A Prototype Development of Reliable Sensor Network Based Structural Health Monitoring System For Railway Bridges

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

MASTER OF TECHNOLOGY

by

Lt Col Raj Kumar

under esteemed guidance of

Dr. Kameswari Chebrolu
Department of EE.
IITKanpur

Dr. Bhaskaran Raman
Department of CSE.
IITKanpur

to the

Department of Electrical Engineering,
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

June, 2007

Certificate

This is to certify that the work contained in the thesis entitled *"A Prototype Development of Reliable Sensor Network Based Structural Health Monitoring System For Railway Bridges"*, by *Lt Col Raj Kumar*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

June 2007

(Prof. Kameswari Chebrolu)
Department of Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur, Uttar Pradesh 208016

June 2007

(Prof. Bhaskaran Raman)
Department of Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur, Uttar Pradesh 208016

ABSTRACT

The Indian Railways consists of about 1,27,000 bridges, of which 40% are over 100 years old. It is critical to have a system to monitor the structural health of these bridges for maintenance and safety of the public. Present systems used for such monitoring are mainly wired systems. These systems are generally bulky and requires expertise in manpower. Also it takes days to deploy these systems on the bridge.

In this thesis work we develop an automated wireless sensor network (WSN) based system, which makes use of sensor motes and MEMS accelerometers for railway bridge monitoring. We call this system as BriMon. The system is easily deployable and requires minimum maintenance. The system though primarily designed for long term monitoring of remotely located bridges, can be used for short term monitoring too. The system has been developed by making use of off the shelf hardware. We have adopted an application driven approach in developing the protocols and components of the system. The implemented solution shows how the design choices dictated solely by the application requirement are different from general solutions that exist in the sensor network domain.

The system incorporates a novel event detection mechanism that triggers data acquisition by the *data acquiring nodes* in response to an oncoming train. The data from the *data acquiring* nodes is then reliably gathered at the *base nodes*. We make use of train itself to automatically transfer the vibration data from the remotely located bridge. For this we employ a simple yet effective multi-channel data transfer mechanism to transfer the data from the *base nodes* onto sensor nodes located on the moving train on 802.15.4 radio. The data from these nodes is transferred to a data analysis centre whenever the train passes a nearby station which provides some network connectivity to the data analysis centre.

In this work we discuss, data acquisition system, data compaction and compression, transport protocol for reliable data transfer, debugging tool, and data analysis tool which we have designed, developed and evaluated. The data acquisition system is DMA based and it acquires the vibration data with high fidelity and precision. We do ten point averaging of acquired data along with compaction to compress the data acquired by the nodes. We further study the possibility of using standard lossless data compression techniques in our application. The transport protocol developed is SACK based and is used for reliable transfer

of vibration data in all the cases mentioned above. In our application the mobile data transfer is time critical in the sense that we need to reliably transfer the data to mobile train within limited contact duration. The protocol achieves this with minimum latency. Also it satisfies all other requirements of the application. The debugging tool is used for testing the application. The data analysis tool we have developed has user friendly GUI and it meets all the requirements of the structural engineers for data analysis.

Finally we discuss the integration work which we did to integrate our work with other components of the BriMon which were developed by Phani Kumar Valiveti. These components are routing, time synchronization and event detection.

Acknowledgments

Firstly, I would like to thank Prof. Kameswari Chebrolu and Prof. Bhaskar Raman who have provided me the great opportunity to work in this emerging field of sensor networks. It was their mentoring and able guidance which helped me in successfully completing this work. Working with them has provided me a very good practical experience in this domain. The applications in sensor networks have recently started finding their use in the army. It was very important for me to work on one prototype sensor network application so that when I go back, I can apply the knowledge gained in our domain.

Next, I would like to thank Prof. C.V.R. Murthy and Dr. K.K. Bajpai for providing us with all the inputs we needed, from structural engineering domain. They provided us all the facilities in the structures lab to conduct our experiments. They were always readily available for interaction on the subject. Prof. C.V.R. Murthy even accompanied us for doing measurements on a road bridge. His encouragement and inputs always boosted our morale and helped us to design the system better. I would also like to thank Prof. Rajat Moona for all the support he provided us during this work. All the technical discussions we had with him helped us a lot in our work.

It has been a memorable working experience with Phani Kumar Valiveti, who has been an encouraging colleague working on same project as mine. A special mention needs to be done about Nilesh Mishra, a senior who has been supporting us throughout the work, in technical as well as logistical aspects. The past work done by Hemanth and Nilesh has also helped us in the initial stages of our work. I would like to extend my thanks to the staff of Media Lab Asia, IIT Kanpur for providing us all the support we needed.

I would like to thank my wife Suman for all her support and encouragement which helped me in completing this work.

This work would not have been possible, if it were not the encouragement, love and affection of my parents, brothers and teachers. It is with their encouragement, love and care throughout my life that I have reached this level.

I dedicate this work to my parents.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Indian Railways is one of the largest enterprises in the world, and railway bridges form a crucial part of its infrastructure. There are about 1,27,000 such bridges spread over a large geographical area. 40% of these bridges are over 100 years old [1] and many are in a weak and distressed condition. Also with the passage of time, the railway equipment loads have often increased significantly above the original bridge design values. It is not uncommon to hear of a major accident every few years due to collapse of a railway bridge.

To ensure safety of passengers and prevent loss of national property, replacement of all the old bridges with new ones is the best solution but it is a time and cost involving exercise. The other feasible solution to the problem is safely extending the life and maximizing load ratings of such railway bridges, while maintaining ongoing, uninterrupted train operations. This will help in replacement of old bridges in a progressive manner. This solution however involves developing and maintaining an effective bridge inspection program to monitor the health of bridges.

Currently structural engineers make use of wired systems to monitor the health of these bridges. The equipment they use is expensive and quite bulky. It takes days [2] to deploy such system. Also to operate the system requires expertise in technically trained manpower. These systems have also other maintenance problems as associated with any wired solutions. All these factors results in infrequent monitoring of these vital infrastructures.

Keeping in mind above problems and large number of railways bridges that need to be monitored, we need an automated approach to keep track of bridges' health. The solution should be simple not requiring much expertise, easily deployable within no time and has minimal maintenance requirements.

1.1 Problem Statement

We state the thesis problem as:

> *Design and develop an automated, easily deployable sensor network based structural health monitoring system, which acquires the vibration data of a remotely located railway bridge and reliably transfer it to the central repository*

Further analysis of the problem points to the following requirements;

1. The data acquired should have high fidelity.
2. The data should be acquired for some minimum duration as per requirement of structural engineers.
3. The data collected from various points need to be time synchronised, within a certain error bound.
4. The system should require minimum maintenance.

1.2 Challenges

In order to provide the solution meeting all the requirements as mentioned above, there are a lot of challenges involved.

- Conserve power: The sensor nodes are required to be deployed in remote location where there is no source of auxiliary power. These nodes are generally powered by batteries whose capacity is not very huge. Hence prolonged operation of these nodes with minimum maintenance is a challenge. We need to conserve the power of these nodes by making them do sleep wakeup with a low duty cycle.
- Low power accelerometers: We need to make use of accelerometers which provide high sensitivity and resolution and at the same time consume minimum power. Also they should be easily interfaced to rest of the hardware (sensor mote).
- Event detection: Nodes need to be awake at the time when the train arrives for doing the vibration measurement. But the arrival of trains is unpredictable. So this demands a reliable event detection mechanism.
- Mobile data transfer: We are required to automatically transfer the vibration data to central repository from the bridge location where there are is generally no network connectivity to rest of the world. This demands considering the approach of transferring the data to mobile train itself. But here there is a problem of limited contact duration between the train and nodes on the bridge. Hence we need to reliably transfer the data to train within this limited contact duration.

- Keeping nodes connected: With the passage of time some of the nodes may fail. The system should function with the remaining nodes. Also there must be a provision to replace the non-functional nodes with new ones.

- Limited capabilities of the platform: The sensor nodes we use have limitation in terms of processing power, storage and radio range. We have to develop our application keeping in mind these limitations.

1.3 Thesis Contributions

In this thesis we provide solution to the problem which was defined, meeting all the challenges listed earlier. We would like to mention here that we used routing, time synchronisation and event detection components from the master's thesis of Phani Kumar Valiveti [3]. Our major thesis contribution is as listed below:

1. Design and implementation of DMA based data acquisition system which provide high fidelity

2. Design and implementation of application specific reliable transport protocol for transfer of data both in static and mobile mode

3. Design and development of data analysis tools, meeting the requirement of structural engineers

4. Design and development of debugging tool which is used for debugging the BriMon system both in short term and long term monitoring mode

5. Study of compression techniques for their applicability in BriMon

6. Integration of above elements with the elements developed by [3]

1.4 Organization of the Report

The rest of the report is organised as follows. Chapter 2 presents the details and design of the application. Chapter 3 presents the design details and implementation of the transport protocol. In Chapter 4 we discuss all the experiments along with the results which we did during the development of the application. In Chapter 5 we present the past work done on

the application and then we present the related work in the application design and transport protocols. In Chapter 6 we conclude and define future scope of our work briefly.

CHAPTER 2

Application Details and Design

In this chapter we present the application details and design of our bridge monitoring system which we call as BriMon. The system can be used for both short and long-term railway bridge monitoring. Short term monitoring here implies monitoring the vibration behavior for a short duration say for some hours or a day. Long term monitoring here implies monitoring the vibration behavior for a long duration say for months or years. Long-term monitoring is the main focus in our thesis.

To begin the chapter, we will briefly talk about structural health monitoring and wireless sensor networks so as to make the reader comfortable in understanding the rest of the report. In the next section we will discuss the hardware platform we have used for the application development. Next we will cover the application details and then we will be discussing the long-term monitoring design. In the next sections we will be discussing the data acquisition system, data compaction and compression, and data organization respectively. Toward the end sections of the chapter we will discuss the debugging, and data analysis tools respectively.

2.1 Background

This section provides the necessary background to the reader. Here we will first briefly talk about structural health monitoring and then we will talk of wireless sensor networks.

Structural Health Monitoring (SHM):

Structural health monitoring implies finding out the present state of the structure directly or indirectly so as to classify it as safe, repairable, or unsafe for usage. Early signs of deterioration in the structure are often not seen visually because of some type of deck overlay, paint, protective wrap or composite make of the structure mask them. So we need the indirect techniques like vibration behavior analysis, for knowing the correct state of the structure. These analysis techniques are predominantly used by the structural engineers for studying the behavior of structures since these provide them deep insight into the structure's current state. Accelerometers are commonly used to pick up the vibrations from different locations on the structures. There are three types of vibrations which are of interest to structural engineers. These are namely

- *Forced Vibrations:* These are the vibrations which are induced into the structures with the help of external source like making use of mechanical shakers, movement of vehicle over the structure etc. The force causing vibration is constantly applied to the structure while it is vibrating.

- *Free Vibrations:* These are the vibrations of the structure caused by the residual energy in the system after the external source of vibration has been removed.

- *Ambient Vibrations:* These are the vibrations which are caused in the structures because of ambient sources like wind

Wireless Sensor Network:

A wireless sensor network (WSN) is a network made of many small independent sensor nodes. The sensor nodes, are self-contained units consisting of a battery, radio, sensors, and a minimal amount of on-board computing power. A sensor node is commonly called as *mote*. Common example of motes are Mica2, MicaZ [4], telos, telosb and tmote [5].

These motes apart from providing on board sensors for temperature, humidity etc also provide external IO ports through which we can acquire other analog signals like vibrations picked up by the accelerometer sensor. The on board analog to digital convector convert the analog signal to digital signal. These motes today being less expensive and rich in features as compared to computers are being used in numerous civil applications. Some of these application have been discussed in Section 5.2.

TinyOS [6] is the most commonly used operating systems for sensor mote platforms. It is developed at University of California at Berkley (UCB) for programming sensor nodes. It has a component based architecture where only application specific components get compiled and transferred to the nodes during programing. Contiki [7] and MANTIS [8] are some of the other operating systems available for the motes. We use TinyOS for our application development.

2.2 Hardware Platform

Having got some idea of structural health monitoring and sensor networks, let us now discuss what all hardware we have used for development of our application. We will first

start with the tmote which we have used as our platform and then will be talking about the accelerometers used.

Tmote:

The tmote from tmote-sky [5] is one of the latest offering from Mote-IV 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, which is an offering from CrossBow Technologies [4]. We choose tmote-sky as our hardware platform primarily because it was successfully used in previous works done on our type of application [9, 10]. Figure 2.1 shows the tmote.

Some of the important features of this platform are as listed below. Further details of the hardware can be found in its data sheet at [5].

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver
- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- 1 MB external flash for data storage
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- 16-pin expansion support and optional SMA antenna connector
- TinyOS support : mesh networking and communication implementation
- Complies with FCC Part 15 and Industry Canada regulations
- Environmentally friendly complies with RoHS regulations

Accelerometers:

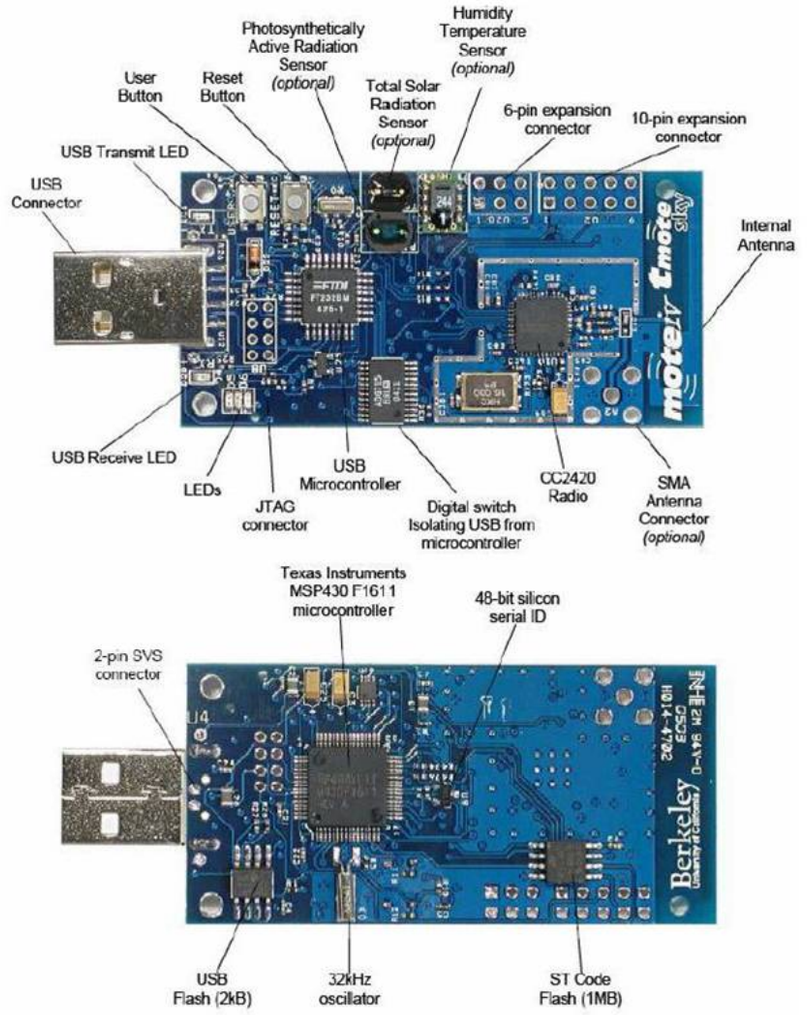

Figure 2.1 Tmote (Source: [5])

The choice of the accelerometer is crucial for 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.

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

- The resolution of the accelerometer

- The sensitivity of an accelerometer
- 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 350 g/$\sqrt{Hz}$ rms is bearable for our application [11]
- The output voltage of the accelerometer at 0g determines whether we require an additional circuitry to adapt this voltage to that of the motes
- The range of acceleration is the maximum and minimum acceleration that an accelerometer can measure. We need an accelerometer that can measure from -1.5g to +1.5g. [11].

We found that accelerometers ADXL203 [12] and MMA7260Q [13] are suitable for our application. The specification of these accelerometers are compared in Table 2.1. In our prototype implementation we have used ADXL 203 because of its better characteristics mainly in terms of sensitivity and noise immunity. Also these accelerometers were readily available to us during our developmental work.

	Accelerometers	
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	110μg/$\sqrt{Hz}$	350μg/$\sqrt{Hz}$
Bandwidth	0.5-2500Hz	XY-350Hz Z-150Hz
Acceleration range	$\pm$1.7g	$\pm$1.5g, $\pm$2g, $\pm$4g, $\pm$6g
Supply voltage	2 to 5 V	2.2 to 3.6V
Output voltage at 0g	2.4V-2.6V	1.485V-1.815V
Current consumption	700μA	500μA

Table 2.1 Accelerometers chosen by us : ADXL203 and MMA7260Q

We decided to use tmote power source itself to power the accelerometer. This greatly simplified the design of our hardware system. Now we had to manage only one power source. Though at 3V the sensitivity of ADXL203 is reduced to 560mv/g but this is adequate in our application. Also now the output voltages of the accelerometer at 0 g are reduced to half of what we get with 5volts power input. So now for maximum value of g:that is 1.7g the variation of voltage form 0 g level = 0.560*1.7 =0.952 volts. So the maximum output of the accelerometer = 1.5 + 0.952 = 2.452. Similarly the minimum output is =1.5-0.952 =0.548 volts. This fits very well in our design since the maximum reference voltage which can be provided to ADC on tmote is 2.5 Volts. We did experiments to do the calibration of this accelerometer. The details are given in Chapter 4 which covers all the experiments. Figure 2.2 shows the interfacing of the accelerometer to tmote.

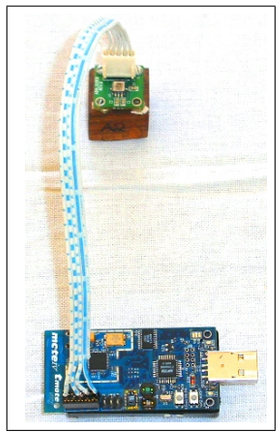

Figure 2.2 ADXL203 interfaced to tmote

2.3 Application Details

In this section we will cover the application details. To start with we will discuss what and where exactly we do the measurement on the bridge. Then we will discuss the short term and long term monitoring.

2.3.1 What and where to measure

A common design for bridge construction is to have several spans adjoining one another (most railway bridges in India are constructed this way). Depending on the construction, span length can be anywhere from 30m to about 125m. Most bridges have length in the

range of a few hundred meters to a few km. Accelerometers are a common choice for the purposes of monitoring the health of the bridges [14, 15]. We consider the use of 3-axis accelerometers which measure the fundamental and higher modal frequencies along the longitudinal, transverse, and vertical directions of motion. The placement of the sensors to capture these different modes of frequencies as well as relative motion between them is as shown in Figure 2.3

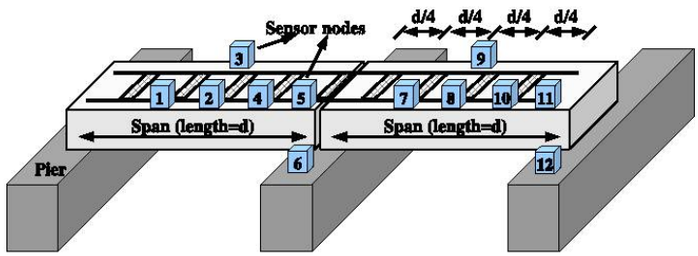

Figure 2.3 BriMon : Spans of Bridge

The data collected by the sensors on each span are correlated since they are measuring the vibration of the same physical structure. In some instances of bridge design, two adjacent spans are connected to a common anchorage, in which case the data across the two spans is correlated. For our data collection, we define the notion of a data-span to consist of the set of sensor nodes whose data is correlated. A data-span thus consists of nodes on one physical span, or in some cases, the nodes on two physical spans. An important point to note here is that collection of vibration data across different data-spans are independent of each other i.e. they are not physically correlated. Here after, when not qualified, the term span will refer to a data-span.

2.3.2 Short Term Monitoring

Many a times we are only interested in monitoring the behavior of a bridge for a short duration i.e. we are only interested to know the condition of the bridge say once in month or a quarter of year to assess the maintenance requirements. Such monitoring is also in great demand when a new bridge is under construction to validate the design parameters in terms of characteristic frequency and modal behavior of the structure. In this scenario we just need a portable bridge monitoring system which can be installed with out much hassle and then the measurements are carried out and data is analyzed then and there only. The first

version of BriMon caters to this requirement.

