BriMon: A Sensor Network System for Railway Bridge Monitoring

Kameswari Chebrolu*, Bhaskaran Raman*, Nilesh Mishra+, Phani Kumar Valiveti†, Raj Kumar‡
*Indian Institute of Technology, Bombay +University of Southern California
†Cisco Systems ‡Indian Army

Abstract: Railway systems are critical in many regions, and can consist of several tens of thousands of bridges, being used over several decades. It is critical to have a system to monitor the health of these bridges and report when and where maintenance operations are needed. This paper presents BriMon, a wireless sensor network based system for such monitoring. The design of BriMon is driven by two important factors: application requirements, and detailed measurement studies of several pieces of the architecture. In comparison with prior bridge monitoring systems and sensor network prototypes, our contributions are three-fold. First, we have designed a novel event detection mechanism that triggers data collection in response to an oncoming train. Next, BriMon employs a simple yet effective multi-channel data transfer mechanism to transfer the collected data onto a sink located on the moving train. Third, the BriMon architecture is designed with careful consideration of the interaction between the multiple requisite functionalities such as time synchronization, event detection, routing, and data transfer. Based on a prototype implementation, this paper also presents several measurement studies to show that our design choices are indeed quite effective.

1 Introduction

Railway systems are a critical part of many a nation's infrastructure. For instance, Indian Railways is one of the largest enterprises in the world. And railway bridges form a crucial part of the system. In India, there are about 120,000 such bridges [1] spread over a large geographical area. 57% of these bridges are over 80 years old and many are in a weak and distressed condition. It is not uncommon to hear of a major accident every few years due to collapse of a bridge. An automated approach to keeping track of bridges' health to learn of any maintenance requirements is thus of utmost importance.

In this paper, we present the design of *BriMon*, a system for *long-term* railway **bri**dge **mon**itoring. Two factors guide the design of BriMon. (1) Given the huge number of existing bridges that need to be monitored, it is important that any solution to the problem should be *easy to deploy*. (2) Next, since technical expertise is both difficult to get and expensive on field, it is equally important that the deployment require *minimal maintenance*.

To facilitate ease of deployment, we choose to build our system based on battery operated *wireless* sensor nodes. BriMon consists of several tens to hundreds of such nodes equipped with accelerometers, spread over the multiple spans of the bridge. The use of *wireless transceivers* and *battery* eliminates the hassle of having to lay cable to route data or power (tapped from the 25 KV overhead high voltage line if available) to the various sensors that are spread about on the bridge. Cables and high voltage transformers typically need special considerations for handling and maintenance: safety, weather proofing, debugging cable breaks, etc.; the use of wireless sensor nodes avoids these issues.

We also reject the possibility of using solar panels as a source of renewable energy. They are not only expensive, they are also cumbersome to use: some sensors may be placed under the deck of the bridge where there is little sunlight. Furthermore, solar panels are also prone to theft in the mostly unmanned bridges.

Given the choice of a battery operated wireless nodes for BriMon, a key goal which drives our design is low energy consumption, so that the maintenance requirements of BriMon (visits to the bridge to change battery) are kept to a minimum.

A significant challenge which arises in this context is the following. The nodes need to sleep (turn off radio, sensors, etc) most of the time to conserve power. But they also need to be ready for taking accelerometer measurements when there is a passing train. BriMon employs a novel event detection mechanism to balance these conflicting requirements.

Our event detection mechanism consists of a beaconing train and high gain external antennae at designated nodes on the bridge that can detect the beacons much before (30s or more) the train approaches the bridge. This large guard interval permits a very low duty cycle periodic sleep-wakeup-check mechanism at all the nodes. We use a combination of theoretical modeling and experimentation to design this periodic wakeup mechanism optimally.

On detecting a train, BriMon nodes collect vibration data. The collected data then has to be transferred to a central location. This data will be used for analysis of the bridge's current health as well as for tracking the deterioration of the bridge structure over time. For this, BriMon uses an approach quite different from other sensor network deployments [2, 3, 4, 5, 6, 7]. We use the passing trains themselves for the data transfer. The data transfer mechanism is also activated through the same event detection mechanism as for data collection.

A very significant aspect of the mobile data transfer model is that it allows us to break-up the entire system of sensor nodes (of the order of few hundred nodes) into multiple *independent* and much smaller networks (6-12 nodes each). This greatly simplifies protocol design, achieves good scalability and enhances performance.

In our overall architecture, apart from event detection and mobile data transfer, two other functionalities play important support roles: time synchronization and routing. Time synchronization is essential both for duty-cycling as well as in the analysis of the data (correlating different sensor readings at a given time). Routing forms the backbone of all communication between the nodes. These four functionalities are all inter-dependent and interfacing them involves several design choices as well as parameter values. We design this with careful consideration of the application requirements as well as measurement studies.

In comparison with prior work in structural monitoring [6, 7, 8], our contributions are three-fold: (a) a novel event detection mechanism, (b) a detailed design of mobile data transfer, and (c) the tight integration of the four required functionalities. While the notion of mobile data transfer itself has been used in prior work (e.g. ZebraNet [9], DakNet [10]), its integration with the rest of the bridge monitoring system, and careful consideration of interaction among various protocol functionalities, are novel aspects of our work.

To validate our BriMon design, we have prototyped the various components of the system. In our prototype, we use the Tmote-sky sensor nodes which have an 8MHz MSP430 processor and the 802.15.4 compliant CC2420 radio, operating in the 2.4GHz band. Our prototype also extensively uses external high-gain antennas connected to the motes [11]. Although BriMon design is more or less independent of the choice of accelerometers, it is worth noting that we use the MEMS-based ADXL 203 accelerometer in our prototype. Estimates based on measurements using our prototype indicate that the current design of BriMon should be deployable with maintenance requirement only as infrequent as once in 1.5 years or so (with 4 AA batteries used at each node).

Though our design has focused on railway bridges so far, we believe that the concepts behind BriMon will find applicability in a variety of similar scenarios such as road bridge monitoring, air pollution monitoring, etc.

The rest of the paper is organized as follows. The next section provides the necessary background on bridge monitoring and deduces the requirements for system design. Subsequently, Sec. 3 describes the overall BriMon architecture. Then, Sec. 4, Sec. 5, Sec. 6, and Sec. 7 present the detailed design of the four main modules of BriMon respectively: event detection, time synchronization, routing, and mobile data transfer. Sec. 8 highlights our contributions vis-a-vis prior work in this domain. We present further points of discussion in Sec. 9 and conclude in Sec. 10.

2 Background on Bridge Monitoring

In this section, we provide a brief background on bridge monitoring. The details presented here drive several of our design choices in the later sections. This information was gathered through extensive discussion with structural engineers.

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 [6, 12]. 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 Fig. 1.

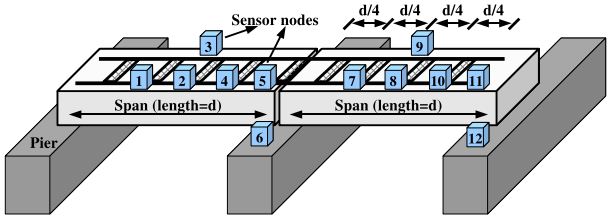

Figure 1. Spans on a bridge

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

When, how long to collect data: When a train is on a span, it induces what are known as *forced vibrations*. After the train passes the bridge, the structure vibrates freely (*free vibrations*) with decreasing amplitude till the motion stops. Structural engineers are mostly interested in the natural and higher order modes of this free vibration as well as the corresponding damping ratio. Also of interest sometimes is the induced magnitude of the forced vibrations. For both forced as well as free vibrations, we wish to collect data for a duration equivalent to about five time periods of oscillation. The frequency components of interest for these structures are in the range of about 0.25 Hz to 20 Hz [6, 7, 12]. 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).

