Intelligent Telemetry for Freight Trains

Johnathan M. Reason[1], Han Chen[1], Riccardo Crepaldi[2], and Sastry Duri[1]

[1] IBM T. J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
{reason,chenhan,sastry}@us.ibm.com
[2] University of Illinois, Urbana-Champaign
201 N. Goodwin Avenue - M/C 258, Urbana, IL 61801
rcrepal2@illinois.edu

Abstract. Within the North American freight railroad industry, there is currently an effort to enable more intelligent telemetry for freight trains. By enabling greater visibility of their rolling stock, including locomotives and railroad cars, railroad companies hope to improve their asset utilization, operational safety, and business profitability. Different communication and sensing technologies are being explored and one candidate technology is wireless sensor networks (WSN). In this article, we present Sensor-Enabled Ambient-Intelligent Telemetry for Trains (SEAIT), which is a WSN-based approach to supporting sensing and communications for advanced freight transportation scenarios. As part of a proof-of-technology exploration, SEAIT was designed to address key requirements of industry proposed applications. We introduce several of these applications and highlight the challenges, which include high end-to-end reliability over many hops, low-latency delivery of emergency alerts, and accurate identification of train composition. We present the architecture of SEAIT and evaluate it against these requirements using an experimental deployment.

Keywords: freight trains, sensor networks, on-board telemetry, outlier detection.

1 Introduction

Train accidents are large sources of lost revenue for the railroad industry. An undetected mechanical issue may lead to a critical failure, such as a derailed train, which requires expensive on-site repairs. In addition, when a train becomes immobile, part of the railroad system becomes inaccessible, thus creating delays for other trains. While the lost revenue due to delays can be substantial, the loss of optimum train speed and balance throughout the train network also affects equipment utilization and customer confidence levels. In addition to revenue loss because of delays, freight trains carry hazardous materials of various kinds, which are at risk of spilling into the environment during an accident, potentially causing detrimental damage to the environment and further economic loss. Train accidents also have a third dramatic cost: loss of human lives. Each year dozens of lives are lost, and hundreds are injured, because of train accidents, which include crossing collisions, equipment failures, and other operating incidents.

S. Ystad et al. (Eds.): MobiCASE 2009, LNICST 35, pp. 72–91, 2010.

Some of the equipment manufacturers for the railroad industry have conducted studies of advanced technologies, such as remote-controlled locomotives and electronically controlled brakes. These prior works have focused almost exclusively on upgrading specific subsystems with modern technology. This is the conventional approach to improving operating efficiencies. On a parallel track, the railroad industry is also searching for a disruptive technology that will enable more advanced applications that currently are not supported with their existing technology. The conventional wisdom is that greater visibility of the health and status of trains would enable business transformation in numerous areas, including predictive maintenance, schedule optimization, and asset utilization.

In this paper, we present Sensor-Enabled Ambient-Intelligent Telemetry for Trains (SEAIT), which is a WSN-based approach to support sensing and communications for novel and advanced freight railroad applications. Freight trains – which are comprised of un-powered and unwired railroad cars – can exploit the ad hoc wireless networking capabilities of WSN technology to provide timely data about the identity and condition of the cars composing a train. SEAIT (pronounced "see it") was designed and realized as a part of an ongoing feasibility study to discern whether WSN is a viable approach to provide the next-generation sensing, computing, and communications infrastructure for the freight railroad industry.

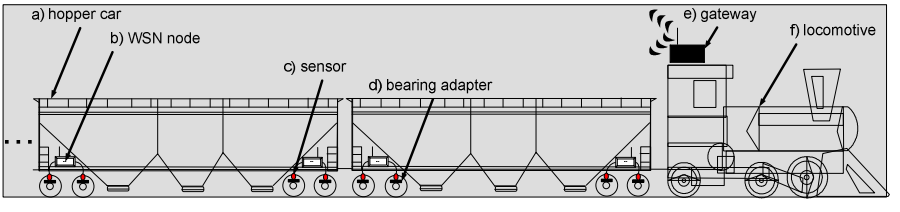

Fig. 1. A freight train hauling hopper cars (a) with WSN nodes (b) attached to the car body and sensors (c) attached to the bearing adapters (d). A gateway (e) is attached to the locomotive (f).

By adding local sensing, computation, and communication at the critical points of each car, SEAIT provides the infrastructure for enabling advanced railroad applications. Fig. 1 represents a possible deployment view of SEAIT on a freight train. On each car, WSN nodes (two per car) are monitoring the temperature of the wheel bearings using a thermocouple connected to the bearing adapter. The WSN nodes form a multi-hop network to communicate data towards the locomotive. The locomotive houses a special node, or gateway, that performs real-time analysis on data and events; thereby yielding faster response time to any outlier conditions (e.g., bearing is too hot). With this in mind, this paper makes the following contributions.

1. Proposes a WSN architecture that offers some novel features and optimizations, including semantics-based wakeups.
2. Proposes novel WSN solutions to key exemplary railroad applications and realizes these solutions using a reference implementation of the architecture.
3. Studies the feasibility of using WSN technology for railroad applications by comparing performance of the realization to key application requirements.

Considering the themes above, the paper is organized as follows. Section 2 describes the more challenging application requirements as exemplified by three advanced railroad applications. Additionally, this section discusses the key characteristics of an onboard WSN for trains. Section 3 describes the architecture of SEAIT and motivates the design choices. This section also cites related work when appropriate. Lastly, Section 4 describes the realization of the architecture and discusses how the experimental results compare to the application requirements.

2 Background

Through collaborative discussion with the railroads, we were able to collect a comprehensive list of desirable application scenarios that their next-generation, on-board sensing and communications infrastructure should support. While this list is too long to describe in detail here, we do describe a few choice scenarios to help highlight the more challenging requirements.

2.1 Application Requirements

On a railroad car, the hub where wheel meets axle encloses lubricated ball bearings that are hermetically sealed. The ball bearings help reduce friction when rolling and are collectively referred to as the *bearing*. Occasionally, the seal on a bearing will crack or break, exposing the bearing to the environment. Once exposed, the lubricant will eventually dry out and the bearing will fail. A failed bearing can cause a wheel to seize, which in the worst case can result in derailment of the train. However, more typically, a failed bearing will lead to stoppage of a train. Detecting (or more desirably predicting) a bearing failure can substantially reduce or eliminate derailments and stoppage delays caused by such faults.

The temperature of a bearing can be a strong indication that a bearing has failed or is about to fail. As the lubricant inside the bearing begins to wane, the temperature of the bearing increases because of increased friction between wheel and axle. It is common for the surface temperature of a failing bearing to exceed 300° F above the ambient temperature. The time between an overheating event and an actual failure may only be tens of seconds to a few minutes. Thus, timely reporting of an overheating (or hot bearing) event is paramount.