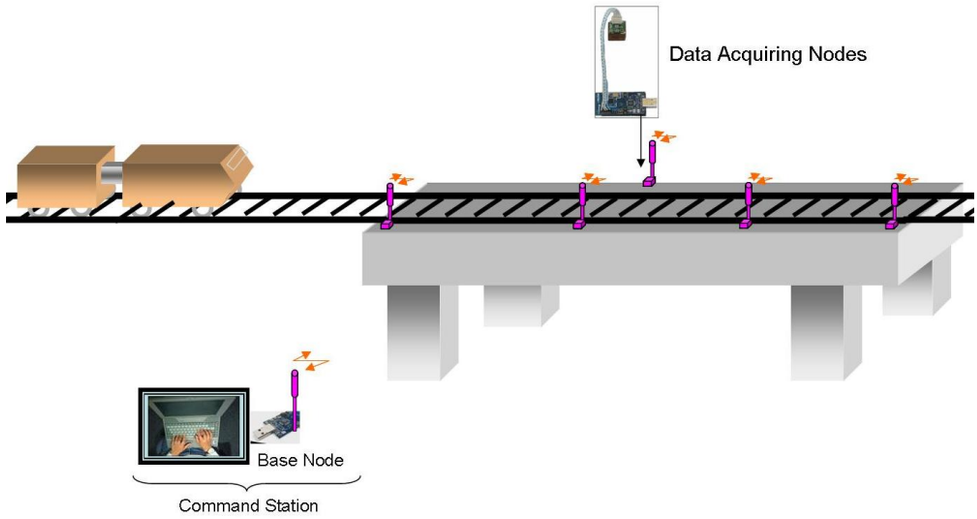

Figure 2.4 BriMon : Short Term Monitoring

Figure 2.4 shows the deployment of BriMon in short term monitoring mode. As shown in this figure the data acquisition nodes are deployed on the bridge at the desired location. Each data acquisition node consist of an MEMS based accelerometer interfaced to it. Accelerometers are used for the measurement of both forced and free vibration of the bridge. These accelerometers are fixed to desired locations on the span of the bridge. Also each node is connected to an 8dbi omni directional antenna in oder to attain good radio range. These data acquisition nodes could form a single or multi hop network. One of the nodes among the data acquisition nodes is called as header node. In the figure there is a command station which consists of laptop and a node which is connected through USB port to it. This node is called as base node. The base node is also connected to the external 8dbi omni directional antenna. The command station is operated by the user and it is established near the bridge. The station runs a java based, console application which is used to issue the commands to the base node.

Figure 2.5 shows the console of command station. The base node sends the commands to the header node which forwards the commands to the remaining data acquisition nodes. To start the monitoring process, the user waits for the train to come on to the bridge. When the train arrives on the bridge, he issues a command to start the data acquisition process by the data acquiring nodes. The nodes acquire the vibration data for a minimum duration of

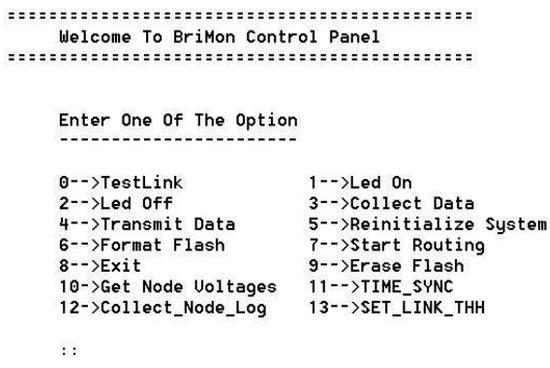

Figure 2.5 BriMon : Command Sation Console

40 seconds. The reasons for choosing this minimum duration is explained later in Section 2.4 which covers the application design. Once the data acquisition by nodes is over, the header node intimates the base node that data acquisition is over. The user then issues command to the header mote to gather the acquired data from all the nodes. The header node gathers the data from all the nodes one by one. Once it has gathered the data, it again intimate the base node that data has been gathered. On this intimation, the user issues the command to download data from the header node to lap top via base node. Thus this way the complete data is downloaded to the laptop. Now on the laptop the user make use of data analysis tool, the details of which are given in Section 2.9 to analyse the data. Once the data has been acquired, the entire system can be dismantled easily.

2.3.3 Long Term Monitoring and challenges

When we need to monitor the structural behavior of bridge for a longer duration i.e. to say for months or years together, there exists the requirement of an automated system which can monitor the bridge and automatically transfers the monitored data to the data analysis centers with minimum maintenance requirements. This version of BriMon meets this requirement in an elegant manner. To facilitate ease of deployment, we choose to build this system based on battery operated wireless sensor nodes. This version of BriMon consists of several tens to hundreds of such nodes equipped with accelerometers, spread over the multiple spans of the bridge. The use of wireless and battery eliminates the hassle of having to lay cable to route data or power (tapped from the 25 KV overhead high voltage line if available) to the various sensors that are spread about on the bridge. Given the choice of a battery operated wireless nodes for BriMon, a key parameter that drives our design is low

energy consumption, so that the maintenance requirements of BriMon (visits to the bridge to change battery) are kept to a minimal. Significant challenges which arise in this version of the BriMon are as listed below.

- The nodes need to sleep (turn off radio, sensors, etc) most of the time to conserve power.
- The need to wakeup and get ready for taking accelerometer measurements when there is a passing train.
- The data collected by the nodes need to be automatically sent to the data analysis centers which are located far away from the bridge.

To meet these challenges this version of BriMon employs a novel event detection mechanism [3] to balance these conflicting requirements. Event detection mechanism consists of a beaconing train with mobile node (refer Figure 2.6) and high gain external antennae at the header nodes on the bridge that can detect the beacons much before (30s or more) the train approaches the bridge. This large guard interval permits a very low duty cycle periodic sleep-wakeup-check mechanism at all the nodes. Figure 2.6 shows the deployment of BriMon in long term monitoring mode. On detecting a train, the data acquiring nodes

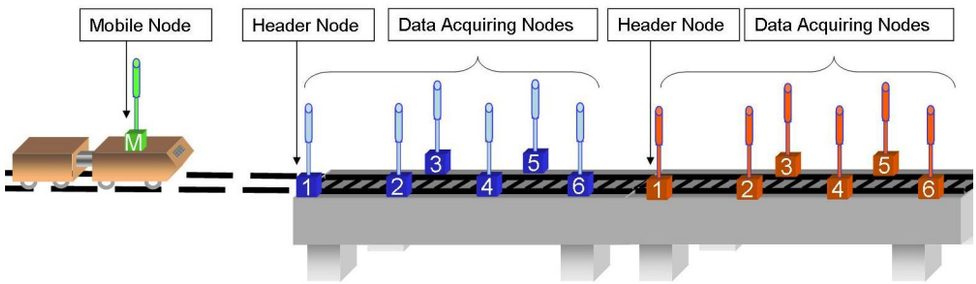

Figure 2.6 BriMon : Long Term Monitoring

collect vibration data. The collected data then has to be transferred to a central location. This data will then be used for analysis of the bridge's current health as well as for tracking the deterioration of the bridge structure over time. We use the passing trains themselves for the data transfer. The data transfer mechanism is also activated through the same event detection mechanism as for data collection.

2.4 Application Design for Long-Term Monitoring

In this section we will cover in detail the design of long term monitoring system. The long term monitoring of a bridge because of inherent challenges involved in it, is the main focus in our application. The short term monitoring is a subset of it. Other than event detection, sleep wakeup and mobile data transfer, everything we discuss about long term monitoring also applies to short term monitoring. Hence while discussing the application design we only talk of long term monitoring.

The prime goal in BriMon design is to have a system which requires minimal maintenance. This translates to two implications. (1) Once installed, the system should be able to run as long as possible without requiring battery replacements. (2) The data collected should be made available from remote bridges to a central data repository where it can be analyzed, faults detected and isolated. Important questions that arise in this context are:

- How do we balance the requirement for duty cycling with the fact that train arrivals are unpredictable?
- How do we transfer the data from the place of collection to a repository?
- How can we achieve scaling, for potentially long bridges?

We answer these questions as follows.

Event detection:

BriMon balance the requirements of having to duty cycle, and being ready when a train arrives, through event detection mechanism. BriMon use the fact that significant radio range is achievable with the 802.15.4 radios [16, 3], on using external antennas. An 802.15.4 node on the train beacons as it arrives, and is detected several tens of seconds in advance (before the train is on the span) by nodes on the span. This enables a periodic sleep/wake-up mechanism for the nodes on the bridge. The event detection mechanism is explored in detail in Phani Kumar Valiveti's thesis [3].

Data transfer to Data Analysis Centre :

In BriMon, we use the passing train itself for transferring the data collected. We call this data transfer as mobile data transfer. The data is then ultimately delivered to data analysis

centre. This could be done, via say an internet connection available at the next major train station. The same event detection used for data collection is also used to trigger the data transfer to a moving train. A subtle point to note here is that the data collected in response to a train is actually conveyed to the central repository via the next oncoming train.

Data span as an independent network:

One fundamental design decision we make in BriMon is to treat each dataspan as an independent network. This is possible primarily due to the fact that the data from each data-span is independent physically (different physical structures). This also fits in well with our mobile data transfer model. The alternative here is to treat the entire bridge (including all data spans) as a single network. We rejected this approach since there is no specific reason for the data-spans to know about one another or inter-operate in any way. Having a smaller network simplifies protocol design, and enables much better performance. A designated node which we call as header node (refer Figure 2.6) on each span gathers all the data collected by the different sensor nodes on that span. It then transfers this data to the node on the moving train. We make different spans operate on different independent channels, so that the transfer on each span can proceed simultaneously and independently.

The event detection as well as data transfer bank on two underlying mechanisms: time synchronization and routing. So there are four main functionalities in BriMon: (a) event detection coupled with periodic sleep/wake-up, (b) mobile data transfer, (c) time synchronization, and (d) routing. Our fellow student, Phani Kumar Valiveti worked on event detection, time synchronisation and routing components of the BriMon in his thesis work [3]. We have used these components as it is in the application. We will now talk of mobile data transfer.

Mobile Data Transfer:

When the train arrives, the event detection mechanism triggers data collection at the nodes. After the data collection phase, this data must reliably be transferred from the (remote) bridge location to a central server that can evaluate the health of the bridge. Most sensor network deployments today do this by first gathering all collected data to a sink node in the network. The data is then transferred using some wide-area connectivity technology to a central server [17, 18, 19, 20, 22, 14, 21].

We reject this approach for several reasons. First, in a setting where the bridges are spread over a large geographical region (e.g. in India), expecting wide area network coverage such as GPRS at the bridge location to transfer data will not be a valid assumption. Setting up other long-distance wireless links (like 802.11 or 802.16) involves careful network planning (setting up towers for line of sight operation, ensuring no interference, etc.) and adds further to maintenance overhead. Having a satellite connection for each bridge is too expensive a proposition. Manual collection of data via period trips to the bridge also means additional maintenance issues.

One more important reason is the following. Recall that we can have bridges as long as 2km, with spans of length 30-125m. This means that overall we could have as many as about 200 sensors placed on the bridge, at different spans. Even if we were to have somehow have a long-distance Internet link at the bridge location, we would have to gather all the sensor data corresponding to these many sensors at a common sink, which has the external connectivity. Such gathering has to be reliable as well. The total data which has to be gathered would be substantial for each measurement cycle. Doing such data gathering over 10-20 hops will involve a huge amount of transfer time and hence considerable energy wastage. Having a large network has other scaling issues as well: scaling of the routing, synchronization, periodic wakeup,etc. Large networks are also more likely to have more fault-tolerance related issues.

Keeping the above two considerations in mind, we consider a mobile data transfer mechanism, where data from the nodes is transferred directly to the train. This then allows us to partition up the entire bridge into several data-spans with independently operating networks. In each data-span, the header node gathers data from all the nodes within the data-span. It then transfers the data gathered for the dataspan onto the train. Since we are dealing with networks of size at most 12 nodes, data gathering time is small, and so is the transfer time to the train itself. We eliminate the need for a back-haul network in such an approach.

One subtle detail to note here is that we seek to transfer the data collected for one particular train, not to the same train, but to a subsequent train. We make use of Tmote-sky's sensor motes [5] in our application. On the Tmote platform,it is not possible to use the radio in parallel with the data collected being written to the flash since there is a common bus. We then need to have different trains for data collection and the data transfer. The

additional delay introduced in such an approach is immaterial for our application.

The transport protocol we use for the data transfer is quite simple. The details of the protocol are given in Chapter 3. The same protocol is used for data transfer from the header node to the mobile node in the train and for the data gathering as well: that is, for getting the data from the individual nodes to the header node.

There are a few challenges however in realizing our mobile data transfer model. One, while one header node is transmitting data of its dataspan to the train, nearby header nodes would also be within contact range of the train and would be transferring their data. This would lead to interference if sufficient care is not taken. In fact, the synchronization,routing, and other operations of multiple spans could interfere with one another. We address this issue by using multiple channels, as we explain below.

Using multiple channels:

We take the approach of using the 16 channels available in 802.15.4. There are at least eight independent channels available [23]. We could simply use different channels on successive spans, and repeat the channels in a cycle. For instance we could use the cycle 1, 3, 5,7, 9, 11, 13, 15, 2, 4, 6, 8, 10, 12, 14, 16. This would ensure that adjacent channels are at least 7 spans apart, and independent operation of each data-span would be ensured. Note that the train needs to have different motes, operating in the appropriate channel, for collecting data from each data-span.

Reserved channel for event detection:

The above approach works except for another subtle detail. We designate that the channel for event detection mechanism for all dataspans is the same, and reserve one channel for this purpose. So as the train approaches, the radio within it can continuously beacon on this reserved channel without having to bother about interfering with any data transfer or other protocol operation within each data span. The head nodes of each span listen on this channel for the oncoming train, and switch to the allocated channel for the data-span after some time duration.

Cycle of operation:

Having discussed the design details let us see how one cycle of operation works in long term monitoring mode. As soon as nodes boot up, the routing gets activated and the routing tree is formed that is each node knows its parent and its children. After routing, nodes get synchronised. Header nodes issue commands to the data acquiring nodes to do sleep wakeup. Also before going to sleep, all the header nodes switch to the reserved channel.

Now when the train arrives, event detection takes place. The header nodes on each dataspan switch to their respective channels and informs data acquiring nodes to remain awake and acquire the vibration data at a particular global time. The nodes then acquire the data. The header nodes then gather the data from the nodes in their respective dataspan. The nodes then again synchronise and again do periodic sleep wakeup. When the next train comes, again event detection takes place. The header nodes switch to their respective channels and transfer the data to their respective mobile nodes.

Figure 2.7 shows the deployment of BriMon for longterm monitoring of railway bridge at remote location.

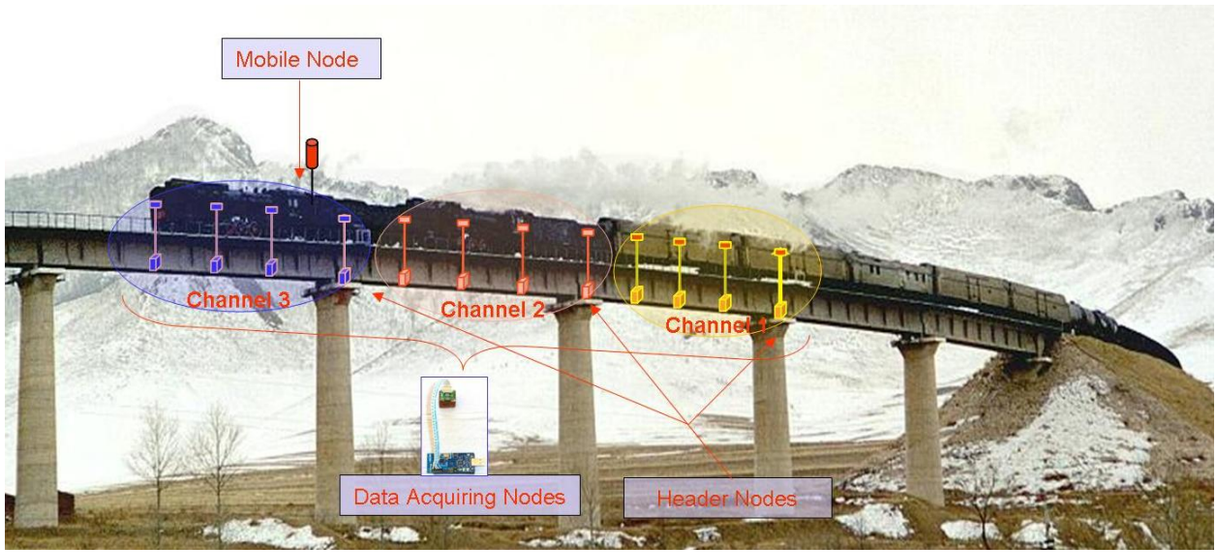

Figure 2.7 Deployment of BriMon for long term monitoring of a railway bridge (only depiction)

2.5 Data Acquisition System

In this section we will discuss the data acquisition system of the BriMon in detail. Here we will first talk of the application requirements and the associated constraints. Then we

will discuss the DMA based data acquisition system of BriMon.

2.5.1 Application Requirements and Constraints

In our application we are interested in measuring the forced and free vibration of the bridge. When a train is on a span, it induces the forced vibrations. After the train passes over the bridge, the structure vibrates freely with decreasing amplitude till the motion stops. Structural engineers are mostly interested in the natural and higher order modes of this free vibration as well as the corresponding damping ratio. Also of interest sometimes is the induced magnitude of the forced vibrations. For both forced as well as free vibrations, we wish to collect data for a duration equivalent to about five time periods of oscillation. The frequency components of interest for these structures are in the range of about 0.25 Hz to 20 Hz [14, 15, 21]. For 0.25 Hz, five time periods is equivalent to 20 seconds. The total data collection duration is thus about 40 seconds (20 seconds each for forced and free vibrations).

Before we proceed ahead let us now see if the 12 bit ADC provided on tmote motes is suitable for analog to digital conversion of signal from the accelerometer. Our application requires resolution of 10 milli g [11]. Note that the acceleration is measured in units of g where g = 9.8 m/s^2. Though we can make use of any accelerometer for this resolution, we make use of ADXL 203 [12] for acquiring the vibration data in our prototype. The analog signal from this accelerometer has resolution of 1 milli g. So the after the data conversion by ADC we should get at least 10 mg resolution as per the application requirement. Let us now see what resolution of digital data we get with the ADC on tmote. The full scale voltage used for ADC on tmote is 2.5 volts. So the resolution provided by the ADC is = 2.5/4096 = 0.00061 volts per division. We power the accelerometer from 3 volts source. At this voltage its sensitivity is 560 milli volts per g So one division of ADC in terms of g is = ≈0.00061/.560 ≈.001g. This value is much lower than 10 milli g resolution required by the application. Hence the ADC is suitable for analog to digital conversion conversion of signal from the accelerometer. We will now estimate the data collected by each data acquiring node.

Each node needs to collect accelerometer data in three different axes (x, y, z). The sampling rate of data collection is determined by the maximum frequency component of the data we are interested in: 20 Hz. For this, we need to sample at least at 40 Hz. Often

oversampling (at 400 Hz or so) is done and the samples averaged (on sensor node itself before transfer)to eliminate noise in the samples. But the data that is finally stored/transmitted would have a much smaller sampling frequency which we set to 40Hz in our case. Each sample is 12-bits (because of use of a 12 bit Analog-to-Digital converter). The total data generated by a node can be estimated as: 3channels12bits40Hz40sec = 57.6Kbits. There are an estimated 6 sensor nodes per span, and a maximum of 12 nodes per data span. Thus the total data we have per data-span per data collection cycle is a maximum of 57.612 /8= 86.4 KBytes.

Since the data within a data-span are correlated, we need time synchronization across the nodes, to time-align the data. The accuracy of time synchronization required is determined by the time period of oscillation above, which is minimum for the highest frequency component present in that data i.e. 20 Hz. For this frequency, the time period is 50ms, so a synchronization accuracy of about 5ms (1/10 of the time period) should be sufficient. Note that this is of much coarser granularity than what is typically described in several time synchronization protocols (e.g. [24]). We have used simple light weight protocol [3] for time synchronisation.

Let us now talk of the constraints which we face in meeting the above requirements while implementing our application on Tmote motes. There is only 10KB of RAM available on these motes which can be used as buffer space. This implies that we can not acquire above quantity of data in one go. To overcome this problem we need to make use of both flash and RAM for acquiring the data. One method of doing this is, we start acquiring ADC data into maximum permissible size RAM buffer and when this buffer is filled up, we write the data from this buffer into flash. After writing data to flash we again start acquiring the data. This process is repeated till the time complete data is acquired. Obviously this method has a flaw. There is loss of ADC data when we are writing data to the flash since writing to flash takes time and the buffer is not free at this time to acquire ADC data. To overcome this problem we create create two equal sized RAM buffers of maximum permissible size. We acquire the ADC data into RAM buffers one after the other. Once a buffer is filled, the second buffer is used to acquire the data from ADC. The data of previous buffer at this time is written to the flash. The process continues till the time complete data is acquired. To make sure that no samples are lost even in this case when the data is written to the

flash (since the micro controller will be busy), we make use of DMA controller to handle the data coming from the ADC. Further details of DMA based data acquisition are given in the subsequent subsection.