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 in our case. Each sample is 12-bits (because of use of a 12 bit Analog-to-Digital converter). The total data generated by a node can be estimated as: $3channels \times 12bits \times 40Hz \times 40sec = 57.6Kbits$. There are an estimated 6 sensor nodes per span, and a maximum of 12 nodes per data span. Thus the total data we have per data-span per data collection cycle is a maximum of $57.6 \times 12 = 691.2Kbits$.

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. Note that this is of much coarser granularity than what is typically described in several time synchronization protocols (e.g. FTSP [13]).

3 BriMon: Design Overview

With the application details as given above, we now present the design of BriMon. The various components of our design are closely inter-related. Given this interaction, we first present an overview of the design in this section, before moving on to the details in subsequent sections.

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

Important questions in this context are:

- How do we balance the requirement for duty cycling with the fact that train arrivals are unpredictable?
- How do we transfer the data from the place of collection to a repository?
- How can we achieve scaling, for potentially long bridges?
- What are going to be the inter-dependencies among BriMon's components, and how do we resolve them?

We answer these questions as follows.

Event detection: We balance the requirements of having to duty cycle, and being ready when a train arrives, through our event detection mechanism. We use the fact that significant radio range is achievable with the 802.15.4 radios [11], on using external antennas. An 802.15.4 node on the train beacons as it arrives, and is detected several tens of seconds in advance (before the train is on the span) by nodes on the span. This enables a periodic sleep/wake-up mechanism for the nodes on the bridge.

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.

Data span as an independent network: One fundamental design decision we make in BriMon is to treat each data-span as an *independent* network. This is possible primarily due to the fact that the data from each data-span is independent physically (different physical structures). This also fits in well with our mobile data transfer model. The alternative here is to treat the entire bridge (including all data spans) as a single network. We rejected this approach since there is no specific reason for the data-spans to know about one another or inter-operate in any way. Having a smaller network simplifies protocol design, and enables much better performance.

A designated node on each span *gathers* all the data collected by the different sensor nodes on that span. It then transfers this data onto the moving train. We make different spans operate on different independent channels, so that the transfer on each span can proceed simultaneously and independently.

Inter-dependence challenges: The event detection as well as data transfer bank on two underlying mechanisms: time synchronization and routing. So there are four main functionalities in BriMon: (a) event detection coupled with periodic sleep/wake-up, (b) mobile data transfer, (c) time synchronization, and (d) routing. In this context, several non-obvious questions arise:

- What protocols should we use for time synchronization and routing?
- More importantly, how should these two interact with any duty cycling?
  - Should routing be run for each wake-up period, each time a node wakes up? Or should it be run periodically, across several wake-up periods? If the latter, can we be sure that routes formed during the prior wake-up period will still be valid?
  - Similarly, when exactly should time synchronization be run? How do we balance between synchronization overhead and having a bound on the synchronization error?
- Also important is the interaction between routing and time synchronization. Which functionality should build on the other? Can time synchronization assume routing? Or should routing assume time synchronization?

To our knowledge, such interfacing challenges have not been addressed in a significant way in prior sensor network deployments [2, 3, 4, 5, 6, 7, 9]. These questions are significant even for the small networks corresponding to each data-span, and we answer them as follows.

Approach to time synchronization: We require time synchronization for two things: for the periodic sleep/wake-up 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 Sec. 4.1 that our periodic sleep/wake-up

has a period of the order of 30-60s, with a wake-up duration of about 200ms. And we show that we can have a light-weight 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 [14] 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 Sec. 2 that we require only about 5ms or less accuracy in synchronization. So this too does not require any drift estimation.

Our approach to time synchronization is simple and efficient, and is in contrast with protocols in the literature such as FTSP [13]. FTSP seeks to estimate the clock drift, and synchronize clocks to micro-second granularity. Due to this, it necessarily takes a long time (of the order of a few minutes [13]). Furthermore, it is not clear how FTSP could be adapted to work in a periodic sleep/wake-up setting such as ours.

Approach to routing: 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. For this, we refer to an earlier study in [11], where the authors show the following results, on the same 802.15.4 radios as used in our work. (1) When we operate links above a certain threshold RSSI (received signal strength indicator), they are very stable, even across days. (2) Below the threshold, the link performance is unpredictable over small as well as large time scales (few sec to few hours).

This threshold depends on the expected RSSI variability, which in turn depends on the environment. In practice, in mostly line-of-sight (LOS) environments such as ours[1], operating with a RSSI variability margin of about $10dB$ is safe. So, given that the sensitivity of the 802.15.4 receivers is about $-90dBm$, having an RSSI threshold of about $-80dBm$ is safe.

With such a threshold based operation, in [11], 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 this paper also corroborate 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.

With this high level description, we now move on to the detailed design of each of the four components of BriMon.

4 Event Detection in BriMon

Event detection forms the core of BriMon. It is needed since it is difficult to predict when a train will cross a bridge (trains can get delayed arbitrarily). It needs to go hand-in-hand with a duty cycling mechanism for extending battery lifetime, and thus minimizing maintenance requirements.

In the description below, we first assume that the nodes are synchronized (possibly with some error). And we assume that we have a routing tree, that is, each node knows its parent and its children. Once we discuss time synchronization and routing, it will become apparent as to how the system bootstraps itself. At the core of our event detection mechanism is the ability to detect an oncoming train before it passes over the bridge. We seek to use the 802.15.4 radio itself for this.

Event detection model: Our model for event detection is depicted in Fig. 2. (For convenience, Appendix A gives a glossary of the various terms/notations we use). 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 Fig. 1), if it were awake. Assume for the time being that the node can detect this instantaneously, as soon as the train comes in range; we shall later remove this assumption. We denote by T_{dc} the maximum time available between the detection of the oncoming train, and data collection. Thus $T_{dc} = D_d/V$ where V is the speed of the train (assumed constant).

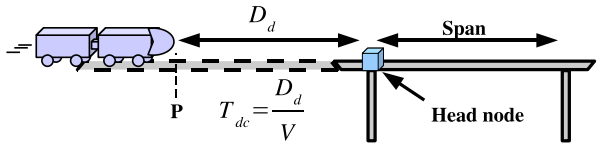

Figure 2. Detecting an oncoming train

In our design, all nodes duty cycle, with a periodic sleep/wake-up mechanism. One node per data-span is designated as the *head* node. This is typically the node which the train would pass first[2] (node-1 in Fig. 2). This head node has the responsibility of detecting an oncoming train. During its wake-up period, if it detects a beacon from a train, it sends out a *command* to the rest of the nodes in the network to remain awake (and not go back to sleep), and start collecting sensor data. So the other nodes have to listen for this command during the time they are awake.

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 T_w of awake time. Thus $T_{cc} = T_{sl} + T_w$. We now have to determine what T_w and T_{cc} have to be. Clearly we would like to have as large a T_{cc} as possible to reduce the duty cycle. We derive this now.

Note that we have to work under the constraint (C0) that an oncoming train must be detected *in time* for data collection:

[1] Within a span, it is reasonable to expect several pairs of nodes with LOS between them.

[2] We assume for now that vibration measurements are triggered only by trains going in one of the two possible directions.

all nodes must be awake and ready to collect data by the time the train enters the data-span.

We ensure constraint C0 by using two sub-constraints. SC1: If the head node detects an oncoming train at the beginning of one its T_w windows (i.e. as soon as it wakes up), then it should be able to convey the command to the rest of the nodes within the same T_w. And SC2: there must at least be one *full* T_w duration between the time the train is in range and the time data collection is due: that is, the train is between point P and the head node in Fig. 2.

Clearly, SC1 and SC2 ensure that C0 is satisfied. Now, SC1 determines T_w and SC2 determines T_{cc}, as we explain below.