The current approach to detecting hot bearings employs trackside hot bearing detectors at sparse locations, on average about every 30 miles. Given the sparse deployment, coupled with slow average train speed (about 20 miles per hour), the current railroad technology does not support timely detecting or reporting of hot bearing events. On average, the current technology offers visibility every 45 minutes. Moreover, the hot bearing analysis takes place at the enterprise level – not in the field, adding to the overall delay of event notification. In fact, event notification is often performed manually via two-way radio between the train's engineer and operations.

While there is certainly more need for algorithm study to detect hot bearings in situ, the low-latency reporting of hot bearing events is the more challenging requirement. Critical alert messages must traverse a potentially long distance and over many communication hops. Additionally, for energy consumption consideration, nodes will

likely employ a sleep schedule with a low duty cycle. Thus, the end-to-end latency will include the time it takes to wake up enough of the network to provide a forwarding path to the gateway.

Hot bearing detection is just one of several fault detection scenarios envisioned by the railroads, each having the requirement of low-latency reporting of alert messages. Other scenarios include cracked wheel detection, flat wheel detection, low/high brake pressure detection, car intrusion detection, high temperature detection on refrigerated cargo, and biological or chemical hazard detection. In addition to outlier detection, the railroads would also like to have periodic reporting of these key operational measurements, where the target reporting interval is every 10 minutes or better. In this paper we will evaluate SEAIT's effectiveness in delivering alert messages with low latency and regular synchronous reports, both over the characteristic distances and number of hops encountered on a real train.

In North America there are more than 1.5 million freight railroad cars and more than 17000 freight locomotives either owned or operated by the nine major Class I Railroads. Locomotives and railroad cars represent a significant portion of the capital assets invested by the freight railroad industry and their customers. A freight train may comprise up to 150 freight cars of different types from different customers to be delivered to different destinations. At every stop of a route, cars may be dropped off or added to the train. Maintaining the accurate and up-to-date information about the train consist (listing of the railroad cars and their order with respect to the lead locomotive) is an essential requirement for railroad operations. Such information may be used for correcting and preventing operational mistakes, logistic planning and optimization, and reconciling billing and other financial settlements. The information about the length and weight distribution of the train can also be derived from the consist, and these measures, along with speed, brake pressure, location, and knowledge of terrain can be used to help operate the train safely at the optimal speed for maximum fuel efficiency and train network throughput. Additionally, consist information is needed to fully utilize applications that perform outlier detection. Knowing the ordering of cars allows service personnel to be directed to the exact car and component that generated the outlier condition.

In the early 1990's, the North American Railroad industry adopted an Automatic Equipment Identification (AEI) system to identify and track railroad equipment while en route. Today, over 95% of the railcars in operation have been tagged with an industry standard-based UHF AEI transponder, one on each side of a car, and more than 3000 AEI readers have been deployed across North America. As a train passes by a trackside reader, the unique identification numbers of the locomotives and railcars are automatically captured by the reader and transmitted to a central server to construct a train's consist.

While useful, this approach suffers from the same drawback as the trackside hot bearing detectors. Namely, sparse deployment leads to gaps in coverage and untimely information. AEI readers are not available at all possible drop-off and pick-up locations. In fact, many waysides, where cars are often dropped off for customers to load/unload, are cut off from communications, including AEI. Alternatively, the railroads would like to identify a train consist in real-time, where real-time is measured in minutes versus the tens of minutes achieved using AEI. However, there are some challenges to indentifying consists in real-time. First, to keep the total solution costs

down, it is desirable that consist identification be performed using radio frequency (RF) measurements only. Second, the communications infrastructure must be able to deliver data reliably over many hops. Lastly, assuming 100% reliability cannot be achieved, any proposed consist identification algorithm must be robust to some packet loss and noisy RF measurements.

The on-board, WSN-based infrastructure may also help the railroads to address another problem: detecting "dark" cars. Railroad cars are routinely left at wayside locations for customers to load or unload. Occasionally the information system would lose track of the locations of some cars after they are detached from a train. There is no existing solution for this problem. AEI is not suitable because the range is inadequate (up to 30 feet). Using WSN-based consist identification, each train network will be able to detect when and where cars are dropped off at waysides, thus reducing the occurrence of dark cars. Additionally, any WSN-equipped train that passes a wayside can potentially query the wayside for the presence of any dark cars.

This scenario is an example of an on-demand query/response communication paradigm, where there is a mobile gateway and one or more nodes at a stationary wayside. This paradigm presents an interesting challenge. Namely, the mobile gateway potentially passes many sensor nodes at the wayside simultaneously and each sensor node may only be in range of the wayside gateway for a few seconds (depending on the train speed). Thus, robust delivery and low latency are also challenges for dark car detection.

In anticipation of a progressive roll-out of WSN, a train might have some railroad cars equipped with WSN technology while others are not. In such a mixed-mode environment, a train might be without a gateway or the network might become partitioned. For example, sensor nodes on a train without a gateway can store pertinent data until they are in proximity of a wayside gateway, which then can query the train for data as it passes. This scenario is similar to the familiar data mule scenario, where the train is the mule. It is also another example of an important mobile railroad scenario that relies on the on-demand query/response paradigm.

2.2 Preliminaries

Conceptually, an onboard WSN can be viewed as a logical representation of the train composition. Recall Fig. 1, WSN nodes are attached to railroad cars, creating a physical association between the node and the car. Thus, tracking a WSN node is logically equivalent to tracking a railroad car. To make this binding concrete, a WSN node must be provisioned with a unique identifier for itself and for the railroad car before deployment. For this purpose, SEAIT uses unique 64-bit identifiers, which are referred to as the *nodeId* and *carId*, respectively. The resolution of each identifier is more than adequate to support uniqueness for the number of railroad cars and locomotives in North America. This binding must be persistent in a database at the enterprise, and gateways must have access to it. A gateway should be able to learn everything it needs to know about a WSN node/railroad car by querying the database with the nodeId/carId. We use node and car interchangeably throughout the discussion. Additionally, the location where each node is attached to the railroad car is needed to help determine the ordering of cars and to identify the location of faulty components (e.g., a bad wheel bearing). In SEAIT, location refers to a designated end, and side. In

particular, railroads designate the end with the hand brake as the *B end* and the other as the *A end*. Similarly, each side of the car is designated as either the *left side* or the *right side* as viewed from the B end.