The second constraint is that, there is only one ADC on tmote. So for acquiring data from more than one axes requires multiplexing of this ADC. We found that the switching time from one axis to another takes more than 10 ms. This clearly results in loss of data when we are sampling at high rates as mentioned above. To overcome this problem we recommend that either we make use of separate hardware for ADC so that there is one ADC per axis to be sampled or we only sample one axis at a time if we use the ADC present on the Tmote. In our implementation at present we are using the latter approach.

2.5.2 DMA based Sampling

As said earlier we make use of both, RAM buffers and flash for handling the sampled data. When one of the buffer is filled with the sampled data, its data is required to be written to the flash. At the same time when this data is being written to the flash, the ADC data has to be acquired into the second buffer otherwise there will be data loss. Since the micro controller cannot handle both these operations simultaneously, we need to make use of DMA controller which is available on tmotes. Also when ADC converts an analog sample into digital form, there is an interrupt raised. Under normal implementation, micro-controller handles this interrupt to transfer data from the ADC conversion register to RAM buffer. So if the sampling rate is too high, the interrupt load produced will overload the system, preventing it from doing any thing else [25]. This will also result in the jitter in the samples.

So we make use of DMA controller to coordinate the transfer of samples from ADC conversion registers to sequential words in RAM. Instead of generating an interrupt after each sample conversion, the DMA generates an interrupt each time it fills a RAM buffer with data. We make use of two RAM buffers for data acquisition. To start with, first buffer pointer is passed to the DMA. So when this buffer gets filled up, DMA controller raises interrupt, at this instant second buffer pointer is passed to the DMA controller so that there is minimal or no loss of data. When the data is being acquired into the second RAM buffer, the data from previous buffer is written into flash by the micro controller. The process repeats till the time complete data is acquired by the node.

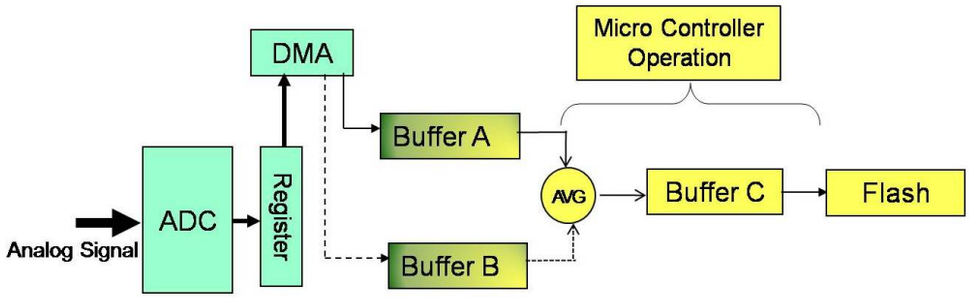

Figure 2.8 DMA based Sampling

Figure 2.8shows how we have implemented DMA based data sampling in our prototype. As seen in the figure, the two RAM buffers used for sampling are buffers A and B. In our implementation we do ten point averaging of data points acquired in the RAM buffer before writing the data to the flash. Also though the ADC we use is of only 12 bits, the samples we get after conversion are 16 bits each. So we convert these samples into 12 bits before writing them to flash. We call this conversion as compaction.

To make sure that empty RAM buffer for sampling is readily available to DMA controller when it raises an interrupt, we make use of third small RAM buffer (named as C in the figure) for temporarily storing the averaged data before writing it to the flash. So this way we are able to simultaneously acquire the data and also write it to the flash.

We tested our prototype with and without DMA based sampling. Refer Figure 2.9. It clearly shows that without using DMA there is unacceptable loss of data samples. In the example shown this figure the loss was for a duration of about 150 ms.

2.6 Data Compaction and Compression

In sensor networks the aim is always to reduce the energy consumption as much as possible so that the life of the network is enhanced. Radio transmission is one of the major source of energy consumption. So in such system we always try to compress (loss less) the data to be transmitted as much as possible. In our application also as mentioned earlier we acquire the data at much higher sampling rate than theoretically required. Then we do ten point averaging of the data to reduce the noise. We further reduce the data size by compaction as described below.

After the averaging operation, the data is placed in buffer C of the data acquisition system. Refer figure 2.10. In this figure at t1 time we get the data into buffer C. Here we see

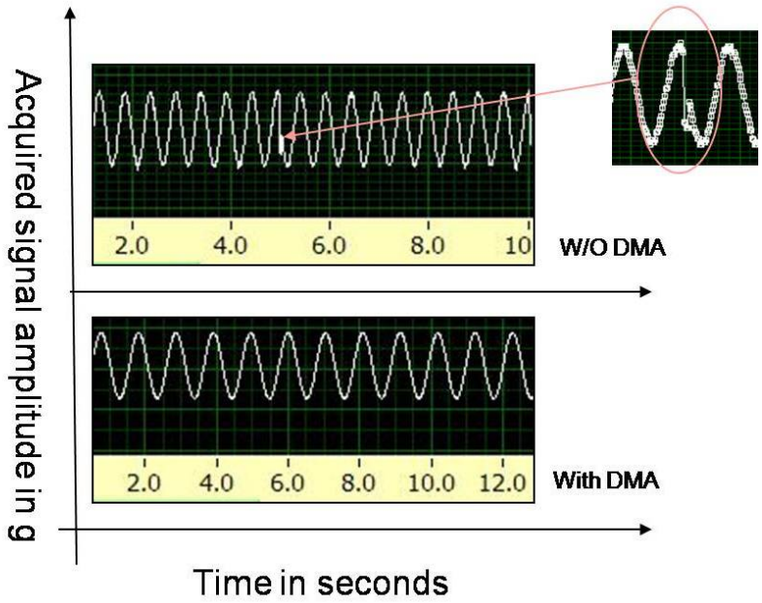

Figure 2.9 DMA based Sampling (Validation)

that in second byte of every data point, upper nibble is unused. We are required to get rid of this unused space. Simplest method to do so would have been to use bit level operations to get rid of this unused space but the bit level operations are not very well supported in tinyOS. Hence we adopt the method as described hereafter. We right shift the third nibble of every odd data point by four bits and at the same time copy the third nibble of every even data point into third nibble of preceding data point. This operation is as shown in figure at t2 and t3 respectively. After this operation we have data arranged as shown in figure at t3. Now we use normal memcopy operation to place all the data related bytes together and hence get rid of free space. The final compacted data is as shown in figure at t4.

To further check possibility of compressing the compacted data, we studied some of the lossless compression techniques. We in fact wrote code to test these compression techniques on sample data which we had collected during an outdoor experiment on the bridge. The compression techniques we studied are as under

- Delta Encoding
- Run Length Encoding
- LZW

Delta Encoding :

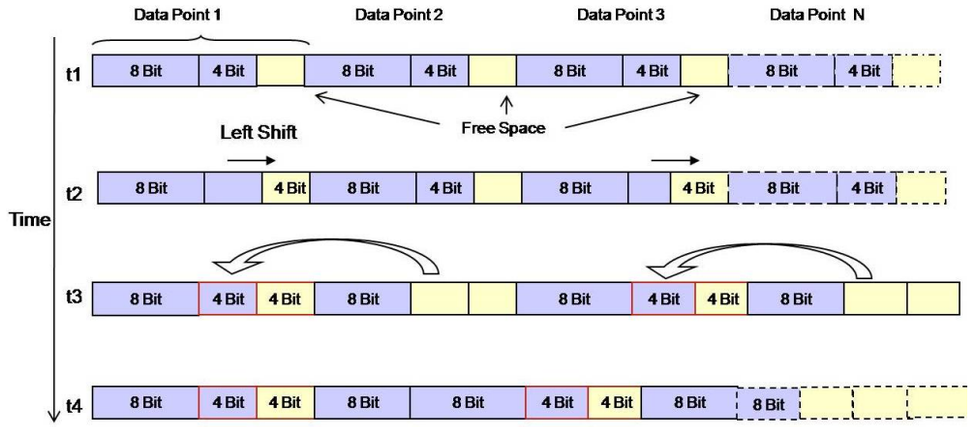

Figure 2.10 Data Compaction

In delta encoding, the first value in the encoded file is the same as the first value in the original file. Thereafter each sample in the encoded file is the difference between the consecutive samples. If the signal is low varying then this encoding provides compression up to almost 50%. We in fact implemented this scheme on tinyos platform and found that if the signal is of very low frequency that is less than 5 Hz then we can get good compression results. We got compression upto 56%. In our application however the frequecy band is from 0.25 Hz to 20 Hz hence at higher frequecies near 20 Hz there is hardly any compresion of data (5 to 7 %). So we ruled out this technique.

Run Length Encoding:

In this encoding we check if there is repetition of data points. If data points are repeated, then those number of data points are just replaced by count value and the value of the data point. It implies that we require a separate variable to represent the count value. In our application again because of varying nature of the signal, the number of repletions especially after compaction are very less. We in fact found that most of the time file was inflated than compressed. So we again ruled out the possibility of using this compression technique.

LZW Encoding:

LZW is a lossless, dictionary-based algorithm that builds its dictionary as data is read in from the input stream [28]. At the start, the dictionary is initialized to 256 entries, one for each eight-bit character. This algorithm has no transmission overhead and is computationally

simple. Since both the sender and the receiver have the initial dictionary and all new dictionary entries are created based on existing dictionary entries, the recipient can recreate the dictionary on the fly as data is received. To decode a dictionary entry, however, the decoder must have received all previous entries in the block which was compressed.

Among all other dictionary based algorithms it is best suited for sensor networks [26]. This algorithm however for implementation requires lot of RAM. In our application we can however implement this compression technique after the complete data has been acquired and stored in flash. It is so because then all the RAM buffers are free and they can be used for the compression purpose. We implemented this algorithm as per [26] in C keeping in mind the memory constraints we will have in its actual implementation on tmotes. Due to RAM size constraint on our platform, we choose each encoded output size as 9 bit value. For this size of encoded output, the dictionary size need to be at least 2^9=512 entries. Each entry in dictionary is of 18 bits. So to store the dictionary we require at lest 1152 bytes (512*18/8). We used hashing algorithm as mentioned in [27] to build the dictionary since it helps in efficient operation of the algorithm while doing the comparisons of input stream characters with the existing codes in the dictionary. Now for implementing the hashing algorithm, it is required that dictionary size should be slightly greater than number of possible encoded outputs. So we used fixed dictionary size of 520 entires ($520 > 2^9$). Since in our application at the maximum we can read a block of data, we selected the input byte stream as equal to one block (2240 bytes). while compressing the actual bridge data we got maximum compression of 19%. Hence we feel that this compression technique can be used in our application to further comprees the compacted data.

2.7 Data Organization

Let us now see how the data is organized at the application layer of BriMon. As said earlier each data acquiring node does the data acquisition and due to limitation of RAM, we acquire data into buffers A and B alternatively (refer figure 2.8). This goes on till the time complete data is acquired. Every time when the data from the RAM buffer C is stored on the flash, we say that a 'File' is acquired. The number of files acquired at all nodes is the same because the amount of data generated for an event at every node is the same. When we store the file on flash, we also store the following information with respect to the file in

the RAM. We call this information as attributes of the file.

- node id
- file no
- Start address of the file in the flash
- Size of the file

After data gathering phase, the files attribute information is stored in flash. Once the data is stored on the flash as files, data is also read as files while the data is transferred from the node. So before reading the files, the files attribute information is read into RAM. We make use of same RAM buffer for data transfer as was used for sampling. In our implementation we make use of buffer A for data transfer. Since we do ten point averaging and compaction, we can read more than one file into the buffer at a time. The unit of data transfer at the application layer is block so the size of block is same as size of buffer A.

A point to note here is that when we receive the block at the application layer on header node or mobile node, the block is again required to be written to the flash. However at this time the file size is same as size of the block.

Also while data transfer we want to transfer information about the bridge on which measurement is being done and the train which triggered the data collection. For this purpose we make use of a structure with the following fields.

- Train id
- Bridge id
- Bridge Span No

After the data gathering phase, this information, is stored into the flash. So when the data is transferred from header node to mobile node, at the end of data transfer, we also reliably transfer this information to mobile node.

2.8 Debugging Tool

During the development of our prototype, we needed a tool to debug the system while testing it. So we came up with this tool. Later we found that same tool could be used to test or debug the system for functionality during actual deployment. The tool consists of command station and base node. Refer Figure 2.5. This figure shows the commands which we can issue from the command station. These commands are executed on the nodes and at the end of execution, we get the results back on the command station.

Let us as an example discuss how we can test the links which nodes form for packet error rates. We have developed a small module which we name as 'LinkTest'. This module is integrated with the BriMon. This module is used for testing the links which the nodes form. From the command station we issue command named as 'linktest'. Here in this command we specify the terminating nodes of the link in which we declare one node as sender and the other node as receiver. Also we specify the inter-packet pause. On reception of this commands, the sender node sends specified number of packets to the receiver node. The receiver node after receiving these packets calculate the packet loss and send the results to the command station. Like this we can test all the links.

Other command which is useful is to get the node voltages. We issue the command to get the node voltages of all nodes. Each node sends its node voltage to the command station. Table 2.2 shows all the commands and their respective purpose. We want to mention here that this tool was very helpful to us in development of our prototype.

2.9 Data Analysis Tool

In this section we will discuss the data analysis tool which we have developed for the data analysis of data collected form the nodes at the data analysis centre. This tool has been developed as per the requirement of structural engineers [11]. The tool was of immense help to us right from the beginning of our work since at every stage it was used for analysis of data. We tested this tool for correctness by using it for analysis of known waveforms and found that it works as per the requirement. This tool has been implemented in LabView version 7.1.

To start the section we will discuss the requirements of the user: that is structural engineers with respect to data analysis tool. Thereafter we will discuss the details of the

Command	**Purpose**
Test Link	For testing any link
Led On	Switch On LED on deployed nodes
Led Off	Switch Off LED on deployed nodes
Collect Data	Start acquiring data (Dummy or Real)
Transmit Data	Used to collect data from the specified node
Reinitialize System	To reboot the system
Format Flash	To format the external flash on nodes
Start Routing	To test the routing module
Exit	Not transmitted, used only to close the console
Erase Flash	To erase the external flash on nodes
Get Node Voltages	To get the node voltages
Time Sync	To time synchronise the nodes as per header node clock
Collect_ node_log	To collect the log from the specified node
Set_Link_THH	To set RRSI level for routing

Table 2.2 Commands and their Purpose

analysis tool.

User Requirements:

The following were the requirements structural engineers [11] with respect to data analysis tool:

1. The tool should have user friendly graphical user interface (GUI)
2. Provision of digital filter with an option to enable or disable it
3. Provision to select portion of waveform for data analysis
4. The tool should provide information both in time and frequency domain
5. It should pick up input data from file
6. Accept input data as compressed or uncompressed
7. The GUI should have view magnifying options
8. There should be provision to select the sampling rate
9. The dominant frequency components of the waveform should be displayed along with their amplitudes.
10. The maximum value of acceleration present in the signal should be displayed

Implementation:

We developed the tool keeping in mind all above requirements. Refer Figure 2.11. It shows how the data is acquired from the file and and what all operations are carried on this data. Here X[] represents the data array to be analysed.

Figure 2.12 shows the GUI of the analysis tool we have developed.

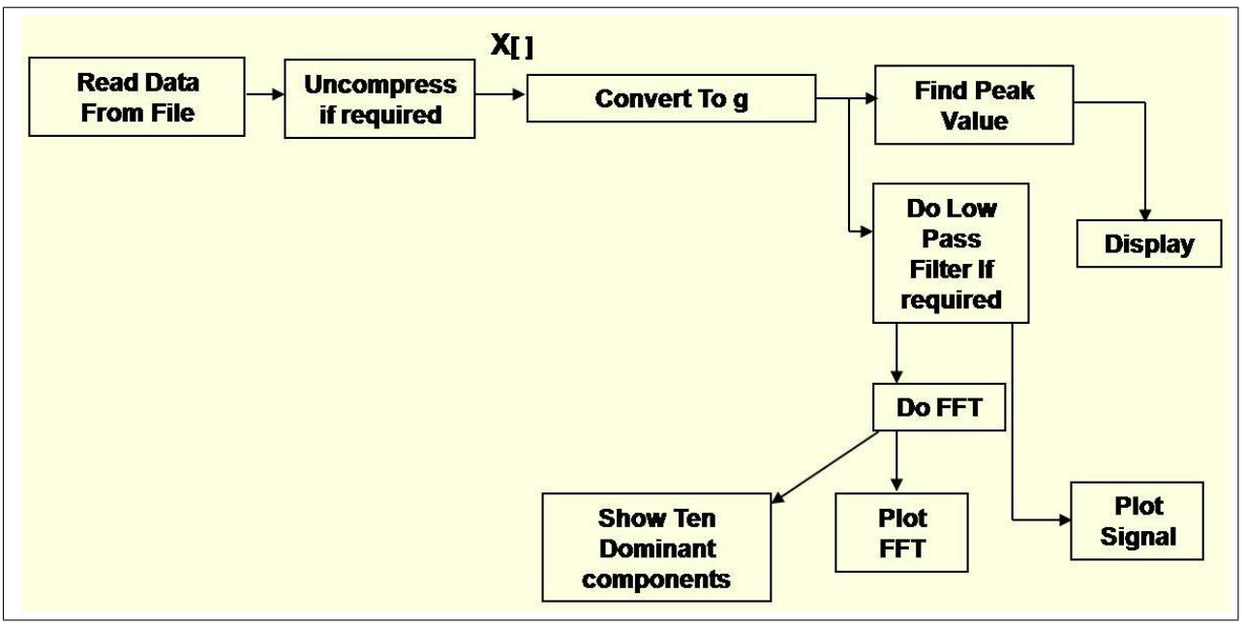

Figure 2.11 Block Diagram: Data Analysis Tool

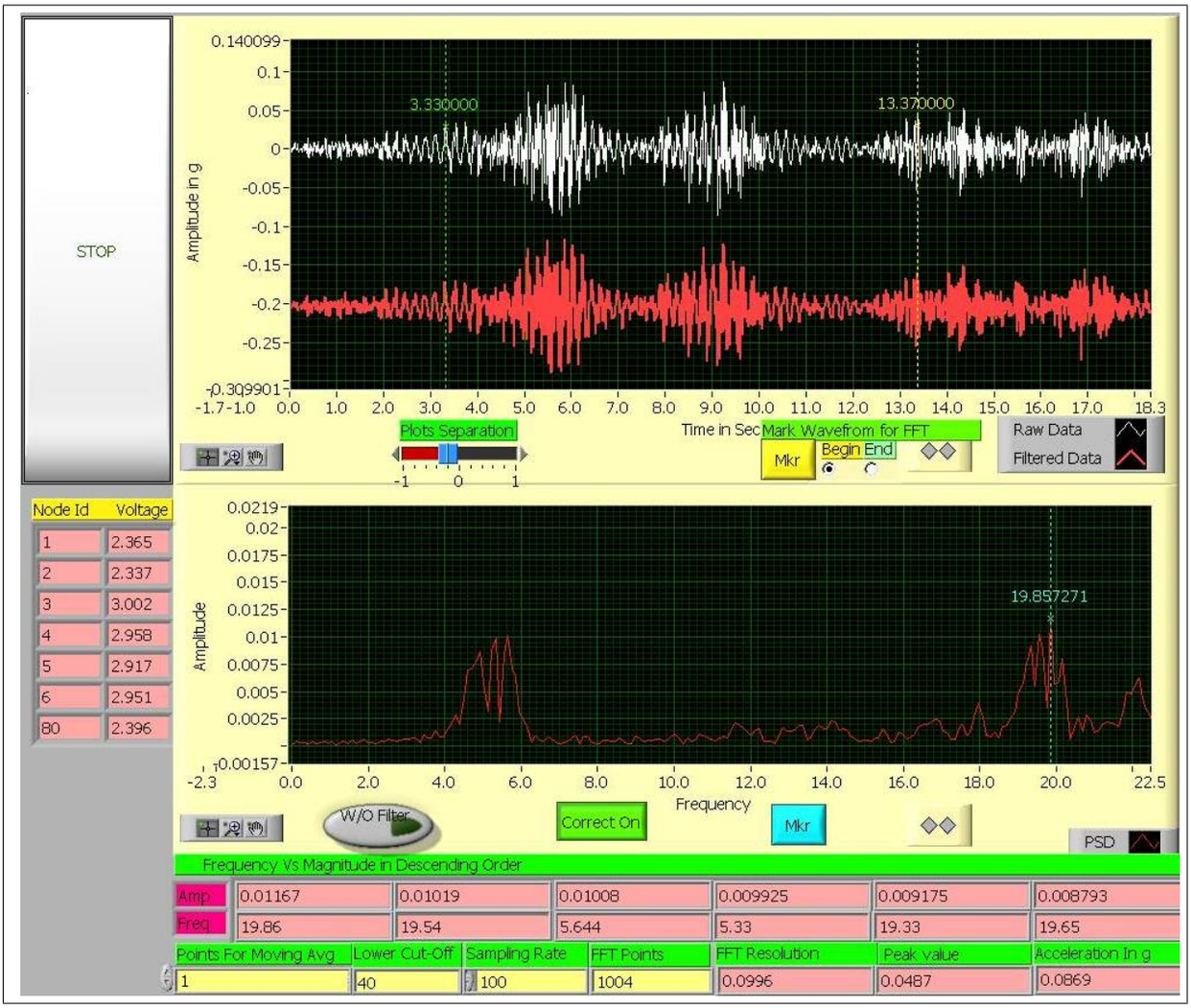

Figure 2.12 Data Analysis Tool:GUI

CHAPTER 3

Transport Protocol

In this chapter we will discuss the transport protocol which we have designed and implemented in our application. We will first talk of design requirements and the associated constraints. Then we will describe the protocol in detail and finally we will present its implementation details. We have done the evaluation of the protocol at Section 4.3 of Chapter 4 which covers all the experiments.