SC1 says that the window T_w should essentially be sufficient for the head node to be able to convey a command. Denote the time taken by the protocol for command issue as T_{pc}. That is, within this time, all nodes in the network would have learnt of any command issued by the head node. In addition to T_{pc}, T_w should also include any possible clock differences between the various nodes (i.e. due to synchronization error). Let us denote by T_Δ the maximum possible clock difference.

In our design, we choose to work with the worst case possible T_Δ, and assume that it is known. The worst case T_Δ can be estimated for a given time synchronization mechanism.

With such an approach, the head node should wait for a duration of T_Δ before starting its command issue, to ensure that the other nodes will be awake. Thus, T_w should at least be $T_\Delta + T_{pc}$. Now, from the point of view of the non-head nodes in the network, T_w should include an additional T_Δ. This is because it could have been the case that the other nodes in fact were awake T_Δ *earlier* than the head node. Thus we have $T_w = 2T_\Delta + T_{pc}$.

Using SC2, we can fix the relation: $T_{cc} \leq T_{dc} - T_w$, as we explain now. Fig. 3 argues why this condition is both necessary and sufficient for ensuring SC2. In the figure, we consider various possibilities for the occurrence of the time window T_{dc}, with respect to the time-line at the head node. t_0 is the start of the first T_w window, before the end of which T_{dc} starts. If T_{dc} starts (the train comes within range) at or before t_0, command issue happens during the first T_w (this is case (a)). Otherwise, T_{dc} starts after t_0 but before t_1, and command issue happens in the second T_w (this is case (b)). Clearly in both cases, we have a full T_w window when the train is in range, and before data collection is due.

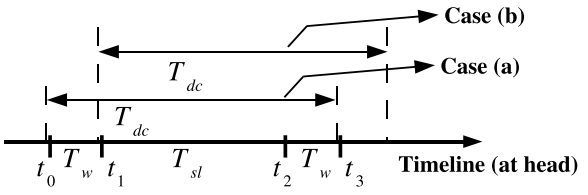

Figure 3. Train detection by head node: $T_{cc} \leq T_{dc} - T_w$

Since we want T_{cc} to be as large as possible, we have $T_{cc} = T_{dc} - T_w$ as the optimal value.

Incorporating detection delays: One aspect which the description above has not considered is the delay which would be involved in detecting the train once the head node wakes up, and the train is in range. Suppose the period of the beacons from the train is T_b. And suppose we define D_d such that the packet error rate (of the beacons) from the train to the head node is at most 20%. Then we can surely say that within 5 beacon periods, the probability that the train goes undetected is extremely small ($< 10^{-3}$). So it would work in practice to set the detection delay T_{det} to be $5T_b$.

Now, to incorporate T_{det} in our sleep/wake-up mechanism, we need to add the following feature: the head node has to be awake for a duration T_{det} ahead of the other nodes in the network. So for the head node, $T_w = T_{det} + T_\Delta + T_{pc}$. Note that the non-head nodes still have $T_w = 2T_\Delta + T_{pc}$.

The above is the essence of our event detection mechanism. In this, we have three important parameters: D_d, T_{pc}, and T_Δ. We now describe detailed experiments to the maximum possible D_d (i.e. how soon we can detect the oncoming train). The next section (Sec. 5) then looks at the other two important parameters: T_{pc}, and T_Δ, in the context of our synchronization mechanism.

4.1 Radio range experiments

The distance D_d essentially captures the distance at which beacons sent from the train can be received at the head node. Measurements in [11] 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, [11] does not consider mobile 802.15.4 radios. Hence we performed a series of careful experiments with one stationary node and one mobile node[3].

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.

The mobile node beacons constantly, every 10ms. It starts from one end of the air-strip, accelerates to a designated speed 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[4]. Fig. 4 shows a plot of the RSSI as a function of the distance of the mobile sender from the receiver.

An immediate and interesting observation to note in Fig. 4 is the pattern of variation in the RSSI as we get closer to the stationary node, for all mobile speeds. Prior to the study, we did not anticipate such a specific pattern since previous mea-

[3]We intentionally describe these experiments here, and not in a later section, since the results of these experiments were used to drive our design in the first place.

[4]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 mote's timestamp and its physical position.

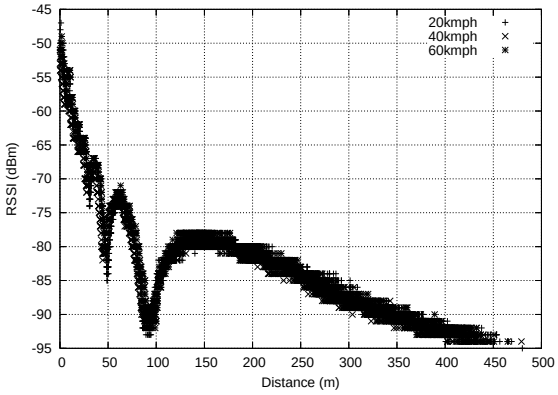

Figure 4. RSSI vs. distance betn. sender & receiver

surement studies have not really reported any such observation [15, 16, 11]. 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 & 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.

We observe from Fig. 4 that we start to receive packets when the mobile is as far away as 450m, and this is more or less independent of the mobile's speed. The RSSI 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, with a threshold of $-80dBm$, as described earlier, we can have link ranges as high as 150-200m.

For the same set of experimental runs, Fig. 5 shows plots of the error-rate versus distance. The error-rate is measured over time windows of 5 packets. To determine D_d, as discussed earlier, we look for the point where the error rate falls to about 20%. From Fig. 5, we find that D_d is about 400m. This too is irrespective of the mobile speed.

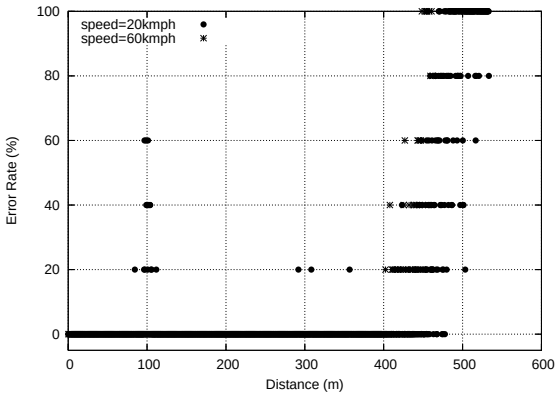

Figure 5. Error rate vs. distance betn. sender & receiver

In Fig. 5, we see that for mobile speed of 20kmph, we see some packet errors at about 100m. This is because of the RSSI dips we explained in Fig. 4. We note that the 60kmph line does not show such packet errors at this distance. This is because at this higher speed, the mobile quickly moves away from the region of the RSSI dip. We observed similar behaviour for higher speeds of 70kmph and 80kmph too.

During all these 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, and the range is likely more than 800m. In this experiment too, we saw no significant effect of the mobile's speed on this range.

What this likely implies is that we can further improve D_d, 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 train's node, we can have a range of the order of 800m or more. (Note that the additional power consumption at the train's node is not a concern).

To summarize the above measurements, when the train is coming at a speed of 80 Kmph, and with $D_d = 800m$, we have $T_{dc} = 36s$.

4.2 Frontier nodes

One other mechanism we propose to further increase T_{dc} is the use of *frontier* nodes. Frontier nodes are essentially nodes placed upstream of the sensor network (upstream with respect to the direction of the train). These nodes do not participate in any data collection, but only serve to detect the oncoming train much earlier.