In many sensor network applications, the WSN is considered to be a single flat network controlled by one or more gateways, where the WSN nodes are free to choose any available gateway to sink their packets [1]-[5]. This approach to network discovery does not apply well to a WSN onboard a freight train because most railroad applications imply each node should belong to a distinct network. For example, let's consider the case when two trains, Train A and Train B, are within radio communications range of each other (as is the case in railroad yards and when trains pass each other). Operational data, such as a hot bearing alert, from a WSN node on Train A has no meaning to Train B, and vice versa. Therefore, nodes on Train A should never choose Train B's gateway as a sink, even if the route to Train B's gateway is the best route. This observation implies that there should be a logical binding between train and network. SEAIT accomplishes this by assigning a unique 16-bit identifier to every train WSN (TWSN) and binding the TWSN ID to a unique train identifier that already exists for every train. The resolution of a TWSN ID is sufficient to statically assign a unique ID to all the locomotives in North America. To support low power operation for motes not associated to a TWSN, there is one common system-wide wakeup channel. All motes not associated to a TWSN operate in a low power sleep mode on the wakeup channel. During association, the TWSN gateway assigns a channel distinct from the wakeup channel for TWSN traffic.

3 SEAIT Architecture

Fig. 2 depicts an overview of the SEAIT software architecture for a sensor node. Message flow is designated by the grey arrows, while the dotted black arrows represent cross-layer configuration. Railroad applications and services send and receive messages through the interface to the communication stack, which provides networking, link delivery, and an IEEE 802.15.4 medium access control (MAC) and physical layer (PHY). Applications also interact with node services, which have global scope. The information and reporting services realize the execution of a novel and uniform information and messaging model (Section 3.1 and 3.2, respectively). Because the messaging and information model is unified, most applications are realized in SEAIT without any application-specific messages; dark car detection is one example (Section 4). The synchronization service realizes robust management of a real-time clock (RTC), which provides application-level alarms (Section 3.3).

The network layer includes the message manager, time-scheduled transmit and receive queues, a list of neighbors, and a router (Section 3.6). The message manager supervises message flow and coordinates interaction between other components in the network layer. The router uses cross-layer configuration to provide a novel approach to optimizing the routing protocol based on the operating mode of the node. A time-scheduled queue in SEAIT is simply one where each message in the queue is assigned a specific time-to-live (TTL) in the queue. The queues use the following service policy: the entry with the lowest TTL gets served first. To break ties, a FIFO policy is employed. The time-scheduled queues provide a mechanism to randomize responses

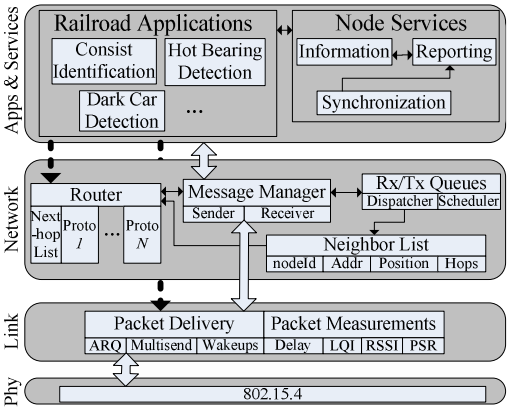

Fig. 2. Applications and services send/receive messages through the communication stack (grey arrows). Cross-layer configuration is used to optimize routing and link message delivery (dotted black arrows). Components within a layer interact with each other through their interfaces (solid black arrows).

to one-shot queries, provide a means to support sub-second timing for multi-phase protocols and applications (Section 3.7), and it support priority-based queuing. The neighbor list stores a node's one-hop neighbors' attributes, including nodeId, network address, position in consist, and hop count to the gateway.

The link layer provides packet delivery and packet measurements. All packets that flow through the communications stack are annotated with the packet measurements. The delivery mechanisms, automatic repeat request (ARQ) and multisend of broadcast messages, are well known, so we do not describe them here. However, we do highlight our approach to measuring delay and wakeup signaling (Section 3.3 and 3.4), which differ from prior works. We use 802.15.4 for the MAC and PHY; however, there is no real dependence on 802.15.4-specific MAC functions. The architecture can easily be adapted to any comparable multi-channel radio that uses carrier-sense multiple-access with collision avoidance.

3.1 Information Service

The goal of the information service is to provide a common information model for managing and reporting sensor, configuration, and application information. Most of the architectures we surveyed did not present an information model. One related work, Tiny Web Services (TWS), uses XML to support web services on resource constrained nodes [6]. However, this approach requires an underlying TCP/IP stack and about a third of the ROM resources on the device (15.8KB of 48KB). In contrast, the information service in SEAIT uses about 300 bytes of ROM using the same device as TWS. Because of this costly resource overhead and the dependence on a specific communications stack, we opted to design a very light weight information model for our study.

The information service manages a collection of information blocks. An information block is an object that describes the attributes and value[s] of some physical

measure or a logical abstraction derived from some physical measure. The latter provides a means to track complex events of interest. From a gateway's perspective, attributes are read/write and values are read only. Each information block has a unique one-byte identifier, allowing for the definition of up to 256 unique information blocks. An attribute is a distinct characteristic of an information block, such as the unit of measure for a block's value. Attributes provide the information necessary to allow a sensor node to properly encode/decode the value of a sensor. For example, to model any analog temperature sensor we can use the following information block, expressed as the tuple {*T, Vref, ADCMax, unit, offset, resolution}*, where *T* is the temperature value, *Vref* is the reference voltage of the analog-to-digital converter (ADC) on the sensor node, *ADCMax* is the maximum range of the ADC, *unit* is a one-byte code representing the unit of measure for *T*, *offset* is the zero calibration offset of the sensor, and *resolution* is the sensor resolution (usually in volts/°C). The temperature value for the analog sensor can be determined as follows

$$V = \frac{ADCValue}{ADCMax} \times Vref \tag{1}$$

$$T = \frac{(V - offset)}{resolution} \tag{2}$$

where *V* is the voltage representation of the digital output (*ADCValue*) from the ADC.

The example above describes how an information block can be used as a generic model for a particular sensor type. That is, if the analog temperature sensor is changed to one having a different specification, the node does not need to be reprogrammed or taken offline. The new sensor can be connected and the analog temperature information block can be dynamically configured with new attributes. Attributes also provide a means to configure the application logic associated with a particular information block. To describe this aspect, we consider a threshold-based hot bearing detection application. We can model a hot bearing detection block using the following tuple {*hasAlert, avgBearingTemp, unit, threshold, window}*, where *hasAlert* is the state of the detector, *avgBearingTemp* is the average temperature of the wheel bearing, unit is a code representing the unit of measure for *avgBearingTemp*, *threshold* is a temperature threshold, and *window* is the time interval to average the bearing temperature over. The attributes *threshold* and *window* are used to configure the threshold detection logic. In particular, *avgBearingTemp* is derived by averaging the wheel bearing temperature over the last *window* milliseconds. For each new sensor sample, *avgBearingTemp* is computed and then compared to *threshold*. If *avgBearingTemp* is greater than *threshold*, then the bearing temperature is considered to be outside normal operating range and the value *hasAlert* is set to true. If *avgBearingTemp* falls back below *threshold*, then *hasAlert* is set to false. This simple example motivates using attributes to configure application logic. The same approach can be used to configure more sophisticated logic, such as trend analysis.