3.1 Design Requirements and Constraints

In this section we will discuss the design requirements and the associated constraints of the transport protocol which we have designed and implemented for reliable transfer of data. In our application, the protocol is used for reliable transfer of data at three places during the entire process of transfer of data from the bridge to the data analysis centre. Figure 3.1 depicts the architecture of our application. In this figure the nodes which are placed on the bridge are called as data acquiring nodes. Among these nodes, the node which is numbered as 1 is called as Header node. The node which is placed inside the train is called as Mobile node. At the data analysis centre we download data from mobile node to PC by making use of a node called as the Base Node.

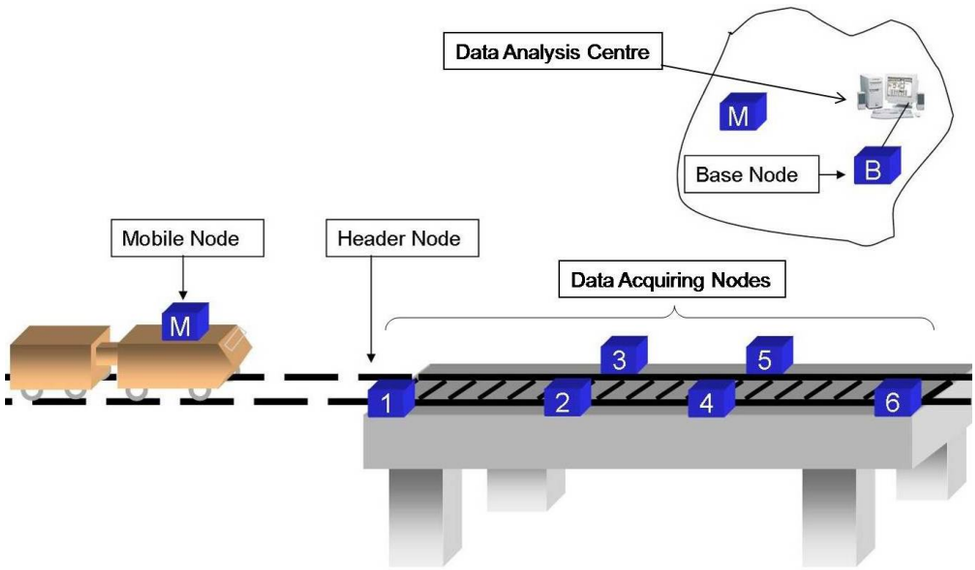

Figure 3.1 Transport Protocol: Data Transfer Modes

In the first place of data transfer, the protocol is used in the transfer of data from the data acquiring nodes to the header node. In the second case it is used to transfer the data from header node to mobile node. Finally it is used for downloading the data from the mobile node to the base node. The first case could involve reliable transfer of data from one node to another both in a single hop or over multiple hops depending upon the routing tree. In second and third cases the data transfer is only over single hop. Among all these cases the mobile data transfer is time critical in the second case since there is a limited contact duration between the mobile and header nodes and within which we have to complete the data transfer.

Let us now talk of the design constraints of the protocol. In our application we have used Tmote-sky's sensor motes [5]. These motes have some limitations which are as listed below.

- There is only 10KB of RAM available on these motes which can be used as buffer space.
- The flash and radio on this platform shares the same SPI bus.
- There is problem of flow control on these motes. That is sending rate is faster than achievable receiving rate.

We will now discuss how these limitations have an affect on our protocol design. The amount of data which we handle at each node is much higher than the available RAM. So it is clear that we can not transfer the entire data stored on the flash of the sender node in one go. Instead we need to transfer the data on chunk by chunk basis. We call chunk of data as *block*. We make use of RAM buffers of sufficiently large size both at the sender and the receiver nodes during the data transfer. These buffers are same as were used during data acquisition (refer Section 2.5). Each buffer can accommodate one block of data. The application layer transfers the data from nodes in terms of blocks. When it receives intimation from the transport layer that a block has been transmitted, it sends the next block. At the receiver when a complete block is received, it is passed to the application layer which then writes this block to the flash.

Due to sharing of SPI bus by flash and radio, there is need of intelligent arbitration between radio and flash. Though most of the arbitration is done by lower layers, we still

need to do some careful programming. It is required because we do not want any interruption while the block is written to the flash. So at the application layer, before beginning to write the block to flash, the radio on the receiver is switched off. The radio is again switched on after the block has been written to the flash.

While the block at the receiver is written to the flash, at the sender the next block is read into the RAM buffer. Thus a parallel flash write and read operation goes on at the receiver and the sender respectively. The next block read at the sender is then similarly transferred to the receiver. This way all the block are transferred from the sender node to the receiver node.

Let us now discuss the problem of flow control. While conducting an experiment involving transfer of data from sender node to receiver node, we found out that when the value of pause between consecutive packet transmissions was less than some certain value, there were packet drops happening at the receiver's radio buffer. When this pause interval was increased there were no drops happening. There were obviously some delays happening on the receiver side which were resulting in drop of packets. Further analysis, the details of which are given in Section 3.3 , proved that we do need to handle flow control while transfer of data between the nodes.

3.2 Protocol Description

In this section we will describe the protocol in detail, explaining how exactly the data transfer takes place. We will consider both single hop and multi-hop data transfer modes. The main emphasis during the discussion will be examining how reliable the data transfer is. That is what all actions are taken whenever there are packet losses during the data transfer. We will first examine the data transfer in single hop case and then we will discuss multi-hop data transfer case.

3.2.1 Single Hop Data Transfer

Mechanism of Data Transfer:

In single hop, there are only two nodes involved in data transfer. One of these nodes acts as sender and the other node acts as receiver. These nodes are referred as 'SNode' and

'RNode' respectively in rest of our discussion. In this case to start the data transfer, the application layer on SNode reads a block of data from the flash and sends it to the transport layer. The transport layer based upon payload size of the packet size used for data transfer, divides this block into n number of packets and sends them to RNode one after the other. On the SNode once a packet has been sent, the next packet is sent after inter packet pause of 'T_p' milliseconds. Inter packet pause means the duration between the 'SendDone' event of a packet send and the function call for send of next packet.

We use selective acknowledgment (SACK) based technique for data transfer instead of stop and wait kind of mechanism. The reason why we use this technique is that it performs better. The number of packet transfers required to get data to the destination is less in selective acknowledgment based mechanisms as compared to Stop and wait where there is requirement of an extra packet transmission for every packet received. In the case of SACK based mechanism only one packet is transmitted for n number of packet losses.

The SNode after the last packet is sent, starts a time-out timer which hereafter is referred to as 'TOut' timer. At the RNode if all the packets are received then it sends acknowledgment message to SNode. We call this message as ACK. The received packets on the RNode are then assembled into a block. This block is then passed on to the application layer. The application layer then immediately switches off the radio for reasons explained earlier and starts writing the block to the flash. After the block has been written to the flash, the radio is again switched on.

If some packets are lost during the data transfer, then the message which we call as NACK is sent to the SNode. This message contains the sequence numbers of all the lost packets. The SNode then resends all the lost packets. On receiving the ACK or NACK, the SNode immediately stops the TOut timer. On the SNode when the ACK message is received, the transport layer intimates the application layer that the block has been transmitted. If now there are still bocks left to be transmitted , the next block is sent to the transport layer and likewise the data transmission progresses.

In case the SNode does not receive ACK or NACK message and TOut timer fires, the sender node again resends the last packet of the block. The RNode whenever it receives the last packet of block, it calculates the number of lost packets. Based upon the number of packets lost it sends a ACK or a NACK message. This way data is reliably transferred from

sender node to receiver node on a single hop.

Recovery of Lost Packets:

Let us now discuss individually the various cases of packet losses and subsequent recovery, one by one. Let us denote the ith data packet of a block by DataPkt i and last packet of block by DataPkt N. The Figure 3.2 shows the case when a block is transmitted from the SNode to RNode and there are no packet losses.

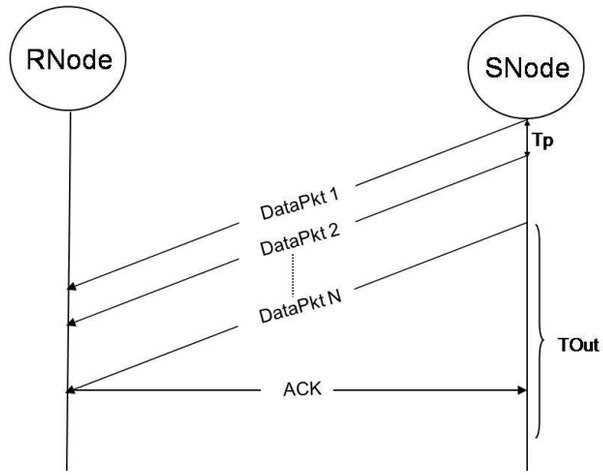

Figure 3.2 No Loss

Next assume that some packets are lost when a block is sent by SNode to RNode but the last packet of the block is received. The Figure 3.3 presents this scenario. The RNode then sends the NACK message. The SNode on reception of NACK message stops the TOut timer and resends the lost packets. The SNode after transmitting the last lost data packet, again restarts the TOut timer. When RNode receives all the lost packets, it sends the ACK message thus completing transfer of the block.

Let us now examine what happens when the last packet of the block is also lost during the transmission. Figure 3.4 shows this case. As said earlier we always resend the last packet of the block being transmitted whenever the TOut timer fires on sender node. The value of this timer is chosen as greater than one round trip time. Also on the receiver side when the last packet of the block is received, it sends ACK or NACK message based upon number of packets lost. So when the last packet is also among the lost packets of the block, RNode

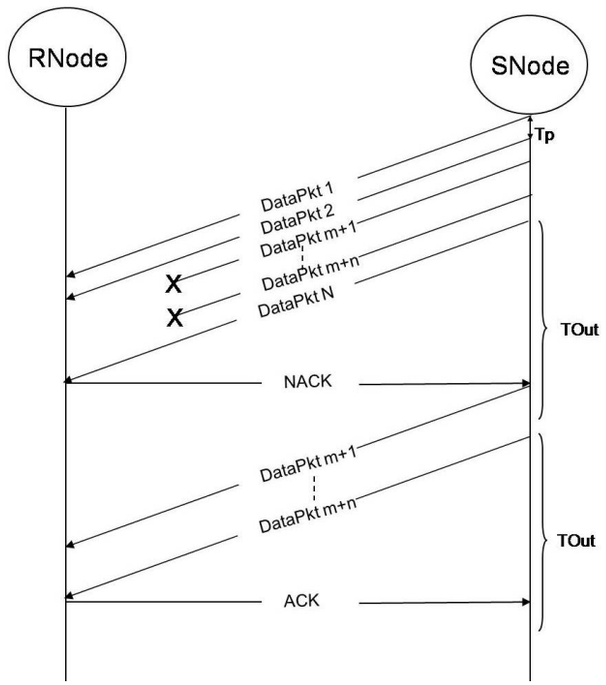

Figure 3.3 Data Packets Lost

neither sends ACK nor NACK message. Subsequent to this the TOut timer on the SNode fires and it resends the last packet of the block. This process continues till the time ACK or NACK message is received by the SNode.

Similar things happen whenever ACK or NACK message is lost. Here again the TOut timer fires on the SNode and it resends last packet of the block. The process continues till the time ACK is received by the SNode. Figure 3.5 depicts the case when ACK or NACK packets are lost.

3.2.2 Multi Hop Data Transfer

Mechanism of Data Transfer:

In case of multi hop data transfer there are more than two nodes involved in data transfer. Apart from sender and receiver nodes there are also other nodes in between them which we call as intermediate nodes. In this case of data transfer we make use of a kind of store and forward technique. The data transfer mechanism between the sender and intermediate node is same as explained in single hop case with the exception that on the intermediate nodes, the block on reception is not forwarded to the application layer. So there is no flash read or write involved on these nodes. The intermediate nodes instead forward it to their parent

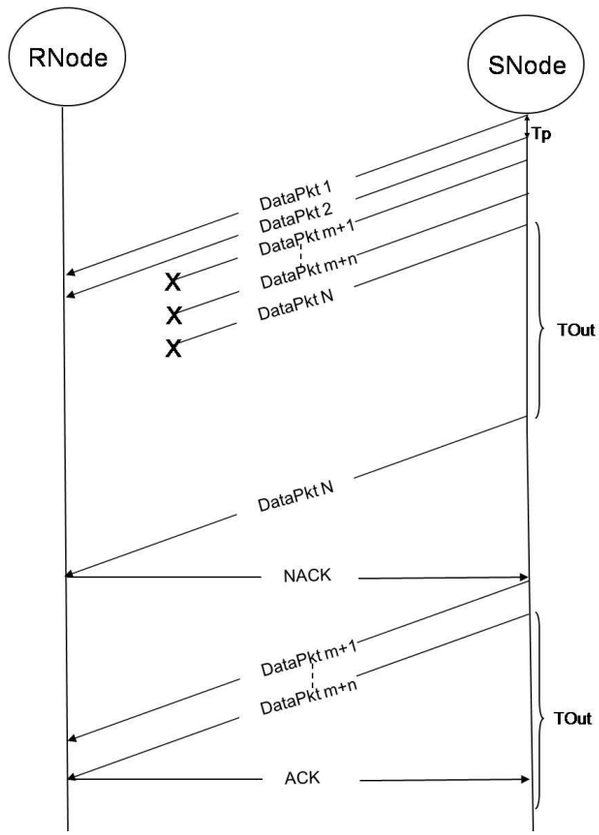

Figure 3.4 Last Packet Lost

when complete block is received.

Figure 3.6 shows the case of multi hop data transfer. Here the sequence numbers which are used to mark the messages indicate the order of occurrence of these messages. The sender node sends a block of data to its next-hop which in fact is a intermediate node in this case. The intermediate node after receiving the complete block forwards it to its next-hop. The process repeats till the time complete block is received by the receiver node. The receiver node then sends the ACK message which also acts as multi hop block ack message which henceforth is called as SINK_MSG message. The SINK_MSG message is forwarded by the intermediate nodes to the sender node. The receipt of SINK_MSG message by the sender node marks the compete transfer of a block from sender to receiver node in this case. Whenever the intermediate nodes forward the SINK_MSG message to their child, they wait for ack message from the child. We call this message as SINKACK. If SINKACK message is not received within a certain time limit, SINK_MSG message is forwarded again. Thus in this case we make use of both hop-by-hop and end to end reliability.

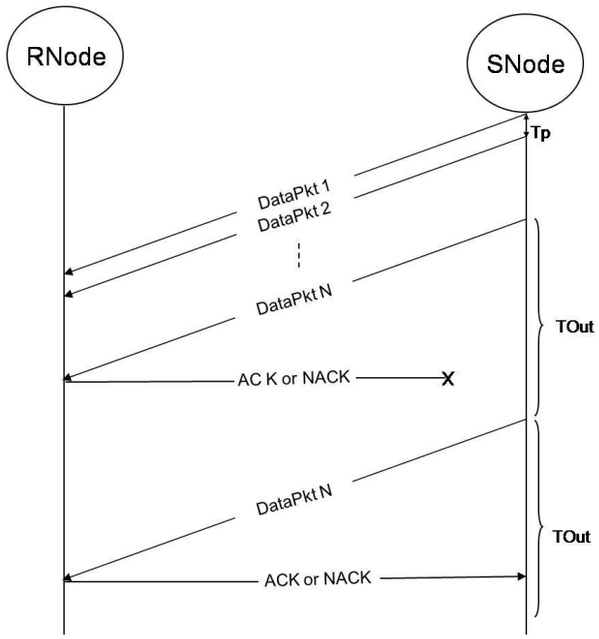

Figure 3.5 ACK or NACK lost

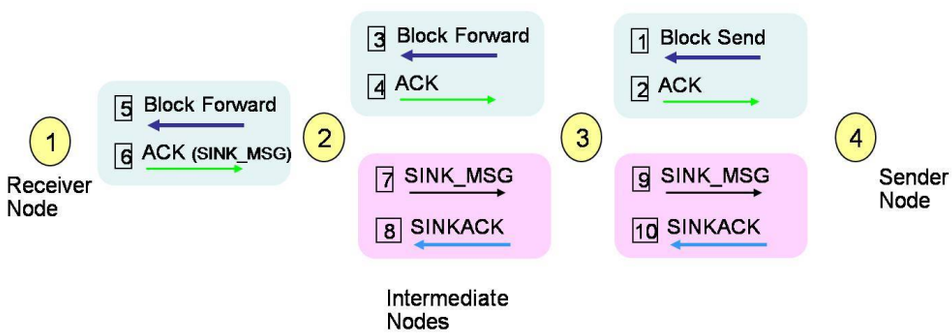

Figure 3.6 Multi-hop Data Transfer

A point to note here is that the unit of hop by hop data transfer is a block and not a packet. This has been done so because the transfer of data in our implementation takes place only at one of the hops at a time. Because of this there is minimal or no inter-hop interference. During the data transfer on a hop, our aim is to transfer as much data as possible in one block at the fastest possible inter-packet pause so as to obtain higher throughput. Had we used the unit of data transfer as packet in multi hop case, then it would have taken lot of time for entire transfer of data because of requirement of larger inter packet pause interval.

As a variation from our implementation, in PSFQ implementation [29] on tmote, the unit of data transfer used on a multi hop network is a packet. Also here the data transfer takes place on multiple hops at the same time resulting in inter hop interference. When multiple

hops interfere with one another, the sender cannot really figure out at what rate to send packets (i.e. what the inter-packet pause should be). PSFQ ends up being very conservative in this regard,and hence has poor performance in practice [29, 22].

Recovery of Lost Packets:

Let us now discuss packet loss recovery in multi hop data transfer case. Figure 3.7 shows the case when there are no losses. Let us now consider an example where there are three nodes involved in the data transfer. The data source and the data sink nodes here again are called as SNode and the RNode respectively. The intermediate node is called as INode. The RNode as usual on receipt of a complete block, sends the ACK message. This ACK message from the RNode in this case also acts as SINK_MSG. INode on receipt of ACK message from the SNode, forwards SINK_MSG to its child, which in this case is SNode. INode then waits for SINKACK for SINK_MSG sent. Also after sending SINK_MSG to its child, it starts a timer called as SinkOut timer. If SinkOut timer fires then Inode again sends SINK_MSG. The SNode on receipt of SINK_MSG informs the application layer that the block has been received by RNode and hence it can send the next block. In this way all the blocks are transferred.

Let us now consider the case when SINK_MSG or SINKACK is lost. When any one of these messages is lost, the SinkOut timer fires and hence INode resends SINK_MSG message. The process continues till the time SINKACK is received by INode. Figure 3.8 shows the case when SINKACK packet is lost. The same sequence of events happens when SINK_MSG is lost. The recovery mechanism with respect to the loss of other types of packets is the same as was discussed for a single hop case.

3.3 Implementation Details

In this section we will cover the implementation details of the transport protocol. We will first talk of the active messages and packet structures used in the implementation of the protocol and then we will discuss the flow control.

3.3.1 Active Messages and Packet Structures

The table 3.1 shows all the active messages which are used in the implementation of the protocol.