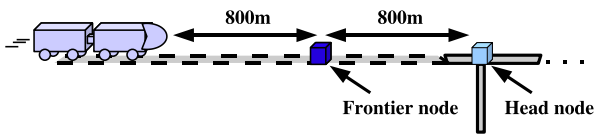

Figure 6. Using frontier nodes to increase T_{dc}

An example in Fig. 6 illustrates the use of frontier nodes. T_{dc} is effectively doubled. Note that depending on the timing, it could be the case that the head node directly learns of train arrival, instead of the frontier node telling it.

A relevant alternative to consider here, to extend network lifetime, is to simply have additional battery installed at each of the nodes instead of having additional infrastructure in terms of a frontier node. Note however that adding a frontier node improves the battery life of *all* nodes uniformly by decreasing the duty cycle, and hence is likely more beneficial.

It is possible to extend the concept of frontier nodes to have more than one frontier node to detect the oncoming train even further earlier. But the incremental benefit of each frontier node would be lesser. Further, each frontier node also adds additional maintenance issues. In practice we expect not more than 1-2 frontier nodes to be used.

We now move on to the issue of time synchronization.

5 Time Synchronization

The next important aspect we look at in BriMon design is the time synchronization. This aspect is related close to the periodic sleep/wake-up and event detection: two of the parameters in event detection, T_{pc} and T_{Δ}, are both related to time synchronization, as we explain now.

There are two separate questions here: *how* to do time synchronization (i.e. the time-sync protocol), and *when* the protocol should be run. We first focus on the protocol.

5.1 How to do time synchronization?

When an 802.15.4 node A sends a message to node B, it is possible for B to synchronize its clock to that of A. This is explained for the Tmote platform in [18]. Thus intuitively it is possible for the entire network to synchronize itself to the head node's clock when a message goes (possibly over multiple hops) from the head node to the other nodes.

Now, in our *command* issue from the head node (Sec. 4) too, the message exchanges involved are exactly the same: a message has to go from the head node to the other nodes. In fact, the same protocol message sequence can be used for synchronization as well as for command issue. Only the content of the messages need to be different: for synchronization we would carry time-stamps, and for command issue, an appropriate byte to be interpreted at the receivers. In fact, the same set of messages can carry both contents (piggybacking one functionality on the other).

Our goal in designing this message sequence for time synchronization and/or command issue is to minimize T_{pc} since this directly translates to a lower duty cycle. We following the guiding principle of optimizing for the common case. The common case in our setting is that packet losses are rare, due to the stable link quality, as explained in Sec. 3.

The protocol consists of simple steps in flooding, and builds on the knowledge of the current routing tree (provided by the routing layer). (1) The head node sends a command/sync message to its children in a broadcast packet. (2) When a node receives a command/sync message, if it has children in the routing tree, it forwards it on.

We have made an important design choice above: there are no acknowledgments in the protocol. Instead, we simply use multiple retransmissions (say, 2 or 3) for each message. We design it this way for several reasons.

First, ACKs and timeouts are likely to increase the overall delay, especially when a node has to wait for ACKs from multiple children. On the other hand, the retransmissions can be sent quickly. Second, since in our network we design such that the links are of good quality, it is appropriate to treat packet losses as corner cases. Third, for the command message, absolute reliability is not necessary. If once in an odd while a train's vibration measurement is missed, it is alright in our application. And last, as an added advantage, just sending a message is much easier to design and implement than having to deal with ACKs and timeouts and the resulting corner cases in the periodic sleep/wake-up mechanism.

The next design choice in the flooding is what exactly is the MAC scheme each node follows while transmitting. Here the choice is quite non-obvious, as we found out the hard way. We initially used the straightforward mechanism where each node uses carrier-sensing before transmitting any message (i.e. CSMA/CA). However, we found that this did not quite work well, even in test cases involving networks of just six nodes. We found that there were several instances where all of the retransmissions[5] were getting lost, and as a result, nodes not successfully synchronizing (or receiving commands).

There was no significant wireless channel errors, so that could not be the reason for the packet losses. We ruled out the possibility of wireless hidden nodes too: such packet losses occurred even when all the nodes were within range of each other. As we delved deeper into the possible reason, the answer surprised us: the packet losses were happening at the receiver's radio buffer! That is, lack of flow-control was a significant issue.

The issue of flow control

To gain an in-depth understanding of the various delays in the system during transmission & reception, we conducted the following experiment. We sent a sequence of packets, of a pre-determined size, from one mote to the other. We had sufficient gap (over 100 ms) between successive packets to ensure that no queuing delays figure in our measurements. We also disabled all random backoffs. We recorded time-stamps for various events corresponding to each packet's transmission as well as reception. These events, termed $S_1, \ldots, S_6$ at the sender and $R_1, \ldots R_5$ at the receiver, are listed in Tab. 1.

Events at sender side	
Event	**Description**
S1	Send command issued at application layer
S2	Start of data transfer from micro-controller to radio, over SPI bus
S3	End of data transfer over SPI bus
S4	SFD start (first few bytes of pkt sent over air)
S5	Tx of pkt over radio done
S6	SendDone event received at application layer
Events at receiver side	
Event	**Description**
R1	Received SFD interrupt (first few bytes of pkt recd. over air)
R2	Interrupt on full pkt reception
R3	Start of data transfer from radio to micro-controller, over SPI bus
R4	End of data transfer over SPI bus
R5	ReceiveMsg event at application layer

Table 1. Events recorded to measure various delays

For an event S_i or R_i, denote the time-stamp as $t(S_i)$ or $t(R_i)$ respectively. Tab. 2 tabulates the various delays corresponding to these events. The different rows in Tab. 2 correspond to experiments with packet sizes of 44, 66, and 88 bytes (including header overheads) respectively. The delay values in the table are the averages over several hundred packets; the variance was small and hence we omit it. Given the average delay value, we then calculate the speed of SPI/radio transfer; these values are also shown in the table.

We note that the radio speed is close to the expected 250Kbps at both the sender and the receiver, for all packet sizes. (It is slightly higher than 250Kbps at the sender side, because our measurement of the $[t(S_5) - t(S_4)]$ delay

[5] We used 3 transmissions: 1 original + 2 retransmissions in our implementation.

	Sender side						Receiver side					
Packet size (bytes)	t(S3)-t(S2): SPI tx delay (ms)	SPI tx speed (kbps)	t(S5)-t(S4): Radio tx delay (ms)	Radio tx speed (kbps)	[t(S2)-t(S1)] + [t(S4)-t(S3)] + [t(S6)-t(S5)] = Other delays (ms)	Total delay (ms)	t(R2)-t(R1): Radio rx delay (ms)	Radio rx speed (kbps)	t(R4)-t(R3): SPI rx delay (ms)	SPI rx speed (kbps)	[t(R3)-t(R2)] + [t(R5)-t(R4)] = Other delays (ms)	Total delay (ms)
44	1.28	275.46	1.40	250.75	1.44	4.12	1.47	239.13	2.29	153.73	0.43	4.19
66	1.74	303.71	2.10	250.89	1.46	5.30	2.17	242.65	3.36	157.78	0.44	5.97
88	2.14	329.59	2.81	250.89	1.40	6.35	2.88	244.56	4.40	160.18	0.44	7.72

Table 2. SPI, Radio delays in tx/rx

marginally underestimates the actual delay on air.) However, a significant aspect we note is that the SPI speed at the receiver is much lower, only about 160kbps, which is much slower than the radio! This means that packets could queue up at the radio, before getting to the processor. To make matters worse, the CC2420 chip used in the Tmote hardware which we used, has an on-chip receive buffer of just 128 bytes [19].

Furthermore, since the time-sync packet sent by a node to its children is *broadcast*, implementing filtering at the radio is not an option (the TinyOS 2.0 software we have used does not yet implement filtering at the radio even for unicast packets). This means that all packets have to come to the micro-controller, even those not destined for this node. For instance, a node's sibling's broadcasts, its parent's sibling's broadcasts, may all reach the micro-controller! All these put together mean that flow-control at the link layer is an issue in this platform.