SEAIT provides a uniform structure for encoding and decoding of information blocks; Table 1 depicts this structure. Each information block contains two parts: 1) a small three byte header and 2) a number of data fields that store a block's values and attributes. The first byte of the header is always the block id and the second byte is always the length of the block in fields. The third byte represents the size of each data field in bytes. When the field size is greater than zero, then the information block is said to have fixed-length encoding, where each field has the same size. For example,

if the field size is two, then each data field will be two bytes wide. When the field size is equal to zero, then the information block is said to have variable-length encoding, where each field can vary in size. In variable-length encoding, the first byte of each data field represents the length of the field in bytes. The first *M* data fields always contain the read only values, where the size of *M* depends on the block definition. The remaining fields contain read/write attributes.

Table 1. General structure of an information block

Header Byte	**Meaning**
0	block id
1	length of block
2	field size
Data Field	**Meaning**
0	read-only value
...	…
M-1	read-only value
M	read/write attribute
...	…
up to packet length	read/write attribute

To configure information blocks, gateway applications issue *Configure Command (ConfCmd)*. Configure Command enables configuration of all attributes for any information block supported by a sensor node, and multiple blocks can be configured with a single message. Using a nodeId or carId and a flag indicating which type of identifier is present in ConfCmd, the message can be targeted towards all nodes, to a specific node, or all nodes on a specific railroad car. End-to-end acknowledgments can optionally be requested for ConfCmd.

3.2 Reporting Service

The reporting service provides the messaging means by which a node's information blocks are sent upstream to a gateway. It keeps track of what information blocks require service and the type of reporting paradigm required by each active block. It also manages the execution of each reporting paradigm (see below). In a typical situation, a gateway will request one or more information blocks by issuing a *Report Request (RepReq)* and nodes respond using *Report Response (RepResp)*. Report Request supports retrieval of just the values in an information block or the entire information block (values and attributes). As with ConfCmd, RepReq can be targeted towards all nodes, to a specific node, or all nodes on a specific railroad car, and multiple blocks can be requested with a single message. RepResp is the unified message format that carries information blocks from a sensor node to the gateway.

SEAIT supports two types of synchronous reporting paradigms: periodic and N-times, where N-times reporting is a special case of periodic reporting that has a preconfigured finite duration. In both paradigms, a node will send all active information blocks periodically in its car's designated time slot, which is assigned when a car associates to a train (Section 3.5). A synchronous paradigm is invoked when the report service receives a periodic alarm event from the synchronization service

(Section 3.3). Because the specific use and granularity of intra-frame slot timing can depend on application semantics, applications can override the default slot timing. Consist identification is one application that employs this option to support robust intra-slot messaging needed to measure neighbor closeness (Section 3.7). Gateways can also modify a node's default duty cycle through an information block.

SEAIT supports asynchronous reports using on-demand and event-trigger paradigms. In the on-demand paradigm a gateway will request one or more information blocks using RepReq and all nodes will respond immediately. In contrast, the event-trigger paradigm is initiated by node applications and is used to communicate alert events. The hot bearing detection block described in the previous section is an example that employs the event-triggered paradigm. When the hot bearing detection logic detects an overheated wheel bearing, it updates the hot bearing detection block and notifies the report service that the block requires service. Since alert events will likely occur outside a synchronous duty cycle (when nodes are sleeping), SEAIT accomplishes fast delivery of asynchronous reports by using wakeup messages (Section 3.4). Any node that has an alert message to report will just send it if the node is in a synchronous duty cycle, otherwise it will send a wakeup message first and then send the alert. An intermediate node that receives an alert first tries to forward the alert to a next hop. At any intermediate hop where a forwarding node fails to find a next hop, the forwarding node will send a subsequent wakeup message and then send the alert message. Each intermediate node follows this procedure until the alert message is delivered to the final destination. This approach allows us to delivery alerts within tens of seconds over a long train (Section 4).

3.3 Synchronization Service

The synchronization service manages and provides an interface to a real-time clock (RTC). The RTC can be realized in software or hardware; SEAIT currently uses a software implementation. Applications can register alarms with the synchronization service, where one alarm is always dedicated for the report service. Alarms can be configured to provide second, minute, or hour granularity. A pre-computed schedule is set up by configuring an alarm to alert at a discrete epoch that evenly divides into one cycle of the next highest granularity. For example, for an alarm configured to a granularity of seconds, the valid periods are every 2, 3, 4, 5, 6, 10, 12 15, 20, and 30 seconds. There is also a notion of a starting epoch for each alarm. The starting epoch represents a shift of the alarm's epoch zero, and it can be any discrete value in one cycle of the clock. For example, an alarm configured to go off every 15 seconds with a starting epoch of 2 will alarm 4 times each minute of each hour at XX:XX:02, XX:XX:17, XX:XX:32, and XX:XX:47. The starting epoch provides a means by which two or more gateways operating distinct networks in close proximity on the same channel can coordinate their synchronous schedules. Using discrete alarm times allows for a simple and robust implementation because the alarm update procedure is memoryless; compute once and the schedule repeats each cycle of the clock. In contrast, when using a continuum of alarm times, the schedule does not repeat each cycle; requiring a node to maintain history and perform computations at each update.

To maintain its clock, our prototype gateway uses NTPv4, which provides a tolerance of 1ms or less over a LAN [7]. On a real locomotive, the gateway would have

access to GPS time, which can provide microsecond tolerance. A synchronization update for each WSN node is achieved by broadcasting a reference clock from the TWSN gateway and measuring the accumulated delay at each hop, where the accumulated delay is used to adjust the clock offset. To combat clock drift, SEAIT uses a two-pronged approach. First, the gateway periodically sends out new reference broadcasts, with a frequency determined by application requirements. Second, each node proactively polls its neighbors for a fresh clock, whenever a node determines its synchronization state is stale. Our measured delay approach is similar in concept to DMTS or Delay Measurement Time Synchronization [8] in that it seeks to accurately measure the clock offset by measuring the delay between transmitter and receiver using a reference broadcast. Our approach differs from DMTS in how the measured delay is propagated throughout the network. DMTS requires the network to organize itself in a parent-child hierarchy, where each parent node updates its clock first, then generates and sends a new reference broadcast to its child nodes. In contrast, our approach does not require such organization. Instead, the network packet header includes a two-byte field that represents the accumulated forwarding delay, as measured from the reference source to each sink. Thus, at any given sink, the clock offset is simply the forwarding delay of the received reference broadcast message plus the queuing delay incurred between hops. Directly measuring the accumulated forwarding delay provides other benefits, such as measuring end-to-end packet latency and providing an accurate temporal notion of packet freshness.

