RAISE: RAIlway infrastructure health monitoring using Wireless SEnsor Networks

Jaime Chen, Manuel Díaz, Bartolomé Rubio, and José M. Troya

Dpto. Lenguajes y Ciencias de la Computación,
University of Málaga, Málaga, Spain
{hfc,mdr,tolo,troya}@lcc.uma.es

Abstract. Wireless Sensor Networks are composed of devices of reduced size, self-powered and with wireless transmission capabilities. Because of these features this technology has been recognised as promising for a large variety of monitoring and surveillance applications. Moreover, WSNs have been identified as having the potential to become an integral part of the protection of critical infrastructures (CIP). In this paper we present the details of an application that makes use of WSNs to monitor railway infrastructures. The WSN collects information about the structural health and behavior of the infrastructure when a train travels along it and relays the readings to a base station. The base station uses the next train(s) as a data mule to upload the information. The information is then processed on the train which does not have the limitations of a sensor node. The use of a train as a data mule is especially suitable to collect information from remote or inaccessible places which do not have a direct connection to the internet. The application has been built using a publish/subscribe middleware called PS-QUASAR over Tmote sky nodes. The results of the simulation using the Cooja simulator are presented in this paper and confirm the feasibility of the application.

Key words: high-level programming abstraction, wireless sensor network, middleware, critical infrastructure protection, infrastructure health monitoring, railway bridge monitoring

1 INTRODUCTION

Wireless Sensor Networks [Akyildiz et al., 2002] are composed of small devices which are self-powered and contain different sensors that can get information from the environment. They can also wirelessly communicate with each other to coordinate themselves and transport the information to a base station.

Monitoring applications can greatly benefit from this technology since a large number of nodes can be deployed in the scenario without the need for wiring [Gaura et al., 2010]. By means of routing and synchronization protocols, sensor nodes can coordinate with each other to sense the environment. In recent years, for example, the future prospects for WSNs as promising for the Critical Infrastructure Protection (CIP) field have been recognised. In this regard, WSNs have the potential to become an integral part of the protection of CIs. Their

distributed nature makes them particularly suitable against failures and attacks as they are much more rarely affected in their entirety, unlike wired systems.

One of the main barriers, researchers and industry need to tackle in order for WSNs to become pervasive in this application domain is the lack of QoS support, mainly due to their wireless nature [Chen et al., 2011]. It is important that the information sensed from the critical infrastructure is reliably sent to the base station. Moreover, because of the embedded nature of sensor nodes, programming applications for these devices is an error-prone task. In order to raise the abstraction level with which these devices are programmed a middleware layer is used on top of the sensor node's operating system [Mottola and Picco, 2011]. The middleware offers a programming model that simplifies the task of developing applications. It can also manage the communication between the devices so that application QoS requirements are met.

In this paper we apply WSN technology to the CIP problem, more specifically the monitoring of railway infrastructures. The application scenario consists of a railway bridge in which structural health is monitored. The WSN is deployed along the bridge and takes periodical readings about the structural health of it. Trains passing through are used as data mules to get the information from the sensors which means that no direct connectivity to the internet is required for the WSN. In order to tackle the lack of QoS support and the low level of abstraction of the sensor devices a middleware called PS-QUASAR [Chen et al., 2013] has been used. PS-QUASAR provides a simple publish/subscribe programming model. Developers can use it to provide QoS requirements that communications need to meet. The application implemented in this paper tackles the use of WSNs in the CIP problem. Also, it proves and defends that the use of middleware abstraction such as PS-QUASAR can considerably simplify the task of developing WSN applications and make it less error-prone. Finally, it makes use of interesting mechanisms that can be used to organize the network, such as clustering (to avoid packet collision and packet loss), data fusion (to minimize the number of sent packets) and QoS support.

The rest of the paper is organized as follows. Section 2 describes the motivation of the application scenario. In Section 3 related work is presented. A brief description of the PS-QUASAR middleware is described in Section 4. The application architecture and implementation details are depicted in Section 5. The evaluation and future work are described in Sections 6 and 7, respectively. Finally, some conclusions are presented in Section 8.

2 Motivation

Railway infrastructures, as any other kind of infrastructure, are affected by the aging process. This is particularly important in this domain. For example, large sections of the railway lines in the United States were built in the late 19th century or beginning of the 20th century. In Europe large sections of the railway lines were reconstructed after the Second World War. Therefore, it is really important to regulate maintenance and restoration guidelines to ensure the safety in the

railway transport. In this regard, more attention has been paid to this issue from the late 20th century. The document containing the guidelines for the maintenance of the Spanish railway lines (ITPF-5) is regulated in the FOM/1951/2005 Ministerial Order [Spanish Official Bulletin of the State (BOE), 2005]. In particular, for railway bridges, the guidelines establish that a visual inspection of elements of the infrastructure needs to be carried out every 15 years by specialized technicians. Furthermore, a general visual inspection is completed every year by non-specialized railway line guards.

This is sufficient for most railway bridges. In structures with unusual topology or particularly high/long structures, however, the information on the evolution of defects is more limited. Moreover the visual inspections are much more difficult to carry out and require a tempory closure to traffic. In these structures, it is common to check the state of the structure using specialized equipment or even install a permanent monitoring system, e.g. fiber optic instrumentation with BOTDA (Brillouin Optical Time Domain Analysis) or distributed sensor instrumentation. One of the main disadvantages of these systems is the high cost. Also, if there is no mobile coverage then data acquired by the system cannot be sent to the remote control center.

The current WSN technology can be used as a permanent monitoring system and considerably reduce the cost of installation and maintenance since no wiring is required. The application presented in this paper seeks to provide a system to monitor railway infrastructures using WSNs cost-effectively. It also copes with the network coverage problem and tackles the transfer of large quantities of data in a reliable manner.