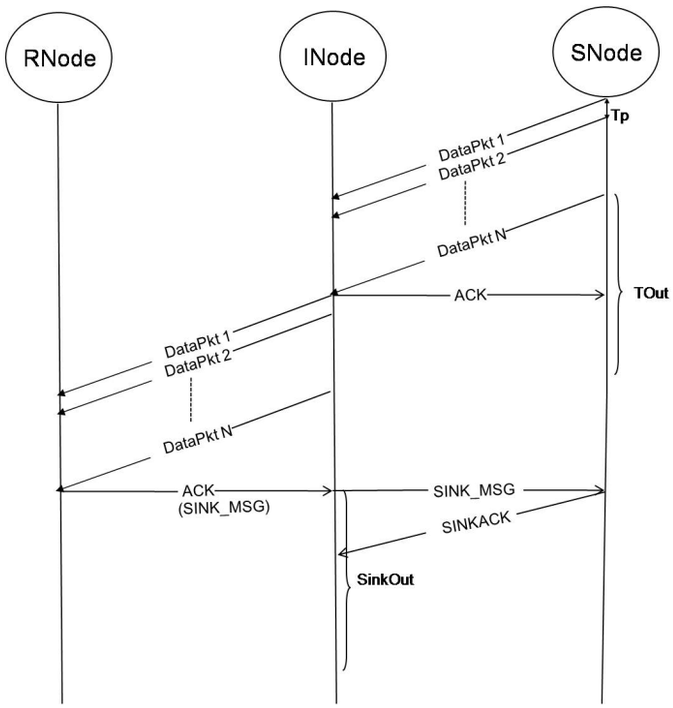

Figure 3.7 Multi Hop Transfer (No Loss)

Message Type	**Used For**	**Mode**
AM_DATA_MSG	Data packets	All
AM_ACK	Acknowledgment Message	All
AM_NACK	Nack Message	All
AM_LAST_HOP	Sink Message	Multi hop mode
AM_LASTHOP_ACK	Sink Ack Message	Multi hop mode
AM_TAIL	Marking End of Block	Downloading data from mobile node

Table 3.1 Active Messages Used

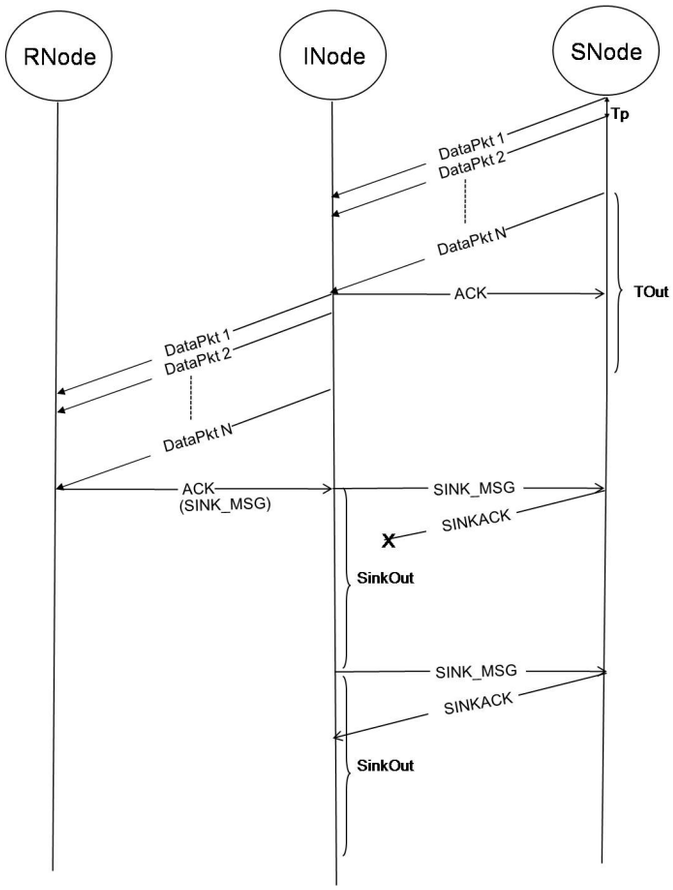

Figure 3.8 Multi Hop Transfer (Packet Loss)

The TOSMsg structure which is explained at B of appendix is used to carry all the messages. It includes the MAC header and the MPDU (MAC Protocol Data Unit). All the active messages are copied into the MPDU before sending the message.

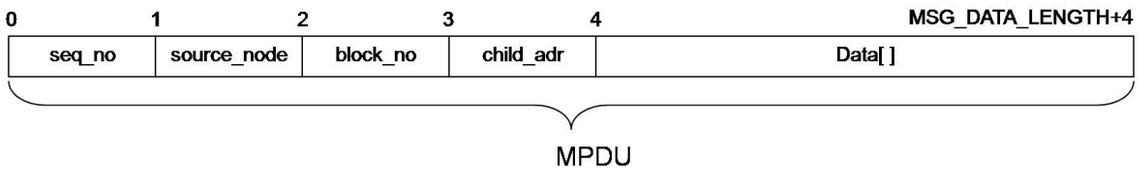

Figure 3.9 Data Message Structure

Let us now consider the case when SNode is trying to send a block to RNode. Figure 3.9 shows the data message structure which is used to carry the data packets. The seq_no field is used to carry the sequence number of the packet being transmitted. This information is used by all the nodes to keep track of what all packets of a block have been transmitted, received or required to be retransmitted. The seq_no is reset to zero when we send last packet of the block. The source_node indicates the node whose data is being transmitted i.e. the node which acquired this data during the data acquisition. The source_node information is required by the RNode to know which node acquired this data. Also this information helps in segregating the data of different nodes into their respective files while downloading data to PC for data analysis purposes. The block_no implies block number. child_addr indicates the TOS_LOCAL_ADDRESS of the node sending the packet. The child_addr information is used at the INodes and also at the RNode to know from which child the data message is being received. Since there can be more than one child of a node. SNode gets the details of source_node and block_no from the application layer. Before sending the packet, it copies these details along with seq no generated and its address into the respective fields of the data packet. Note that in case of multi hop data transfer, child_addr and source_node are same at the SNode since in this case the SNode transfer only its data. In case of single hop data transfer when the data is transferred from header node to mobile node or mobile node to base node, the child_addr and source_node fields will be different when they transfer data of other data acquiring nodes which has been stored in their flash.

Data[ ] is an array with each element of two bytes. The length of the array is equal to

maximum number of such elements that can be transmitted in a packet. It is denoted by MSG_DATA_LENGTH and is obtained as

$$MSG\_DATA\_LENGTH = (TOSH\_DATA\_LENGTH - 4)/2 \tag{3.1}$$

where TOSH_DATA_LENGTH is the payload size of the data message.

The ACK message is used to indicate to the child node that the block has been successfully received by the parent node. Figure 3.10 shows the ack message structure. Here the parent_addr implies the TOS_LOCAL_ADDRESS of the node which sent the ack message. Other fields information is used in similar way as was used in data message packet case.

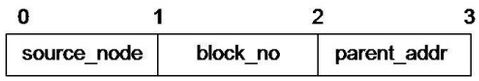

Figure 3.10 Ack Message Structure

Figure 3.11 shows the NACK message structure. NACK message as said earlier is used to inform the parent node about the packets which have been lost during the block transmission and hence required to be retransmitted. Here the ctrpktloss field indicates the number of packets lost and seqno[] is an array which contains the sequence numbers of the packets which are lost. So once the NACK message is received by the child node, it resends the lost packets. NACKMSGLENGTH implies maximum number of packets which can be lost. This number is same as the number of packets per block.

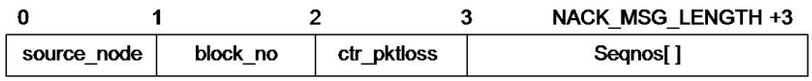

Figure 3.11 NACK Message Structure

The packet structures used for SINK_MSG and SINKACK are same as that of ACK message. However while transmission of these messages AM_TYPE values used are different so that they can be distinguished from each other.

Let us now talk about the tail message. This message is used only when data is downloaded from mobile node to PC. On the base node, which basically runs the TOSBase application code, we have implemented the functionality of the transport protocol. The protocol is not implemented as a separate layer. This is so because while using genericcom

component of TinyOS which is used in the code loaded on other nodes, we found that there is some problem in using the UART interface which is used to send and receive the packets from PC. Though we could not exactly pinpoint the cause, we solved the problem by using the radio component one layer below the layer used in normal implementation. The same problem was reported by others also [9].

The base node is connected to the PC. All the data transfer related packets received by this node are forwarded to PC (since here the aim is to download data to PC). However the transport component makes sure that there is reliable data transfer between the mobile node and the base node. Which means that whenever there is any packet loss, there is recovery of data. There is no RAM buffer maintained on this node. The similar buffer is however maintained on the PC. On PC we run a java application which handles all the data packets received by the base node. So when all the packets of a block have been received by the base node and hence the PC, we need to write the data block from PC buffer into the data file corresponding to the node id of the data source node. So to indicate when to write this buffer data into the respective node file is indicated whenever we receive a tail message from the mobile node. This message is sent by mobile node when it receives the ACK message from base node.

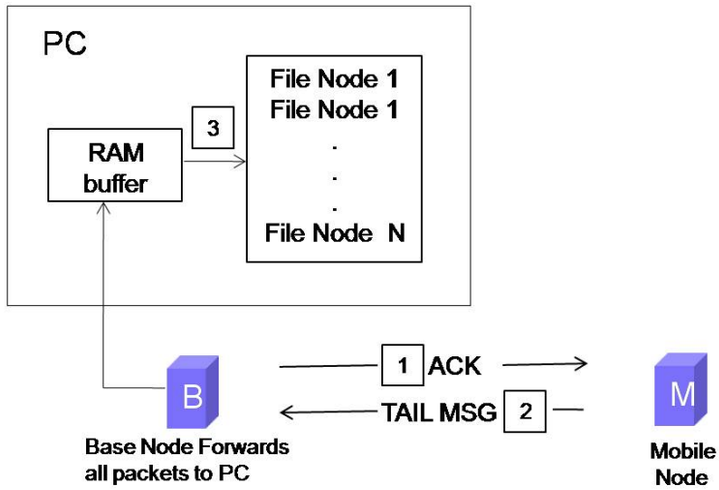

Figure 3.12 Data download to PC

Figure 3.12 shows the data download to base node. Here sequence numbers in the figure

indicate the order of occurrence of respective events. One can argue that on transmission of ACK message we could do the same thing. However since the data transfer from mote to PC is slower than data transfer from mote to mote, there are chances that before the buffer data is written to file, we start receiving the next block. Hence use of tail message solves this problem. The message structure used for the tail message is as shown in Figure 3.13. Here from_addr field implies the source node address which sent this message. Same message structure is used for sending ack of the tail message to mobile node. Other fields have same meaning as in other messages.

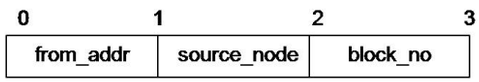

Figure 3.13 Tail Message Structure

3.3.2 Flow Control

In this section we will examine the problem of flow control about which we have mentioned briefly in Section 3.1. To examine the problem in detail we needed to find out the various delays which happened on both sender and receiver side when a packet was transmitted. The following experiment was conducted by Nilesh [10]. He transmitted packets of known size one after the other with large value of T_p parameter (over 100 ms) to ensure that there are no queuing delays. Whenever a packet was transmitted, the time stamps at various events on both sender and receiver nodes were recorded . The various events for which the time stamps were taken on sender node and receiver node are as shown in Table 3.2 and Table 3.3 respectively.

Table 3.4 and Table 3.5 shows the average delays which occur on sender side and receiver side respectively for different packet sizes. The average was found out over transmission of about 500 packets of each size. Let us now analyse this data by taking example of 128 packet size. Figure 3.14 shows the time line of events happening on both sender and receiver side when a packet of 128 bytes is sent. As seen from this figure, when SendDone event is triggered on the sender side, at almost the same time event R3 happens on the receiver side. Which means that though the SendDone event has triggered on sender side but on the receiver side, this packet still needs to be transferred from the radio buffer to micro controller

Event	Description
S1	Send Command Issued
S2	Start of transmission from Micro-controller to Radio on SPI bus
S3	End of Data Transfer over SPI bus
S4	SFD start i. e. sending a few packets over air
S5	Tx of packet over radio done
S6	SendDone Event triggered

Table 3.2 Events On Sender Side

Event	Description
R1	Received SFD interrupt i.e. first few bytes received over radio
R2	Complete packet received
R3	Start of Data Transfer from radio to micro-controller over SPI bus
R4	End of Data Transfer over SPI bus
R5	ReceiveEvent Message Triggered

Table 3.3 Events On Receiver Side

RAM. Even after this transfer there is an additional delay of 0.240 ms before receive event i.e. R5 is triggered. So R5 event on receiver is triggered after a duration of B=6.55 ms from event R3. Now on the sender side all the events up to S4, covering duration of A = 4.12 ms can happen for the next packet transmission without affecting at the receiver, the transfer of previous packet from radio buffer to micro controller RAM. So to avoid the next packet drop at the receiver's radio, we need the value of T_p which satisfies the equation 3.2.

$$Tp > (B - A) = 2.43ms \tag{3.2}$$

Since the timer interface to which we pass T_p as parameter takes only integer values , the value of T_p used is 3 ms for a packet size of 128 bytes. Like above example the value of T_p can be found for other packet sizes.

Table 3.6 shows all the timers we have used in implementation of the protocol. The optimal values are shown for a packet size of 128 byte (with this packet size we got the best throughput of 45kbps in the experiment the details of which are given in Section 4.3).

Let us now talk of TOut timer. As said earlier this timer fires whenever we do not receive

Packet Size in bytes	**(S2-S1) in ms**	**(S3-S2) in ms**	**(S4-S3) in ms**	**(S5-S4) in ms**	**(S6-S5) in ms**
44	0.735	1.278	0. 519	1.404	0.183
66	0.763	1.739	0. 519	2.105	0.183
88	0.734	2.136	0. 488	2.806	0.183
128	0.732	2.9	0.49	4.090	0.190

Table 3.4 Sender Side Delays for Different Packet Sizes

Packet Size in bytes	**(R2-R1) in ms**	**(R3-R2) in ms**	**(R4-R3) in ms**	**(R5-R4) in ms**
44	1.472	0.201	2.290	0.234
66	2.176	0.202	3.346	0.235
88	2.879	0.202	4.395	0.235
128	4.15	0.210	6.31	0.240

Table 3.5 Receiver Side Delays for Different Packet Sizes

Timer	**Mode**	**Optimal Value in ms**
Data Timer	All	3
TOut	All	25
SinkOut	Mutihop	25
Tail Timer	Data Download	25

Table 3.6 Timers Used

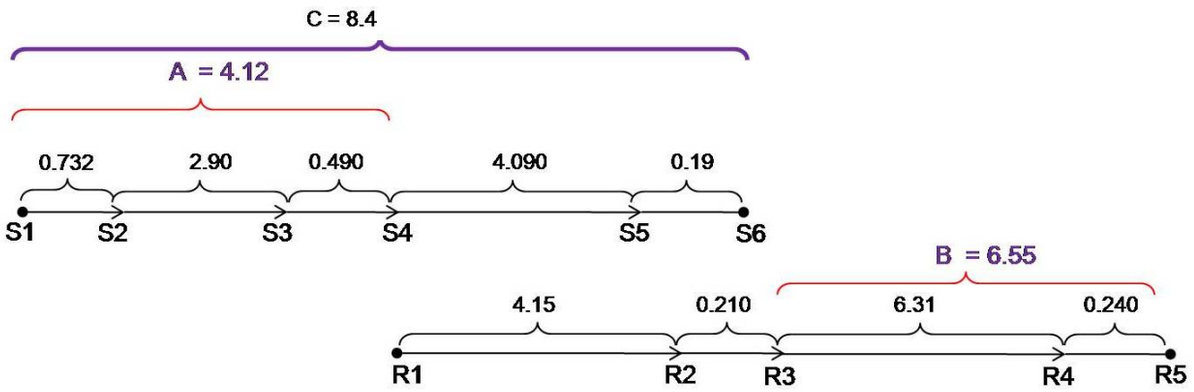

Figure 3.14 Events Time Line at Sender and Receiver Nodes (128 bytes packet)

ACK or NACK from the receiver node, after transmission of a last packet of a block. This timer is started when SendDone event is triggered after the last packet is transmitted. The value of this timer if chosen very small will result in unnecessary transmission of last packet and hence delay in recovery of packet losses. Also if value chosen is too large again it will result in delay in recovery of lost packets. So we need to choose some optimal value of this timer. It should be greater than one round trip time. Let us now find out the value of round trip time. As seen from Figure 3.14 after last packet is transmitted, on the receiver side the ReceiveEvent i.e. R5 is fired after A=6.55 ms. After this R5 event is fired, the packet is copied into the RAM buffer and packet losses are determined. If there are packet losses we need to send a NACK message. For sending the NACK or ACK message, we make use of post task method which is the recommended method in TinyOS.

We conducted a small experiment to find out maximum value of time taken between the event when Send NACK task is posted and the event when NACK message SendDone event is triggered. Let us call this time as 'MAX_NACKSEND_TIME'. We had introduced artificial loss of data packets on the receiver side so that NACK messages are generated. We found that the maximum time taken was around 12 ms in most of the runs except one case where this time was around 15 ms. At this time surprisingly the number of packets lost were only two. We ran this experiment for ten times. During the experiment the value of TOut was kept as 30ms. After the SendDone event of NACK message is triggered on sender side there is an additional delay between events R3 and R5 on receiver side. So for a NACK message reporting maximum loss, this delay is about 2.5 ms. so the TOut value should be

such that it satisfy the equation 3.3

$$TOut > (6.55ms + (MAX\_NACKSEND\_TIME = 15ms) + 2.5) = 24.5ms \tag{3.3}$$

Hence the optimal value of TOut is 25 ms. For uniformity and similar reasons, the value of other time out timers is taken as same as TOut timer.

CHAPTER 4

Experiments and Results

In this chapter we will present all the experiments which we conducted while developing our prototype. To start the chapter we will discuss the sensor calibration experiments and the experiment which we did on the road bridge to test our prototype operation in short term monitoring mode. Next we will be evaluating the transport protocol which we developed for our prototype. Finally we will discuss the mobile data transfer experiment.

4.1 Sensor Calibration

In this section we will present the calibration experiments which performed for the calibration of accelerometer ADXL203. We first did gravity test for the accelerometer for different range of voltages so as to find out how the sensitivity of the accelerometer varies with the applied voltage level. Next we wanted to find out the noise level of this accelerometer so that we know the minimum value of acceleration which it can measure. As mentioned earlier the acceleration is measured in units of 'g' which is equal to 9.8 m/s^2.

Gravity Test:

Experiment setup and results: In this experiment we did the normal gravity test for the accelerometer for different voltages. In this test when accelerometer chip is placed parallel to earth surface, at that time value of g measured is 0 so the output voltage of ADXL203 should be half of the exciting voltage as per specifications [12]. When the accelerometer is tilted by +90 degrees from this position then the acceleration measured is 1g. So for excitation voltage of 5 volts, the output voltage should be close to 2.5+1 = 3 volts. When accelerometer is tilted by -90 degrees from the horizontal position then acceleration measured is -1g. So the output voltage of the accelerometer should be 2.5-1 = 0.5 volts.

As per the data sheet of ADXL 203, the output voltages and sensitivity of the accelerometer are ratio metric. We applied different voltages to the accelerometer while conducting the gravity test for both X and Y axes. Each time we measured the output voltages and calculated the sensitivity using the formula as sensitivity = (output voltage at +1 g position - output voltage at -1 g position)/2g. Figure 4.1 shows the variation of sensitivity with

the applied voltage. The plot is almost linear. We in our prototype use 3 volts for exciting the accelerometer. As seen from the graph at 3 volts the sensitivity of ADXL203 is 570 mg. This closely matches the specified value of 560 mg at 3 volts in the data sheet [12]of the accelerometer.

Noise Level Measurement:

Experiment setup and results: We wanted to conduct this experiment at a place which was free from any vibrations. So we conducted this experiment at IIT Kanpur air strip. This place is quite open and also away from the main campus of IIT Kanpur hence the vibration level here is expected to be quite low. We conducted this experiment in the evening at around 8 pm.