Using a similar measured delay approach, the authors in [8] and also in [9] show that, when packet reception/transmission timestamps are taken as close to the actual event as possible, the worst case offset error can be limited to two clock ticks per hop, where clock tick is measured in the local time base of the sink node. Alternative approaches try to algorithmically estimate the clock offset and drift by using multiple broadcast references during synchronization update [10][11]. In our implementation, the local time base and the forwarding delay field have the resolution of a 32KHz clock, which gives a worst case error per hop of ~61μs. Assuming a liberal number of hops (about 30) to cover a maximum length train (150 cars with average length 60 feet), the worst case offset error would be ~1.8ms.

3.4 Semantics-Based Wakeups

Preamble sampling is a common low power, asynchronous approach used to wake up low duty-cycling (sleeping) nodes on-demand using a long preamble that is continuous or pulsed. Sending long preambles as a communication wakeup was first proposed in [12] and has since been refined in numerous works [13]-[16]. SEAIT's approach to wakeup signaling differs from these prior works in two respects: 1) a node's decision to wakeup is based on application semantics and 2) a novel packet structure to embed the semantics and wakeup signaling. In the common approach, wakeups would normally be obscured in the MAC layer, not accessible to applications. All the prior works cited above wakeup a node's radio from the MAC layer, which, from a layering perspective, seems like the appropriate layer to exercise control over the radio. However, our experience in designing a WSN for railroad applications shows that wakeup signaling can have greater utility if the decision to wakeup is realized at the application layer. In prior works, wakeup signaling is used

exclusively as a mechanism to facilitate asynchronous, multi-hop transfer during upstream communications from WSN nodes to a gateway. In addition, SEAIT uses wakeups for a swath of applications and network operations, including the dissemination of network commands (e.g., reset), fast delivery of alert events from outlier applications (e.g., hot bearing detection), and as an on-demand wakeup for a group of nodes that share the same mode of operation (e.g., all dark cars).

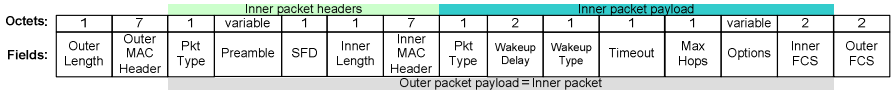

Fig. 3. The inner packet is preceded by a long preamble and a full MAC header. The inner packet payload contains the application semantics.

A wakeup message in SEAIT is a specially formatted data message, whose structure is depicted in Fig. 3. The structure shown in the figure was designed for use with IEEE 802.15.4, but the same approach can be generally applied to other radios. Each packet is a maximum length message that contains a complete nested packet within a packet. The nested packet is called the inner packet and it is said to be contained in the outer packet. The outer packet has a normal PHY and MAC header and footer. The inner packet is the payload of the outer packet and is self contained. It includes its own PHY and MAC headers followed by the inner payload and ending with the standard two-byte frame control sequence (FCS).

In SEAIT, the first byte after the MAC header is always the packet type field, which indicates the type of packet. The packet type field provides a flat classification for all packets, leaving a designer free to decide which layer in the stack will decode a packet. To accommodate variable-sized inner packet payloads (and optionally variable-sized MAC addressing), the inner packet's preamble is variable (up to 100 bytes). To be complete, the inner PHY header must also contain the start of frame delimiter (SFD) and the length of the inner MAC frame. For our application space, we found 802.15.4 destination MAC addressing with short addresses sufficient for most applications. The inner packet payload has four other fixed fields and one variable length field. The wakeup delay field of milliseconds remaining in the wakeup burst. This field allows a receiver to schedule when it should wakeup and listen for data. The authors in [17] use a similar approach to reduce receiver idle listening time while waiting for completion of the wakeup burst. The wakeup type field indicates the type of command/application/protocol that is associated with the wakeup. The timeout field allows each wakeup to have deterministic duration of arbitrary length. The max hops field indicates the maximum number of hops for forwarding the wakeup. This field is decremented before forwarding to the next hop. When the field's value is zero, the wakeup is not forwarded. The options field carries optional application data, which is wakeup type dependent.

Because full MAC addressing is used in the wakeup signaling, a wakeup can target a specific node or an entire TWSN. Additionally, a wakeup can target only those nodes that have interest in the specific wakeup type, which is an effective approach to reduce idle listening that is not supported by prior works. Our realization of consist identification and dark car detection (Section 3.5 and 4) use this approach to target

only nodes that share the same mode of operation. Forwarding of wakeups over a finite number of hops is another useful feature not supported by prior works. Wakeup forwarding can be used to ensure that the entire TWSN is awake before the gateway issues an important command. Applications can override the default parameters for the number of packets in a wakeup burst, the channel, and the destination addressing fields – on a per packet basis. When waking up a single node, the authors in [16] propose a pulsed approach to transmitting long preambles to allow time for the target node to acknowledge the wakeup, thereby shortening wakeup time and energy consumption. Alternatively, SEAIT also supports this approach by using the time-scheduled queues in the network layer to schedule multiple wakeup bursts that can be separated by gaps of arbitrary length, based on application requirements.

3.5 Associating Cars to a Train

The process of automatically associating cars with a train is a vital first step in the important application of consist identification. This process depends on the train's situational awareness: Is the train moving? How fast is the train moving? Did the train just start/stop moving? Does a manifest exist for the expected consist? Has the train received a clear-to-go message from yard operations? These business and operational constructs are most effectively assessed at the application level. However, because of the binding between WSN node and car and between train and network, network discovery is inherently coupled to the process of associating cars to a train. In particular, SEAIT employs a coordinated approach, where the network layer tracks networks (via gateways), but the application layer determines the association to a specific train/network. Thus, the network layer's default gateway is set by the application layer. Throughout the discussion, we use association and discovery interchangeably.

