Routing and Time Synchronization Protocols for Low Duty Cycle Operation of a Sensor Network Based Bridge Monitoring System

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

MASTER OF TECHNOLOGY

by

Phani Kumar Valiveti
Y5104042

to the

Department of Electrical Engineering,
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

May, 2007

Certificate

This is to certify that the work contained in the thesis entitled *"Routing and Time Synchronization Protocols for Low Duty Cycle Operation of a Sensor Network Based Bridge Monitoring System"*, by *Phani Kumar Valiveti*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

May 2007

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

May 2007

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

ABSTRACT

Railways form a crucial part of transport infrastructure in many parts of the world. A large fraction of bridges used by railways may be considerably old and may be in use for decades. For instance, Indian Railways has about 127,000 bridges of which 51,000 are about 100 years old. For smooth functioning of the system, safety of travel over these bridges must be assured. For this, a structural health monitoring system is required, that is capable of indicating any deterioration of physical condition of the bridges, thereby calling for maintenance. Existing techniques are mostly wired solutions, requiring technical personnel to be present at the bridge site during the inspections.

In this work, we present BriMon, a Wireless Sensor Network based structural health monitoring system that has the essential features like ease of deployability, long life with minimum maintenance, and remote monitoring. It also satisfies the constraints imposed by structural engineers on data collection and analysis. The solution is based on wireless sensor motes and MEMS accelerometers. We have designed various mechanisms and protocols required for providing the afore-said features in an application specific manner. The implemented solution shows how the design choices dictated solely by the application are different from the general solutions that exist in the sensor network domain. We implemented the solution on Moteivs Tmote-sky and used TinyOS and nesC for programming the motes.

In this work, we discuss the routing and time synchronization protocols that we have designed, implemented and evaluated. These are very simple, light-weight and efficient in our application setting, when compared to existing protocols in literature. We also describe a signal strength based event detection mechanism in order to detect the oncoming train. This subsequently triggers either data collection or mobile data transfer to a node on the train depending on the context. The event detection mechanism allows us to detect the oncoming train at a distance of about 1Km, which allows the nodes to sleep-wakeup with a very low duty cycle of 1-2%. The moving train itself is used as a carrier of the vibration data collected at the bridge, which is one of the novel aspects of this design. We also discuss about the careful integration of the above components of routing, sleep-wakeup and time synchronization with the other aspects of BriMon namely high fidelity data aquisition, reliable data transfer onto moving train.

Acknowledgments

Firstly, I would like to thank Prof. Kameswari Chebrolu who has given me the opportunity to work on a cutting-edge technology. It was with her encouragement in the initial stages that I felt confident to enter a domain that was completely new for me. Next, I would like to thank Prof. Bhaskaran Raman for his excellent mentoring and guidance throughout the work. Working with Prof. Kameswari Chebrolu and Prof. Bhaskaran Raman has provided a very good practical experience, given the interest they have shown in conducting outdoor experiments without any compromise. Thanks to both of them for a memorable Bangalore trip. Thanks for the critical reviews of the report by Prof. Bhaskaran Raman, due to which it has taken a good shape. I would like to thank Prof. C.V.R. Murthy and Dr. K.K. Bajpai for providing us with critical inputs from structural engineering domain, which have helped us a lot in designing our system. Also, thanks to Prof. C.V.R. Murthy for accompanying us on a trip to bridge for an initial validation of our work.

It has been a memorable working experience with Maj Raj Kumar, 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 has also helped us in the initial stages. Experiments conducted in the initial stages of my work with Naveen and Gokhale have been very useful in the later stages. I would like to extend my thanks to the staff of Media Lab Asia, IIT Kanpur for managing logistics during some of our experiments.

Life at IIT Kanpur has been full of fun with friends from different places. Endless treats with friends and our *batch* from Hall-7 have taken place in these two memorable years of stay. Also, a special mention needs to be done about the fun-filled get-togethers followed by delicious dinners at Bhaskar and Kameswari's home. Also, thanks to Raj, Naveen and their families for the affection that they have shown.

This work would not have been possible, if it were not for the love and affection of my parents and brother. It is with their encouragement and care throughout my life that I have come up to this level.

I dedicate this work to my parents, brother and teachers through out my academic life.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Railways are the most prevalent transport infrastructure in many parts of the world. For instance, Indian Railways is one of the largest enterprises in the world. And bridges form a critical part of railways. Indian Railways has approximately 127,000 bridges of which about 51,000 are more than 100 years old [25]. In order for this system to operate smoothly, assurance of safety of travel over bridges is crucial.

The life of a bridge is not based on its age but on physical condition as ascertained by inspections. Generally, the vibrations of these structures are measured using sensors called accelerometers during the inspection. Bridges showing signs of deterioration of physical condition indicate need for repair and are classified as distressed bridges. A recent report from the Indian Railways [25] says that about 50 billion rupees have been budgeted over the period of 2001-2008 for repair and maintenance of about 11,000 bridges. This shows the criticality of the problem.

Most of the current systems used for this purpose of structural health monitoring (SHM) require technical personnel to be physically present and inspect the bridges. They also use very expensive and bulky equipment. Thus, there is an utmost requirement for an easily deployable system with minimum maintenance, that monitors the physical condition of bridges and indicates the need for maintenance.

1.1 Problem Statement

The statement of the thesis problem is as follows:

> *To build an easily deployable, scalable and maintenance-free system that measures the structural vibrations of a bridge located at a remote place and transfer the data to a repository*

The vibration data that is measured should satisfy the constraints on the features of data imposed by the structural engineers for the purpose of analysis like

1. Time synchronization across readings from several points

2. Duration for which data is collected

3. Required sampling rate

1.2 Challenges

While providing a solution with the features described above, using the technology of Wireless Sensor Networks, there are several challenges that need to be addressed:

- Limited source of energy: Since the sensor nodes are placed in a remote location and are expected to work for long time with minimum maintenance, the availability of power becomes a problem. Generally these are powered up using batteries or such limited power sources. So, there is a need for the nodes to conserve energy for a prolonged life time. They have to be made to follow a periodic sequence of low power and normal power consuming modes, with the duty cycle of the nodes being minimum.
- Detection of Event: This is a challenge that arises due to the duty cycling described above. The nodes should know when exactly to sense the vibrations, since they are required to be measured only for a specific duration of time, when there is a passing train. But the time of train arrival is unpredictable.
- Keeping the nodes connected: Even if a few nodes in the network fail, the remaining nodes need to stay connected. At the same time, there must be flexibility to replace the non-functional nodes with new ones.
- Limited platform capabilities: Sensor motes which we use in this work, have certain limitations in terms of computation, program and data memory and communication range. We need to be abide by these limitations or enhance the capabilities by using complementary hardware (for example, external antennas are used to enhance range).

1.3 Thesis Contributions

In this thesis, a solution that meets the goals described, while also addressing the challenges is provided. The solution set consists of the following components:

- A routing protocol for network formation of sensor nodes
- Implementation of sleep-wakeup for duty-cycling.

- An event detection mechanism that aids the data transfer as well as triggers the data collection
- Time synchronization of the sensor nodes

Apart from design and implementation on tmote platform, we also justify and substantiate the usability of these components via various evaluation experiments.

Note that apart from the components listed above, the solution set requires the following, for completeness. These are developed in a separate and parallel work by Raj Kumar [22].

- Data acquisition with high fidelity using DMA based method
- Reliable data transfer on to a mobile node as well as within the network of nodes
- Data analysis system as per the requirements of structural engineers

These components are once again developed on tmote platform and are integrated to the other components mentioned.

1.4 Organization of the report

Chapter 2 provides relevant background information on Structural Health Monitoring and Wireless Sensor Networks. This also presents the design overview of the entire system of BriMon in detail. Chapter 3 presents the past work done by Hemanth Haridas [23] and Nilesh Mishra [24] on BriMon. This also presents the related work in the domains of routing protocol and time synchronization mechanisms. Chapter 4 gives the design and implementation details of the routing protocol used for BriMon. Chapter 5 gives the design and implementation details of the time synchronization protocol used for BriMon. Chapter 6 describes the experiments done for validation of event detection, and evaluation of routing and time synchronization protocols. Finally we conclude the work in Chapter 7 and specify the future work.

CHAPTER 2

Background and Design Overview

This chapter provides the necessary background to the reader to understand the rest of the report. Section 2.1 provides background on Structural Health Monitoring as a whole, highlighting the case of bridges. Section 2.2 provides the necessary background on Wireless Sensor Networks. Next the design overview of BriMon is explained in Section 2.3.

2.1 Structural Health Monitoring (SHM)

By definition, in SHM we monitor, directly or indirectly, various parameters of a structure for identification of its current state. Based on the state we can assess whether the structure is damaged or not. Subsequently, we can locate the damage and estimate the remaining useful life for safe usage of the structure. Since SHM techniques fall under the non destructive analysis domain, they are gaining prominence amongst the structural engineering community with advancement in theory and equipment technology. To summarize SHM is used for:

- Damage detection i.e. is there change in the state of the structure.
- Damage localization i.e if there is a damage, can we locate the site of damage.
- Damage assessment i.e. how extensive is the damage and what is its impact.
- Lifespan prediction i.e. what is the life remaining of the structure for safe use.

The life of a structure is not dictated by its age but by its physical state. In order to operate safely one needs SHM methods for assessment of risk from time to time. The property of the structure often monitored in SHM systems is vibration. This is measured using sensors called accelerometers. From the point of view of the civil engineers there are three types of vibration which are important:

1. *Forced vibration* is the vibration induced in the system when the source of vibration is still pumping energy into the system. For e.g., this situation comes up when a moving train passes over a bridge and thus imparts it some energy to make it vibrate.

2. *Free vibration* happens when the source of energy is removed from the structure in question but the residual energy remaining in the system keeps it vibrating. E.g. when a locomotive crosses a bridge, the latter keeps vibrating for some time even after the vehicle has moved away from the bridge.

3. *Ambient vibration* is the vibration in the bridge when no intentional source is pumping energy into the structure. The source of the vibration can be attributed to wind, passage of vehicle on a nearby road, seismic vibrations, etc.

SHM of Bridges :

We now turn to provide a brief background on bridge monitoring. The details presented here drive several of our design choices in the later chapters. This information was gathered through extensive discussion with structural engineers [42].

General information on bridges: 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 metres to a few km.

What & where to measure: Accelerometers are a common choice for the purposes of monitoring the health of the bridges. 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 2.1.

The vibration 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. An important point to note here is that vibration data across different spans are independent of each other i.e. they are not physically correlated.

When, how long to collect data: When a train is on a span, it induces forced vibrations as described earlier. After the train passes the bridge, the structure vibrates freely that creates natural (free) vibrations with decreasing amplitude till the motion stops.

With regard to SHM of bridges, Structural engineers are mostly interested in

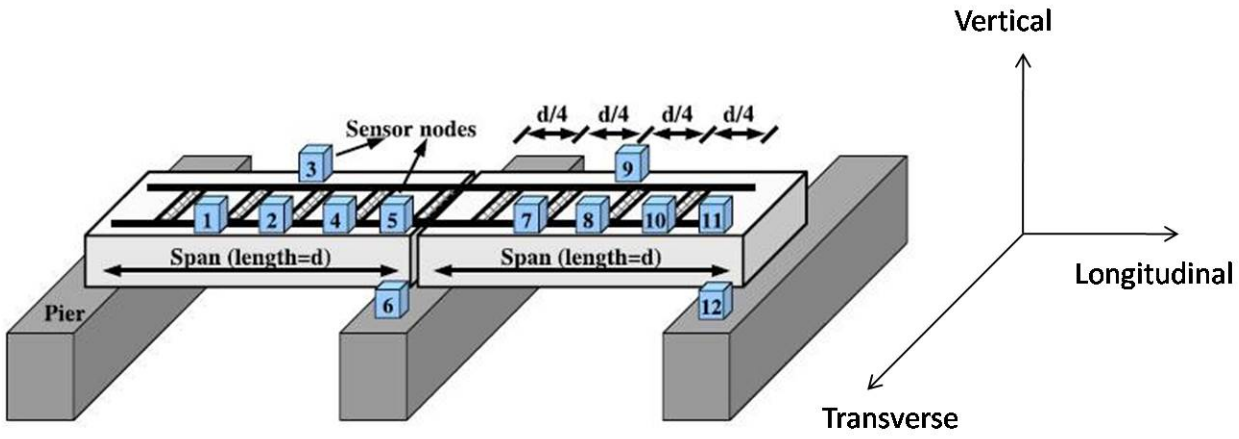


Figure 2.1 Nodes on a double-span

- The natural and higher order modes of this free vibration as well as the corresponding damping ratio.
- Peak magnitude induced by the forced vibrations.
- Data duration equivalent to about five time periods of oscillation for both forced as well as free vibrations.

The frequency components of interest for these structures are in the range of about 0.25 Hz to 20 Hz [26, 27]. 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).

Time synchronization requirement: 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.

2.2 Wireless Sensor Network Technology

A wireless sensor network (WSN) [38] is a *wireless network* consisting of spatially distributed autonomous devices using *sensors* to cooperatively monitor/measure physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants, at locations of interest.

Wireless sensor networks are currently being developed in many civilian application areas, including environment and habitat monitoring, health care applications, home automation, structural health monitoring and traffic control as well as military applications such as battlefield surveillance. Some of the foremost deployments are the habitat monitoring applications such as the great duck island experiment [33], studying the redwood tree macroscope project [32], Zebranet project [34] etc. Some of the recent deployments falling under category of structural health monitoring are [31, 27, 15].

Components of a Sensor node

Sensor nodes are small modules having the capabilities of

- Sensing physical quantities like temperature, pressure, vibration etc.
- Storage and processing of the data collected.
- Communication with another such module.

They are generally powered up by limited sources of energy like batteries, considering the form factor acceptable by applications. In applications requiring long-term deployments, limited power supply poses a requirement on the programmer to put off the power consuming components like radio, sensors, when not in use.

Generally, all the components of processing, storage and communication are included into a single board device called *mote*. Some of the platforms also have onboard sensors for temperature, humidity or light. A mote can be equipped with external or internal radio antennas depending on the compactness of the design and the range of communication required.

Many such platforms exist of which Mica, Mica2, telos, telosb, tmote are a few. One of the most commonly used operating systems for sensor mote platforms is TinyOS [40]. 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. The components have been developed for a number of common platforms like Mica, Mica2, telos, telosb, tmote. It provides interfacing at various levels including direct interfaces with hardware, interface for abstraction of common features and programming level application interfaces.

Other operating systems include Contiki [36], MANTIS [37], BTnut [39]. We use TinyOS in this development.

2.3 Design Overview of BriMon

With the goals of the system as enlisted in Section 1.1, we now proceed to explain the complete design overview of BriMon. As was mentioned, the design of all the components of BriMon is entirely based on the specific application needs and customized to it. For clarity, we first explain all the stages of the application exactly as it happens in a real deployment. Later, we discuss the design approach of the main components that build-up the system.

2.3.1 Stages of BriMon

Figure 2.2 gives a schematic of the entire system of BriMon (the exact placement of nodes is better visible in Figure 2.1 though).

Components of a sensor node in BriMon : Each node in BriMon consists of a sensor mote, an accelerometer chip, an external antenna (internal antenna of tmotes may suffice for bridges having short spans) and power supply batteries. All this hardware is packaged in a moisture proof box while deploying.

Deployment of sensor nodes on bridge : The sensor nodes each containing the components described above should be placed on the spans of the bridge at the strategically chosen positions shown in Figure 2.1, such that the position of each node (identified by its address) is known. When the nodes boot up, they are initially idle.

The vibration data to be collected by nodes on a given span will be correlated as was mentioned in Section 2.1. So, we need to time synchronize only the six nodes (twelve in case of a double-span) on a given span. We can do so if these six nodes (or twelve nodes as appropriate) constitute a network.