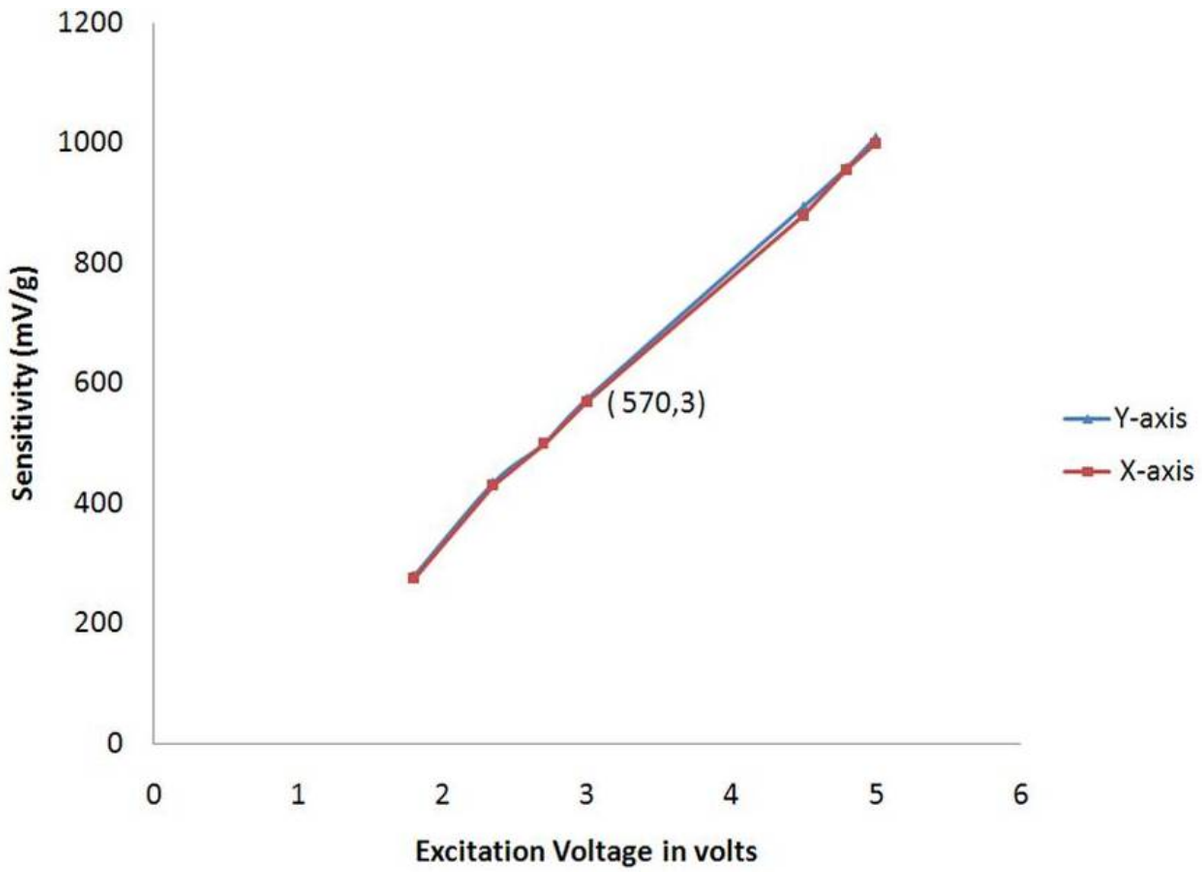

Figure 4.1 Variation of sensitivity with Voltage:ADXL203

We interfaced the accelerometer to tmote as shown in Figure 2.2. We ensured that battery voltage of mote was around 3 volts. We acquired the data from the accelerometer for 20

seconds at the sampling rate of 500 Hz. While analysing the data we further used ten point averaging of this data to reduce the noise. We measured the rms value of noise level as 2.93 mg under these conditions. Under ideal circumstances (i.e. free from any vibrations) as per the data sheet, the value at 50HZ bandwidth (used in our case) should be 0.983 mg. So this variation in the measured value seems to be because of prevalence of non ideal conditions. However this value is acceptable to us in our application since this value is much lower than the 10 mg value which is the limit in our case as per [11].

4.2 Experiment on a Road Bridge

While we were developing our prototype, we did extensive measurements in the structures lab. All the measurements were very positive and encouraging. To further boost our confidence we wanted to test our system on an actual bridge in the presence of structural engineers from IIT Kanpur [11] with whom we were constantly interacting at every stage of the development. We got an opportunity to test our system in short-term monitoring mode on a road bridge at river Ganga at Kanpur. We choose road bridge instead of railway bridge primarily because there were less administrative hassles involved in getting the permission to do measurement on road bridge than railway bridge. Also in this experiment the main aim was to test the system for all functionalities and also to demonstrate the technology to the structural engineers. The data collected in this experiment was preserved for later references.

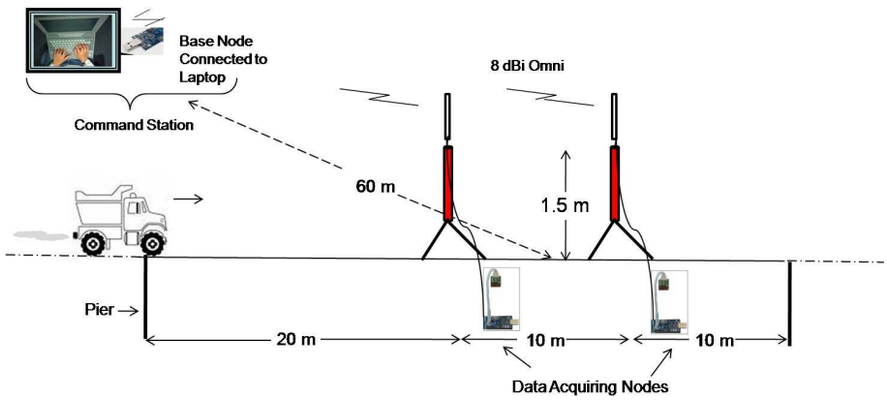

Figure 4.2 Bridge Experiment Layout

Figure 4.2 shows the layout of this experiment. In this experiment we used two data acquiring nodes for the data acquisition. As per direction of structural engineers, we selected

a bridge span and the two nodes along with the accelerometers were placed at the middle and 1/4 of the length of the span respectively. Both the nodes were fitted with external 8 dBi omni directional antennas. The height of the antennas was around 1.5 meters. The length of span of the bridge was about 40 meters.

The command station comprising of the base node and a laptop was established in a stationary van and was located about 60 meters from the span of the bridge. For data acquisition, we selected 500 Hz as the sampling rate so that later we could study the affect of averaging the data, for noise reduction. There was no averaging of data done on nodes during this experiment. During each measurement we acquired the data for 20 seconds. We measured the vibrations of the span both in vertical direction and horizontal directions.

To start the experiment we waited for a heavy vehicle to come near the span. When the vehicle entered the span, we issued a command from the command station to the data acquiring nodes to start acquiring the data. After the data has been acquired by the nodes, we get a message at the command station that data has been acquired by the nodes. Following this we issued a command to these nodes to reliably transfer their data to the command station via the base node (refer Section 2.3.2). We used the data analysis tool to analyse the data at the bridge itself. Like this we repeated the experiment for both vertical and horizontal vibrations of the span ten times each.

4.2.1 Results

The results from this experiment were very encouraging. Every time we were able to reliably issue the commands to the data acquiring nodes and were able to reliably transfer the acquired data from these nodes to the command station. The entire system worked as was expected to work.

Figure 4.3 shows a snap shot of data being analysed with data analysis tool. Our data analysis showed the characteristic frequencies of the span as 5.75 Hz and 2.56 Hz for vertical and horizontal directions respectively. The value of acceleration measured in the vertical direction was 98 milli-g. We did not have any data of the bridge for cross checking. However the feel which we got of these vibration while standing on the span, pointed out that our measurements were in right half park.

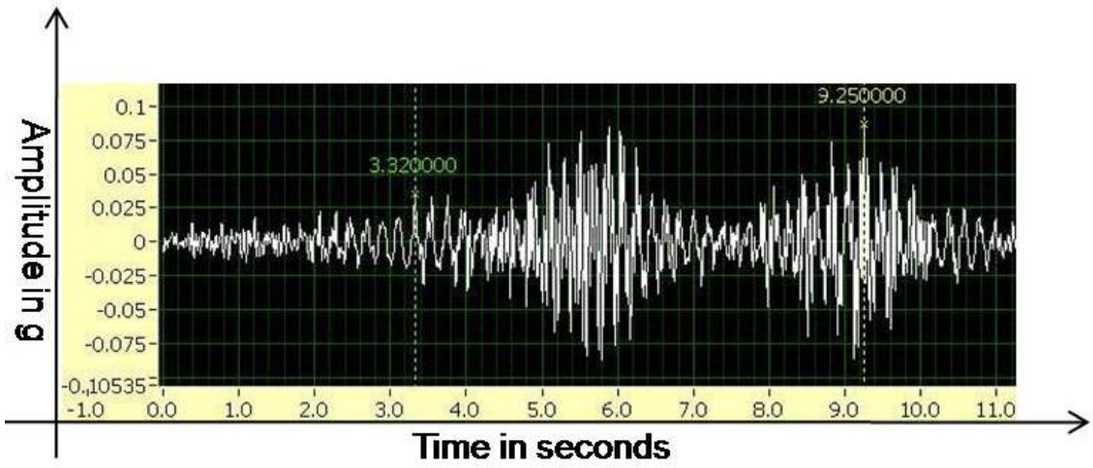

Figure 4.3 Bridge Data Analysis

4.3 Transport Protocol Evaluation

In this section we will analyse the transport protocol in detail. We will be examining it for throughput, reliability and performance in single hop and multi hop cases. Finally we will compare it with PSFQ [29].

Throughput:

Let us first talk of throughput. We conducted an experiment to determine the maximum throughput which could be achieved with this protocol in a single hop case. Since in our application mobile data transfer is time critical, we wanted to achieve maximum throughput in single hop case so that the data is transferred at the earliest from header node to mobile node.

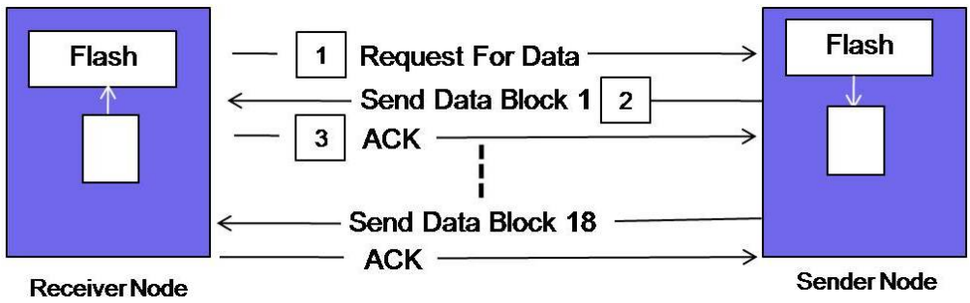

Figure 4.4 Setup for single hop throughput measurement

For conducting the experiment, we configured one node as the sender node and the other node as the receiver. These nodes were separated by a few meters. Figure 4.4 shows the

setup. The sequence numbers in the figure indicate the order of occurrence of associated events. On the press of user button on receiver node the request for data transfer was sent to the sender node. The sender node on receiving the request, started the data transfer on block by block basis. It read the block from the flash and sent it to receiver node. The receiver node on receipt of block, wrote it to its own flash. A total of eighteen blocks of data were transferred.

The figure of eighteen blocks results from the fact that each data acquisition node acquires three blocks of data and there are six data acquisition nodes on a data-span. The header node would require to upload its own data and data gathered from remaining five data acquisition nodes. So total blocks it needs to upload is $6 \times 3 = 18$. We used block size of 2240 bytes which is significant chunk of the 10KB RAM in the MSP430 chip of Tmote sky.

We repeated the above experiment for different packet sizes and noted down the total time taken for data transfer in each case. Table 4.1 shows the total transmission times which were taken for each packet size. There were no packet losses in each case. Figure 4.5 shows the variation of maximum throughput that can be achieved with each packet size. Clearly the maximum throughput achieved increases with the increase in packet size, with maximum throughput of around 45.8 kbps at 128 bytes packet size. We could not use packet size greater than 128 bytes since this is the maximum size of radio buffer on tmotes. The trend in the graph is so because there are fixed delay overheads on sender and receiver side irrespective of packet size used. This is clearly observed from the delay tables 3.4 and 3.5 for sender and receiver respectively. So if we send more number of packets per block, more time it will take to transfer the complete block.

Let us now account for the total time taken to transmit eighteen blocks of data with packet size of 128 bytes. Due to flow control problem as mentioned in Section 3.3 we choose inter packet pause of 3 ms for this packet size. The payload size for this packet size is 116 bytes. So a block fits into 20 data packets. Also in our implementation, except for the last block the flash read and write operations on the sender and receiver nodes respectively, overlap with each other. The average time taken to read and write a block of 2240 bytes in Tmote's flash was experimentally found out to be about 100 ms and 70 ms respectively. The transmission time for a packet including the inter packet pause comes out to be 12 ms.

So the transmission time per block comes out to be 240 ms. So total transmission time for eighteen blocks is = 18x100+1x70+18x240= 6190 ms. This is close to experimentally observed transmission time of 7032 ms. The difference of around 840 ms is likely to processing overhead which we did not include in the calculations.

The throughput obtained in this case is =2240x18x8/7.032 = 45.8 kbps . This value is much lower than 250 kbps allowed by 802.15.4 radio due to various inefficiencies namely:

- Various header overheads.
- The flash and radio share the same SPI bus due to which flash read/write cannot go in parallel with radio operation.
- Use of pause timer to address flow control.

Packet Size in Bytes	**Time to Transmit in seconds**	**Throughput in kbps**
44	18.1	17.85
72	10.96	29.43
96	8.83	36.53
128	7.032	45.8

Table 4.1 Packet size vs time taken to transmit (18 blocks of data)

Reliability:

In order to check the reliability aspect of the protocol we conducted an indoor experiment in which we dropped all types of packets with probability of 80% both on sender and receiver nodes. In this experiment the data was transferred form the data acquisition nodes to header node and then header node uploaded complete data to the mobile node. We found that there was full recovery of packets even at such higher error rates. We repeated this experiment for various other error rates and found the similar results.

As said earlier in our application the transfer of data from the header node to mobile is very time critical. So in oder to check whether there is timely recovery of lost packets in this case, we repeated above experiment for different error rates, for single hop case which

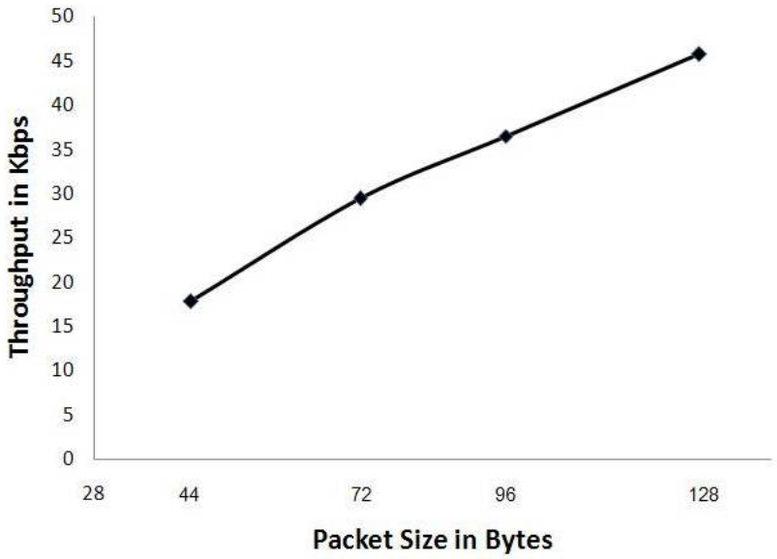

Figure 4.5 Throughput vs Packet Size Plot

involved only header node and mobile node. The header node was made to upload eighteen blocks of data to the mobile node. Each block of data was of 2240 bytes. Packet size used was of 128 bytes since this packet size gave us maximum throughput in our protocol. We dropped the packets with error rate of 0% ,10% ,20% and 40% . Such error rates are likely to occcur in actual deployment.

Figure 4.6 shows how the data was reliably transferred under the lossy environment. For above mentioned error rates we found out that data was reliably transferred within maximum time of 12 seconds. In an outdoor mobile data transfer experiment the details of which are given at Section 4.4, we found that mobile node starts receiving the data from header node at a distance of around 500 meters irrespective of speed of the mobile node. So if mobile node is moving at a speed of 72 Km/Hr, the contact duration between the header node and mobile node is (0.500x60x60/72) 25 seconds. Within this time we should be able to reliably transfer the eighteen blocks of data. Which we have been able to do within just 12 seconds as mentioned above.

Performance of protocol for different data sizes:

In order to check the performance of our protocol for different data sizes, we conducted the following experiment for single hop and multi hop cases

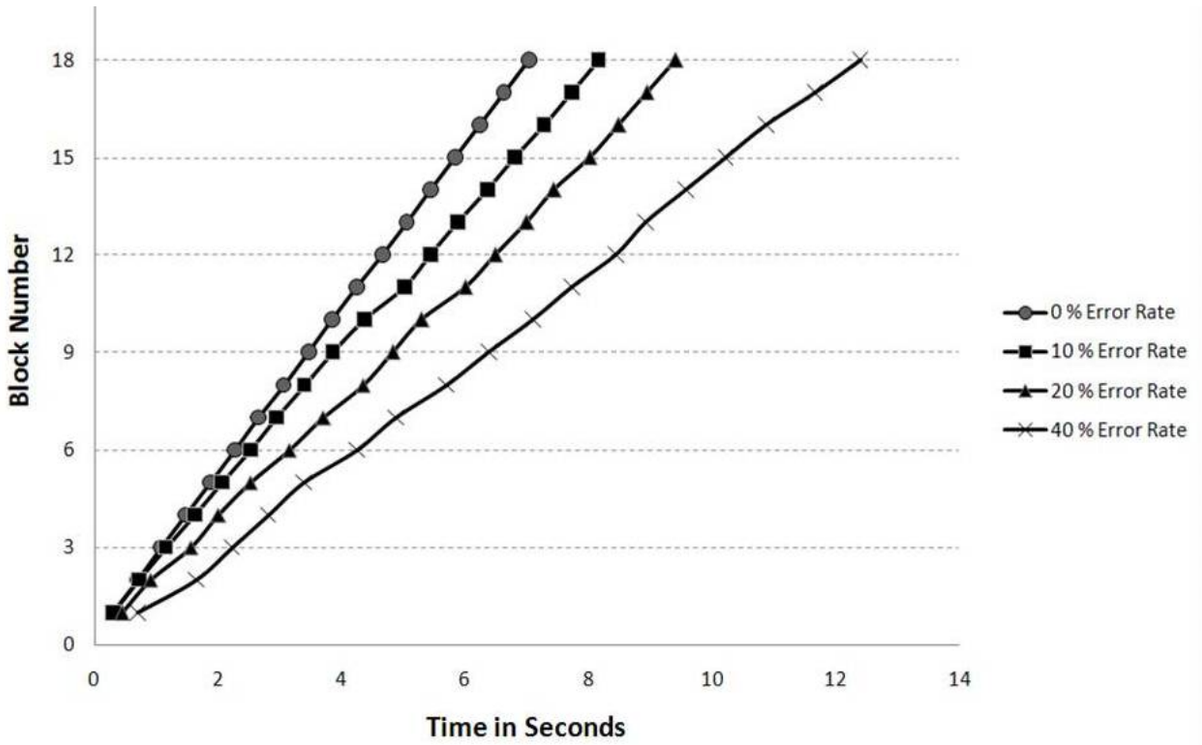

Figure 4.6 Data Recovery in Lossy Environment

Single hop performance: For single hop case we configured one node as sender node and the other node as receiver node. The sender node was made to transmit different numbers of blocks of data to the receiver node. We noted down the time taken to transmit in each case. Again the block size and packet size used were same as in the earlier experiment.

We plotted the number of blocks transmitted vs the time taken to transmit. Figure 4.7 shows this plot. The plot obtained is a straight line. With respect to mobile data transfer it implies that if header node uploads data of more data acquiring nodes then more is the time taken to upload complete data.

Multi hop performance: For multi hop case we used the routing tree as shown in Figure 4.8 as an example. Here node number one represents the header node. The header node was made to collect data from the remaining nodes one after the other. Each node transferred three blocks of data .

