the same group. By default it is set to TOS_DEFAULT_AM_GROUP;

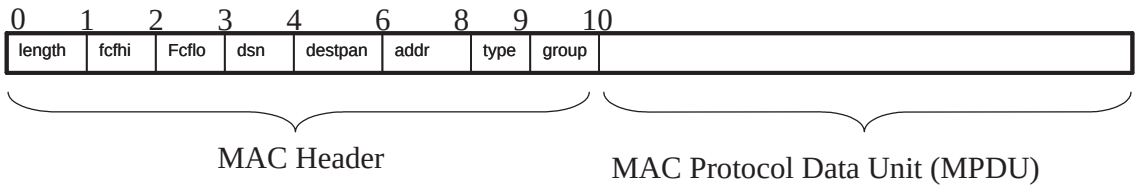

Figure A.1: TOSMsg Packet Structure

There are some of the fields in the TOSMsg structure that are not actually transmitted but are there only for internal use. The "strength" field give the signal strength of the received packet and the "lqi" field give the link quality indicator , which is also used as a measure of packet loss, however we have found it a bit unreliable. The ack field is used for acknowledgment and the "time" field is used for time stamping. The actual TOSMsg structure is reproduces as it is.

```
typedef struct TOS_Msg
{
  /* The following fields are transmitted/received on the radio. */
  uint8_t length;
  uint8_t fcfhi;
  uint8_t fcflo;
  uint8_t dsn;
  uint16_t destpan;
  uint16_t addr;
  uint8_t type;
  uint8_t group;
  int8_t data[TOSH_DATA_LENGTH];
```

```
  /* The following fields are not actually transmitted or received
   * on the radio! They are used for internal accounting only.
   * The reason they are in this structure is that the AM interface
   * requires them to be part of the TOS_Msg that is passed to
   * send/receive operations.
   */
  uint8_t strength;
  uint8_t lqi;
  bool crc;
  bool ack;
  uint16_t time;
} __attribute ((packed)) TOS_Msg;
```

TOSMsg Structure reproduced from AM.h

Appendix B

Some Facts on Bridges/Trains

In world,

1. Longest length - Second Lake Pontchartrain Causeway- 38.42 kms (span 45.7 mt)
2. Longest span - Akashi-Kaikyo Bridge - 3.91km length, 1.99 km span

In India, the longest bridge is Mahatma Gandhi Setu 5.85 km length

Railway bridges in India

- Longest length - Nehru Setu - 3.06 km
- Longest span - Sone, Dehri - 93 spans of 30.5m (upcoming 18 spans of 120 m + 2 spans of 30.5 m in Assam)

Fastest train

1. In world, Shanghai's Maglev 299.33 km/hr (186 miles/hr) (France is testing 321km/hr or 200 miles/hr train)

2. In India, Bhopal Shatabdi 140km/hr (test runs upto 184km/hr were conducted in 2000)

Bibliography

[1] http://www.analog.com/en/prod/0,2877,ADXL203,00.html.

[2] B. Akan and Ian F. Akyildiz. Event-to-sink reliable transport in wireless sensor networks. *IEEE/ACM Trans. Netw.*, 13(5):1003–1016, 2005.

[3] http://www.cs.wright.edu/ ~phe/EGR199/Lab_2/.

[4] http://www.chipcon.com.

[5] http://www.ftdichip.com/.

[6] Lakshman Krishnamurthy, Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Nandakishore Kushalnagar, Lama Nachman, and Mark Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 64–75, New York, NY, USA, 2005. ACM Press.

[7] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM Press.

[8] Nilesh Mishra. Brimon : Application design for railway bridge monitoring. Master's thesis, IIT Kanpur, 2006.

[9] Nilesh Mishra, Kameswari Chebrolu, Bhaskaran Raman, and Abhinav Pathak. Wakeonwlan. In *WWW '06: 15th International World Wide Web Conference*, 2006.

[10] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MMA7260Q.

[11] www.moteiv.com.

[12] http://tinyos.cvs.sourceforge.net/tinyos/tinyos-1.x/tos/lib/MultiHopLQI/.

[13] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244, New York, NY, USA, 1994. ACM Press.

[14] Personal communication with Dr Bajpai and Abhisheck Singhal Dept of Civil Engg IIT Kanpur.

[15] http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2006-January/014213.html.

[16] http://www.chipcon.com.

[17] http://www.rediff.com/news/2001/jun/25spec.htm.

[18] F Stann and J Heidemann, RMST: reliable data transport in sensor networks in Proc. IEEE SNPA, May 2003, pp. 102-112.

[19] A Shekawat. Packet delivery performance in sensor networks. Master's thesis, IIT Kanpur, 2006.

[20] http://mail.millennium.berkeley.edu/pipermail/tinyos-help/2006-January/013999.html.

[21] http://www.st.com/stonline/books/pdf/docs/8495.pdf.

[22] www.ti.com/msp430.

[23] http://www.tinyos.net.

[24] C. Wang, K. Sohraby, Bo Li, and W. Tang, Issues of Transport Control Protocols for Wireless Sensor Networks.

[25] Chieh-Yih Wan, Andrew T. Campbell, and Lakshman Krishnamurthy. Psfq: a reliable transport protocol for wireless sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 1–11, New York, NY, USA, 2002. ACM Press.

[26] Chieh-Yih Wan, Andrew T. Campbell, and Lakshman Krishnamurthy. Psfq: a reliable transport protocol for wireless sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 1–11, New York, NY, USA, 2002. ACM Press.

[27] Chieh-Yih Wan, Shane B. Eisenman, and Andrew T. Campbell. Coda: congestion detection and avoidance in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 266–279, New York, NY, USA, 2003. ACM Press.

[28] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27, New York, NY, USA, 2003. ACM Press.

[29] www.xbow.com.

[30] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor network for structural monitoring. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24, New York, NY, USA, 2004. ACM Press.