Cluster and *Cluster Head*: We designate one of the six nodes on a span as a *head node* and we call the group of six nodes a *cluster*. The cluster head has several additional responsibilities (that become clear as we proceed), other than collection of data itself.

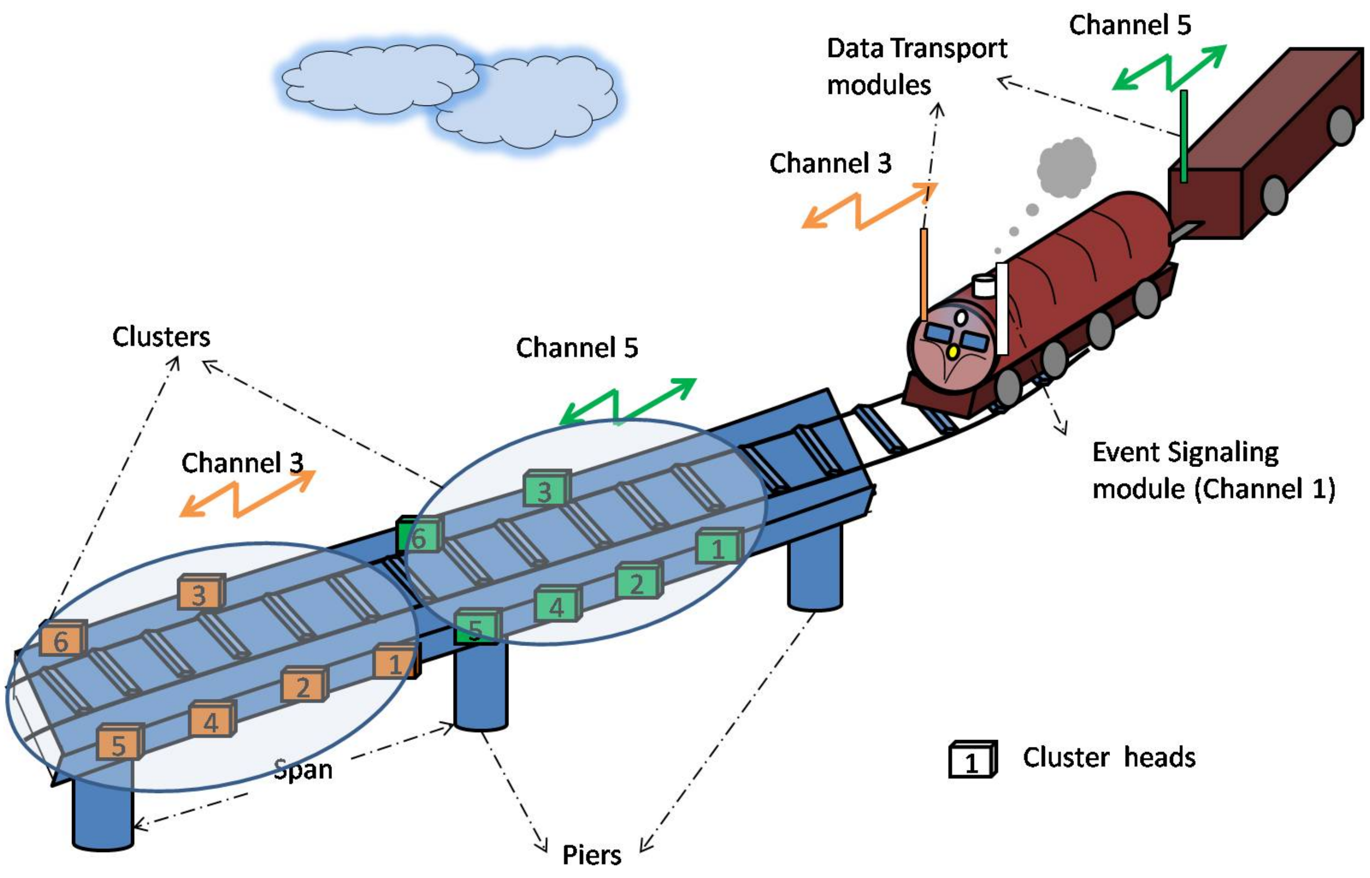


Figure 2.2 Architecture of BriMon

Thus, we will have as many clusters as the number of spans in a bridge. The nodes in different clusters are programmed to communicate on different radio channels. Importantly, the adjacent clusters must be programmed to operate on channels far from each other among the sixteen possible 802.15.4 channels available in the frequency range of 2.4 GHz (CC2420 radio chip, used by tmote operates in this frequency range). Normally a channel separation of 2 should be enough because CC2420 provides an alternate channel rejection of about 54 dB [19].

Network Formation : The next task is to make the nodes in a cluster *connected*. The routing protocol will be run in all the clusters independently in individual clusters. This results in a formation of topology in which each node of a cluster will have a path towards the head node of the cluster and the head node acts as the root of the routing tree. The reason for this will become apparent as we proceed. This topology information has to be available on every node in a cluster.

Thus, the head node can give commands to the nodes in the cluster to carry out the necessary tasks like

- *Data Collection*: During this, each node will sample the ADC at a rate of at least 400Hz*. The duration in general can be 40 seconds, as was mentioned in the previous section. Also, the nodes must perform the sampling operation in synchrony, with in an error of 5ms as was explained in Section 2.1.

- *Data Gathering*: After the data has been collected, the head node holds the responsibility of transferring the data from all the nodes in its cluster to a repository (how it is done will be explained later). Before doing this, the head node *gathers* the data collected by individual nodes. So, it issues a command to each node to pass on its data. A node on hearing to this command should react by either forwarding the command if it is not intended to it, or sending its data via its parent node to the root node.

- *Sleep-Wakeup*: Each node will go through a periodic sleep (radio, ADC and sensor are turned off) and wakeup (all components are on) on getting the command from head node. Moreover, so as to be able to receive any further commands, they do so in synchrony. The durations of sleep and wakeup are decided as per the constraints that will be explained as we proceed.

Time Synchronization: Thus, for the above operations to happen, the next crucial component required is time synchronization. Time synchronization of the nodes is carried out by our custom designed light-weight protocol. At the end of this, all the nodes in a cluster will be time synchronized apart from being connected.

Command System: For the commands to propagate through the cluster there must be a mechanism that does so in minimum possible time. We use a TDMA based flooding scheme for command propagation. This scheme is exactly the same as used by the time synchronization module and is explained in Chapter 5.

Once the nodes are time synchronized, the head node may issue the command for sleep-wakeup and the nodes go through duty-cycling. Now, it is the responsibility of the head node to identify when the train comes, so as to issue the command for data collection. For this, the mechanism of event detection will be required.

*As was mentioned in Section 2.1, the sampling rate requirement is 40Hz. But in order to reduce noise, we do a ten-point averaging of the data for which we have to do sampling at ten times more than 40Hz

Event Detection: The head node will have to detect the arrival of an oncoming train. For this, we equip the oncoming train with an event signaling module, that continuously emits beacons on 802.15.4 radio. The head node, during one of its wakeup cycle will hear to such beacons and ceases its duty cycling. It further issues the command to the nodes in its cluster to start data collection at a particular time of its clock (note that all the nodes are time synchronized to head node).

There is a subtle problem of channel assignment in the above setting. As explained, the nodes in a cluster are programmed to operate in a channel that is different from that of the other clusters. But the operation of event detection is common to all the clusters. Thus, we will have to assign a channel especially for this purpose of event detection and all the head nodes of clusters on a bridge, during their awake period will be listening on this common channel. Once they detect an oncoming train, they switch their operating channel to that assigned to their respective clusters.

Once the data collection is over, the head node will issue command to individual nodes to forward their data. It thus gathers the entire data and writes it into its flash memory.

Quantity of data: As mentioned earlier, each node collects 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. 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: $3channels * 12bits * 40Hz * 40sec = 57.6Kbits$. As explained there will be six sensor nodes per span, and a maximum of twelve nodes. Thus the total data we have per span per data collection cycle is a maximum of $57.6 * 12 = 691.2Kbits = 87KBytes$.

With all the data written into the flash, the head node once again will issue the command of sleep-wakeup to the nodes. The cluster thus continues its duty cycling once again. The head node now is operating on the common channel assigned for event detection. The remaining task for it is to transfer this data to some repository.

We use the trains themselves as the carriers of this data. The next train coming on that bridge will be equipped with Data Transport modules as shown in Figure 2.2. So, when the train arrives, all the cluster heads will detect that and switch to the respective channels of their clusters.

Mobile Data Transfer : The train will be equipped with separate data transfer modules for individual clusters, operating in respective channels. Thus, each cluster head will upload the data in its flash on to the corresponding mobile node, which stores it.

Later, when the train reaches some station, that has Internet connectivity, this data can be collected and transfered to a central repository where the analysis is done.

Having seen the entire elements of the architecture, let us proceed to the details of the design approach for important elements of the design namely event detection, routing, time synchronization and mobile data transfer.

2.3.2 Design Approach to Elements of BriMon

Event Detection :

Our model for event detection is depicted in Figure 2.3. We have an 802.15.4 node in the train which beacons constantly. Let D_d denote the maximum distance from the bridge at which beacons can be heard from the train at the first node (node-1 in Figure 2.2), if it were awake. We denote by t_{dc} the maximum time available between the detection of the oncoming train, and data collection. Thus $t_{dc} = D_d \div V$ where D_d is the maximum distance at which train can be detected and V is the speed of the train (assumed constant). When we think in the context of sleep/wakeup, the maximum duration for which the nodes can sleep, is t_{dc}. In our design as explained, all nodes duty cycle, with a periodic sleep/wake-up mechanism

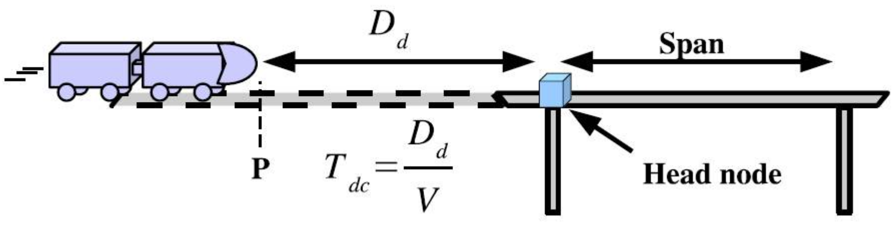


Figure 2.3 Event Detection Model

as was mentioned. Let us denote the duration of the sleep/wake-up/check cycle as t_{cc} which consists of a duration t_{sl} of sleep time and a duration Tw of awake time. Thus $t_{cc} = t_{sl} + t_w$.

Clearly we would like to have as large a t_{cc} as possible to reduce the duty cycle. Thus, we should increase this detection range as much as possible. We will explain how the minimum value of t_w is chosen, in the discussion to follow.

Routing Protocol :

We now explain the approach to the routing protocol used in BriMon. The first significant question which arises here is what is the expected stability of the routing tree; that is how often this tree changes. This in turn depends on link stability. We observed (and will be explained in detail in Chapter 4) that when we operate links above a certain threshold RSSI (received signal strength indicator), they are very stable, even across days. Below the threshold, the link performance is unpredictable over small as well as large time scales (few sec to few hours).

With such a threshold based operation, it is observed that link ranges of a few hundred metres are easily achievable with off-the-shelf external antennas. The measurements we later present in Chapter 4 also prove this. Note that using external antennas is not an issue in BriMon since we are not particularly concerned about the form-factor of each node. Now, recall that a physical span length is about 125m in the worst case, and a data-span length can thus be about 250m maximum. Given a link range of 100m or so, this implies that we have a network of at most about 3-4 hops. In such operation, the links will be quite stable over long durations of time, with close to 0% packet error rate.

This then answers most of our questions with respect to routing. The protocol used can be simple, only needing to deal with occasional node failures. We need to run the routing protocol only occasionally. And time synchronization can effectively assume the presence of a routing tree, which has remained stable since the last time the routing algorithm was run.

Time Synchronization :

We require time synchronization for two purposes as explained : for the periodic sleep/wakeup mechanism, and for time-aligning the sensor data from different nodes.We adopt the design approach of not seeking to estimate the exact clock drifts, as this normally adds considerable complexity to the time synchronization protocol. We justify this as follows. We shall show in Section 6.1 that our periodic sleep/wake-up has a period of the order of 30 seconds. The minimum wakeup duration of the nodes is the duration taken by a command to propagate (t_{pc}) in the entire network along with any synchronization error (t_{Δ}). Since the error can be positive or negative, we should have $t_w = t_{pc} + 2 * t_{\Delta}$. As will be shown in Section 6.3, t_{pc} will be of the order of 80ms, where as t_{Δ} will be of the order of 0.2ms. Leaving some margin, we chose to have a wake up duration of around 200ms.

Thus, we can have a lightweight time synchronization mechanism run during every wake-up duration, at no extra cost. In the time-period between two wake-up durations, of about a minute, the worst case clock drift can be estimated. The work in [13] reported a worst-case drift of about 20ppm for the same platform as ours. This means a maximum drift of 1.2ms over 60s. This is negligible as compared to our wake-up duration, and hence exact drift estimation is unnecessary. With respect to our application too, the time synchronization requirement is not that stringent. Recall from Section 2.1 that we require only about 5ms or less accuracy in synchronization. So this too does not require any drift estimation.

Mobile Data Transfer :

In BriMon, we use the passing train itself for transferring the data collected. The data is then ultimately delivered to a central repository. 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. The implementation of this component of the design is done in a parallel work by Raj Kumar [22].

CHAPTER 3

Past and Related Work

3.1 Past work on BriMon

There was some past effort [23, 24] towards the design of BriMon. The architecture assumed in their work uses a combination of 802.11 and 802.15.4 nodes. A schematic of the architecture used in their work is shown in Figure 3.1.

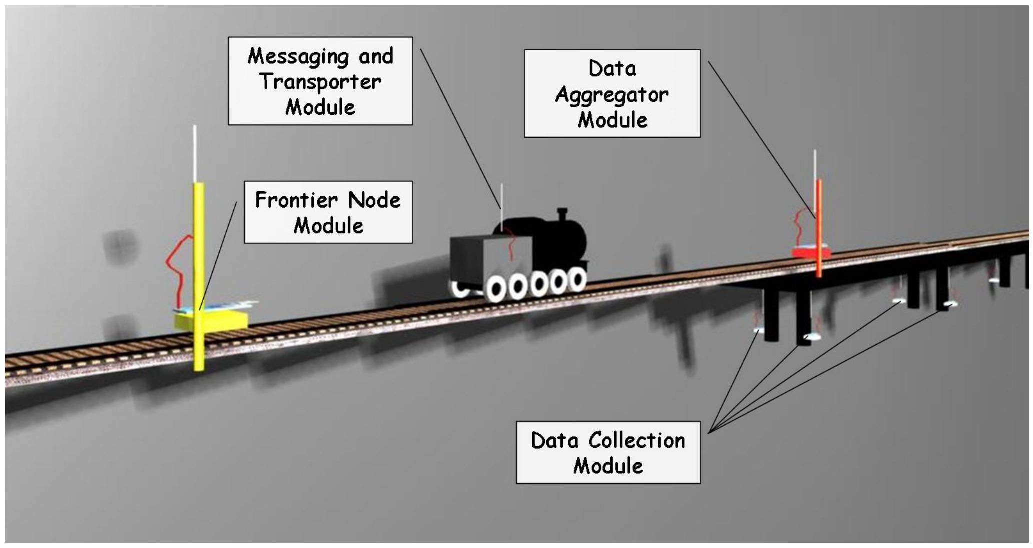


Figure 3.1 Earlier Architecture of BriMon (*courtesy: Nilesh Mishra* [24])

The main differences of this old architecture from the current design presented in Section 2.3 are the following:

- All the nodes on the bridge are identical in terms of their components, in the current architecture whereas in the earlier architecture, the data collection modules and data aggregator modules differ: The data aggregator module makes use of additional hardware called Soekris, which has a serial interface with mote and an 802.11 compliant radio.