In Tab. 2, a few other aspects we note are the following. At the sender side, the SPI bus shows slightly higher speed than the radio; and this SPI speed increases with packet size, which suggests a fixed overhead for each such transfer. We also note that there is an almost constant overhead for each packet due to software processing: about 1.45ms at the sender side, and 0.45ms at the receiver side. The table also shows the total average per-packet time taken at the sender & receiver. We can see that the total delay can be as high as over 2.5 times the over-the-air radio delay.

The experiment above essentially means that flow-control is required. But this is absent in a CSMA/CA based flooding mechanism. This explains the packet losses we observed. It is worth noting that other time synchronizing approaches such as FTSP [13] have not reported such a problem with CSMA/CA, because they do not have synchronization messages sent back-to-back. In contrast, we need to send the synchronization messages as quickly as possible, to minimize T_w, and thus achieve very low duty-cycling.

TDMA-based flooding

The issue of flow-control arises essentially due to the use of a radio which is faster than the processor's (or bus's) capabilities. To circumvent this issue, we use the approach of having a TDMA-based flooding mechanism. The idea is to have the head node come up with a schedule, based upon which each node in the network will transmit. The schedule ensures that only one node in the network transmits at a time, and that the time-slot duration for each packet is sufficient for successful reception (including all delays listed in Tab. 1).

The head node can embed the schedule information too in the time-sync (or command) messages. This works as follows. As mentioned in Sec. 3, the time-sync mechanism assumes that we already have a routing tree. The routing protocol will in fact ensure that the entire tree information is known at the head. The head then computes the schedule using a depth-first traversal of the routing tree. For instance, in Fig. 7, one possible schedule is 1, 5, 9, 10, 3, 4. Note that only non-leaves have to transmit. Also, note that the schedule is arranged so that each node gets to know both the synchronization information as well as its slot assignment before it has to transmit its synchronization packet.

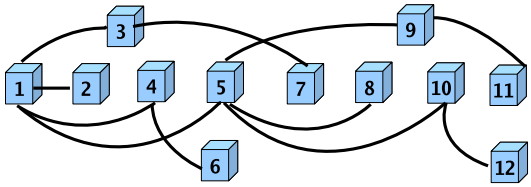

Figure 7. Test network used for measuring T_{pc}

In the above mechanism, we have consciously chosen a centralized approach. Given the size of the network we expect (up to 12 nodes), the fact that the schedule is centrally computed and conveyed in a packet is not an issue. (In fact, the O(n) DFS computation, and having a one-byte slot information per node, can accommodate larger networks too, of say 100-200 nodes).

Measuring T_{pc}

We have implemented the protocol on our Tmote sky platform. We used 3 retransmissions from each node, to be on the safer side. We determine the slot size as follows. A slot needs to accommodate 3 packets. And each packet involves the time taken for reception. The total packet size was 42 bytes, which has a total delay at the receiver, of about 4ms. So we used a slot time of 12ms. This resulted in negligible packet losses.

A slot time of 12ms gives $T_{pc} = 12 \times 6 = 72ms$, for the test network of 12 nodes shown in Fig. 7. This is because, in this network, the six non-leaf nodes need to transmit one after the other. We observed this value of T_{pc} experimentally too, in our prototype implementation. Recall that we expect a data-span to have at most 12 nodes, and not more than 3-4 hops. Our test network is designed to resemble a data-span with two physical spans. In the test, all of the 12 nodes were placed within range of each other, but we artificially imposed a routing tree on them. For a six-node network, we can expect T_{pc} to be much lesser, about $36ms$.

5.2 When to do time synchronization?

We now turn to the question of *when* we should run the above time-sync/command protocol. The command to collect data is issued only when a train is detected. With respect to the synchronization protocol, there is the question of how often we should synchronize. For our CC2420 platform, the answer

is simple. We just run the synchronization mechanism during each wake-up period. This does not incur any additional overhead since anyway all nodes are awake for $T_w = T_\Delta + T_{pc}$, in expectation of a potential command from the head node. And if nodes are awake and listening, the power consumption in the CC2420 chips is the same as (in fact slightly higher than) that of transmitting [20].

We can now estimate T_Δ too. It is the sum of the possible clock drift in one check cycle (T_{cc}), and T_{err}, the error in the synchronization mechanism. We estimated $T_{dc} = 36s$ in Sec. 4.1. Since $T_{cc} < T_{dc}$, the worst case T_{drift} can be estimated as $20 \times 10^{-6} \times 36s = 0.72ms$. Here we have used a worst case clock drift rate of 20ppm [14].

In the same experiment above, where we estimated T_{pc}, we also measured the worst case error in our synchronization to be at most 5-6 clock ticks over the 3-hop, 12-node network. This is about $6 \times 30.5\mu s \simeq 0.18ms$. The overall T_Δ is thus about $0.9ms$. It is worth noting that this is much smaller than T_{pc}.

In our prototype implementation, we have also tested (in-lab) that the time-sync and periodic sleep/wake-up mechanism indeed work stably: we have tested for several hours (close to 1 day) at a stretch.

6 Routing

We have noted above that both the event detection (command) mechanism and the time-sync protocol depend on the existence of a routing tree rooted at the head node. The crux of our routing mechanism is the fact that we use stable links. That is, unlike in [16], we do not have a situation where we have to distinguish between links of error rates in-between 0% and 100%. In fact, the RSSI and LQI variability measurements in [11] suggest that trying to make such distinctions in a dynamically varying metric can produce unstable behaviour (in the 802.15.4 platform).

Routing phases: In designing the routing, we make the design decision of using a centralized routing approach, with the head node controlling decisions. We have the following simple steps in routing. (1) *Neighbour-discovery phase:* The head node initiates this phase, by periodically transmitting a HELLO. Nodes which hear the HELLO in turn periodically transmit a HELLO themselves. After some time, each node learns the average RSSI with its neighbours, which we term the link-state. (2) *Tree construction phase:* The head node now starts constructing the routing tree. The construction goes through several stages, with each stage expanding the tree by one hop. To begin with, the root node knows its own link-state, using which it can decide its one-hop neighbours. Now it conveys to each of these chosen one-hop neighbours that they are part of the network. It also queries each of them for their link-state. Once it learns their link-state, it now has enough information to form the next hop in the network. And this process repeats until all possible nodes have been included in the network. In each link-state, based on the RSSI threshold, as described in Sec. 3, links are classified as good or bad. The root first seeks to extend the network using good links only. If this were not possible, it seeks to extend the network using bad links.

Although simple, the above mechanism has two properties essential for us. (1) The head node *knows the routing tree* at the end of the two phases. This is essential for our time synchronization and command mechanisms (e.g. to send a command to the nodes in the network to start collecting data, after detecting an oncoming train). It is also essential when it is time for the head node to gather all data from the nodes in the network, before transferring it to the train. (2) More importantly, the head node *knows when the routing protocol has ended operation*. We stress that this is a property not present in any distributed routing protocol in the literature, to our knowledge. And this property is very essential for power efficient operation: once the head node knows that routing has ended, it can then initiate duty cycling in the network. Such interfacing between routing and duty cycling is, we believe, an aspect which has not really been looked at in-depth in prior work.

When to run the routing protocol? Like with the time-sync protocol, we also need to answer the question of *when* to run the routing protocol. The routing protocol can be run infrequently, or whenever a failure is detected. Now, how to detect failures in the network? A node can detect itself to be disconnected from the current routing tree, if it fails to receive the time synchronization messages for a certain timeout. It can then cease its duty cycling, and announce that it has been 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.

We wish to stress that such a laid-back approach to fixing failures is possible because we are relying upon stable links. We also once again stress that scaling is not an issue since we consider each data-span to be an independent network.