Figure 4.9 shows how the data transfer takes place in case of multi hop case. Here the node number three and five transmitted data on multi hop. The remaining nodes transmitted data on single hop. We can clearly identify region of different slopes in the plot. These

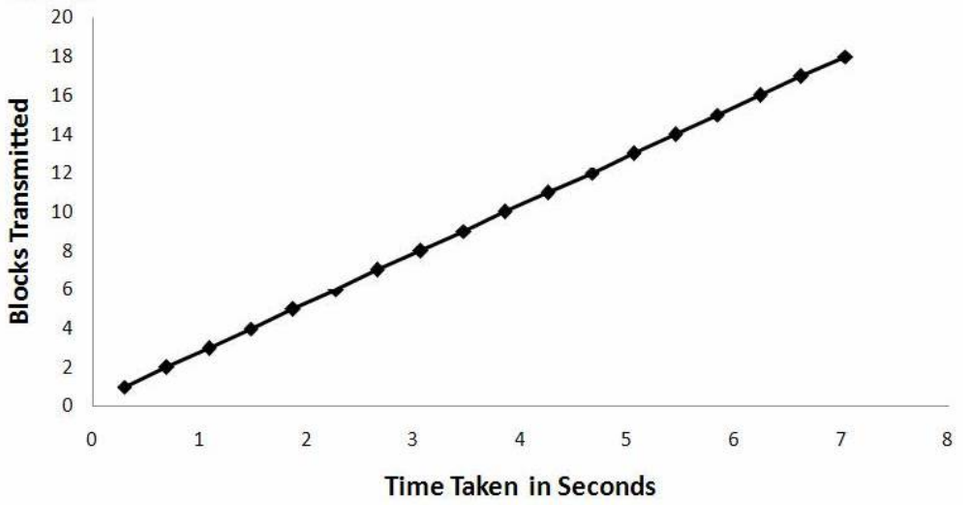

Figure 4.7 Data size vs time taken to transmit: Single Hop

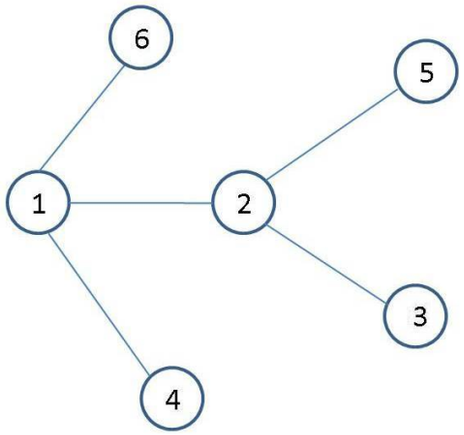

Figure 4.8 Routing Tree (With Multi Hops)

different slopes in the plot correspond to the single hop and two hop cases. The node from which data transfer was happening in each region of the plot is marked alongside that region in the plot. As expected the slopes of the multi hop cases are less than the slope of those nodes which transmitted data on single hop. It implies that more the number of hops between a data acquiring node and header node, more time it will take to transfer its complete data to the header node.

Performance Comparison with PSFQ:

While our transport protocol was being developed, a parallel work was also going on the implementation of PSFQ on tmote platform [29]. The aim of the implementation was to see if this protocol could be used in our type of application. Let us now compare the

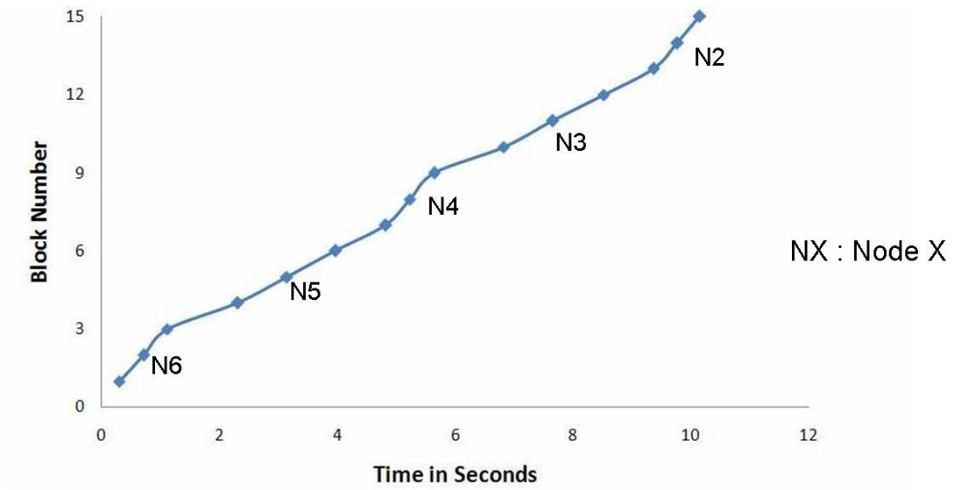

Figure 4.9 Multi Hop Data transfer

performance of PSFQ implementation on tmote and our protocol for both single and multi hop cases one by one.

Single Hop: In the implementation of PSFQ, in single hop case they could not achieve higher transmission rates for error rates greater 10% (no specific reason cited in [29]) where as in our implementation we have been able to acheive the best possible (limited only due to flow control problem) transmission rate and this rate is not affcted by the error rates. Even at 0 % error rate in single hop case, the time taken to transmit 1150 bytes with packet payload of 24 bytes (maximum packet payload used by them) was around 800 ms. With almost similar packet size(lowest in our case) our protcol took around 514 ms to transmit the same amount of data. This value in our case further reduces to 195 ms when we use packet size of 128 bytes. This clearly shows that we are able to keep latency very low as compared PSFQ in single hop case. PSFQ performance seems to be bad beacuse of inherent pump slowly paradigm of the protocol. The proceesing overheads are probaly higher in this protocl due to which they are not able to achieve the higher transmission rates even at 0% error rates.

Also as seen from Figure 4.10 in our protocol though the latency increases linearly with packet error rates but the rate of increase is not very high. The verification of the fact in the study, that in PSFQ latency is unaffected by the error rate in single hop is also not of great significance to us since the minimum level of latency they are able to achieve is much

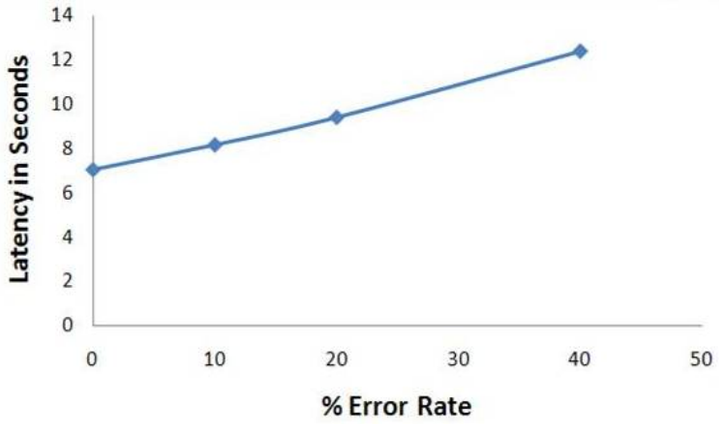

Figure 4.10 Percentage Error rate Vs Latency plot (18 blocks of data transfer)

higher than we are able to achieve with our protocol.

Multi Hop: As per the study in [29] the performance of PSFQ in multi hop case is very poor primarily because of inter-hop interference. In our case the transfer of a block is hop by hop and hence there is minimal or no inter-hop interference. In the study they could not achieve complete data transfer on multi hop when the error rates were greater than 30% (they say due to implementation bugs [29]) . In our case we have been able to achieve the reliable transfer of data on muti hops even at error rates as high as 80 to 90% (we did not test beyond this).

Let us now compare the performance of these protocols at 0% error rates in muti hop case. Figure 4.11 shows the best possible latencies which could be achieved by both the protocols in multi hop case for transfer of equal amount of data (1150 bytes). In this plot our protocol is marked as BTP (block transfer protocol). As seen in the figure, though the latency in both implementations increases linerly with the network size but the rate of increase in PSFQ is much higher. This is because of inherent inter hop interference and lower transmission rates used in PSFQ implementation to ensure completion of data transfer.

So both in single hop and multi hop cases our protocol performs much better than PSFQ for our application. Even at higher error rates and larger network size, with higher transmission rates we are able to keep the latency within acceptable limits which is a very important aspect in our application especially in mobile data transfer case.

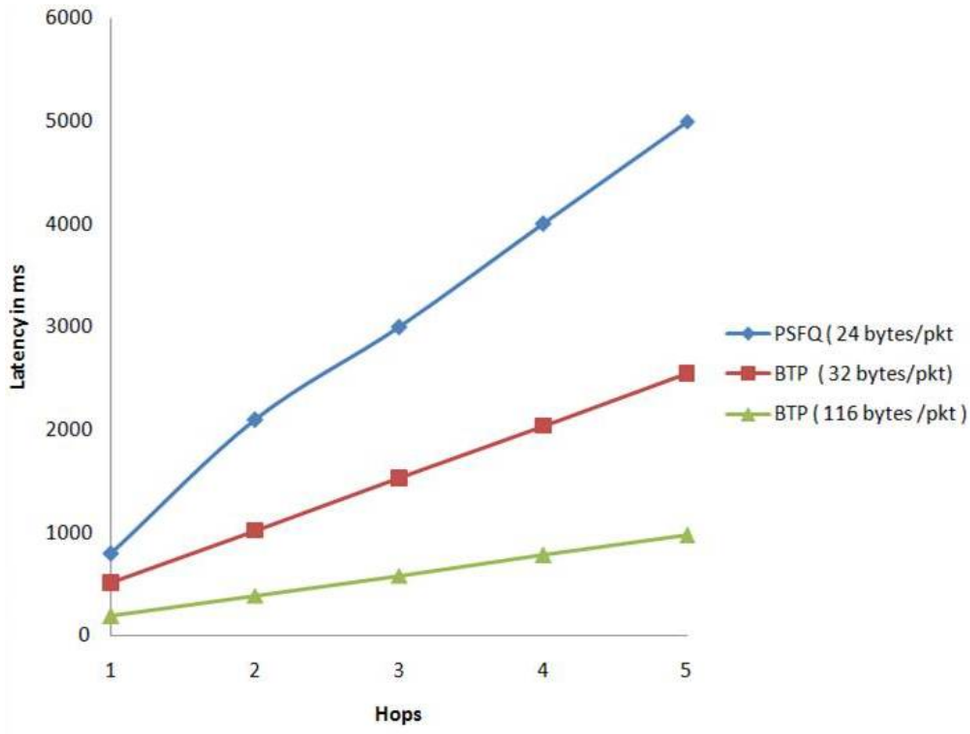

Figure 4.11 Performance comparison of PSFQ with BTP (our transport protocol)

4.4 Mobile Data Transfer

To see if we are able to achieve similar throughput as in indoor environment under a realistic scenario, we conducted an outdoor mobile data transfer experiment. We will first present the experimental setup and then the results.

4.4.1 Experiment setup

The setup mimics the situation where the header node (refer Figure 3.1) of a *cluster* uploads the total data collected by all the nodes within its cluster on to mobile node on a train. The data transfer is initiated when the mobile node (on the arriving train) requests the header node to transfer the collected data. The header node then reads the data stored as files from flash and uploads them one by one using the reliable transport protocol. In this experiment, the header node was made to transfer 18 blocks of data, each of 2240 bytes. The total data transfer was thus 18x2240 = 40,320 bytes.

Figure 4.12 shows the experimental setup layout. In the setup the header node as well as mobile node were equipped with external 8dBi omni directional antennas. The antenna of the header node was mounted at a height slightly over 2.5 m. The mobile node was fixed on

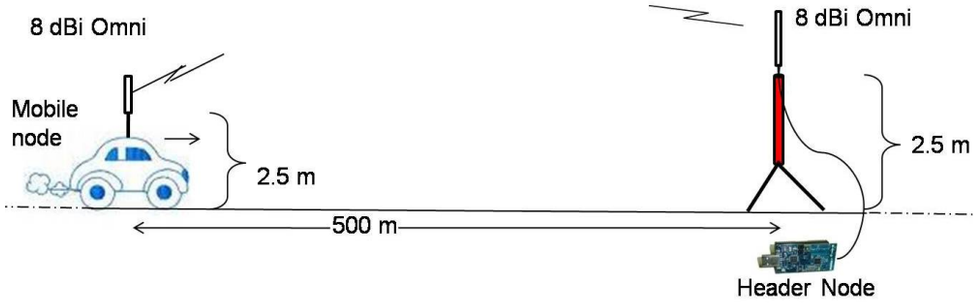

Figure 4.12 Experimental Setup: Mobile Data Transfer

a vehicle and the antenna was at an effective height of slightly less than 2.5 m. The header node (stationary) with the complete data of 40,320B in its flash, is initially idle. The mobile node is turned off until it is taken out of the expected range from the header node, and then turned on. The range experiments done in [3] show that the range can be around 400m.

We start the vehicle at a distance of 800m from the header node. The vehicle is made to attain a constant known velocity before the point of 500m, while coming toward the header node. At this point we boot the mobile node. On booting, the mobile node starts sending the request for data, every 100ms, until it receives a response from the header node. The header node, on receiving the request, uploads the data block by block, using the reliable transport protocol. The transport layer at both nodes takes a log of events like data requests, ACKs, NACKs and retransmissions, for later analysis.

4.4.2 Results

Figure 4.13 shows a plot of the block sequence number received at the mobile node, versus the position of the mobile with respect to the header node. We first note that although in [3] the contact range was conservatively estimated to be 400m, the data transfer begins at about 490m, irrespective of the mobiles speed. We next note that after the mobile is within 450m from the header node, the rate of data transfer (slope of the line) is more or less constant. We have calculated this slope to correspond to the same data rate (about 46Kbps) as in the stationary throughput test. The regions in each graph where the slope is different from this rate correspond to instances where a NACK was sent since some of the packets in a block were lost and were retransmitted. The experiment proves that the header node can reliably upload the complete data to mobile node irrespective of speed of the mobile node. Also the data transfer is completed comfortably within the minimum expected contact duration of 25

seconds (for train speed of 72 Km/h).

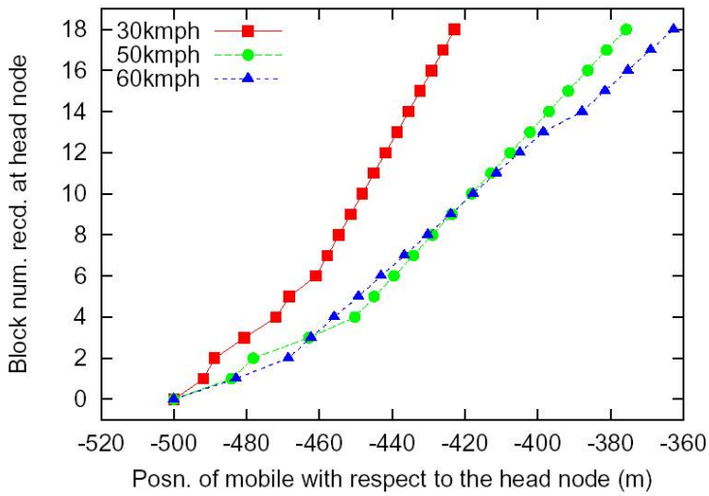

Figure 4.13 Mobile Data Transfer

CHAPTER 5

Past and Related Work

In this chapter we will first discuss the past work done on BriMon. Thereafter we will present the related work.

5.1 Previous work on BriMon

H Hemanth [9] and Nilesh Mishra [10] did some work in their respective thesis work on design of BriMon. Figure 5.1 shows their architecture. They used tmote nodes and soekris [30] (single board computer) in their architecture. In their architecture, all the nodes deployed on the bridge formed one network. Tmote nodes were primarily used for data acquisition and transfer of acquired data to the soekris through a base node. The soekris was used as an aggregator and gateway at the base node. The data collected by soekris was transferred to another soekris kept inside the train on 802.11 radio. They used event detection for triggering data acquisition and transfer of acquired data to the train.

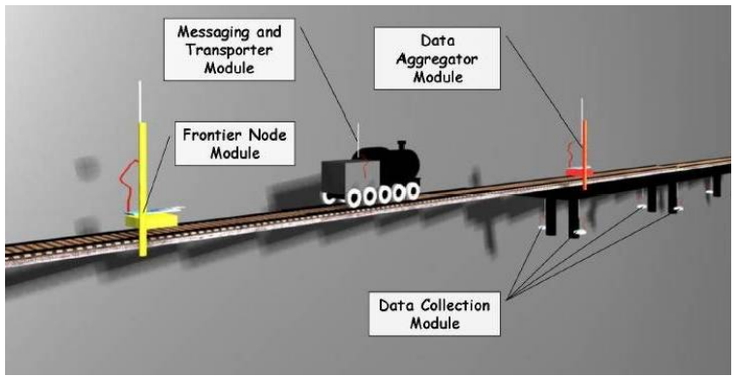

Figure 5.1 Previous BriMon Architecture. Figure reproduced from [10]

The main differences between the old architecture and current design presented in Section 2.4 are as under:

1. In current design we use only tmotes for both data acquisition and data transfer. Hence we use only one type of radio: that is 802.15.4. In the earlier architecture, they used tmotes for data acquisition and data transfer to the base node. The soekris at the bridge (which is connected to base node) has 803.11 radio on which aggregated data is transferred to another soekris board in the train.

2. In current design we consider the fact that vibrations in different spans of the bridge are not correlated. Hence we consider each span as an independent element for vibration measurement. The nodes deployed on a span form independent network. In earlier architecture entire bridge was considered as one element and hence all the nodes deployed formed a single network.

3. The current design uses separate 802.15.4 channels for each independent network mentioned above. Due to this our design scales better in terms of routing, time synchronisation and reliable data transfer. The earlier architecture used only one channel for all the nodes on the bridge hence there are scalability problems.

4. In the current design we use multiple channels on 802.15.4 radio for mobile data transfer whereas in earlier design they used only 803.11 for mobile data transfer.

5. In our design the requirement of frontier nodes is not mandatory. In earlier design the use of frontier nodes was mandatory since the boot up time of soekris board was considerable.

6. In our design we need to handle only power requirement of tmotes whereas in earlier design we need to handle the power requirement of both tmotes and soekris which consumes a lot of power (5 to 6 watt [31]).

Some of the takeaways from the earlier design are:

1. We use the same motes and accelerometers as used in earlier architecture.

2. We use the same switching circuit as used by them for switching the power supply to the accelerometers during sampling and non sampling periods.

5.2 Related Work

In this section we will discuss the existing literature available on wireless sensor network applications and current research work on similar problems elsewhere. We first present the survey of various sensor network based applications available in the literature. We will then compare these applications with our application and design. There are many such applications present in the literature but we choose four application among these for the

comparison. Later we preset the survey of various transport protocols available and then discuss why they are not suitable for our application.

5.2.1 Application Design

The BriMon application requires to acquire the vibration data with the help of accelerometers placed on the spans of the railway bridge. The data acquired needs to be reliably transferred to the central repository for data analysis purposes. The deployment is at an isolated location which lacks power and network connection.

Habitat monitoring [17] represents one of the important class of sensor network applications. The application aims to study the habitat and micro-climate of sea birds habitat at Great Duck Island, Maine. The application has tiered architecture. The lowest level consists of sensor nodes which transmit their data to the gateway. The gateways then transmit the sensor data to the base station through a transit network. The application uses mica motes and current deployment is with 32 motes. Sensor nodes sense temperature, pressure and humidity. The data collection is periodic. Compression techniques have been used to reduce the amount of data to be transferred. The application is a long term deployment, ran for more than six months.

WISDEN [21] is a multi hop structural health monitoring system for large structures. The aim of this work is to replace the existing wired system meant for collecting the vibration data from the different part of the structure. A separate 16bit ADC resolution, accelerometer data collection card is used with an on board microprocessor. The sampling rate used is 200-500 Hz which is similar to the one used in BriMon (400 Hz). The system uses a reliable data transfer and light weight synchronisation mechanisms. Nodes do not perform duty cycling and continuously monitor and stream data. As the data transfer requirements surpass the radio bandwidth, compression techniques like run length encoding and wavelet compression are used. The deployment is short term consisting of 14 micaZ nodes. The architecture is flat, sensor nodes send data directly to PC through the base node.

The deployment of industrial sensor networks [22] is one of the concrete sensor network applications. The application aims to use sensor network technology for predictive maintenance of the equipment. The deployment focuses on vibration analysis for predictive maintenance and uses piezo electric accelerometer to measure vibrations. It uses PSFQ [32]

for reliable data delivery. The network is tiered wherein the motes act as leaf nodes collecting data and route it via a cluster head. The cluster head sends this data to a gateway node which is a Stargate node. The gateway transfers the data using 802.11 radio, ethernet and over internet to the data collection point.