- In earlier architecture, nodes on the entire bridge were made to synchronize rather than nodes on a span. However, this is not required since the vibrations across different spans will anyway not be correlated.

- In earlier architecture, they were trying to form a network of all the nodes on the bridge, where as in the new architecture, we make only the nodes on a span (or double span depending on bridge) to form a network.

- 802.11 was intended to be used for mobile data transfer in the earlier architecture, where as we use 802.15.4 in the new architecture.
- The earlier architecture makes use of only one radio channel at a time, where as we use multiple channels simultaneously in new architecture.
- In the earlier architecture, scalability was a problem since there will be only one central node that holds the responsibility of storing and uploading data of all the nodes on the bridge. The present architecture scales well to any length of bridge, since a head node will just have to manage its own cluster of six nodes (or twelve at maximum).

Coming to the takeaways for us from the past work, we are using the same platform (tmote) and other hardware for the development. The accelerometer (ADXL 203) used is the same. The present design also makes use of the same power-switching circuitry as that in the earlier design which is required to manage the power supply to the sensor during sleep-wakeup.

3.2 SHM Applications using Wireless Sensor Networks

A number of architectures have been provided in literature for SHM using WSNs. In [30] Xu et al have proposed an architecture called WISDEN data acquisition system to stream MEMS based accelerometer data. The system has a reliable data transport protocol using end-to-end and hop-by-hop recovery. A separate 16-bit ADC resolution, accelerometer data collection card is used with an on-board microprocessor. The maximum sampling rate that they have used was 160Hz. The nodes do not perform duty cycling and continuously monitor and stream data. They also study the use of wavelet based data compression techniques in this setting.

The same group has designed the NetSHM [27] architecture which is a programmable and re-usable architecture which can evolve with the change in sensor network technology. The system is designed such that the application level programing of SHM related algorithms can be done in C/MATLAB transparent to the intricacies of underlying sensor network. The requirements of these applications is broken down into tasks for the sensor nodes. Thus an interface layer can be designed such that changes in the underlying sensor network techniques, algorithms and hardware or application layer programs do not effect each other. The authors also implement a damage detection and localization method to test the applicability of the system.

Another recent implementation of WSN based bridge monitoring is in [31]. They have deployed a network consisting of 64 nodes, spread on a 4200ft span of Golden gate bridge in San Francisco and monitored the ambient vibrations. They collect and store the data acquired by the nodes into a base station located at one end of the bridge. They have used the existing protocols for routing (MintRoute [1]) and time synchronization (FTSP [10]). A few other projects [28, 29] have also addressed the use of WSN in MEMS accelerometer based SHM studies but they stress more on the mote building aspect rather than the complete structure of the application.

The significant features of our work that differ from the earlier implementations, (particularly [30] and [31]) are the following:

- The data collection in BriMon is event based: our nodes collect data only when there is a passing train. We have designed a novel event detection technique for this purpose. Rest of the existing works do the data collection continuously. The other applications collect data continuously [30, 31].
- The means by which we transfer the data collected at a remote bridge to a repository is novel: we use the moving trains to carry the data.
- We custom design all the protocols required viz. routing, time synchronization, transport and interface with sleep-wakeup with low duty cycling. The existing implementations use the protocols existing in literature like MintRoute [1], FTSP [10] etc. Also, there is no sleep-wakeup in WISDEN [30] and [31].

3.3 Existing WSN Protocols and (ir)relevance to BriMon

3.3.1 Routing Protocols

There are many routing protocols available in sensor network literature. The ad hoc routing protocols like [1, 5] run in background along with the application, like in traditional networks. They continuously measure a routing metric and monitor its changes and accordingly change the route information. Moreover, a point to note is that these protocols are more generic, in the sense that they are designed keeping in view that the flow of traffic can be there from any node to any node in the entire network. However, our application needs the traffic only towards a single destination, which is the root node.

SPIN (Sensor Protocol for Information via Negotiation) [4] is a flooding based family of protocols used to efficiently disseminate information in a wireless sensor network. This assumes that all the nodes in the network are always active, which is not possible in our setting. Also, these family of protocols are not energy aware.

LEACH (Low Energy Adaptive Clustering Hierarchy) [3] is designed for sensor networks where an end-user wants to remotely monitor the environment. In such a situation, the data from the individual nodes must be sent to a central base station, often located far from the sensor network, through which the end-user can access the data. LEACH includes distributed cluster formation, local processing to reduce global communication, and randomized rotation of the cluster-heads. The disadvantages of LEACH are that cluster head selection is a difficult problem to optimize. However, in our setting this dynamic clustering is out of question since we always want the nodes on a particular span to be a cluster, since it is their data which is correlated.

Also, none of these protocols have been implemented and evaluated in context of any specific application. [1] presents implementation of MintRoute protocol and evaluates (though not specifically for an application) with a setup of 50 nodes. This has been implemented in TinyOS as MultiHopLQI [41]. This argues that expected number of retransmissions can be a useful cost metric for making routing decisions. However, this protocol needs to be run continuously in background during any message transfer so as to dynamically estimate the routing metric. Thus, it will not be possible to have an explicit phase of routing that *ends*, as is desired in our application.

Considering the arguments (of using RSSI as a metric to decide the route) in [7, 6] we designed a light weight routing protocol that would cater to the needs of the other components as were explained in Section 2.3. This routing protocol will have an explicit *end* state, unlike other protocols in literature.

3.3.2 Time Synchronization Protocols

There are primarily two classes of techniques by which a message exchange can achieve time synchronization between two nodes.

- Receiver-receiver based: When a sender broadcasts a message, all the receivers hearing to that will get synchronized to the clock of the sender, since all of them receive the message at the same instant.

- Sender-receiver based: When a sender sends a message *containing the current time stamp* at which the message is being sent, the receiver on hearing it can find the current time at the same instant, thus being able to find its offset from the sender.

Reference Broadcast Synchronization (RBS) [9] is a protocol based on the receiver-receiver based technique of achieving synchronization through message exchange. For a multi-hop network, it achieves synchronization by grouping the nodes into clusters and common nodes that fall into several clusters will be used for synchronization across the clusters. The existing implementation on Mica platform, with the clock of 4 MHz (corresponding to a clock tick duration of 0.25μseconds) gives an average error of about 30μseconds.

Time-sync Protocol for Sensor Networks (TPSN) is one of the latter class of protocols, which is implemented for Mica motes, running at their maximum clock rate of 4 MHz. For more accuracy, it makes use of mutual message exchange between the two nodes under the process of synchronization, and finds the average of the offset obtained in the two message cases. It reports synchronization of nodes to within 17μ seconds, giving a two-fold better performance than RBS. For multi-hop scenario, the nodes form tiers and a spanning tree of nodes is formed, and the two nodes on every edge of this tree are synchronized in the manner similar to a two node case. It makes use of randomized backoffs at the nodes in a same tier to decide the order in which they participate in synchronization. This increases the duration required for synchronization. Also, the first sender of the time sync pulse will send it again after a time-out, if it does not get a time sync pulse reply. However, this is not required in our application, since we do not want the synchronization process to take long time and it is acceptable if a node gets out of sync once in a while.

FTSP [10] performs better than above protocols by combining the good features of both the protocols. It uses broadcast nature of the medium to disseminate time stamps, sender-receiver type model for updates and regression based clock skew estimation. It also performs MAC level time stamping and betters it by averaging the time stamps put in the messages to remove interrupt handling related jitter errors. FTSP seeks to estimate the clock drift through linear regression, and synchronize clocks to granularity of one micro-second. In FTSP, the propagation of synchronization messages over a multi-hop network is in an ad hoc manner. Due to this, it necessarily takes a long time for synchronization (of the order of a few minutes) [10]. Furthermore, it is not clear about how FTSP could be adapted to work in a periodic sleep/wake-up setting such as ours. Also, the existing implementation of FTSP is specific to Mica2 platform, which uses CC1000 radio.

Our approach to time synchronization is simple and efficient when compared to the protocols in literature. We use a TDMA based flooding as opposed to random backoff used by TPSN and ad hoc scheme used by FTSP for propagation of time synchronization messages over a multi-hop network. Also, in comparison with TPSN, our protocol does not involve a message exchange, but only a single message transfer from one node to another (or a few other) node(s) is required. One common property that our protocol shares with TPSN is that the nodes need to have some information about network topology.

CHAPTER 4

Routing Protocol

This chapter discusses the Routing Protocol used to form the network of nodes within a *cluster* of BriMon. The chapter starts with explaining the measurements carried on link range and stability. The results from these experiments would be useful not only for BriMon, but also for any other wireless sensor network application. Then we explain the purpose of routing protocol in BriMon. Then comes the discussion on the basis of the protocol. This constitutes of the implications of the experimental studies on link ranges and variations in Received Signal Strength Indicator (RSSI) and packet error rates, the details of which were explained in Section 4.1 and also in [6]. Then we discuss the algorithm for topology formation named C2P Routing (for *Centralized Two Phase Routing*), followed by the implementation details.

4.1 Link range and (in)stability measurements

In these experiments, we sought to measure the link ranges achievable and to estimate the quality/stability of the links in different environments. In all the experiments described, we used Moteiv Tmote Sky motes [21] for our experiments. These motes come with a Chipcon CC2420 radio chip that is compliant with the 802.15.4 standard. To connect the external antennas to the motes (wherever applicable), we soldered an SMA (Sub-Miniature ver-A) connector while also disconnecting the internal antenna. Note that the Tmote Sky comes with a 3.1 dBi internal antenna [21].

Range Measurements :

For this, we just had to put two motes to use in all the individual experiment runs. One mote designated as transmitter, was programmed to continuously transmit a configurable number of packets with 20ms inter-packet pause. All packets were broadcast by the transmitter. The transmit power was fixed at 0 dBm, the maximum allowed by the CC2420 radio. Each packet contained a 12 Byte MAC header and 14 Bytes of data that included a sequence number to help calculate packet loss rate. The other mote designated as receiver, was programmed with TOSBase, which forwarded the captured packets to a laptop along with the individual

RSSI and LQI values of each packet. The receiver was mounted along with its antenna on a tripod at a height of 1.5m to 1.7m. A schematic showing the setup is shown in Figure 4.1.

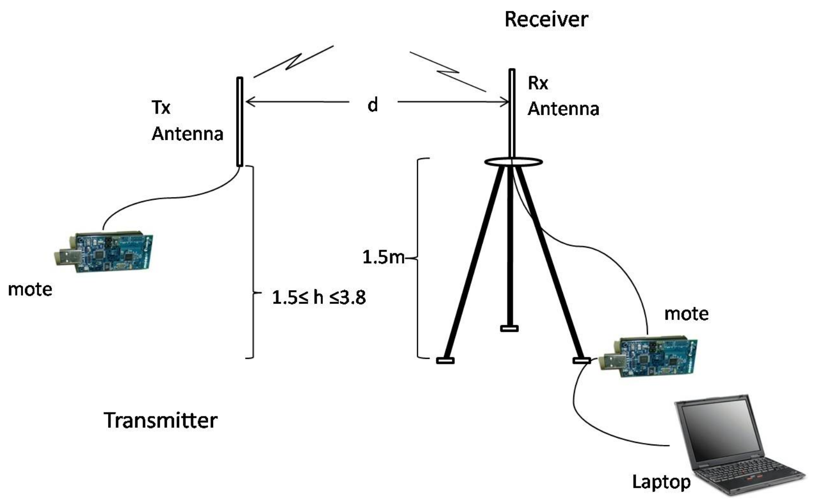


Figure 4.1 Setup for Link range and stability measurements

These experiments are done in two environments:

(1) *Dense Foliage environment*: In this environment, we experimented with just the internal antenna at the receiver, though the transmitters antenna was varied between internal, omni (8dBi), sector (17dBi) and grid (24dBi). The transmitter antenna was fixed at a distance of 1.5m above the ground. For each choice of the transmitter antenna, we took readings at several receiver locations. We stopped at a distance where the received signal strength fell to about -85dBm or worse. The experiments in this setup used 6000 transmitted packets: about 2 minutes, with one packet sent roughly every 20 ms.

In Table 4.1, we show the packet error rate and the RSSI for various configurations at the approximate range achieved with the set of antennas in use. The average and the standard deviation of these metrics are calculated as follows. We first partition the transmitted 6000 packet sequence into bins of 100 packets each. We compute the error-rate and average RSSI within each bin. We then compute the average as well as the standard deviation of the resulting 60 values. We ignore bins which do not have any received packets, in which case, the computation of the average RSSI/LQI within the bin would not make sense.

Given the high gain of the sector and parabolic grid antennas (17 dBi and 24 dBi), we had expected better ranges than 60m or 90m respectively. However, it is worth noting that in our environment the foliage was so dense that beyond about 35m, the people standing

Tx Antenna - Dist	Avg Pkt Error % (Std. Dev)	Avg RSSI dBm (Std. Dev)
Internal - 35m	0.3 (1.25)	-78.79 (3.43)
Omni - 40m	0 (0)	-79.42 (2.35)
Sector - 60m	0.53 (2.6)	-80.77 (3.55)
Grid - 90m	1.6 (4.08)	-85.05 (4.19)

Table 4.1 Range measurement results in dense foliage environment

near the transmitter and the receiver could not see each other.

(2) *Narrow road environment*: Similar to the foliage environment, we conducted various range measurements on a narrow road. In this environment, we experimented with both internal and omni-directional antenna at the receiver. The transmitting antenna was placed at a height of 1.5m when using internal and omni. However we had to increase the height to 3.8m for the sector and grid antenna as the ground reflections at the lower height of 1.5m introduced losses in our measurements even at short distances.

Internal Antenna at receiver		
Tx Antenna - Dist	Avg Pkt Error % (Std. Dev)	Avg RSSI dBm (Std. Dev)
Internal - 75m	1.37 (4.34)	-83.74 (3.61)
Omni - 75m	0 (0)	-80.64 (2.47)
Sector - 210m	0 (0)	-81.92 (0.49)
Grid - 500m	0 (0)	-85.67 (0.94)
8 dBi Omni Antenna at receiver		
Tx Antenna - Dist	Avg Pkt Error % (Std. Dev)	Avg RSSI dBm (Std. Dev)
Omni - 90m	0.04 (0.33)	-80.92 (0.88)
Sector - 500m	0.13 (0.68)	-82.16 (0.37)
Grid - 800m	0.13 (0.39)	-85.76 (0.31)

Table 4.2 Range measurement results in narrow road environment

Table 4.2 shows our various range measurements in this environment. We note that the line-of-sight range measurements are much higher than in the case of our foliage measurements. This is true even in the case where we only used the internal antenna at the receiver. There was an increase of about 410 m in range, going from a foliage environment to direct line-of-sight environment, when using a grid antenna at the transmitter and internal antenna at the receiver. In the line-of-sight environment, replacing the internal 3.1dBi antenna to an 8dBi omni at the receiver, increased the range further by 300m.

Signal strength and Error rate variability :

We have conducted another series of experiments in different environments, with the same setup as that in Figure 4.1 in order to observe the behaviour of links over time. The environments we have considered are air-strip, road, foliage, corridor of a student hostel and structures lab.

With the same setup as described, the data analyzed this time is the RSSI and the packet error rate. The division of the packets into bins and further analysis is similar to that of the link range experiments. Since RSSI can be observed only for the packets successfully received, to capture the full extent of RSSI variability, we restrict ourselves to data points where we observed an overall error rate of less than 0.1%. The results that we got in all these environments are having almost the same patterns. So, we show here a representative set of readings that we got in case of narrow road environment.