In order to build tolerance to failure of the head node, one can provision an extra head node. An alternative would be to have one of the other nodes detect the head node's failure (say, on failure to receive synchronization messages for a certain timeout period), and take-over the head's functionality.

Benefits of a centralized approach: A centralized approach has several benefits, apart from the simplicity in design and implementation. For example, recall from Sec. 5 that T_{pc} is proportional to the number of non-leaf nodes in the network. So we need to minimize the number of non-leaf 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.

Routing protocol delay: Since we expect to run the routing protocol only infrequently, the delay it incurs is not that crucial. But all the same, we wish to note that in our prototype implementation, the routing phases take only about 1-2s, for the test network shown in Fig 7. This is much smaller compared to say, the duration of data collection (40s) or data transfer (Sec. 7).

7 Mobile Data Transfer

We now discuss the last important component of BriMon. The event detection mechanism triggers data collection at the nodes. After the data *collection* phase, this data must reliably be *transferred* from the (remote) bridge location to a central server that can evaluate the health of the bridge. Most sen-

sor network deployments today do this by first *gathering* all collected data to a sink node in the network. The data is then transferred using some wide-area connectivity technology to a central server [2, 3, 4, 5, 6, 7].

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

A more important reason is the following. Recall that we can have bridges as long as 2km, with spans of length 30-125m. This means that overall we could have as many as about 200 sensors placed on the bridge, at different spans. Even if we were to have somehow have a long-distance Internet link at the bridge location, we would have to *gather* all the sensor data corresponding to these many sensors at a common sink, which has the external connectivity. Such gathering has to be reliable as well.

The total data which has to be gathered would be substantial: $57.6Kb \times 200 \simeq 1.44MB$ for each measurement cycle. Doing such data gathering over 10-20 hops will involve a huge amount of transfer time and hence considerable energy wastage. Having a large network has other scaling issues as well: scaling of the routing, synchronization, periodic wake-up, command, etc.. Large networks are also more likely to have more fault-tolerance related issues.

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

One subtle detail to note here is that we seek to transfer the data collected for one particular train, not to the same train, but to a subsequent train. In fact on the Tmote platform, it is not possible to use the radio in parallel with the data collected being written to the flash since there is a common bus. We then need to have different trains for data collection and the data transfer. The additional delay introduced in such an approach is immaterial for our application.

The transport protocol itself for the data transfer can be quite simple. In BriMon, we use a block transfer protocol with NACKs for the data transfer from the head node to the train. We use a similar protocol, implemented in a hop-by-hop fashion, for the data gathering as well: that is, for getting the data from the individual nodes to the head node.

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

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

Reserved channel for event detection: The above approach works except for another subtle detail. We designate that the channel for event detection mechanism for all data-spans is the same, and reserve one channel for this purpose. So as the train approaches, the radio within it can continuously beacon on this reserved channel without having to bother about interfering with any data transfer or other protocol operation within each data span. The head nodes of each span listen on this channel for the oncoming train, and switch to the allocated channel for the data-span after the T_{det} duration[6].

Throughput issues: Another challenge to address in our mobile data transfer mechanism is whether such transfer is possible with sufficient throughput. The amount of data that can be transferred is a function of the contact duration of the train with the mote, which in turn is a function of the speed of the train and the antennae in use.

In order to determine the amount of data that can be transferred using our hardware platform, we have conducted experiments with a prototype. We have implemented the NACK-based reliable block transfer protocol. Although simple conceptually, the protocol involves some subtleties in implementation. We need to transfer blocks of data from the flash of the sender to the flash of the receiver. As mentioned earlier, in the Tmote platform, we cannot perform flash read/write simultaneously with radio send/receive because of a shared bus. However, our protocol does parallelize flash write at the receiver, with flash read of the next block at the sender.

In our implementation, we use blocks of size 2240 bytes; this is significant chunk of the 10KB RAM in the MSP430 chip of Tmote sky. And we have used packets of payload 116 bytes. So a block fits in 20 packets. In our data transfer mechanism, the sender simply uses a *pause* between successive packets to implement flow control. We compute the pause duration as the excess total delay at the receiver side as compared to the sender side. Using an extrapolation of Tab. 2, we calculate the required pause duration for packets of payload 116 bytes (total 126 bytes) to be about 3ms (sender side delay: 8ms, receiver side delay: 11ms). We use a value of 4ms,

[6]Such channel change takes only a few hundred micro-sec on the CC2420 chips.

as a safety margin.

To get an estimate of the various delays involved in the protocol, we first ran a data transfer experiment using an overall file size of 18 blocks. The total time taken was 6,926ms. This consists of the following components. (1) A flash read-time for each block of about 100ms. (2) A flash write-time for each block of about 70ms. This is overlapped with the flash read-time for the next block, except of course for the last block. (3) The transmission + pause time for each packet is about 12, resulting in an overall transmission + pause time of $20 \times 12 = 240ms$ per block. So we expect a delay of $18 \times 100ms + 1 \times 70ms + 18 \times 240ms = 6190ms$, which is close to the experimentally observed delay of $6,926ms$.

The above experiment thus gives an effective throughput of $2,240 \times 18 \times 8/6926 = 46.6Kbps$. Note that this value is much lower than the $250Kbps$ allowed by the 802.15.4 radio, due to the various inefficiencies: (a) various header overheads, (b) the shared bus bottleneck mentioned above due to which flash read/write cannot be in parallel with radio operation, and (c) the use of a pause timer to address the flow-control issue.

Mobile data transfer experiment: To see if we are able to achieve similar throughput under a realistic scenario, we conducted the following experiment. The setup mimics the situation where the header node (root node) of a cluster in BriMon uploads the total data collected by all the nodes within its cluster on to mobile node on a train. The data transfer is initiated when the mobile node (on the arriving train) requests the head node to transfer the collected data. The head node then reads the data stored as files from flash and uploads them one by one using the reliable transport protocol.

In this experiment, the head node was made to transfer 18 blocks of data, each of 2240 bytes. The total data transfer was thus $18 \times 2240 = 40,320B$. The head node as well as mobile node were equipped with external 8dBi omni antennas, much like in Sec. 4.1. The antenna of the head node was mounted at a height slightly over 2m. The mobile node was fixed on a vehicle and the antenna was at an effective height of slightly less than 2m.

The head node (stationary) with the complete data of 40,320B in its flash, is initially idle. The mobile node is turned off until it is taken out of the expected range from the head node, and then turned on. The experiments in Sec. 4.1 show that the range can be around 400m. So we started the mobile node at a distance of 500m from root node. The vehicle is made to attain a constant known velocity at this point of 500m, while coming towards the head node. On booting, the mobile node starts sending the request for data, every 100ms, until it receives a response from the head node. The head node, on receiving the request, uploads the data block by block, using the reliable transport protocol. The transport layer at both nodes takes a log of events like requests, ACKs, NACKs and retransmissions, for later analysis.

Fig. 8 shows a plot of the block sequence number received at the mobile node, versus the position of the mobile with respect to the head node. We first note that although we conservatively estimated the contact range to be 400m, the data transfer begins right at about 490m, irrespective of the mobile's speed. We next note that after the mobile is within 450m from the head node, the rate of data transfer (slope of the line)

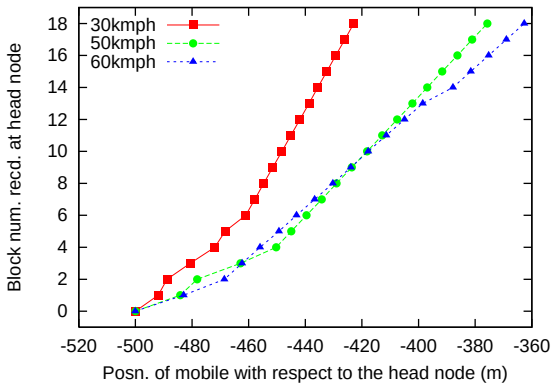

Figure 8. Mobile data transfer measurement

is more or less constant. We have calculated this slope to correspond to the same data rate (about 46Kbps) as in the stationary throughput test. The regions in each graph where the slope is different from this rate correspond to instances where a NACK was sent since some of the packets in a block were lost.