For obvious safety reasons, cars can be physically attached to (or removed from) a train only when the train is stationary. The TWSN gateway in the locomotive is best equipped to discern a train's situational awareness because it has access to GPS speed and location, as well as connectivity to the enterprise; therefore, association is initially directed by a TWSN gateway. There are numerous scenarios under which a gateway might start associating cars to a train; however, the basic approach is the same. The TWSN gateway detects a start condition (e.g., clear-to-go message, manual push-button, train begins moving, etc.). The TWSN gateway optionally sends a wakeup message on the wakeup channel, followed by one or more Car Association (CarAsc) messages (described below). The wakeup has type WAKE_CARASC, which indicates that CarAsc will follow. Any nodes not requiring association can ignore the wakeup and go back to sleep to conserve energy. Car Association contains the TWSN ID, synchronization data, the gateway's carId, the gateway's position in the consist, and an optional list of carIds. The synchronization data includes the clock reference and optionally the alarm time and duty cycle for the synchronous reporting paradigm. The list of carIds specifies which cars should associate to the train. This list is only available when a manifest exists. When no manifest exists, the list is empty, and the message is referred to as the all join CarAsc. When receiving an all join CarAsc a node will optionally join a network based on its car's motion. A SEAIT node uses motion detection (via an accelerometer) to determine its status. If an unassociated node is experiencing persistent motion upon receiving an all join CarAsc, then the

node will join the network. To extend the range of CarAsc, associated nodes periodically take turns at rebroadcasting their association data on the wakeup channel, with an updated timestamp and optionally preceding it with a wakeup.

3.6 Routing

A railroad car experiences three distinct modes of operation: 1) it is not associated to a train (dark car), 2) it is associated to a train with a known consist (known linear topology), and 3) it is associated to a train with unknown consist (unknown linear topology). This observation led to a modal design approach to routing messages in SEAIT. Specifically, SEAIT supports multiple protocols for selecting the next-hop and the appropriate routing strategy is selected based on the operating mode of the TWSN. When the consist is unknown, SEAIT employs a hop-based routing strategy, where the next-hop always has a hop count one less than the current hop. During consist identification, this simple protocol is used because it has low complexity and it can quickly adapt to a changing topology. Once the consist is known, very efficient geography-based routing is used to multi-hop the ordered linear topology. A car's position in the consist represents it's one-dimensional coordinate along a line. The next-hop is selected by choosing a node that is P positions closer to the gateway. When a car is dark, it employs single-hop communications only because the typical distance from a wayside to the track does not require multi-hop (40 feet or less).

Through the cross-layer configuration interface (Fig. 2), the application layer configures the operating mode of the network, along with any operational parameters such as the car's position in the consist. While we found existing hop- and geography-based routing protocols adequate for our evaluation, SEAIT's modal approach does not preclude using other techniques.

3.7 Consist Identification Application

Consist identification is an iterative application that has four major phases: 1) associating cars to the train (Section 3.5), 2) measuring closeness between each pair of nodes, 3) reporting closeness measurements, and 4) determining the consist composition and order. The gateway sets up the second and third phase by sending a RepReq for the closest neighbor information block using the periodic reporting paradigm. This request tells each node to perform a series of RF closeness measurements and report the results. An RF measurement is a broadcasted request for neighbor information followed by unicast responses from each neighbor in range; each node takes a total of four measurements. During the measurements, sender and responder use the same power level, which is set to cover a range of about three cars. A closest neighbor information block contains a ranked list of each node's closest neighboring nodes. Each list entry contains the pair *{nodeId, measurement}*, where the measurement is the bi-directional link quality. We define bi-directional link quality as the average of the sender and responder link quality indicator, which is produced by the 802.15.4 radio. In the final phase of the consist identification application, an ordering algorithm (described below) at the gateway inputs the closest neighbor information blocks and produces the consist as its output.

Consider n cars in a train $\mathbf{N}=\{N_i \mid i=1,...,n\}$. The ordering algorithm operates in three steps: 1) compute a car closeness metric $\{d_{ij}\}$ from the node measurements, 2) refine the car closeness metric using a correlation based operator, and 3) construct a weighted digraph, $G=(\mathbf{N},\mathbf{E})$, where edge $e_{ij} \in \mathbf{E}$ has a weight of d_{ij}. The car closeness metric reflects the closeness between a pair of cars N_i and N_j. The closer the two cars are, the greater the value for d_{ij}. Forming a consist from $\mathbf{N}$ is then equivalent to finding the maximum Hamiltonian path for graph G.

Step 1: Each car N_i is equipped with two nodes. We denote q_{kl} as the bidirectional link quality between nodes k and l. We define the car closeness metric as the combined link quality measurements between all the nodes on N_i and N_j, such that,

$$d_{ij} = \sum_{\substack{k=2i,2i+1 \\ l=2j,2j+1}} q_{kl} \tag{3}$$

where d_{ii} is scaled to four times the maximum link quality value. Under ideal conditions, for each pair of cars, this summation has four terms. When any of the pair wise measurements are missing, q_{kl} is set to zero. Because the link quality measure is asymmetric and the scale may vary across nodes, we apply two conditioning operations. First, we scale the rows of $\{d_{ij}\}$ so that the diagonal becomes all ones. Second, we make the matrix symmetric by computing the average of the original matrix and its transposition.

$$\phi(\{d_{ij}\}) = \{d'_{ij}\}, \tag{4}$$

$$d'_{ij} = \frac{\sum_{k=1}^{n} d_{ik} d_{jk}}{\left(\sum_{k=1}^{n} d_{ik} \sum_{k=1}^{n} d_{jk}\right)^{1/2}} \tag{5}$$

Step 2: We could construct the graph using the metric directly and perform the maximum Hamiltonian path search. However, this often leads to unstable results, as the metric is based on noisy measurements. We introduce a refinement operator on the metric to improve the search algorithm's stability. Consider the function $g_i(k) = d_{ik}$. It can be viewed as a distribution of the closeness with regard to N_i over the entire set $\mathbf{N}$. The correlation between g_i and g_j gives a second-order measurement of the closeness between N_i and N_j. It makes the graph searching algorithm more stable as it incorporates a number of d_{ij}. The operator is thus formally defined by (4) and (5).

Step 3: The general case of maximum Hamiltonian path problem is equivalent to the Traveling Salesmen Problem, which is NP-Hard. For simplicity, we use a greedy algorithm to construct a maximum Hamiltonian path. The algorithm starts from a known starting node (the gateway on the locomotive). Each successive node is found by following the edge with the maximum weight. This simple algorithm is $O(n^2)$.

4 Evaluation

We implemented SEAIT in TinyOS [18] and the gateway software in Java. We deployed 32 WSN platforms along the front metal railing of the roof at our facility.

Each platform contained a TmoteSky sensor node, a sensor board, two 5.4 Ah batteries, an embedded antenna, an input/output connection board, and a weatherproof enclosure. The sensor board included temperature, light, and accelerometer sensors. The WSN node code fits into 38 KB of ROM and 7 KB of RAM. On average, freight railroad cars are about 60 feet long, ranging from as little as 40 feet up to 90 feet. Deploying two sensor platforms per car, we emulated a train of fifteen cars plus a locomotive, with each car 60 feet long and a node spacing of 10 feet between adjacent cars. The entire deployment spans about 0.25 miles across the roof.