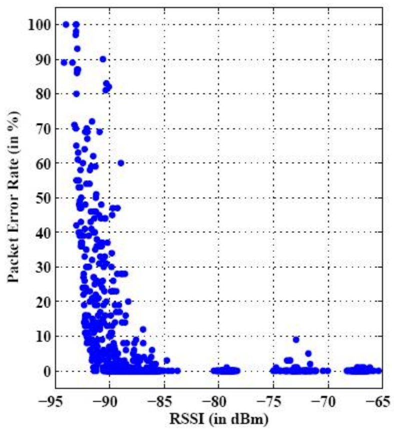


Figure 4.2 Correlation between RSSI and Error rate(road)

Figure 4.2 shows the plot between packet error rate and the RSSI, which implies that there is a very sharp region of RSSIs where the error rate varies a lot. Thus we can observe that there is a good correlation between RSSI and the packet error rate.

Figure 4.3 shows the variation of RSSI with time. We can observe that the variation is too much even in time scale of 2 seconds (which was the duration of each run of the

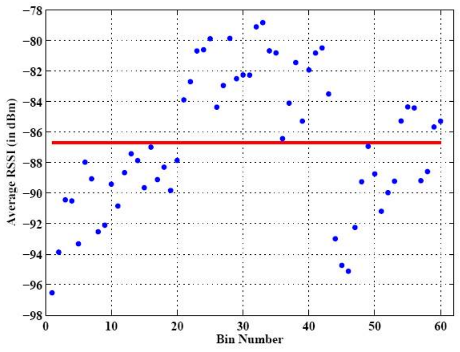


Figure 4.3 Temporal variability of RSSI

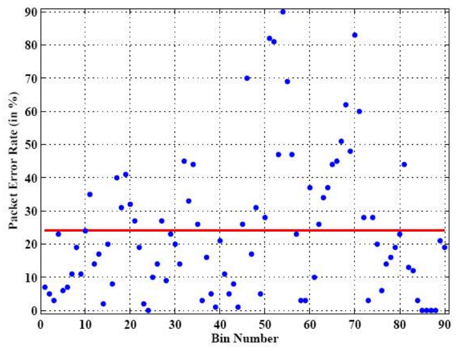


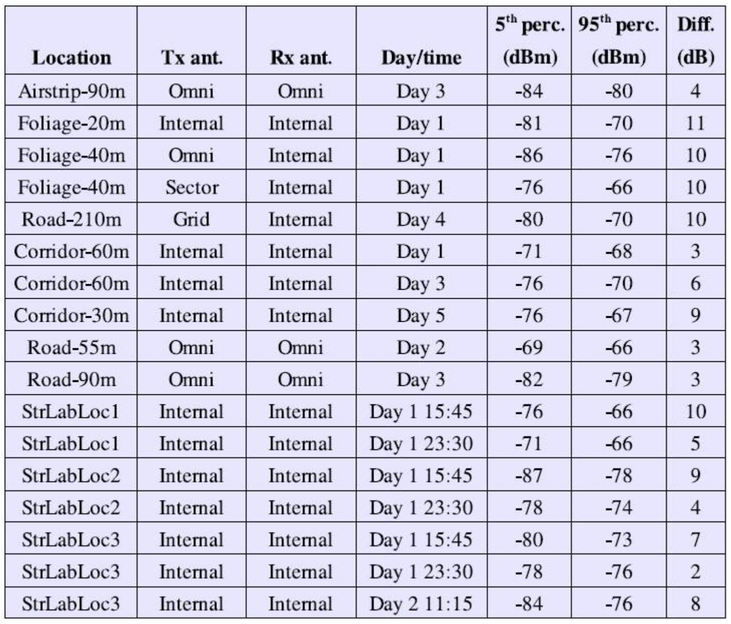
Figure 4.4 Temporal variability of error rate at average RSSI of -87dBm

experiment). Having seen the correlation between RSSI and error rate, we expect a similar temporal variation in packet error rate too and this is indeed true, as shown by the plot of Error rate Vs. time in Figure 4.4. Note that this variability in RSSI is observed only when the average RSSI (-87dBm in case of Figure 4.4) falls into the steep region in Figure 4.2.

The table shown in Figure 4.5 captures the RSSI variability in terms of the difference between the 95th percentile and the 5th percentile in the range of RSSI values. This metric, instead of the standard deviation, removes the effect of outliers. In the table, the different distances indicated in the first column are the distances between the transmitter and the receiver.

4.2 Purpose of Routing Protocol and Challenges

Referring to the the clustered architecture in BriMon presented in Section 2.3, the nodes within a *cluster* have to be connected with the cluster head, so as to enable the head node perform the cluster management tasks. These include *gathering* the data collected by the nodes, *time synchronizing* the nodes in the cluster etc. The link range results in Section 4.1



Location	Tx ant.	Rx ant.	Day/time	5th perc. (dBm)	95th perc. (dBm)	Diff. (dB)
Airstrip-90m	Omni	Omni	Day 3	-84	-80	4
Foliage-20m	Internal	Internal	Day 1	-81	-70	11
Foliage-40m	Omni	Internal	Day 1	-86	-76	10
Foliage-40m	Sector	Internal	Day 1	-76	-66	10
Road-210m	Grid	Internal	Day 4	-80	-70	10
Corridor-60m	Internal	Internal	Day 1	-71	-68	3
Corridor-60m	Internal	Internal	Day 3	-76	-70	6
Corridor-30m	Internal	Internal	Day 5	-76	-67	9
Road-55m	Omni	Omni	Day 2	-69	-66	3
Road-90m	Omni	Omni	Day 3	-82	-79	3
StrLabLoc1	Internal	Internal	Day 1 15:45	-76	-66	10
StrLabLoc1	Internal	Internal	Day 1 23:30	-71	-66	5
StrLabLoc2	Internal	Internal	Day 1 15:45	-87	-78	9
StrLabLoc2	Internal	Internal	Day 1 23:30	-78	-74	4
StrLabLoc3	Internal	Internal	Day 1 15:45	-80	-73	7
StrLabLoc3	Internal	Internal	Day 1 23:30	-78	-76	2
StrLabLoc3	Internal	Internal	Day 2 11:15	-84	-76	8

Figure 4.5 RSSI variation (95th - 5th percentile)

show that individual direct links can almost always be established between the cluster head and remaining nodes, like in a star topology. So, a question of what is the purpose of routing may arise.

Once again, refering to the Figure 4.4 in Section 4.1, we find that the there is a lot of scope for temporal variation of packet error rates, while operating on weak links. If we operate a star topology on long spans of the order of 100m, the links between root node and nodes at the other end of the span would certainly be so weak that they can face temporal instabilities (the degree of weakness being decided by the environment) [6]. This will be discussed in more detail in Section 4.3.

A better solution is thus to have a simple routing algorithm that is based on the use of strong and stable links, even if it may lead to a multihop network topology. Another resultant problem that has to be addressed by the algorithm is to keep the active nodes in a cluster connected, even during the failure of intermediate nodes in their path towards the cluster head.

Further, there are more challenges that the Routing Protocol in BriMon need to address:

- The application requires the duty cycle to be low. Due to this, the routing protocol cannot be a continuous process like in [1] & [5], but it has to run only for a finite duration.

- The routing module at each node must be able to provide the children information of the node, apart from its path to the cluster head. This information will be used by the Time Synchronization protocol as will be explained in Section 5.2 and also by the command system as explained in Section 2.3.

4.3 Link Stability and Routing

4.3.1 Link stability studies

Let us now look into the results from our experiments on link stability, given in Section 4.1. It can be seen from Figure 4.3 that in a given environment, there could be large RSSI variability. This means that in the course of a run, we could have cases where the RSSI falls in the steep region of Figure 4.2, although the average RSSI may well be above the sensitivity limit (-95dBm) of the CC2420 radio [19] used. Now, put together the observation that there is a good correlation between RSSI and the packet error rate, there will an variability in the observed error rate.

We thus have the implication that we would observe variability in the error rate too (at the same time scales as variability in RSSI). This is illustrated by the plot in Figure 4.4. In case the overlap of the RSSIs with the steep region is higher, the variability is even higher. For instance, we can see that the error-rate variation is all the way from about 0% to about 90% in Figure 4.4, where the average RSSI was -87dBm, falling into the steep region of Figure 4.2. In fact, throughout all our experiments in these studies, we observed high variability in error rate whenever the error rate was significant (say, over 5%). This observation essentially implies that a routing metric such as 1/PSR (PSR is the Packet Success Rate) [1] would not really work. That is, by the time we end up measuring the packet error rate meaningfully, it would have already changed.

These measurements of RSSI and error-rate variability have far reaching implications with respect to routing protocol design. The variability means that we cannot reliably use a mechanism whereby we measure the RSSI (or the packet error rate) at one wake-up time, and use it for routing during the next wake-up time. The measurement done earlier would likely not be valid by the time of the next wake-up period.

But then, an important point to note here is that the instability in error rate is only when the RSSI variability window overlaps with the steep region of Figure 4.2 i.e., when the link sometimes operates close to the receive sensitivity, and the overall error rate is significant (over 5%). The immediate next question then is whether it is possible to say when a link has a stable error-rate. The answer to this is suggested by the data in Figure 4.5. We could take into account the RSSI variability to provide sufficient link margin to operate away from the steep region. We could make a worst-case assumption of say 11 dBm (as suggested by the data in the Figure 4.5) for the RSSI variability. If the average RSSI measured at given time does not come closer with the steep region by this margin of 11dBm, we could say for sure that the link would have stable and low error rate. With such a high margin, we might be erring on the side of safety while determining a link to be stable i.e., we could potentially miss out on some stable links, but would not classify unstable links as being stable. Thus, we are in a position to distinguish between good and bad links.

4.3.2 Applicability to Routing in BriMon

We now consider the usability of above implications for the routing protocol. The basis of C2P Routing protocol is the use of *stable* (good) links. The methodology of distinguishing between good/bad links is based on a threshold RSSI as described. Thus the variability in link behaviour would be considerably less. Also, since the RSSI for these links is high, the packet error rate is inherently low.

Since we operate six nodes in a span of bridge (which in general is less than 120m) with the locations as mentioned in Figure 4.6, the maximum distance between adjacent nodes can be 30m. Given this, the link range and RSSI results in Section 4.1 Table 4.2 shows that a (multihop) network topology that contains (only) stable links can be most likely formed.

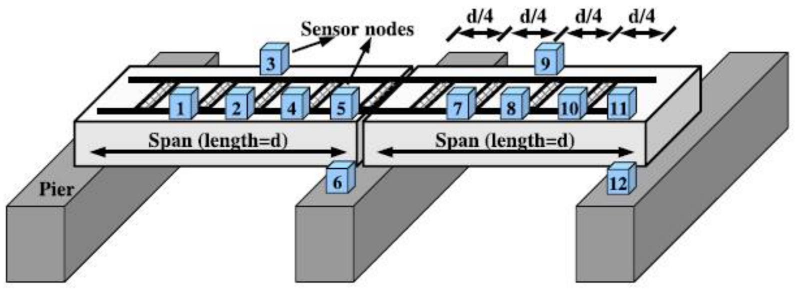


Figure 4.6 Nodes on adjacent spans of a multi-span long bridge

Also, since we are using only stable links a topology formed at some wakeup duration can be used for communication during many subsequent wakeup cycles, thereby avoiding the necessity of routing protocol running continuosly in background (like in [1] & [5]) which is what is required for BriMon as the section 4.2 points out.

We now look into the design of the routing protocol.

4.4 Protocol Design: C2P Routing

An important design decision made here is to go for a centralized routing. It is the head node of the cluster that completely manages the routing protocol. It will always be the root of the routing tree. It assigns parents to the rest of the nodes in its cluster as explained below.

The reason for choosing a centralized approach is simplicity. Since it is only six nodes (or twelve in case of double span) that constitute a cluster, it would be possible for a central node to manage the routing process, given its simplicity, while spending very less resources in terms of computation and memory. Also, as will be explained later in Section 5.3, the time taken for synchronization denoted by t_{ps} is proportional to the number of non-leaf nodes in the network. So we need to minimize the number of nonleaf nodes. A centralized routing approach can optimize this much more easily as compared to a distributed approach. Similarly, any load balancing based on the available power at the nodes is also more easily done in a centralized routing approach.

We now turn into the description of the protocol.

As the name indicates, this algorithm consists of two phases: *Neighbourhood Discovery* & *Link State Query*.

Neighbourhood Discovery :

This phase is initiated by the cluster head, during which each node sends a finite number of beacons that contain the respective node address. Also, on reception of such beacon from other nodes, the average of the RSSI at which the beacons are received from the sender of that beacon is calculated. This phase ends when all the nodes are done sending their finite number of beacons.

At the end of this phase each node will have the information of who all its neighbours are and the respective averages of RSSI. This information will be called as *Link State* in the following description.

Link State Query :

The central idea of the protocol is to classify links as good/bad depending on a threshold of RSSI (R_{th}). All the nodes whose *link states* indicate an average RSSI of above R_{th}, will be called good neighbours. After the neighbourhood discovery, the head node proceeds with the formation of the tree. It starts by deciding who all can be its children. It essentially chooses all of its good neighbours as children. It then queries and obtains the link states of the children nodes and updates the tree, once again with the good neighbours. This procedure continues until all nodes with good neighbours are connected.

Thus, the root node initially tries to make a path to all nodes in the cluster through stable (good) links, even if it may be multiple hops away. After this, if some nodes are not yet connected, the root node assigns parents to such nodes based on the criteria of minimum hops, since anyway they are bad neighbours to all the other nodes.

A flow chart showing the algorithm run by the cluster head is shown in Figure 4.7.

Handling Node Failures :

Following through the algorithm in Figure 4.7, we see the case where the *tree is incomplete* (meaning that some of the nodes have not yet been assigned parent) as well as *no other nodes remained to query.* This joint case occurs when a node fails, and subsequently have not been heard by any node during the neighbourhood discovery phase. At this point, the root node finishes the routing algorithm by dispatching the routing tree formed. The tree will be flooded to all the nodes in the network.