Feasibility of mobile data transfer: Assuming the throughput of about $46Kbps$ which we have been able to achieve, it means that if we have $691.2Kb$ per data-span, as estimated in Sec. 2, we need a contact duration of $691.2/46 \simeq 15s$. This is achievable with a contact range of about 330m for a train speed of up to 80kmph. Or, with a contact range of about 250m for a train speed of 60kmph.

Note from our discussions in Sec. 4 that in the worst case, the head node detects the oncoming train only just before the train passes over the span. Combining this observation with the fact that data transfer starts right from about 490m (Fig. 8), we have sufficient contact range, larger than the 330m requirement estimated above. So we can conclude that our data transfer is achievable, with significant leeway. Our analysis above has in fact been a worst-case analysis. For instance, the leeway would be much higher if we have only a 6-node network, or if we use frontier nodes.

There are also other possibilities to further improve the effective data transfer throughput. One obvious performance enhancement is the use of data compression. Another possibility is use different hardware. We could use a mote which allows simultaneous operation of the flash and the radio. Or we could even use a completely different radio, say for example the Bluetooth enabled Intel motes, as considered in [5]. These possibilities could give higher throughput than what we have been able to achieve with 802.15.4 in our prototype.

Another possibility to increase the effective amount of data which can be transferred to a train is to employ the following trick. We could have one data transfer to a receiver in the front coach of the train, and another in a rear coach sufficiently far apart from the front coach. This is feasible since trains are often about 1km or more long.

8 Related Work

Prior to BriMon, several projects have looked at the issue of automated structural health monitoring. The work in [21, 7] uses MEMS-based sensors and Mica2/MicaZ motes to study vibrations in buildings. It focuses on data compression tech-

niques, reliable data transfer protocol, and time synchronization. The work in [6, 8] has looked at bridge monitoring in-depth. They have presented extensive studies of hardware, the required data sampling rate, and data analysis.

BriMon builds on this body of prior work in structural monitoring, and the techniques of data compression, data transfer protocol, data analysis, etc are complementary to our contributions. The novel aspects in our work are the event detection, mobile data transfer, as well as the integration of these aspects with low duty cycling. These have not been considered in earlier work.

Low duty cycling by itself is not novel by any means. B-MAC [22], SCP-MAC [20] and AppSleep [23] are MAC protocols to achieve low duty cycling. Since these protocols have been designed without any *specific* application in mind, they are necessarily generic. For instance, SCP-MAC uses complex schedule exchange mechanisms with neighbouring nodes. This is designed for an arbitrary traffic pattern, and hence does not apply (optimally) in our setting.

Similarly, mobile data transfer too is not novel by itself. The ZebraNet [9] and DakNet [10] projects too have used similar strategies. There is a growing body of literature in this context for 802.11 (WiFi) [24, 25]. With respect to 802.15.4 too, [26] presents some preliminary measurements indicating that mobile data transfer in 802.15.4 is feasible, and that the throughput is independent of the speed. Our measurements in Sec. 4.1 are in broad agreement with these.

In contrast to the above work on low duty cycling or mobile data transfer, our primary goal is to integrate the requisite functionalities for a specific application. BriMon integrates vertically with *all* aspects relevant to bridge monitoring. It uses *only* the necessary set of mechanisms, thus simplifying the design. We use extensive application information and cross-layer optimizations across the four functionalities of event detection, mobile data transfer, time synchronization, and routing. To our knowledge, we are the first to have carefully looked at the interaction between such network protocol functionalities.

9 Discussion

We now present several points of discussion around the system design described so far.

Lifetime estimation: It is useful to get an idea of how well the system we have designed is able to achieve our goal of minimal maintenance. For this, we estimate the time for which a set of batteries will last, before requiring replacement. Consider the sequence of events depicted in Fig. 9. We have a data collection phase, followed by a data gathering phase (data goes from each node to the head node). Then when the next train arrives, the data is transferred to the moving train.

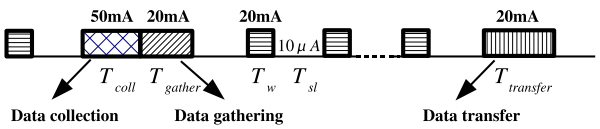

Figure 9. Estimating node lifetime in BriMon

We assume that we use a battery, or a series of batteries of sufficient voltage, say 6V (at least 3V required for the Tmote sky motes, and at least 5V for the accelerometers). For example, this can be achieved by using 4 AA batteries in series. The various current values shown are rough estimates derived from the data specification sheets of the accelerometer and Tmote sky respectively. We also verified many of these values using a multimeter in our lab.

In Fig. 9, we can estimate T_{coll}, T_{gather}, and $T_{transfer}$ as follows. T_{coll} starts when the train is detected and extends until 20 sec after it has crossed the span. Hence it depends on when exactly the train is detected, train speed, the train's length, and the span length. Assuming the worst case when the train is detected very early (T_{dc} before it enters the span), and assuming train speed to be 60kmph, train length to be 1km, and data-span length to be 250m, we have:

$T_{coll} = 36s + (250m + 1km)/(60kmph) + 20 = 131s$

We assume that once collected, the data is truncated to only the last 40s of data (which is of interest).

Then we estimate T_{gather} to be the time it takes for this data to be gathered at the head node. Now, pushing 40s worth of collected data over one hop toward the root takes $57.6Kb/46Kbps = 1.25s$. In Fig. 7, there is one head node, 4 nodes one-hop away, 5 nodes two-hops away, and 2 nodes three-hops away. Thus if we transfer the collected data hop-by-hop, one after another, the total time taken would for this would be $(1\times 0 + 4\times 1 + 5\times 2 + 2\times 3)\times 1.25s = 32.5s$. For $T_{transfer}$, we use the value of 15*s*, as estimated in Sec. 7.

Now, using $T_w = T_{pc} + 2T_\Delta + T_{det}$ will work for both head nodes as well as non-head nodes (see Sec. 4). Recall that $T_{pc} = 72ms$ (for a 12-node network), $T_\Delta \simeq 1ms$, and $T_{det} = 5T_b = 50ms$ for an inter-beacon period of $T_b = 10ms$. We take $T_w = 125ms$ as an upper bound on the above estimate.

Assuming that we have data collection once per day[7]. There are thus at most $1day/T_{cc} \simeq 2400$ durations of T_w and T_{sl}. So the total energy drawn, expressed in $mA \times sec$, at 6V, can be estimated as:

$T_{coll} \times 50mA + T_{gather} \times 20mA + T_{transfer} \times 20mA + 2400 \times T_w \times 20mA + 2400 \times T_{sl} \times 10\mu A$
$= 6550\ (collect) + 650\ (gather) + 300\ (transfer) + 6000\ (wakeup) + 864\ (sleep) mAsec$
$= 14364 mAsec \simeq 4mAh$

The AA batteries have about 2500mAh capacity. Hence we can expect that the lifetime in this setting would be about $2500/4 \simeq 625days$, which is over 1.5 years.

We note that in the above equation, the main components to the overall energy consumption are the data collection and the periodic wake-up. In fact, if we have further infrequent data collection, say once a week, in the above estimation the periodic wake-up will constitute an even larger fraction of the power consumption. So it was indeed useful that we sought to minimize the wake-up duration in our design!