Volcano monitoring [19] uses a wireless sensor network deployment for monitoring volcanic eruptions using low-frequency acoustic sensors. A sensor array of 16 nodes were deployed over 3 km which collected data for three weeks. Due to the nature of event, the system was used with 100% dutycycle i.e. the sensor nodes and sensors are always kept on. The nodes are deployed in a linear fashion. Each node is fitted with an external antenna and the last hop uses a low frequency long distance link. The application uses tmotes for sensor nodes. Due to high resolution requirement of the data, they make use of external ADC cards for data acquisition. A threshhold based event detection method is used to detect happening of interesting events such as volcanic erruptions. An interesting event is reported to the base node. If sufficient number of nodes report an event, the base initiates a data retrieval node-by-node. In BriMon we do not use signal threshhold level for event detection, instead we use the signal from the mote on the train for even detection.

Structural Health Monitoring of the Golden Gate Bridge [33] is another recent bridge monitoring application for road bridge. The deployment consists of 64 nodes deployed in a linear fashion, spread over 4200ft long span of Golden gate bridge in San Francisco. The nodes form multi-hop network and transfer data reliably to base station located on one end of the bridge. The monitoring of bridge is continuous. They monitor the ambient vibrations. The sampling rate used is 1KHz. The architecture of the application is flat.

The comparison of these application is done in Table 5.1. The Golden gate [33] and WISDEN are similar kind of applications. They differ from each other in terms of sampling rates and the size of the network. In the comparison table hence we compare our application with only one of them that is WISDEN.

5.2.2 Transport Protocol

Our application needs a transport protocol for reliable transfer of data at three places during the entire process of transfer of data from the bridge to the data analysis centre. Refer Figure 3.1 for the architecture of our application. First the header node needs to reliably

gather the data from data acquiring nodes and then it needs to reliably upload the complete data to the mobile node(inside the train) within a limited contact duration. Finally the mobile node needs to reliably download data to PC by making using a base node at the data analysis centre. The amount of data we handle is greater than available RAM on all the nodes. Hence the protocol needs to tightly integrate with tmote‘s flash. The fact that tmote's flash and radio share the same bus, requires some intelligent arbitration of radio and flash. Though most of the arbitration is done by lower layers, we still need to do some careful programming exercising cautions.

There are large number of transport protocols available in the literature. Some of these are PSFQ [32], RMST [35], ESRT [36], CODA [34], SenTCP [38] and GARUDA [37]. CODA and SenTCP do not provide reliability and hence were ruled out. ESRT is a event to sink reliable transport and provides only event reliability hence also ruled out. RMST is designed to perform optimally in upstream transfers hence seemed suitable. But it's performance with writing data to flash had not been studied. Also its implementation was not available. GARUDA and PSFQ are designed for downstream transfers and hence were not expected to perform well on upstream transfers.

PSFQ integrates well with flash but for mica platform where the implementation of arbitration for flash and radio handling is different . PSFQ has been used for upstream transfers in an application deployment [22] but its performance was worse than expected. Detailed comparison of all these protocols can be found at [39]. The summary is given at Table 5.2.

While our application was being developed, PSFQ was implemented on tmote platform [29]. But the performance of this implementation for our kind of application has been worse because of inter hop interference and pump slowly paradigm of the protocol. So we set out to design a new reliable transport protocol which was optimal for our application and hardware borrowing ideas from the existing literature. Some of the conceptual differences between PSFQ and our protocol are as under.

1. PSFQ is designed basically for downstream data transfer where as our protocol is designed for upstream data transfer.

2. PSFQ follows pump slowly fetch quickly paradigm whereas in our protocol we follow

pump quickly paradigm.

3. In PSFQ the data transfer on multi hops is simultaneous due to which there is inter hop interference. In our protocol the data transfer is hop by hop hence no or minimal inter hop interference.

	Habitat Monitoring	**WISDEN**	**Industrial Sensor Network**	**Volcano Monitoring**	**BriMon**
Deployment Duration	Long Term (6 Months)	Short term	Few Months (2 Months)	Few Weeks (3 Weeks)	Long Term
Architecture	Tiered	Flat	Tiered	Tiered	Flat
Platform	Mica2	Mica2 and Micaz	Micaz and Imotes	Tmotes	Tmotes
Sensors Used	Temperature, Pressure& Humidity	Accelero-meters	Accelero-meters	Seismo-acoustic sensors	Accelero-meters
Data Collection Model	Periodic	Continuous	Periodic	Continuous and Event based	Event based
Mobile Data Transfer	No	No	No	No	Yes
Multi-channel Data Transfer	No	No	No	No	Yes
On Mote Computing	Raw data collection	Raw data collection	Raw data collection	Event detection	Raw data collection, averaging & compaction
Compression	Yes	Yes	No	No	Partial*

*10 point averaging and compaction

Table 5.1 Comparison of wireless sensor network based applications with BriMon

Attribute	**CODA**	**ESRT**	**RMST**	**PSFQ**	**GARUDA**	**SenTCP**
Direction	Upstream	Upstream	Upstream	Downstream	Downstream	Upstream
Reliability	No	Yes*	Yes	Yes	Yes	No
Congestion control	yes	Passive	No	No	No	Yes

*only event reliability

Table 5.2 Summary of transport protocols

CHAPTER 6

Conclusion and Future Scope Of Work

The Indian Railways consists of large no of bridges, many of which are quite old and are in weak and distressed condition. It is important to have an automated system which is easy to deploy, to monitor the structural health of these large number of bridges spread over a large geographical region. The present systems used for such monitoring are mainly wired systems. These systems are generally bulky and requires expertise in manpower. Also it takes many days to deploy these systems on the bridge.

In this thesis work we develop an automated wireless sensor network (WSN) based system, which makes use of sensor motes and MEMS accelerometers for railway bridge monitoring. The system is easily deployable and requires minimum maintenance. The system, though primarily designed for long term monitoring of remotely located bridges, can also be used for short term monitoring. The system has been developed by making use of off the shelf hardware. We have adopted a application driven approach in developing the system. All the protocols and components of the system have been developed keeping in mind the application requirements. The implemented solution shows how the design choices dictated solely by the application requirement are different from general solutions that exist in the sensor network domain. The system incorporates an event detection mechanism [3] that triggers data acquisition by the *data acquiring nodes* in response to an oncoming train. The data from the *data acquiring* nodes is then reliably gathered at the *base nodes*.

We make use of train itself to automatically transfer the vibration data from the remotely located bridge. For this we employ a event based simple yet effective multi-channel data transfer mechanism to transfer the data from the *base nodes* onto sensor nodes located on the moving train on 802.15.4 radio. The data from these nodes is transferred to data analysis centre when ever the train passes a near by station which provides some network connectivity to the data analysis centre.

The data acquisition system used in the system is DMA based which acquires the vibration data with high fidelity and precision. The transport protocol we have developed is SACK based and is used for reliable transfer of vibration data in all the cases mentioned above. The protocol though application specific can be used in other sensor network applica-

tions such as [19, 21, 22] which require upstream data transfer. The data analysis tool which we have developed has user friendly GUI and it meets all the requirements of the structural engineers for data analysis. This tool being configurable in terms of sampling frequency, FFT points etc, can be used for data analysis from other applications.

In a parallel work on the same project [3], the elements of event detection, routing and time sysnchronisation have been developed. We integrated all the components of the BriMon to make it a complete functional system.

Though many design choices in our prototype were made specific to this application, we believe that the same set of protocols and components would be relevant to many other applications too. In view of this, we are in the process of making a software release to the open source community.

Future scope for work :

In the data acquisition system, the time at which all the nodes start acquiring data is synchronised. However, due to variation of the timing of interrupt handling when a RAM buffer is filled, from node to node, the end times of acquiring data are not synchronised. This needs to taken care of.

There is only one ADC on tmote, using this ADC for acquiring data from more than one axes results in loss of data due to delay (more than 10 ms)in switching of axes fed to ADC. Due to this problem, at present at a node we are able to acquire data of only one axis at a time. This can be improved by making use of a separate card which has 3 ADCs, one for each axis.

We have mentioned in the study that with LZW compression algorithm, we can achieve compression up to 19% in our kind of application. Hence this compression technique, if implemented will further result in reduction of data that need to be transferred. Also at present we have not considered any lossy compression technique. Some of the lossy compression techniques like linear predictive encoding can result in huge reduction of data to be transferred without much loss of inforamtion content.

The present system has been designed to measure the vibration of the bridge during the passage of train (forced vibration measurement) and after passage of train (free vibration measurement). But there can be instances where an earth quake occurs in the vicinity of

the bridge due to which the bridge may get damaged. During such a situation, one would like to assess the condition of the bridge before passing on any train over that bridge. The current system does not have the feature of data collection being triggered by events like earth quake. The future work may thus involve having a complementary add-on providing such a feature to the current system.

In present architecture we make use of two separate trains, one for triggering the data acquisition and the second one for transfer of vibration data from the bridge to data analysis centre respectively. The possibility of using the same train for both processes can be further explored.

Appendix A

BriMon Integration

In this appendix, we describe the guidelines/assumptions under which the current integrated version of BriMon is implemented. We need to mention here that the integration work has been done jointly with Phani Kumar Valiveti [3]. Let us first briefly discuss the interaction that takes place between various modules of BriMon.

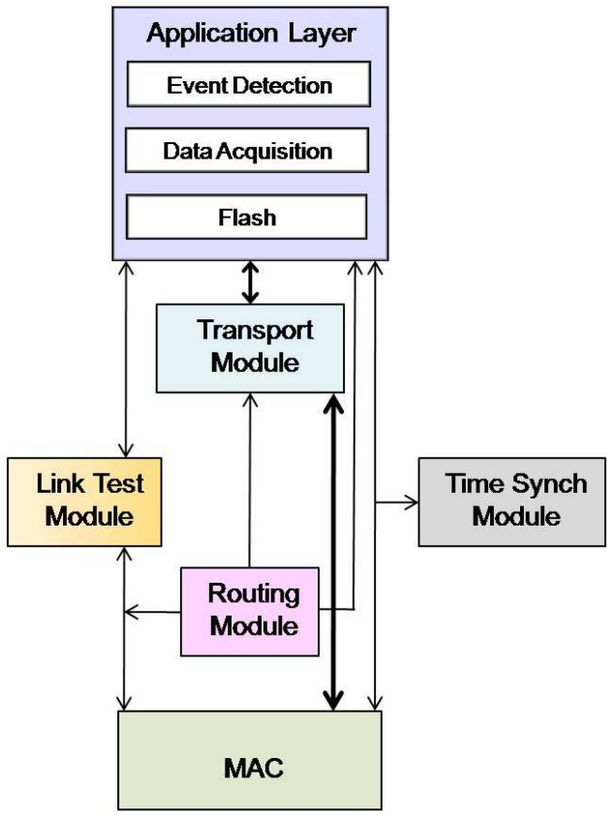

Figure A.1 BriMon Modules

Figure A.1 shows all the modules of the BriMon. Here the architecture is not strictly layered. Transport module is used only for reliable transfer or receive of vibration data. All other messages from the different modules are sent directly to MAC layer. Routing module is responsible for the routing. It provides parent and child information to other modules as shown in the figure. Link Test module is used for debugging. It interacts with the application, routing and MAC layers. Time sync module is used for time synchronisation of the nodes within the network. Application layer handles other functionalities like sleep

wakeup, data acquisition and data storage etc.

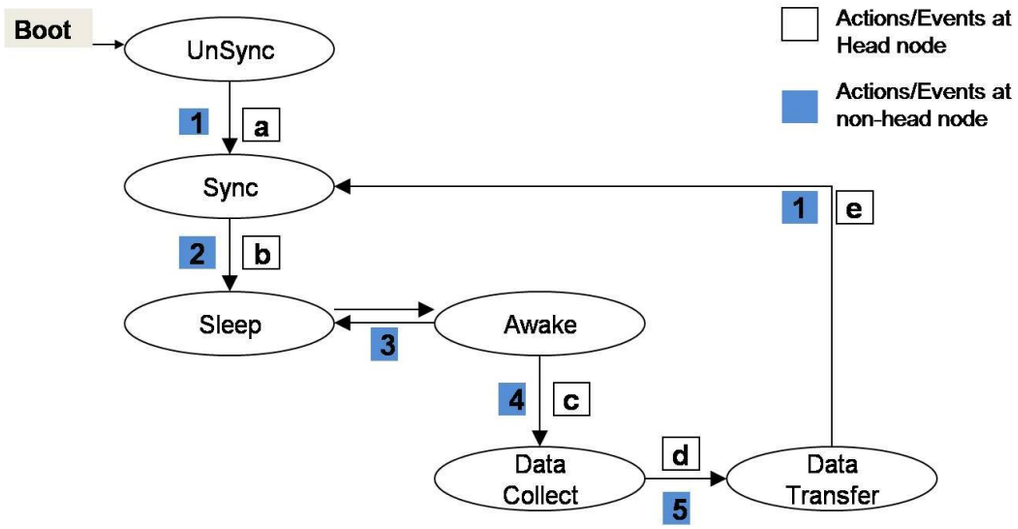

Figure A.2 State Diagram of BriMon cluster (Ideal case of no message losses)

Let us now discuss the integration in a bit detail. To start with we identified different states in which a node in BriMon can find itself, at any point of time. Later, the actions and events at various layers/modules that are responsible for a node to move from one state to the other are identified. Thus, at this point we can be clear about what are the interfaces to be provided/used by any module of BriMon. The state diagram of a BriMon cluster, showing different actions/events at head node and a non head node is shown in Figure A.2. Note that this diagram depicts the case in which a node does not get out of synchronization/unconnected due to message losses. The actions/events are now listed as follows.

Actions taken & events generated at a non-head node :

1. The event *SyncMsg_Received* is generated by TimeSync layer and signaled to the BriMonApp layer. (*Offset* to the global clock reported)

2. BriMonApp receives a *SleepCommand* from parent node. Then, the BriMonApp issues the command for the node to start *Sleep_ Wakeup* cycle after X interval, where

$$X = (currentGlobalTime/SleepWakeupPeriod + 1) * SleepWakeupPeriod - currentGlobalTime \quad (A.1)$$

and $currentGlobalTime = LocalTime + Offset$.

3. The *SleepTimer/AwakeTimer* fires at the layer in which sleep-wakeup is implemented.

4. BriMonApp receives *Data_CollectMsg* message from parent node, mentioning the global time Y to start sampling the ADC. Then, the BriMonApp stops the sleep-wakeup cycle and issues the command to Data Collection Module to start sampling after T2 interval, where *T2 = Y - presentGlobalTime*.

5. BriMonApp receives a Data Transfer message from parent/head node. The BriMonApp then issues the command to Transport Layer to send the collected data.

Actions taken & events generated at a head node : :

(a) On boot up, BriMonApp issues a command to TimeSync Layer to start synchronization. Then the TimeSync layer reports back to the BriMonApp that synchronization is done.

(b) BriMonApp issues a command to start *sleep/wakeup* cycle.

(c) BriMonApp receives an E*vent_Detected* message from EventDetection layer. Then, it stops its own sleep-wakeup cycle and will issue a command to the nodes to start data collection at a global time Y. The head node itself also does the data collection.

(d) On completion of data collection, the BriMonApp issues a command to each node, one by one, requesting for the transer of data.

(e) After the data transfer from all the nodes is done, the BriMonApp issues the command to the TimeSync layer to start synchronization.

Appendix B

TOSMsg Packet Format

TOSMsg packet structure is as shown below. It includes the MAC header and the MPDU (MAC Protocol Data Unit). All the active messages we transmit, 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;

```
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

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.

References

1. Forty percent railway bridges over 100 years old.
   http://economictimes.indiatimes.com/40\%5C_railway\%5C_bridges\%5C_over\%5C_100\%5C_years\%5C_old/articleshow/1967908.cms

2. 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 1324, New York, NY, USA, 2004. ACM Press.

3. Phani Kumar Valiveti. Routing and Time Synchronization Protocols for Low Duty Cycle Operation of a Sensor Network Based Bridge Monitoring System. Mater's Thesis, Department of Electrical Engineering, IIT Kanpur, India, May 2007.

4. Micaz nodes.
   http://www.xbow.com

5. Tmote-sky nodes.
   http://www.moteiv.com/products-tmotesky.php

6. TinyOS operating system for Sensor Networks.
   www.tinyos.net

7. A Dynamic Operating System for Memory-Constrained Networked Embedded Systems.
   http://www.sics.se/contiki/

8. MANTIS.
   http://mantis.cs.colorado.edu/tikiwiki/tiki-index.php

9. H. Hemanth. BriMon: Design and Implementation of Railway Bridge Monitoring Application Mater's Thesis, Department of Computer Science and Engineering, IIT Kanpur, India, May 2006.

10. Design Issues and Experiences with BRIMON Railway BRIdge MONitoring Project. Mater's Thesis, Department of Computer Science and Engineering, IIT Kanpur, India, Aug 2006.

11. Dr. C.V.R. Murthy, Professor, Dept of Civil Engineering, IIT Kanpur Dr. K.K. Bajpai, Structures Lab, Dept of Civil Engineering, IIT Kanpur

12. Analog Device's low power MEMS $\pm$2g accelerometer.
    http://www.analog.com/UploadedFiles/Data_Sheets/ADXL103_203.pdf

13. MMA7260Q Accelerometer.
    http://www.freescale.com/files/sensors/doc/fact_sheet/MMA7260QFS.pdf.

14. Sukun Kim. Wireless Sensor Networks for Structural Health Monitoring. Masters thesis, U.C.Berkeley, 2005.

15. Exploration of Sensor Network Field deployment on a large Highway Bridge and condition assessment.
http://healthmonitoring.ucsd.edu/documentation/public/VincentThomasTesting.pdf.

16. B. Raman, K. Chebrolu, N. Madabhushi, D. Y. Gokhale, P. K. Valiveti and D. Jain. Implications of Link Range and (In)Stability on Sensor Network Architecture In *WiNTECH 2006*, A MOBICOM'06 Workshop, Sep 2006.

17. R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson and D. Culler. An analysis of a large scale habitat monitoring application In *SenSys'04*, Nov 2004.

18. G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay and W. Hong. A macroscope in the redwoods In *SenSys'05*, Nov 2005.

19. G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring Volcanic Eruptions with a Wireless Sensor Network In Proc. European Workshop on Sensor Networks (*EWSN'05*), January 2005.

20. G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz and J. Lees. Deploying a Wireless Sensor Network on an Active Volcano In *IEEE Internet Computing*, March-April, 2006.

21. Jeongyeup Paek, Krishna Chintalapudi, John Cafferey, Ramesh Govindan, and Sami Masri. A Wireless Sensor Network for Structural Health Monitoring: Performance and Experience. In EmNetS-II, May 2005.

22. 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, Nov 2005.

23. Hoi-Sheung Wilson So, Giang Nguyen, and Jean Walrand. Practical Synchronization Techniques for Multi- Channel MAC. In MOBICOM, Sep 2006.

24. M. Marti, B. Kusy, G. Simon and . Ldeczi. The flooding time synchronization protocol In *SenSys'04*, Nov 2004.

25. Capturing High-Frequency Phenomena Using a Bandwidth-Limited Sensor Network Ben Greenstein, Christopher, Mar Alex, Pesterev, Shahin Farshchi, Eddie Kohler, Jack Judy, Deborah Estrin. University of California, Los Angeles

26. Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks Christopher M. Sadler and Margaret Martonosi Department of Electrical Engineering Princeton University.

27. LZW Data Compression.
http://marknelson.us/1989/10/01/lzw-data-compression/

28. T. A. Welch. A Technique for High-Performance Data Compression. IEEE Computer, 17(6):819, June 1984.

29. Manu Bansal, B. Raman : A PSFQ Implementation Study,CS397, IIT Kanpur, India, Dec 2006.
http://braman4.cse.iitk.ac.in/~bhaskar/temp/psfq/ psfqimpl_report_cs497.pdf

30. WiFi and ethernet enabled single board computers.
http://www.soekris.com

31. Nilesh Mishra, Kameswari Chebrolu, Bhaskaran Raman, Abhinav Pathak Indian Institute of Technology, Kanpur: Wake-on-WLAN

32. 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 111, New York, NY, USA, 2002. ACM Press.

33. Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon. Structural Health Monitoring of Golden Gate Bridge (using MEMS accelerometers). In *International Conference on Information Processing in Sensor Networks (IPSN '07)*, Cambridge, MA, April 2007.

34. 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 266279, New York, NY, USA, 2003. ACM Press.

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

36. B. Akan and Ian F. Akyildiz. Event-to-sink reliable transport in wireless sensor networks. IEEE/ACM Trans. Netw., 13(5):10031016, 2005.

37. Garuda.
http://www.ece.gatech.edu/research/GNAN/work/garuda/garuda.html

38. C. Wang, K. Sohraby, and B. Li. SenTCP: A hop-by-hop congestion control protocol for wireless sensor networks. IEEE INFOCOM 2005 (Poster Paper), March 2005.

39. C. Wang, K. Sohraby, Bo Li, andW. Tang, Issues of Transport Control Protocols for Wireless Sensor Networks.