When to run the Routing Protocol :

The routing protocol can be run infrequently, or whenever a failure is detected. This is because the links in the network are often stable, as per the protocol explained. A node can detect itself to be disconnected from the current routing tree, if it fails to receive any messages for a certain timeout. It can then cease its cycle of sleep-wakeup and join the network during the next run of the routing protocol, since it is not accommodated in the present routing tree. A better approach would be to announce immediately that it has been

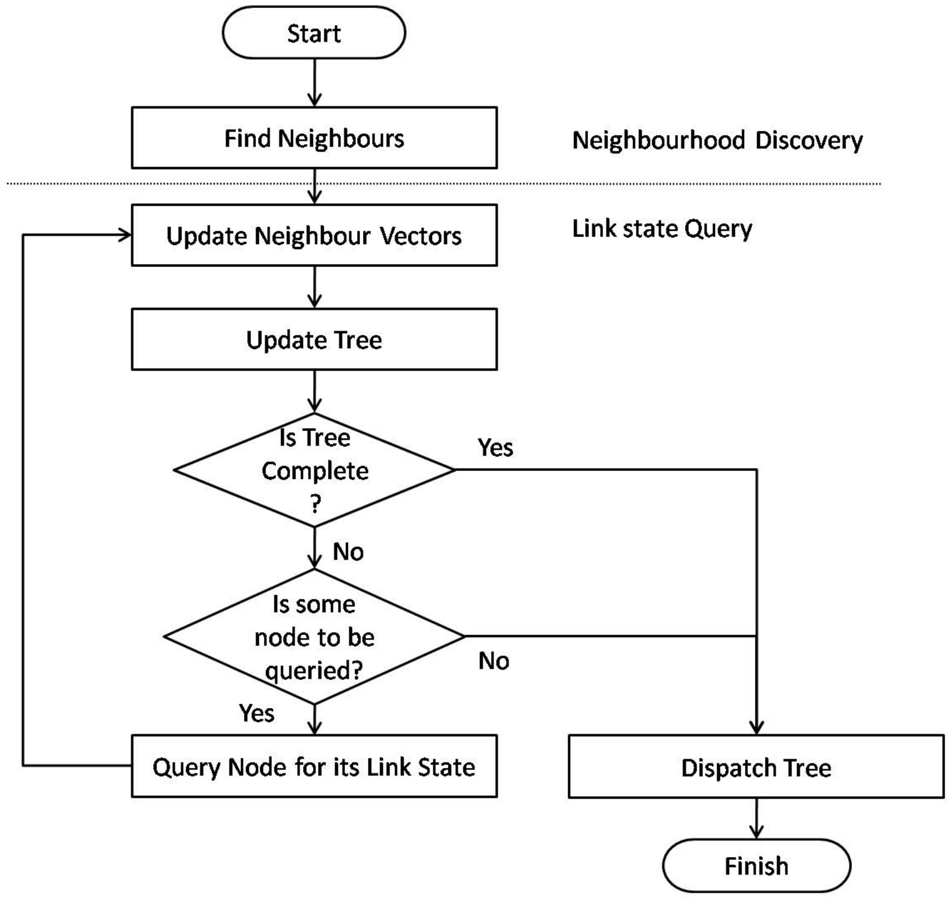


Figure 4.7 C2P Routing Algorithm run by the *Cluster Head*

orphaned. This announcement, if heard by a node connected to the routing tree, is passed on to the head node. The head node can then initiate the above routing protocol again. The latter is just a suggestion, while the former is the implemented version. Such a laid-back approach to fixing failures is possible because we are relying upon stable links.

4.5 Implementation Details

This section discusses various message types and the values of parameters used in the implementation of C2P Routing. For clarity, the description proceeds in the order of actual stages of the protocol. The discussion below will also go through the stages of routing in an example of six nodes cluster shown in Figure 4.8(a).

Neighbourhood Discovery :

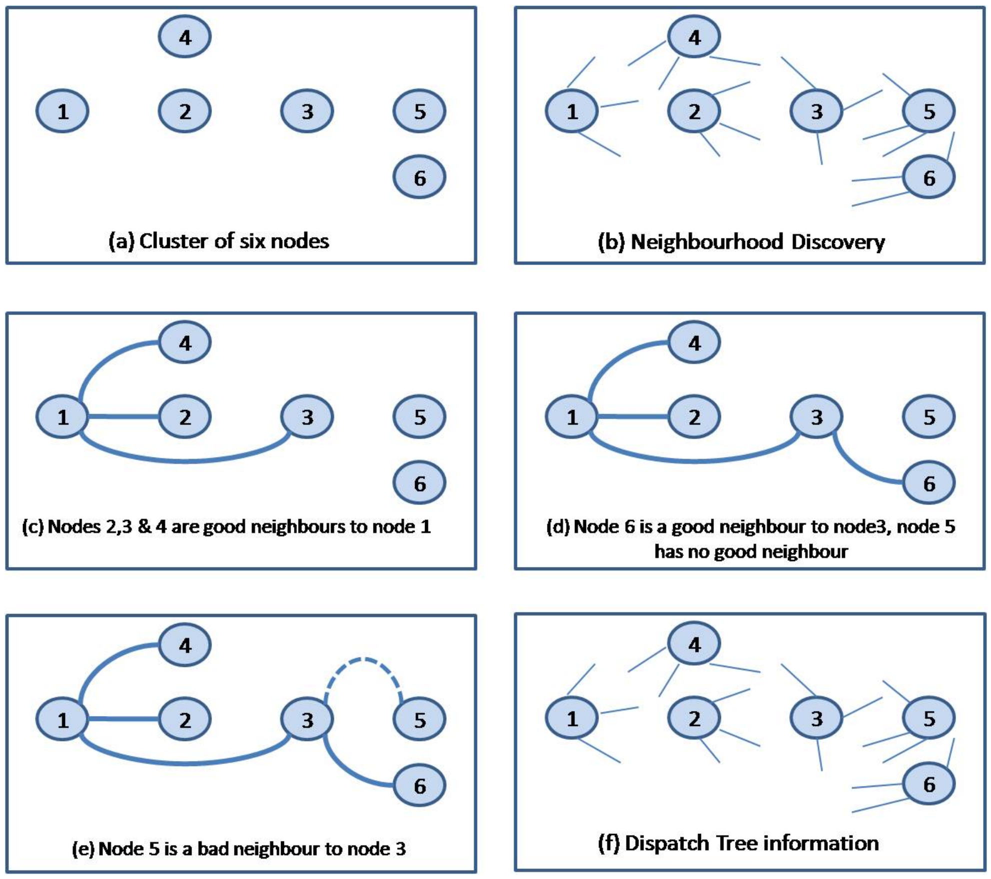

4
1
2
3
5
6
(a) Cluster of six nodes
(b) Neighbourhood Discovery
(c) Nodes 2,3 & 4 are good neighbours to node 1
(d) Node 6 is a good neighbour to node3, node 5 has no good neighbour
(e) Node 5 is a bad neighbour to node 3
(f) Dispatch Tree information

Figure 4.8 Stages in routing - an example

The beacons used for neighbourhood discovery, here after called 'Hello' messages consists of the source address, which can be from 1 to 6. It also contains the sequence number of that Hello message from its source.

The phase is initiated when the root node starts sending the Hello messages. Any node that hears to these will start sending its own Hello messages. This flooding *type* of process further continues till all the nodes in the network are done with sending the designated number of messages. We have chosen it to be a value near to 10 in the current implementation, since if it is very low, the nodes may not be able to hear to each other due to losses. Also, it cannot be much high because it will increase the duration of routing and may not be of much added advantage. Figure 4.8(b) shows this stage.

Since we are using flooding for sending Hello messages, there may be contention among nodes while sending. The underlying MAC layer (CSMA/CA) will have the provision of randomizing the instants at which the nodes send so as to avoid contention among the nodes.

But even then, our observations reveal that the MAC layer does not have flow control, due to which, if receiver is overwhelmed with packets, it simply drops them. More details of this is given in Chapter 5 on time synchronization. To overcome this problem in a simple way, a random inter-message pause ranging from 0ms to 10ms is also incorporated at the routing layer during this phase. Note that this timing is of higher granularity than the back-off periods (fraction of ms) of MAC layer.

On receiving a beacon, nodes calculate the current average RSSI of the beacons received from that source and form a link state table that looks like Table 4.3.

Node Address	RSSI (dBm)
1	..
2	..
..	..

Table 4.3 Link State table formed by each node during neighbourhood discovery phase

Before proceeding to the next phase, the root node must make it sure that this phase has ended. Since there is randomness involved in the timing of the messages, we do the following estimate for the total time of the phase. We assume that a hello message reaches the farthest (in terms of hops) node out of the remaining 5 nodes from the root node in a maximum of 3 hops. This assumption is based on the location of the nodes on a span as given in Figure 4.6. With this in mind we do the worst case estimate of the duration of the phase. In the worst case, the nodes at each of the 4 stages of the 3 hops transmit one after another, with maximum inter-message pause between the nodes. This would take $4 * 10 * 10 = 400ms$ (10 packets with inter-packet pause of 10ms). Since, there would be a transmission time involved per each packet, which is of the order of 0.5ms*, we add a further 50ms interval. Thus, the root node proceeds to the next phase only after this much amount of time after initiating neighbourhood discovery.

Table 4.5 shows the details of the message structures used in this phase.

Link State Query :

*Hello message to be sent on air consists of 2 Byte source ID + 2 Byte sequence number + 12 Byte MAC header = 16 Bytes. For CC2420 at 250Kbps, this would take $16 * 8 \div 250K = 0.5ms$

Now we shall look into the details of the Link State Query phase. By the starting of this stage, all the nodes in the network would have formed the Link state Table 4.3. This phase is under the sole control of the root node. It maintains the information given in Table 4.4, which it updates while going through the phase.

Node Address	Parent	Good neighbour vector	Bad neighbour vector
1	0	0—1—1—1—0—0	0—0—0—0—0—1
2	1	1—0—1—1—0—0	0—0—0—0—0—1
3	1	1—1—0—0—0—1	0—0—0—1—1—0
4	1	1—1—0—0—0—0	0—0—1—0—0—1
5	3	0—0—0—0—0—0	0—0—1—0—0—1
6	3	0—0—1—0—0—0	1—1—0—1—1—0

Table 4.4 Information maintained by root node during C2P Routing - Entries specific to example in Fig 4.8

We make use of bit vectors to maintain the neighbourhood information. The root node starts with setting the appropriate bits of its own neighbourhood bit vectors using its link state table. For this, it will have to decide whether a node is a good or bad neighbour to it, based on a threshold RSSI R_{th} (with a typical value of around -75dBm) and then set the appropriate bit of good/bad neighbour vector. It may occur to us that there is no need to maintain a bad neighbour vector separately, since we are anyway setting the good neighbour vector bit only if current RSSI $> R_{th}$. But there may be cases where a node may not be heard at all during the neighbourhood discovery, meaning that it is not a neighbour at all. So, it is required to maintain a separate bad neighbour vector to distinguish a bad neighbour from a non-neighbour node.

After updating the its own neighbour vectors, the root node uses the good neighbour vector to update the routing tree, which is represented as a list of node addresses against their parent node addresses. Thus the root node, as explained in Section 4.4, proceeds forming the tree with good neighbours. At the same time, it forms a Breadth First List of the nodes getting added to the tree, which will infact be the order in which the nodes will be queried as explained later.

In the example, the nodes 2,3 and 4 happened to be good neighbours to node 1, as can be seen from the Table 4.4. Thus, node 1 chooses nodes 2,3 and 4 as its children and the resultant state of the network is shown in Figure 4.8(c).

At this stage, the root node has chosen who all its children are. Now, it checks if the tree is complete, essentially by looking into the Parent column of Table 4.4. If it finds that some nodes are not yet connected, it will *query* the next node in the list of nodes to be queried, by sending a *LinkStateQueryMessage* that has the format shown in Table 4.5. When the destination receives this query message, it will reply by sending its *LinkStateMessage*.

In our example, node 3 reports on query that node 6 is its good neighbour. Thus, node 3 is assigned as parent to node 6. The resultant network is as shown in Figure 4.8(d).

Once again, seeing the new link state obtained, the process of updating the neighbour vectors, and subsequent tree updation will be performed by the root node. During this course, if the LinkStateMessage does not arrive with in some duration, the root node will query the node again. The duration for which it has to wait is dependent on the number hops that this query message has to go (which is same as the number of hops over which the reply comes) and the message transfer time at each hop. In our current implementation, the time out is $(2 * Numberofhops * 10ms)$. Thus we are giving 10ms time for the message transfer at each hop, even though it can be reduced. There is also a maximum limit to the number of times the query is retried, which is currently 20 (such a high value since it is the common limit that we use for querying bad nodes also).

When the root node is done forming the tree with nodes having good neighbours, it then proceeds to add bad neighbours to the tree with minimum hop criteria. For this, it just needs to look into the bad neighbour vectors of the nodes as per the Breadth First List. A bad node may also need to be queried, if the tree formed is still incomplete. The procedure is repeated until the tree formed is complete.

In our example, node 1 could not find node 5 as good neighbour to any of the nodes queried (2,3,4,6 in order). Thus it starts looking at the bad neighbour vectors built, as shown in Table 4.4. It finds that node 3 is a bad neighbour to node 5, and also leads to shortest path towards node 5. Thus, node 3 will be assigned as parent to node 5 as shown in Figure 4.8(e).

Later, it dispatches the tree using *RouteMessage* structure given in Table 4.5. The remaining nodes on hearing to this, will again flood the same (with the random inter-packet interval, just like the Hello messages), for a fixed number of times and this phase ends when all the nodes are done. Thus, all the nodes will have a knowledge of the complete tree formed.

Message	Contents(Payload in Bytes)	Stage	Tx type
Hello	Source(2) & Seq No(2)	Neighb Disc	Broadcast
LinkStateQuery	HopsToGo(2) & Path(6)	Link State Query	Unicast
LinkState	Source(2) & avgRSSI($6 * 2$)	Link State Query	Unicast
Route	Source(2) & Parent($6 * 2$)	Routing Complete	Broadcast

Table 4.5 Summary of message types used in C2P Routing

CHAPTER 5

Time Synchronization Protocol

This chapter deals with the Time Synchronization protocol used for BriMon. The chapter starts by describing the purpose of time synchronism in our application. Then, having justified the unusability of the existing protocols in literature like FTSP [10] & RBS [9] in Section 3.3.2, we proceed to explain the light-weight time synchronization protocol that we have designed. Later the implementation details of the protocol are explained in detail.

5.1 Role of Time Synchronization in BriMon

The Time Synchronization protocol in BriMon has two purposes to serve:

- The application requires that the nodes on a span, i.e. a cluster of nodes need to collect the data in synchrony, so as to be able to correlate the samples collected across the nodes. The degree of synchrony, often specified as the maximum permissible error, depends on the frequency range of vibrations that are of interest. As explained in Section 2.1, for structures like bridges the maximum permissible synchronizaion error can be 5ms.

- As explained in the architecture of the application in Section 2.3, the nodes need to do sleep-wakeup in order to conserve energy. While doing so, the nodes in a cluster will have to follow the *commands* from the cluster head that specify the operations to be carried out by the nodes like *Data Collection*, *Data Gathering* apart from participating in *Routing* and *Time Synchronization* itself. For this to happen it will be required that the nodes in a cluster have to be AWAKE during the same period of time while going through their sleep-wakeup cycle. Time Synchronization is thus a necessity here.

  The minimum AWAKE duration of the nodes will be directly related to the synchronization error as per the discussion in Section 2.3 and shown in Figure 5.1. Thus the error has to be kept to as minimum as possible. (Note that the SLEEP duration is dependent more on the range of Event Detection, as opposed to the *clock drift* as explained in Section 2.3, which will affect the duty cycle).

The following sections describe the design and implementation details of the protocol, whereas the evaluation of the protocol will be presented in Chapter 6.

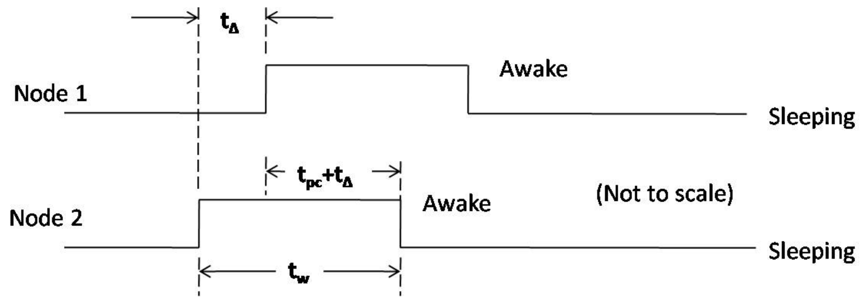


Figure 5.1 Sleep-Wakeup Cycle showing different components

5.2 Protocol Design

One of the main goals in designing the time synchronization protocol is to keep the time required for synchronization, denoted by t_{ps} as minimum as possible so as to keep the duty cycle to a minimum. The time synchronization protocol assumes the following:

- Each node knows who its parent is. And it knows its children set. This assumption is valid since, as explained in Chapter 4, each node will have information about the entire routing tree.
- The packet error rate on individual links would be very low. This assumption is valid in our application, since only strong links are formed most of the times as was described in detail in Chapter 4.