Measurements on a bridge: Most of the experiments we have presented above have used an air-strip (or have been performed in lab). This was convenient since the air-strip was nearby. However, we also tested our prototype on a road

[7] Measuring once a day is sufficient for long-term bridge health monitoring. Also note that we need not collect data for every passing train; we can easily build logic to look only for trains with certain ids in their beacons.

bridge on a nearby river[8]. We had two ADXL 203 accelerometer modules integrated into our system at that time. Prior to our trip to the bridge, we thoroughly tested the accelerometer modules in-lab, using calibrated shake-tables. At the bridge, we used two motes for data collection, and a separate sink (head) node.

We were successfully able to measure the vibrations on the bridge induced due to passing traffic. We observed a dominant free vibration frequency of about 5.5Hz. The amplitude of forced vibration we observed was as high as 100 milli g (vertical). For healthy bridge spans, the expected amplitude is about 30 milli g. So our measurement indicates the need for maintenance operations on that span.

The above measurement on the bridge, as well as the prototype implementations of the various functional components, gives us a measure of confidence in the overall design.

Wider applicability: We have designed BriMon specifically for railway bridge monitoring. This environment is particularly challenging since most bridges are away from an urban environment (poor wide area network connectivity). More importantly, train traffic is sporadic and unpredictable. On the other hand, road bridges are likely to have constant and/or predictable (time-of-day related) traffic patterns. However, even in such scenarios, we believe that many of our mechanisms are likely to find applicability. For instance, we use our event detection mechanism to trigger data transfer, to a designated mobile node. This would be applicable in road bridges too. Also applicable would be the consideration of multiple channels, splitting up the set of nodes into multiple independent networks, and the resulting architecture with the time synchronization and routing functionalities integrated.

Apart from structural monitoring, our event triggering and data transfer approaches are also likely to find applicability in, say road-side pollution monitoring. MEMS-based sensors are currently still evolving in this domain [27].

Ongoing and future work: Admittedly, many aspects of our work require further consideration. While the prototype implementation and detailed experimentation have given us a measure of confidence in our design, actual deployment on a railway bridge is likely to reveal further issues. Related to the issue of fault-tolerance, while we have outlined a possible approach to deal with head node (see Sec. 6), we have left optimizations in this regard for future consideration.

On application specific versus generic design: BriMon is a case study in application specific design. We have consciously taken this approach. Our methodology has been to start from scratch, and pay full attention to the application requirements, and design the required mechanisms. And importantly, we design *only* the required mechanisms, thus keeping the design relatively simple and, arguably, optimal.

Literature in protocol design for sensor networks is abundant. We have not tried to retro-fit any protocol designed in a generic fashion into BriMon. Our approaches to time synchronization and routing are two cases in point.

Although we have not sought to generalize our solutions at this stage, this is not to say that generality is impractical. But we believe that in complex systems, generality emerges from in-depth studies of specific systems.

On layered versus integrated design: Along the same vein as above, we have also paid little heed to traditional protocol layering. Cross layer interactions are well known in wireless systems in general. We have intentionally taken an integrated approach. Once again, this is not to devalue the benefits of layering. But we believe that the right protocol layering and interfacing will emerge from specific in-depth sensor network system designs such as ours. Hence we have not tried to retro-fit the layering which currently exists for wired networks, in our system.

[8]The logistics for doing the same on a railway bridge are more involved.

10 Conclusion

This paper presents the design of BriMon, a wireless sensor network based system for long term health monitoring of railway bridges. The paradigm we have followed is that of application specific design. We believe that this is the right way to understand the complex interaction of protocols in this domain.

In the design of BriMon, we identify the requisite set of functionalities from the application's perspective, and we propose mechanisms to achieve these. We build on several aspects of prior work in automated structural monitoring. Our novel contributions are three fold: (1) an event detection mechanism which enables low duty cycling, (2) a mobile data transfer mechanism, and (3) the interfacing of these two mechanisms with the time synchronization and routing functionalities.

Our design choices have been based on application requirements as well as on several measurement studies using prototype implementations. Based on preliminary measurements, we estimate that our current design should be deployable with minimum maintenance requirements: with the battery lasting for over 1.5 years.

11 References

[1] George Iype. Weak, distressed, accident-prone. `http://www.rediff.com/news/2001/jun/25spec.htm`, 25 Jun 2001. The Rediff Special.

[2] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[3] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A Macroscope in the Redwoods. In *SenSys*, Nov 2005.

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

[5] Lakshman Krishnamurthy, Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Nandakishore Kushalnagar, Lama Nachman, and Mark Yarvis. Design and Deployment of Industrial Sensor Networks: Experi-

ences from a Semiconductor Plant and the North Sea. In *SenSys*, Nov 2005.

[6] Sukun Kim. Wireless Sensor Networks for Structural Health Monitoring. Master's thesis, U.C.Berkeley, 2005.

[7] 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.

[8] Manuel E. Ruiz-Sandoval. *Smart Sensors for Civil Infrastructure Systems*. PhD thesis, University of Notre Dame, Indiana, 2004.

[9] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi. Hardware Design Experiences in ZebraNet. In *SenSys*, Nov 2004.

[10] Alex Pentland, Richard Fletcher, and Amir Hasson. DakNet: Rethinking Connectivity in Developing Nations. *IEEE Computer*, Jan 2004.

[11] 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*, Sep 2006.

[12] Exploration of Sensor Network Field deployment on a large Highway Bridge and condition assessment. `http://healthmonitoring.ucsd.edu/documentation/public/Vincent_Thomas_Testing.pdf`.

[13] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. The Flooding Time Synchronization Protocol. In *SenSys*, 2004.

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

[15] Jerry Zhao and Ramesh Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks. In *SenSys*, Nov 2003.

[16] Alec Woo, Terence Tong, and David Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *SenSys*, Nov 2003.

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

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

[20] Wei Ye, Fabio Silva, , and John Heidemann. Ultra-Low Duty Cycle MAC with Scheduled Channel Polling. In *SenSys*, 2006.

[21] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A Wireless Sensor Network For Structural Monitoring. In *SenSys*, Nov 2004.

[22] Joseph Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *SenSys*, Nov 2004.

[23] Nithya Ramanathan, Mark Yarvis, Jasmeet Chhabra, Nandakishore Kushalnagar, Lakshman Krishnamurthy, and Deborah Estrin. A Stream-Oriented Power Management Protocol for Low Duty Cycle Sensor Network Applications. In *EmNetS-II*, May 2005.

[24] Vladimir Bychkovsky, Bret Hull, Allen Miu, Hari Balakrishnan, and Samuel Madden. A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks. In *MOBICOM*, Sep 2006.

[25] Richard Gass, James Scott, and Christophe Diot. Measurements of In-Motion 802.11 Networking. In *WMCSA*, Apr 2006.

[26] Michael I. Brownfield and Nathaniel J. Davis IV. Symbiotic Highway Sensor Network. In *Vehicular Technology Conference*, Sep 2005.

[27] Sensor Network Testbed Proposed. `http://www.calit2.net/newsroom/article.php?id=28`. CalIT2 Newsroom.

A Glossary of Terms

D_d distance from span, at which the train can be detected

V the speed of the train

$T_{dc} = D_d/V$ the maximum time between train detection and when the train is on the span (i.e. when data collection is due)

T_{cc} the period of the sleep/wakeup/check cycle

T_{sl} the duration of sleep in T_{cc}

T_w the duration of wakeup in T_{cc}

T_{pc} the time taken by the protocol for command issue (from the head node to the rest of the nodes in the network)

T_{drift} the maximum drift possible between any two nodes

T_{err} the maximum error in clock synchronization

$T_\Delta = T_{err} + T_{drift}$ the maximum clock difference possible between any two nodes

T_{det} the time taken by the head node to detect train arrival, once it is in range

T_b the train beaconing period