The consist identification application (Section 3.7) was run for almost two hours using the 16-car deployment on the rooftop. The application was invoked via an electronic clear-to-go message. The periodic synchronous reporting paradigm was setup with a period of 2 minutes and a slot time of 384ms. To evaluate the ordering algorithm, we used the following criteria: 1) the latency before reaching a stable consist and 2) the accuracy of the consist when compared with the expected consist. We define the error as the number of cars that must be moved to achieve the expected consist. We define the ordering algorithm output to be stable when the error over the last 5 cycles is 2 cars or less. We use the term flip to denote the case when the algorithm transposes the order of two cars that are physically adjacent; 1 flip equals 2 cars in error.

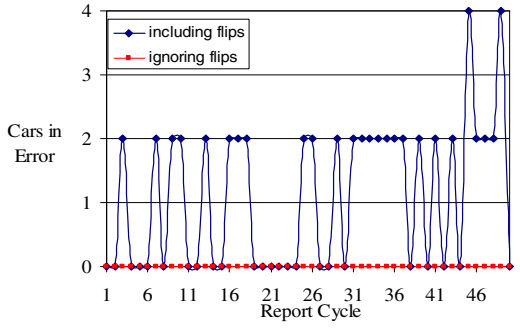

Fig. 4. Within 5 cycles, the algorithm is stable and 100% accurate, when ignoring flips

Fig. 4 shows details of a consist formed during 50 cycles of the application. When flips are included, the algorithm output is stable by the fifth cycle and the accuracy averaged over 50 cycles is 93%. A close look at the data revealed that all of the errors where the result of flips, which is encouraging because a small number of flips are tolerable; no single cycle had more than 2 flips (4 cars in error). If we ignore flips, the generated consist still stabilizes within five cycles, but the accuracy increase to 100%. During this experiment, the end-to-end reliability for each node ranged from 95.4% to 100%. Overall, these results satisfy the application requirements, namely accurate and timely detection is possible using only RF measurements. Because consist identification occurs on a stationary or slow moving train (5 mph or less), we expect similar results on a real train.

We conducted a series of experiments to study the most challenging requirements for the outlier detection class of applications. Namely, for asynchronous alert reports, can we achieve high reliability (95% or better) and low latency (10s of seconds to minutes) over the worst-case length train (150 cars). We configured trains of 5, 7, 10, 12, and 15 cars. For each configuration, one node was configured to simulate a critical event by sending a maximum length (128-byte) alert message every 30 seconds. To stress the system, we imposed a worst-case routing strategy (from a reliability perspective) by forcing all nodes to attempt a route through their closest neighbor first. If the initial route fails, then up to three alternative routes are tried in succession, each alternative being one car closer to the gateway. This routing strategy assumes consist ordering is known. The alerting node was configured at 10, 15, 20, 25, and 30 hops. We conducted 120 trials at each configuration using a wakeup burst of 2 seconds. The preamble sampling duty cycle was 1.2% (wakeup 12 ms every second). For robustness over long distances in a sparse deployment, we found the duty cycle should be large enough to capture 6 wakeup packets per wakeup burst.

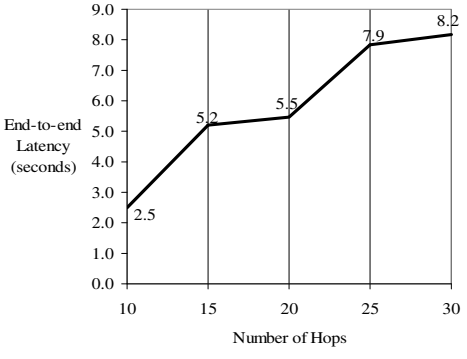

Fig. 5. Every 5 cars, an additional wakeup is required to continue forwarding the alert

Fig. 5 shows the relationship between the number of hops and the latency to reach the gateway, averaged over 120 trials. As expected, more hops to the gateway increased the delivery latency, in a monotonically increasing fashion. Roughly every 10 hops or 5 cars (350 feet) one additional wakeup is required to continue forwarding the alert. Extrapolating these numbers out to a 150-car train, we would expect the alert latency for a node on the 150th car to be about 75 sec, roughly 2.5 seconds per 5 cars. These extrapolated results satisfy requirements and are encouraging because they suggest an upper bound on latency, performance optimizations are certainly plausible. In particular, at the expense of more energy, shorter wakeups (or pulsed wakeups) can be made more robust by doubling the duty cycle to 2.4%. Additionally, in practice, we would use more efficient routing, such as choosing the furthest neighbor first. We also measured the reliability or packet success rate (PSR). The end-to-end PSR was virtually 100% over all experiments; only one packet was lost over all trials. This result is very encouraging because we do not expect to exceed 30 hops for a 150-car train.

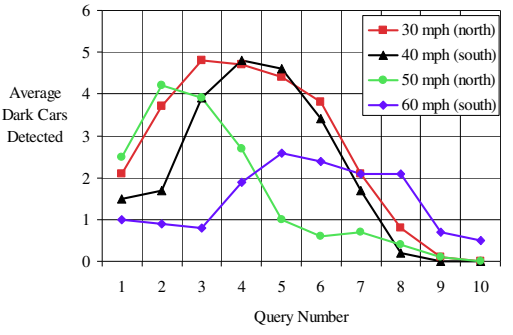

Fig. 6. The response distribution is sensitive to speed and direction. Shapes are similar for the same direction, with lower speeds having greater peaks and tails.

To measure the performance of the dark car detection application we performed experiments using a mobile gateway and five emulated dark cars, each car equipped with two nodes. The spacing between nodes was approximately 49 feet, and each node was elevated about 5 feet from the ground. The dark cars were placed on the wayside of a busy four-lane street. The perpendicular distance from the wayside to the path of the mobile gateway was 48 or 56 feet, depending on which direction the mobile gateway was traveling (northbound or southbound). In practice, waysides are typically no more than 30 to 40 feet from the track. The mobile gateway was a laptop secured in a car with an antenna mounted on the roof. Each WSN node employed a preamble sampling duty cycle of 2.4% (wakeup 24 ms every second). The dark car application was implemented using the following query sequence: 1) send 1-second wakeup, 2) send 3 on-demand RepReq messages for the dark car block, one every 200ms, and 3) wait 400ms for RepResp messages. This sequence was repeated every two seconds as the mobile gateway moved past the dark cars. We conducted 10 trials each for this experiment at 30, 40, 50, and 60 mph. Note, dark cars use a different duty cycle compared to cars associated to a train (i.e., alert latency experiments) because we wanted to use a 1-second wakeup to enable faster detection. Wakeup type WAKE_DARKCAR was used to indicate only dark cars should wakeup. A dark car information block contains nodeId, carId, the last known TWSN ID, and a timestamp indicating when the car went dark. Additionally, to stress the system, the block was zero padded to the maximum packet length.