With these assumptions and also observing the fact that it just takes a single message transfer for any two given nodes to get synchronized, the protocol proceeds in the following steps:

1. The head node sends a few *Sync* messages to its children in a broadcast packet.
2. When a node receives a *Sync* message from its parent, if it has children, it generates its own Sync message and broadcasts it. This procedure will continue until the Sync messages reach the leaf node.

A subtle point here is that all the nodes in a cluster are made to synchronize to the clock of the cluster head, i.e. the *Global time* observed by the nodes is the *Local time* of the head node. Each Sync message contains the Global time (as assumed by the sender) at the instant when the message is just about to be sent on air. The receiver of the message can thus have a knowledge of the global time at the same instant. Message transmission time can be accounted for, as given in the next section on implementation details.

Two important design choices that we have made here are that:

1. There are no acknowledgements and further complexities like time-outs, which renders the process simple, thus keeping t_{ps} low.

2. The nodes follow a TDMA scheme while sending Sync messages. This is possible since each node has knowledge about the entire routing tree. The order in which the nodes transmit the sync messages can be a simple scheme like a breadth-first or depth-first, the only constraint being that a node will transmit only after its parent. The reason why we had to make this choice, rather than having the nodes just flooding randomly (which is more simpler) will be explained in the next section under the *flow control problem*.

Thus, at the end of execution of the protocol, all the nodes in a cluster will be synchronized to the clock of the head node, thus satisfying the purposes mentioned in the previous section. We now move on to the implementation details.

5.3 Implementation Details

The implementation details mentioned below are specific to the *tmote* platform, using a CC2420 radio chip. These should in general also be similar for any other 802.15.4 compliant platforms.

For simplicity, let us start by understanding how two nodes can be made to get synchronized by a single message transfer, before proceeding to the case of a cluster of nodes.

5.3.1 Time Synchronizing two nodes

Consider two nodes 'A' and 'B'. Each node will be having its own clock, running at 32khz*. The state of the clock i.e. the current time is specified as the number of *clock*

*All precisions presented are in *binary* units with respect to one second i.e. one second contains 1024 binary milliseconds. Thus, Time period of the 32khz clock = $1 \div 32768 \approx 30.5\mu$ seconds, often called a *clock tick* duration *ticks* elapsed after the node has been booted (which in TinyOS is represented by a 32 bit unsigned integer). Thus, depending on the instant when the nodes are booted, there will be a difference in their local times. This is often called *offset* and is represented as a difference between the local times of the two nodes at the same instant. Thus in order to synchronize to node A's clock, B has to know this offset. But how does the node B know about the instantaneous time at node A?

The answer is to have node A transfer a message containing its local time stamp and the other node on reception, will take its own local time stamp at the same instant and finds the difference so as to know the offset.

This all may seem simple, and indeed is, if we can tolerate high synchronization errors of the order of few tens of milli seconds if the time stamps are taken at the time sync layer. This is so because, there would be a non-zero message transmission time, meaning that the local time stamps taken at nodes A and B are not exactly at the same instant. The break up of the events that take place during a message transfer and their respective durations are provided in Figure 5.2 and Table 5.1.

However, the degree of error (of the order of tens of ms) obtained in the above manner will not be acceptable to the application itself, keeping aside the requirement from low duty cycle operation.

The solution for this problem will be to take the time stamp at the sender as late as possible (i.e. at lower layers than time sync) before sending the message over the air and taking the receiver time stamp as early as possible after the reception. The solution proposed in [12] for platforms using CC2420 radio is to use the Start of Frame Delimiter (SFD)* interrupt from the radio chip as a reference to take the time stamp, both at the sender and the receiver. The specifications of the CC2420 radio [19] say that the SFD interrupt at the receiver lags that at the sender only by few micro seconds. Thus, making use of this will significantly reduce the synchronization error, putting it down to the order of clock ticks.

Implementation in TinyOS:

What we need for high degree of synchronization is *capturing* of SFD interrupt, and *adding* a time stamp to the *same out going message* corresponding to that SFD interrupt. The set

*The PHY layer header in 802.15.4 consists of four Bytes of preamble used for synchronization purposes and one Byte of SFD to mark the start of a frame of components that does it for us are readily available in the Boomerang version of TinyOS from moteiv corporation [20], which is supposed to have the key features of TinyOS 2.x.

The Time Sync message must be having a field that carries the local time stamp of the sender. It is suggested by the documentaion of TinyOS that the location of this field in the message to be sent must be put towards the end of the payload of the message. This is because, the process of inserting the time stamp into the out-going message involves modifying the content of RAM of the CC2420 radio. This has to be done after the packet transmission over air has started. So, the more time we give for this writing, the more reliably it is written. Thus in our implementation, we use the last 4 Bytes of the message structure (which is of a total of 38 Bytes in case of 6 node network and 44 Bytes in case of a 12 node cluster) for this purpose. The exact components used for this purpose and relevant details are given in Appendix A. We now proceed to explain our method to synchronize a cluster of nodes.

5.3.2 Time Synchronizing a cluster

Let us consider a six node cluster for simplicity. All the nodes in the cluster will be made to synchronize with the clock of the cluster head, hereafter called *global clock*. As we have seen, it takes a single message exchange between two nodes for them to get synchronized. Thus, one may expect that synchronizing a cluster may be just done by flooding of Time Sync messages across the entire cluster, with each sender mentioning its local time stamp and its offset from the global clock in the time sync message, hearing to which a receiver can calculate the current global time.

The exact procedure how it proceeds might be that the head node starts sending the time sync messages, since it's is the global clock. Whatever nodes that listen to it will in turn send their time sync messages. The procedure should continue until all the nodes in the network *receive* atleast one time sync message. Since finding *when* this ending happens itself is not clear (similar to what we had in the Routing Protocol in Chapter 4), we might have to make a worst case estimate of the time it takes and would have to live with it.

It may seem that it would be enough if each node sends only a few time sync messages (to be sure that atleast one time sync message would reach its peers even if some of them are lost), thereby reducing the total time it takes even in the worst case. We did exactly the same, where in we kept the number of time sync messages from each node to a moderate value of 3.

In the above mechanism, there is a subtle implementation detail, that of flow control at the MAC Layer:

Flow Control Problem :

While experimenting with the above said time sync method of flooding with 3 messages each from all the senders, we anticipated that even if there would be contention among the senders, it would be taken care of by the underlying CSMA/CA in the MAC layer.

But even with this, we found that some nodes were not getting synchronized, even in a small network of six nodes, all placed in close vicinity to each other.

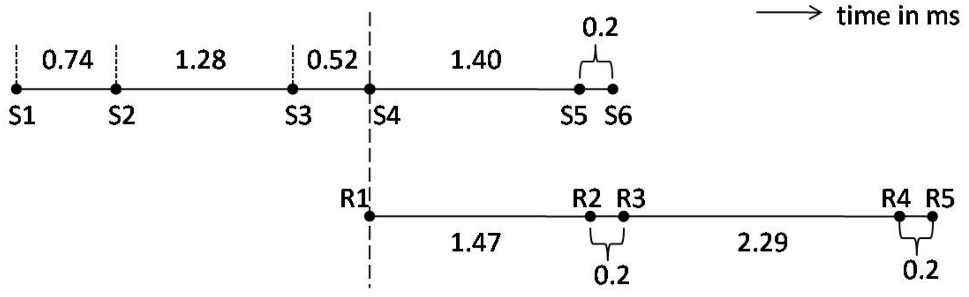


Figure 5.2 Timeline of events at sender and receiver during a message transfer (Packet size of 44 Bytes)

The experiments conducted by Nilesh* in order to find out the time interval between several events during send/receive of a packet over air, the results of which are presented in Figure 5.2 and Table 5.1, have shown that the problem is not with the senders but it is with the receivers. He explained from these observations that since the receiver radio has a limited buffer of 128 Bytes [19], if it receives several messages before they are transmitted over SPI bus to the RAM, it flushes off the entire buffer! This can happen in the method described above since the senders are just relying on CSMA/CA, which has backoffs only in the range of fraction of milliseconds (of the order of 15 clock ticks = 450 μs), where as the transfer time from radio to RAM over the SPI is a few milliseconds as shown in Figure 5.2.

Thus, we chose to have a simple but effective scheme of TDMA-based flooding.

*Nilesh Mishra, Research Associate, CSE & lab colleague. We report the results from these experiments here since there are no prior reports on the same

Sender Side	
Event	**Description**
S1	SendMsg command issued at time sync layer
S2	Start of data tx from Micro-controller to Radio over SPI bus
S3	End of Data Transfer over SPI bus
S4	SFD interrupt (sent the preamble 4 Bytes and SFD 1 Byte over air)
S5	Tx of packet over air done
S6	SendDone Event signalled to time sync layer
Receiver Side	
Event	**Description**
R1	SFD interrupt (Preamble and SFD received)
R2	Complete packet received by radio
R3	Start of data tx from radio to micro-controller over SPI bus
R4	End of data tx over SPI bus
R5	ReceiveMsg signalled to time sync layer

Table 5.1 Events during a message transfer over air

TDMA based Flooding :

If we can allot individual time slots to different senders, it will be a robust solution to the problem described above. Since there would be no contenders during a node's time slot, we further move a step ahead to disable the backoffs at the MAC layer during the execution of time synchronization protocol. Now the question arises, how to allocate the slots to the nodes? A related question is whether it is needed to allocate slots to all the nodes in the network.

The answers to these questions lie in the fact that each node will have information about the entire routing tree of the cluster ss explained in Chapter 4. Thus a simple scheme like Depth First would do, which is indeed used in our current implementation.

The process starts with the head node sending its time sync message, (which may or may not contain the slots information), that contains its local time stamp. All the children of head node, on hearing to this message, will know when the process has started by looking at the time stamp of the head node, called *base time* and then calculate when their respective slots start, if they have one. Only the non-leaf nodes in the network need not be alloted slots because the leaf nodes are in the last tier of the network and they get synchronized when they receive time sync messages from their respective parents. The leaf nodes do not need to transmit any message.

In our current implementation, the inter time sync message pause is kept to be 0ms (meaning the packets are sent back-to-back), which has been chosen since the time difference between the send time (between the events S1 and S6) and receive time (between the events R1 and R5) for a 44 Byte packet is almost zero, as shown in Figure 5.2. We chose to send 3 of such time sync messages from each sender because, there may be packet losses due to noise in the environment. A too high value is also not needed because the links used are mostly strong as explained in Chapter 4. The duration of each slot is 12ms, since each time sync message takes 4.14ms of send duration (as shown in Figure 5.2) and there are 3 such messages in a slot.

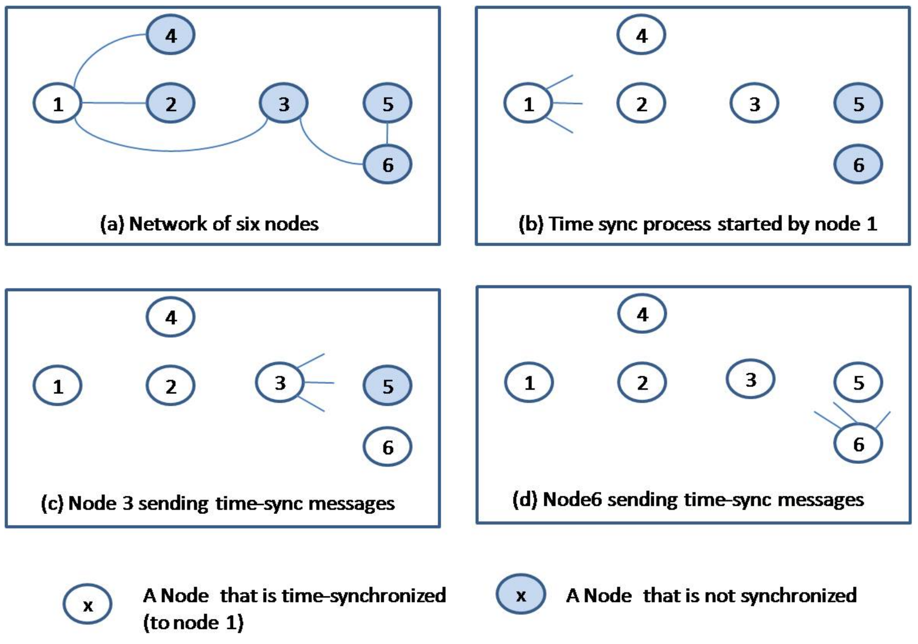

(a) Network of six nodes
(b) Time sync process started by node 1
(c) Node 3 sending time-sync messages
(d) Node6 sending time-sync messages
A Node that is time-synchronized (to node 1)
A Node that is not synchronized

Figure 5.3 Time sync in an example six node cluster

To illustrate with a an example, consider a six node cluster as shown in Figure 5.3(a). Here, only the nodes 1,3 and 6 will be alloted slots (since they are the only non-leaf nodes) and in the same order(since its the only possible Depth First order) as shown by the stages in Figure 5.3 as per Table 5.2.

Thus, we can observe that the time taken for synchronization is directly proportional to

Node Address	Children	Slot number
1	2,3,4	0(initiator)
2	-	-
3	6	1
4	-	-
5	-	-
6	5	2

Table 5.2 Slot information for network shown in Figure 5.3

the number of non-leaf nodes in the network. The same is the case with duration for command propagation because we use the same method of TDMA based flooding for command propagation also as explained in Section 2.3.

CHAPTER 6

Experiments and Results

This chapter gives the details about the experiments conducted and the results obtained. Next, we proceed to the experiments carried out to strengthen the realizability of event detection mechanism in BriMon. Then, we proceed to the evaluation of the protocols we have designed especially for BriMon: the routing protocol and the time synchronization protocol.

In all the experiments described, we used Moteiv Tmote Sky motes [21] for our experiments. These motes come with a Chipcon CC2420 radio chip that is compliant with the 802.15.4 standard. To connect the external antennas to the motes (wherever applicable), we soldered an SMA (Sub-Miniature ver-A) connector while also disconnecting the internal antenna. Note that the Tmote Sky comes with a 3.1 dBi internal antenna [21].

Since the design decisions and the implementation details are explained in the previous chapters, the contents of this chapter serve mostly as a documentation of the experiments and results, but may not explain the reasons about the choices made or the implications of the results obtained, unless they were not covered in the previous chapters.

6.1 Event Detection Experiments

These experiments are intended to find the distance D_d at which a mobile train arrival is detected by the head node of the cluster before commanding the nodes in its cluster to start sleep-wakeup. Link range measurements in the Section 4.1 indicate that if we use external antennas connected to 802.15.4 radios, we can achieve radio ranges of a few hundred metres in line-of-sight environments. However, there we did not consider mobile 802.15.4 radios. Hence we performed a series of careful experiments with one stationary node and one mobile node.

6.1.1 Experiment Setup

We note that we usually have about a 1Km approach zone ahead of a bridge. This is straight and does not have any bends. This is true for most bridges, except in hilly regions. For our experiments too, we use a line-of-sight setting. We used a 900m long air-strip. We mounted the stationary node on a mast about 3m tall. We placed the mobile node in a car, and connected it to an antenna affixed to the outside of the car at a height of about 2m. Both nodes were connected to 8dBi omni-directional antennas. During all these experiments, the transmit power at the mobile node was 0 dBm, the maximum possible with the CC2420 chips.

The mobile node beacons constantly, every 10ms. It starts from one end of the air-strip, accelerates to a designated speed at a specified point and maintains that speed (within human error). The stationary node is 100m away from the other end (so that the car can pass the stationary node at full speed, but still come to a halt before the air-strip ends). For each beacon received at the receiver, we note down the sequence number and the RSSI value. We marked out points on the air-strip every 100m, to enable us to determine where the sender was when a particular beacon sequence number was sent*. Figure 6.1 shows a schematic of the experiment setup.