The application detected the majority of cars at each speed, with the average ranging from 4.4 to 5.0 detected dark cars per trial. A detailed look at the distribution of responses to each query is presented in Fig. 6. The most effective queries occur when the mobile gateway is close to the center of the deployment. The figure also reveals sensitivity to speed and direction. For the 30 and 50 mph trials, the more effective queries are slightly biased towards the beginning of the run because the gateway was moving northbound, where the center of the deployment was reached during the initial part of the run. In contrast, data for the other speeds was taken from the southbound direction, where the center of the deployment was reached midway through the run. We also note that the slowest speeds have a higher peak and longer tail, which suggests a slower query rate could have been used. In contrast, the highest

speeds require the fast query rate because more queries are needed to successfully detect most of the dark cars. These results satisfy the applications requirements because it shows that a high detection rate is achievable, over a range of plausible train speeds and over a typical track to wayside range.

5 Conclusion

This paper presented SEAIT, a WSN-based architecture and system built as a part of feasibility study to determine if WSN technology is viable for use in advanced freight railroad applications. While this study is ongoing, the set of experiments reported in this paper suggest the affirmative. The results showed that application requirements can be met for several exemplar scenarios, citing four key results: 1) reliable message delivery of 95% or better is achievable over a long train, 2) low-latency delivery of tens of seconds over 150-car train is plausible, 3) accurate and timely (within minutes) identification of a train consist is achievable when only using RF measurements, and 4) accurate dark car detection is achievable at a range of typical train speeds and for the characteristic distance form track to wayside.

While these key results are the culmination of substantial investigation, there are still areas for continued exploration. Since this work was part of a feasibility study, where satisfying application requirements are foremost, some aspects of the implementation can benefit from optimization. The specific routing protocols used in the experiments were deliberately made sub-optimal to help project an upper bound on latency and a lower bound on reliability. Similarly, we hope to improve the complexity of the ordering algorithm to O(n). Automation of the scenarios, while maintaining stability, is another area for further study. For example, the dark car scenario should be automatically invoked when a train approaches a waypoint and automatically stopped as a train moves away from a waypoint. Using GPS and predetermined waypoints, a gateway can identify these proximity events and invoke the scenario at the appropriate time.

Acknowledgements. We would like to thank Lynden Tennison and Dan Rubin of Union Pacific Railroad for providing invaluable industry insights and support, Keith Dierkx for making this effort possible, and Maria Ebling and Paul Chou for their guidance and support throughout the project.

References

[1] Arora, A., et al.: ExScal: Elements of an Extreme Scale Wireless Sensor Network. In: Proc. of the 11th IEEE Intl. Conf. on Real Time Computing Systems and Applications, Hong Kong, August 17-19, pp. 102–108 (2005)

[2] Krishnamurthy, L., et al.: Design and Deployment of Industrial Sensor Networks: Experiences from a Semiconductor Plant and the North Sea. In: SenSys 2005: Proc. of 3rd Intl. Conf. on Embedded Networked Sensor Systems, November 2005, pp. 64–75 (2005)

[3] Kim, S., et al.: Wireless Sensor Networks for Structural Health Monitoring. In: SenSys 2006: Proc. of the 4th Intl. Conf. on Embedded Networked Sensor Systems, Boulder, Colorado, October 31-November 3, pp. 427–428 (2006)

[4] Werner-Allen, G., et al.: Deploying a Wireless Sensor Network on an Active Volcano. IEEE Internet Comp. 10(2), 18–25 (2006)
[5] Stoianov, I., Nachman, L., Madden, S., Tokmouline, T.: PIPENET: A Wireless Sensor Network for Pipeline Monitoring. In: IPSN 2007: Proc. of the 6th Intl. Conf. on Information Processing in Sensor Networks, Cambridge, MA, April 25-26, pp. 264–273 (2007)
[6] Priyantha, N.B., Kansal, A., Goraczko, M., Zhao, F.: Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks. In: SenSys 2008: Proc. of the 6th Intl. Conf. on Embedded Networked Sensor Systems, pp. 253–266 (2008)
[7] Mills, D.: Network Time Protocol Version 4 Protocol And Algorithms Specification (September 5, 2005), http://www.ietf.org/internet-drafts/draft-ietf-ntp-ntpv4-proto-11.txt
[8] Ping, S.: Delay measurement time synchronization for wireless sensor networks. Tech. Rep. IRB-TR-03-013, Intel Research, Berkeley, CA (June 2003)
[9] Chebrolu, K., Raman, B., Mishra, N., Valiveti, P.K., Kumar, R.: BriMon: A Sensor Network System for Railway Bridge Monitoring. In: MobiSys 2008: Proc. of the 6th Intl. Conf. on Mobile Systems, Applications, and Services, pp. 2–14 (2008)
[10] Maroti, M., Kusy, B., Simon, G., Ledeczi, A.: The Flooding Time Synchronization Protocol. In: SenSys 2004: Proc. of the 2nd Intl. Conf. on Embedded Network Sensor Systems, pp. 39–49 (2004)
[11] Ganeriwal, S., Kumar, R., Srivastava, M.B.: Timing-Sync Protocol for Sensor Networks. In: SenSys 2003: Proc. of the 1st Intl. Conf. on Embedded Network Sensor Systems, pp. 138–149 (2003)
[12] Intl. Telecommunication Union. Codes and formats for radio paging, ITU-R Rec. M.584-2 (11/97) (November 1997), http://www.itu.int
[13] El-Hoiydi, A.: Aloha with Preamble Sampling for Sporadic Traffic in Ad Hoc Wireless Sensor Networks. In: Proc. IEEE Intl. Conf. on Communications (2002)
[14] Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: SenSys 2004: Proc of the 2nd Intl. Conf. on Embedded Networked Sensor Systems, pp. 95–107 (2004)
[15] El-Hoiydi, A., Decotignie, J.: Low power downlink MAC protocols for infrastructure wireless sensor networks. ACM Mobile Networks and Appls. 10(5), 675–690 (2005)
[16] Buettner, M., Yee, G.V., Anderson, E., Han, R.: X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In: SenSys 2006: Proc. of the 4th Intl. Conf. on Embedded Networked Sensor Systems, pp. 307–320 (2006)
[17] Hui, J.W., Culler, D.E.: IP is Dead, Long Live IP for Wireless Sensor Networks. In: SenSys 2008: Proc. of the 6th Intl. Conf. on Embedded Network Sensor Systems, pp. 15–28 (2008)
[18] Levis, P., et al.: TinyOS: An operating system for sensor networks. In: Ambient Intelligence. Springer, Heidelberg (2004)