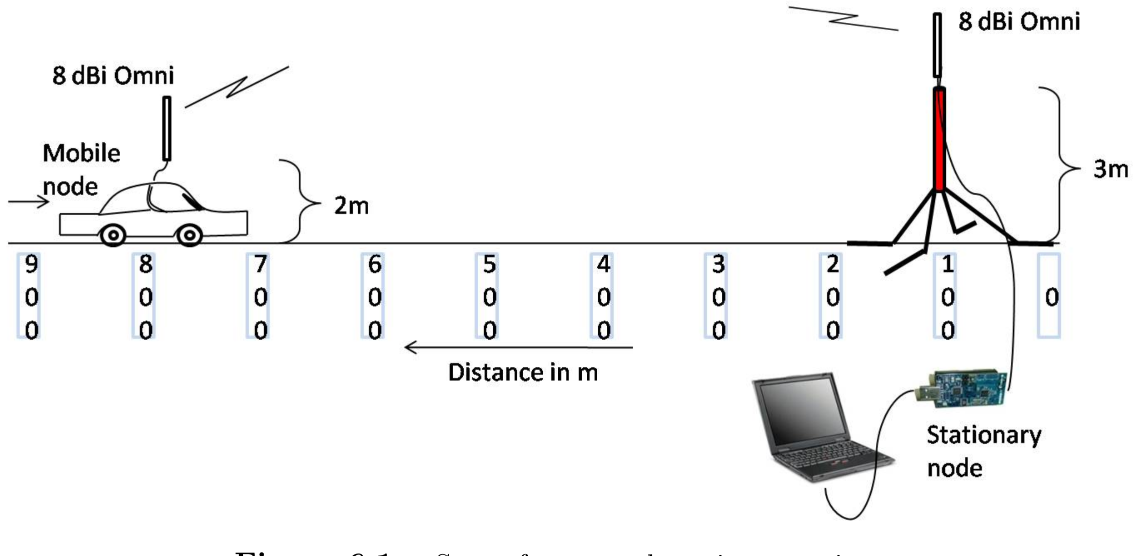


Figure 6.1 Setup for event detection experiments

6.1.2 Results

Figure 6.2 shows a plot of the RSSI as a function of the distance of the mobile sender from the receiver. We observe from the figure that we start to receive packets when the mobile is as far away as 450m, and this is more or less independent of the mobiles speed. The RSSI

*We had a person sitting in the car press the user button of the Tmote sky whenever the car passed a 100m mark; this gives us a mapping between the motes timestamp and its physical position measurements versus distance also have implications for the link range in the (stationary) network on the bridge. If we follow a threshold-based link model (as we are doing in our routing as explained in Chapter 4, with a threshold of -80dBm, as described earlier, we can have link ranges as high as 150-200m.

The difference in range from that reported in the Table 4.2 is due to the difference in environments as well as antennas' heights.

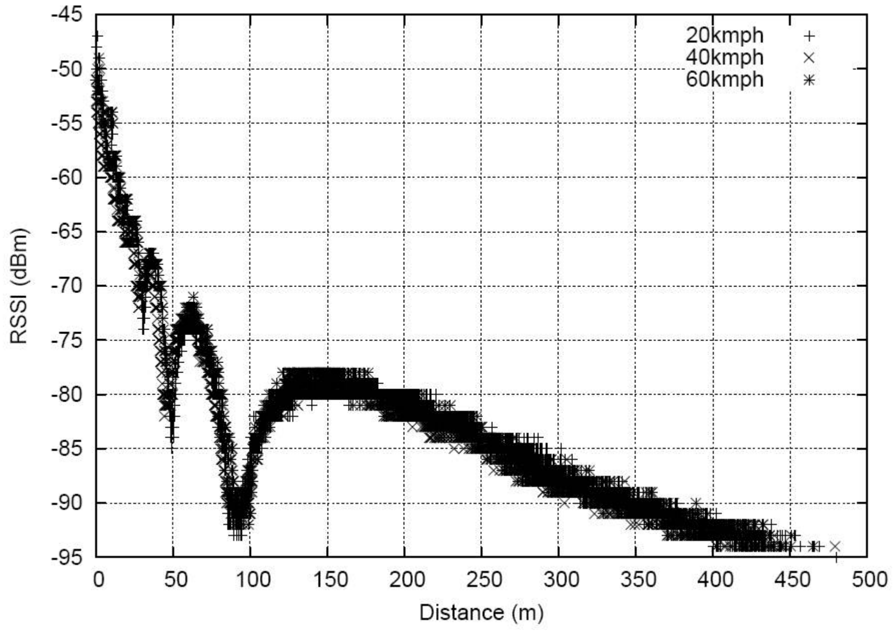


Figure 6.2 RSSI vs. distance between sender & receiver

Another interesting observation to note in Figure 6.2 is the pattern of variation in the RSSI as we get closer to the stationary node, for all mobile speeds. Any RSSI variations observed are generally attributed to unpredictable environmental aspects. In our experiment however, the pattern is entirely predictable: these are due to the alternating constructive and destructive interference of ground reflection which happens at different distances. The exact distance at which this happens depends on the heights of the sender/receiver from the ground. Such variations can be eliminated by using diversity antennas, but the tmote sky hardware does not have such a facility.

During all the above experiments, the transmit power at the mobile node was 0 dBm, the maximum possible with the CC2420 chips. We also tried an experiment with an 802.11 transmitter, which allowed transmission at 20dBm. Now, it is possible to detect transmissions from an 802.11 sender at an 802.15.4 receiver since they operate in the same frequency

(2.4GHz). For this, we can use the CCA (Clear Channel Assessment) detection at the 802.15.4 receiver, as explained in [17]. We used such an arrangement for our experiment, and determined the range to be at least 800m. At this distance, we were limited by the length of the air-strip (900m), and the range is likely more than 800m. In this experiment too, we saw no significant effect of the mobiles speed on this range. What this likely implies is that we can further improve the detection range, if we have a built-in or an external amplifier for the CC2420 chip. We expect that with the use of an external amplifier at the trains node, we can have a range of the order of 800m or more. (Note that the additional power consumption at the trains node is not a concern).

To summarize the above measurements, when the train is coming at a speed of 80 Kmph, and with $D_d = 800$m, we have $t_{dc} = 36s$, where $t_{dc} = D_d \div V$, as per notation in Section 2.3.

6.2 Evaluation of Routing Protocol

From this experiement, we would like to know what could be the frequency with which routing has to be performed and the hop distribution that we can expect. We also intend to obtain an estimate of the duration for which routing takes place, so as to know the extent of overhead caused.

Setup :

For these experiments, we placed six nodes as shown in Figure 6.3. This setup is intended to mimic the way we expect nodes to be placed on a bridge span of about 60m length and 10m width. The nodes go through a sleep-wake up cycle, with a sleep duration of 36 seconds (corresponding to the results of event detection range we have obtained in Section 6.1.2). On every wakeup, the operations of routing and time synchronization happen and then the nodes go to sleep again on getting the appropriate command from their parent. The RSSI threshold for good neighbour is kept at a high value of -75dBm because the adjacent nodes are in a close range of 15m. The setup is shown in Figure 6.3.

Routing was made to run at every wakeup and the head node logs all the routing trees formed. This is so as to know the hop distribution during the course of experiment, along with the data of which all are bad links i.e., links operating below the designated threshold (as explained in Chapter 4). The idea behind running routing on every wake up was to run

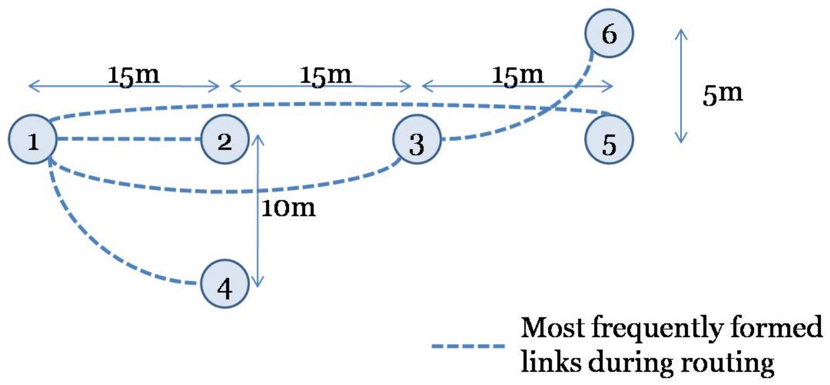


Figure 6.3 Setup for routing protocol evaluation (dotted lines show the links of the most frequently formed tree)

it at the minimum granularity possible and on observing the frequency of changes in the routing trees formed, we can decide upon the required frequency at which routing has to be run, in that particular setting. Also, the time taken for routing during each run is also logged by the root node.

Observations and implications :

The cycle of routing and time synchronization, followed by sleep-wakeup was run continuously without any interruption for about 7 hours. The sleep duration was fixed to 36 seconds following the result from event detection experiments explained in Section 6.1. Thus, during the 7 hours, we had about 700 of these cycles.

We observed from the logs that :

- Most of the time (about 99%), the tree as shown by dotted lines in Figure 6.3 was formed.

  What this implies is that routing protocol can be run quite infrequently, only when a node fails or when a node gets disconnected. Thus, our routing protocol meets the application requirement.

- In the most frequently formed tree (shown in Figure 6.3), all the links were good i.e. the link RSSIs were greater than -75dBm.

  Good links i.e. links with RSSIs greater than -75dBm imply that the error rate is close to 0%, from the plot of RSSI vs. Error rate from Figure 4.2.

- In the most frequently formed tree shown in Figure 6.3, four out of the five nodes (other than cluster head which in our setup is node 1) are having a direct link to root node, where as one node has a 2 hop path to root.

  This implies that most of the nodes in the network may have a direct link to root node. Thus most of the nodes in the network may be leaf nodes, thus reducing the non-leaf nodes. As explained in Chapter 5, this has a direct effect on the time of command propagation (t_{pc}), duration for time synchronization (t_{ps}) and thus the duty cycle.

- The average duration of routing was about 567ms, with a standard deviation of 51ms. The minimum duration was 467ms, whereas the maximum duration was 819ms.

  Thus, the duration for topology formation is less than a second on average, which is a desired feature to keep the duty cycle minimum, even if we run routing protocol frequently (than required).

6.3 Evaluation of Time Synchronization Protocol

The experiments carried for the evaluation of time synchronization protocol fall into two categories: measurement of t_{ps} (time taken for synchronization) and estimation of t_{Δ} (synchronization error). Recall from Section 5.1 that the synchronization error does not account for clock drift and is just the maximum difference between the global times as thought by the nodes in the entire cluster.

Measurement of t_{ps} :

We chose to measure t_{ps} in a cluster of twelve nodes with the topology fixed as shown in Figure 6.4. All the twelve nodes are kept in close vicinity as the static routing tree has already been imposed. In this network, there are five parent nodes apart from the root node, which means that six time sync slots have to be allocated as was explained in Section 5.3.2.

The process of time synchronization starts on pressing the user button on the root node. All the nodes transmit time sync messages during their respective slots as per the protocol.

The setup also includes a passive Logger node, that hears to each and every packet on air and logs it into its RAM, along with a local time stamp attached to each message. Later, on pushing the User button on the logger, it transmits the log collected at a low transfer rate with an inter-packet pause of 20ms (so as not to miss any packet due to buffer overflow at the receiver) to a node running TOSBase, which forwards all the packets to a computer. Figure 6.4 shows the setup.

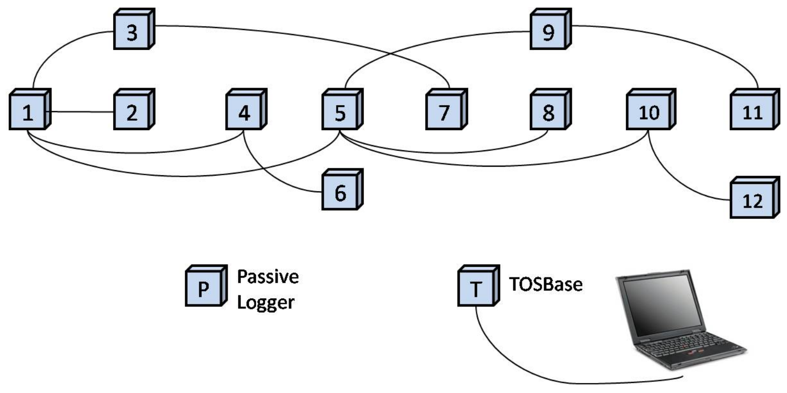


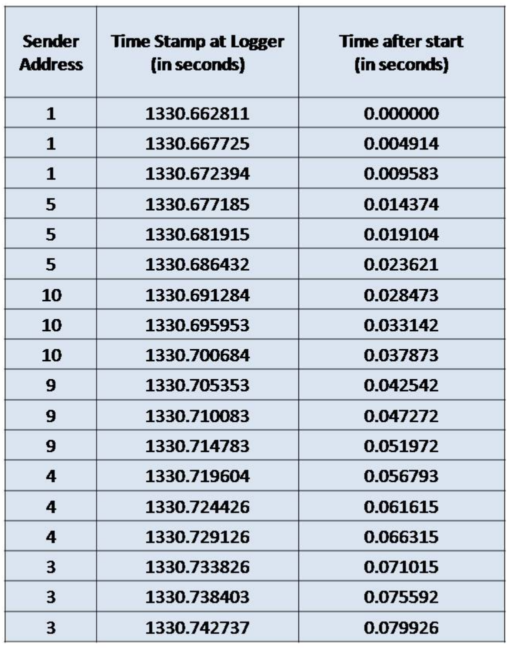
Figure 6.4 Setup for t_{ps} measurement

In the experiment, we gave the inter packet pause for the sync messages from a node to be 0ms (i.e. the packets are sent back-to-back) as explained in 5.3.2 and each node (which has a slot) sends 3 Time Sync messages in its slot duration, which is kept as 12ms. This is due to the fact that each time-sync packet (of size 44 Bytes) takes 4.14 ms as was shown in Section 5.3.2. Figure 6.5 shows a log collected along with a column showing the *time from start* of the process.

From this table, we can see that the process took approximately 80ms. This is tallying with the expected value since there are six nodes, and each node is given a slot of 12ms, we expect it to take around $12 * 6 = 72ms$. The same results are obtained on several runs.

Measurement of t_{Δ} :

We use the same topology of twelve nodes for these experiments too, the setup of which is shown in Figure 6.6. All the nodes in the setup are in close vicinity to each other. First, all the nodes are synchronized as per the protocol. Immediately after that, a poller node will send a single broadcast message. On hearing this message, each will take its respective local time stamp and global time stamp (=local time stamp + offset). Then, after a duration of $20ms * nodeID$, nodeID = 1,2..6, all the nodes transfer their time stamps to a TOSBase node, which forwards them to a PC. This duration of $20 * nodeID$ is based on the node address just to avoid contention among the nodes while sending their time stamps. The



Sender Address	Time Stamp at Logger (in seconds)	Time after start (in seconds)
1	1330.662811	0.000000
1	1330.667725	0.004914
1	1330.672394	0.009583
5	1330.677185	0.014374
5	1330.681915	0.019104
5	1330.686432	0.023621
10	1330.691284	0.028473
10	1330.695953	0.033142
10	1330.700684	0.037873
9	1330.705353	0.042542
9	1330.710083	0.047272
9	1330.714783	0.051972
4	1330.719604	0.056793
4	1330.724426	0.061615
4	1330.729126	0.066315
3	1330.733826	0.071015
3	1330.738403	0.075592
3	1330.742737	0.079926

Figure 6.5 The log taken during t_{ps} measurement

factor 20 ms is chosen so as to provide adequate interval between the nodes sending their time stamp log.

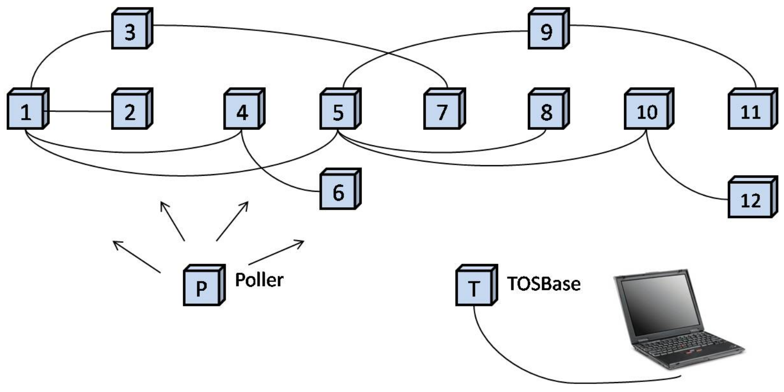
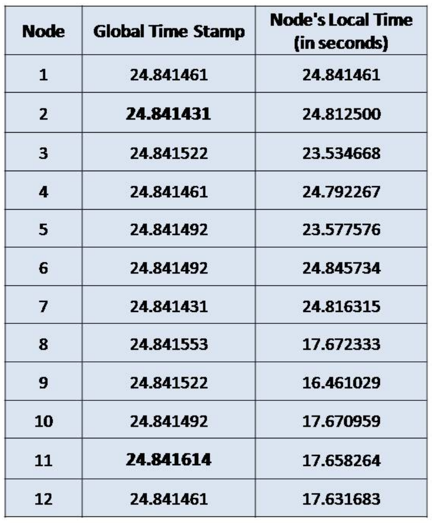


Figure 6.6 Setup for $t_{\triangle}$ measurement

Since the message sent by the poller node is heard by all the nodes at the same time instant, they all should report the same global times, since they are synchronized. A log obtained during a run is shown in Figure 6.7. From the figure, we can see that the global time stamps reported by the nodes differ by a maximum of 0.183ms or 6 clock ticks. This is the difference between the maximum and the minimum values of the global time stamps reported by all the nodes in the network, shown in the second column in the Figure 6.7 (maximum and minimum values are indicated in bold font). Infact this is the worst case that we obtained during several runs carried out.



Node	Global Time Stamp	Node's Local Time (in seconds)
1	24.841461	24.841461
2	**24.841431**	24.812500
3	24.841522	23.534668
4	24.841461	24.792267
5	24.841492	23.577576
6	24.841492	24.845734
7	24.841431	24.816315
8	24.841553	17.672333
9	24.841522	16.461029
10	24.841492	17.670959
11	**24.841614**	17.658264
12	24.841461	17.631683

Figure 6.7 The log taken during t_Δ measurement

This had been intuitively clear for us, since when we tried to synchronize two nodes (one hop), we obtained that the maximum synchronization error was two clock ticks. Now, in the twelve node test network used for the above experiment, the longest path is of three hops. So, in the worst case, the adjacent nodes can differ by two clock ticks and the difference increases as we go toward the other end of the path.

Thus, in general we can say that the maximum synchronization error in the cluster depends on the *depth* of the routing tree: the more the depth, the more might be the error.

Significance: As explained in Section 2.3, the command system also uses the same TDMA based flooding as done by time synchronization. Thus, the importance of this result is also with respect to the duty cycle of the nodes. The wakeup duration, $t_w = t_{pc} + 2 * t_\Delta$, is directly affected by the values $t_{ps} (= t_{pc})$ and t_Δ obtained in these experiments. Thus, for the values obtained, we anticipate that the wake-up duration can be $t_w = t_{pc} + 2 * t_\Delta = (72 + 2 * 0.183) \approx 72.5ms$. With the sleep duration of 36 seconds from the results in Section 6.1.2, we obtain that the minimum duty cycle can be $72.5 \div (36000 + 72.5) \approx 0.2\%$.

CHAPTER 7

Conclusion and Future Work

Railways are the most prevalent transport infrastructure in many parts of the world. For instance, Indian Railways is one of the largest enterprises in the world. And bridges form a critical part of railways. Indian Railways has approximately 127,000 bridges of which about 51,000 are more than 100 years old. In order for this system to operate smoothly, assurance of safety of travel over bridges is crucial. Existing techniques for this are mostly wired solutions and require technical personnel to be present at the inspection site. These suffer from the disadvantages the use of bulky equipment and long setup duration.

In this work, we present a new system for long term monitoring of railway bridges, named BriMon. The system is based on the technology of Wireless Sensor Networks. The proposed system has a number of advantages over the currently used wired systems. Our system is easier to deploy, lower in cost and autonomous in its operation. The design allows the whole setup to be left on site and the data can be retrieved on demand.

We propose a novel event detection mechanism that allows a very low duty cycle of the nodes. We also propose a new approach to fetch the data from the remote site using the train itself.

Our design choices for protocols have been made to aid specific application requirements. The protocols designed for routing, time synchronization are very much application specific, which rendered them simple in design. These protocols have been implemented and evaluated separately. Then they are integrated to the other components of duty cycling and event detection.

In a parallel work on the same project [22], the elements of data acquisition, reliable data transfer, and a data analysis tool have been developed. We integrated all these components to make it a complete system as per the guidelines we have formulated. These guidelines are discussed in Appendix B.

Although many design choices were made specific to this application, we believe that the same set of protocols would be relevant to many other applications too, particularly by grouping a few tens of sensor nodes into a cluster. In view of this, we are in the process of making a software release to the open source community.

Future Work :

In the current system, we do not estimate the speed at which a train is coming. This will be required since the trains may some times slow down at some distance from the bridge due to traffic signals. If the event detection happens by then, the nodes will already start collection, even before the train arrives over the bridge. Thus, estimation of speed and/or position of the train during the event detection will be a good feature that can be added.

Also, the present system has been designed to measure the vibration of 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 feature to the current system.

Appendix A

Time Synchronization in TinyOS

This appendix gives a description of the modules in Boomerang version of TinyOS [20], which are used for time synchronization purpose. These modules were not present in the TinyOS 1.x version. This discussion is specific to the platforms using CC2420 radio.

A common technique employed for time synchronization is using transfer of messages containing time stamps. As was explained in Chapter 5, the uncertainities involved at the sender side and the receiver side can be removed by putting the time stamp at MAC layer of the communication protocol stack. In TinyOS, the implementation of MAC layer is done in a module called CC2420RadioM.

In order to perform time stamping at the MAC layer, we need a component that detects the rise of Start of Frame Delimiter (SFD) pin of the CC2420 radio and instantaneously takes the time stamp of the local clock. The component which does it for us is the CC2420TimeStampingC. This capturing of SFD is already done at the MAC layer (i.e. in CC2420RadioM) for its own operation. CC2420RadioM provides an interface that signals about the change in state of this SFD to the CC2420TimeStampingC, which is called RadioCoordinator. Apart from this interface, the other interfaces required by the CC2420TimeStampingC component are the LocalTime (in order to take time stamp during sending and/or reception) and the HPLCC2420RAM (used to write the time stamp taken during SFD of a send event to the specified byte position in the RAM of CC2420 radio chip). The whole implementation of this process is in CC2420TimeStampingM.

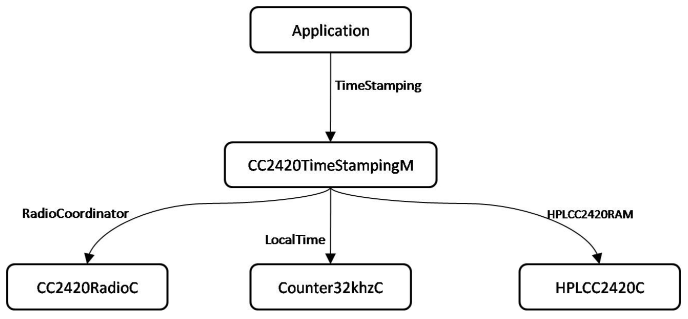


Figure A.1 Wiring of TinyOS components (in the context of time synchronization)

Now, whenever an application needs to use the service of MAC level time-stamping provided by CC2420TimeStampingC component, it needs to use the TimeStamping interface provided by CC2420TimeStampingC. The interface consists of the commands addStamp(TOS_MsgPtr msg, int8_t offset) and getReceiveStamp(TOS_Msg *msg) which can be used to put send time stamp and get receive time stamp respectively.

A diagram showing the wiring of these components is shown in Figure A.1.

Appendix B

BriMon Integration Guidelines

In this appendix, we describe the guidelines/assumptions under which the current integrated version of BriMon is implemented.

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 B.1. 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.

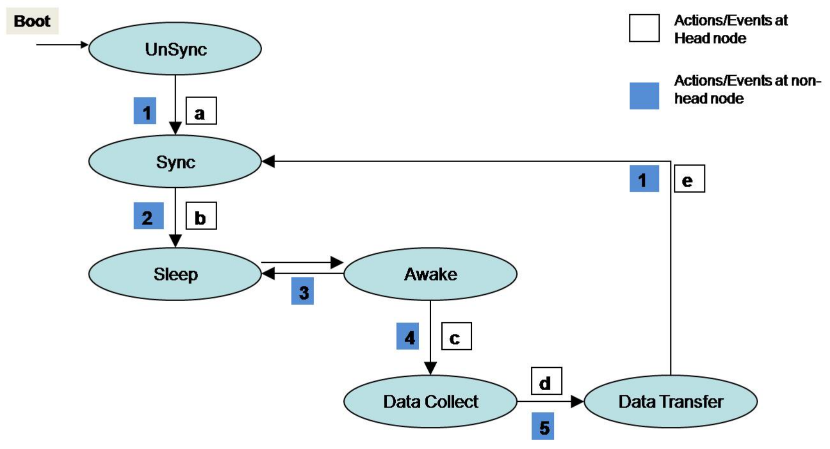


Figure B.1 State Diagram of BriMon cluster (Ideal case of no message losses)

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

$X = (\frac{currentGlobalTime}{SleepWakeupPeriod} + 1) * SleepWakeupPeriod - currentGlobalTime$ 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 BriMon-App 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 *Event_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.

References

1. Alec Woo, Terence Tong and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks In *SenSys 03*.

2. Ivan Stojmenovic, Amiya Nayak and Johnson Kuruvila. Design guidelines for routing protocols in ad hoc and sensor networks with a realistic physical layer In IEEE Communications Magazine *(AdHoc and Sensor Networks Series)*, March 2005.

3. W. R. Heinzelman, A. Chandrakasan, H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks In IEEE Proceedings of the Hawaii *International Conference on System Sciences*, Jan 2000.

4. W. Heinzelman, J. Kulik, and H. Balakrishnan Adaptive protocols for information dissemination in wireless sensor networks In *MobiCom99*.

5. Perkins, C. E. and Bhagwat, P. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers In *SIGCOMM '94*.

6. Bhaskaran Raman, Kameswari Chebrolu, Naveen Madabhushi, Dattatraya Y Gokhale, Phani K Valiveti, and Dheeraj Jain. Implications of Link Range and (In)Stability on Sensor Network Architecture In *WiNTECH*, Sept 2006.

7. K. Srinivasan and P. Levis. RSSI is Under Appreciated. In *EmNets*, May 2006.

8. D. L. Mills. Internet time synchronization: The Network Time Protocol *IEEE Computer Society Press*, 1994.

9. J. Elson, L. Girod and D. Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts In the *proceedings of the fifth symposium on Operating System Desgn and Implementation (OSDI 2002)*, Dec 2002.

10. M. Marti, B. Kusy, G. Simon. The flooding time synchronization protocol In *SenSys04*.

11. S. Ganeriwal, R. Kumar, M. B. Srivastava. Timing-sync protocol for sensor networks In *SenSys03*, Nov 2003.

12. Dennis Cox, Emil Jovanov, and Aleksandar Milenkovic. Time Synchronization for ZigBee Networks In *SSST*, Mar 2005.

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

14. Arsalan Avatoii, Jingbin Zhang, Sang H. Son. Group-Based Event Detection in Undersea Sensor Networks In *International Workshop on Networked Sensing Systems (INSS'05)*, San Diego, California, 2005.

15. Geoffrey Werner-Allen, Konrad Lorincz, Matt Welsh, OmarMarcillo, Jeff Johnson,Mario Ruiz, and Jonathan Lees. Deploying a Wireless Sensor Network on an Active Volcano In IEEE *Internet Computing*, Mar/Apr 2006.

16. Abadi, D., Madden, S., and Lindner, W.. REED: Robust, efficient filtering and event detection in sensor networks In *VLDB, 2005.*

17. Nilesh Mishra, Kameswari Chebrolu, Bhaskaran Raman, and Abhinav Pathak. Wake-on-WLAN In the *15th Annual Interntional World Wide Web Conference (WWW 2006)*, May 2006.

18. Fernando Martincic, Loren Schwiebert. Distributed Event Detection in Sensor Networks In *International Conference on Systems and Networks Communication (ICSNC'06).*

19. Chipcon. *Chipcon AS SmartRF(R) CC2420 Preliminary Datasheet (rev 1.2)*, Jun 2004.

20. Moteiv Boomerang. http://www.moteiv.com/software/.

21. Moteiv Tmote. http://www.moteiv.com/products/tmotesky.php.

22. Maj Raj Kumar. A Prototype development of Reliable Sensor Network Based Structural Health Monitoring System for Railway Bridges Master's thesis. Department of Electrical Engineering, IIT Kanpur, May 2007, India.

23. Hemanth Haridas. BriMon: Design and Implementation of Railway Bridge Monitoring Application Master's thesis. Department of Computer Sciences and Engineering, IIT Kanpur, May 2006, India.

24. Nilesh Mishra. Design Issues and Experiences with BRIMON Railway BRIdge MONitoring Project Master's thesis. Department of Computer Sciences and Engineering, IIT Kanpur, June 2006, India.

25. News Release by Indian Railways on 27th April, 2007. http://economictimes.indiatimes.com/40_railway_bridges_over_100_years_old/articleshow/1967908.cm

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

27. 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.

28. E. Sazonov, K. D. Janoyan,and R. Jha. Wireless Intelligent Sensor Network for Autonomous Structural Health Monitoring Smart Structures. In *NDE 2004*, San Diego, California 2004.

29. C. Park, Q. Xie and P. H. Chou. DuraNode: Wi-Fi-based Sensor Node for Real-Time Structural Safety Monitoring In *IPSN05*, Apr 2005.

30. N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan and D. Estrin. A wireless sensor network For structural monitoring. In *SenSys04*, Nov 2004.

31. 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.

32. 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*, Nov 2005.

33. A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. An analysis of a large scale habitat monitoring application. In *SenSys04*, Nov 2004.

34. P. Zhang, C. M. Sadler, S. A. Lyon and M. Martonosi. Hardware design experiences in ZebraNet. In *SenSys04*, Nov 2004.

35. L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and Deployment of Industrial Sensor Networks: Experiences from a Semiconductor Plant and the North Sea. In *SenSys*, Nov 2005.

36. Contiki Operating System. http://www.sics.se/contiki/

37. Mantis Operating System. http://mantis.cs.colorado.edu/tikiwiki/tiki-index.php

38. Definition of Wireless Sensor Network. http://en.wikipedia.org/wiki/Sensor_network

39. BTnut Operating System. http://www.btnode.ethz.ch

40. TinyOS operating System. http://www.tinyos.net

41. NesC Implementation of MultiHopLQI. http://tinyos.cvs.sourceforge.net/tinyos/tinyos-1.x/tos/lib/MultiHopLQI/

42. Personal Communication with Dr. C.V.R. Murthy, Professor, Dept of Civil Engineering, IIT Kanpur and Dr. K.K. Bajpai, Structures Lab, Dept of Civil Engineering, IIT Kanpur