3 Related work

The use of WSNs for infrastructure health monitoring has been extensively studied. This section covers some of the existing proposals that focus on WSNs monitoring the infrastructure health of bridges.

In [Whelan et al., 2007] a WSN consisting of 20 sensor nodes is deployed on a road bridge to gather accelerometer and strain data. Nodes are assigned a sequential time offset based on their local addresses to enable them to transmit without collisions. Although TinyOS is used as the operating system, low level software is programmed to achieve higher data throughput. An actual deployment of a WSN for railway bridge monitoring is described in [Bischoff et al., 2009]. The WSN consists of 8 nodes that are deployed on the bridge and collect strain information whenever a train crosses the bridge. The network self-organizes as a routing tree to relay the information to a sink node. The information is then relayed from the sink node to the remote control centre using UMTS. In [Aboelela et al., 2006] a WSN is used to monitor railway track status. Sensor devices are hierarchically organized with redundant paths. Multi-path routing is used to send the information to the remote base station. Fuzzy logic techniques are employed to aggregate data collected. In BriMon [Chebrolu et al., 2008] a wireless sensor network composed of Tmote-sky devices is deployed on a railway

bridge. Information is collected by nodes and retransmitted to the train that acts as a mobile sink node. The routing protocol forms a tree rooted at the head node of the WSN by periodically transmitting a message which is flooded down the WSN. The feasibility of the mobile data transfer from the WSN to the train is studied by means of an experiment that only takes into account the mobile head node and the WSN head node. Other real deployments of WSNs on road bridges are presented in [Lee et al., 2007] [Kim et al., 2007] [Lynch et al., 2006] and [Kundu et al., 2008].

Although, some of the these approaches and the proposal covered in this paper share some commonalities there are some important differences. Most of these proposals concentrate on the sensor processing part and a great number of them lack a general purpose routing protocol. BriMon, is the only one to take the data muling technique into account. Although it studies many different issues and aspects of the application by means of isolated testing of components in the system, no general testbed is mentioned in the paper. In our work we have simulated the application scenario as a whole including mobility and the mobile data transfer protocol. In addition, unlike other proposals, we make use of a middleware layer to automatically handle QoS requirements and to simplify the task of developing the application. Finally, other proposals use an application specific design whereas the use of PS-QUASAR allows us to have a more generic design. This in turn, allows us to add more nodes to the WSN, for example nodes to cover new sections of the bridge without having to reprogram already deployed nodes.

4 PS-QUASAR middleware

PS-QUASAR is a middleware for WSANs that offers a high level simple programming model based on the publish/subscribe paradigm. The publish/subscribe programming model provided by PS-QUASAR is really simple and easy to use (Figure 1). The simplicity of the model helps developers to implement WSAN applications without having to worry about common low-level issues such as data packet encoding/decoding, message handling, etc. In this model all nodes in the network are aware of the existing subscribers and can become publishers of each of the topics.

The proposed publish/subscribe programming model is based on two different mechanisms: publish/subscriber primitives and listeners. The publish/subscribe primitives allow information to be transparently sent from publishers to subscribers. These two entities can be located in different nodes or in the same one. In the proposed publish/subscribe model, the QoS requirements are only specified on the publisher's side. This simplifies the task of delivering information and avoids time-consuming QoS-matching algorithms. The QoS parameters offered are deadline, reliability and priority. Listeners are functions that are executed whenever a message is received by a subscriber. Only subscribers can make use of listeners to process received data. Listeners are specified as a parameter in the ps_subscribe method.

```
// SUBSCRIBER
ps_subscribe(char* topic_name, listener_function listener);
ps_unsubscribe(topic_type topic_id);
ps_unsubscribe(topic_type topic_id, listener_function listener);

// PUBLISHER
ps_publish(char* topic_name, char* data,
           unsigned short data_size, struct QoS_policy qos_policy );

Qos_policy{
    // Deadline expressed in ms
    Number deadline;
    // Indicates the number of retransmissions
    Number reliability;
    Number priority;
}

// OTHERS
ps_notify_listener(char *topic_name, char* data_to_listener, rimeaddr_t* addr);
ps_print_status();
```

Fig. 1. PS-QUASAR programming model API

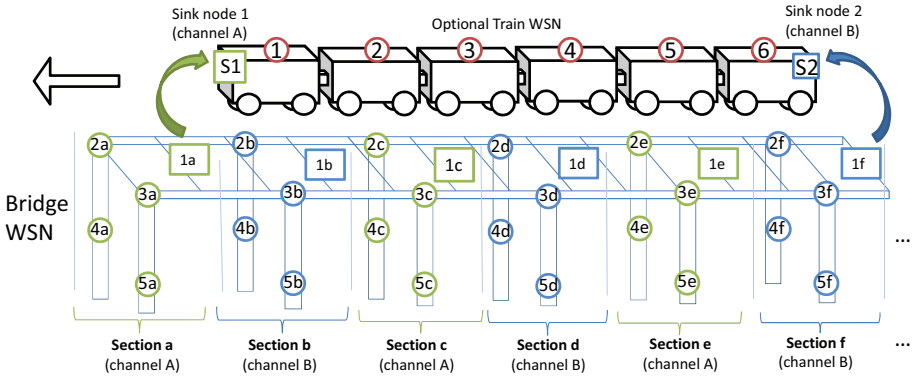


Fig. 2. Architecture of the application prototype

The PS-QUASAR middleware is composed of three different modules: PS-QUASAR Maintenance Protocol, PS-QUASAR Routing Module and the API. Figure 3 shows the PS-QUASAR module diagram and how the different modules connect to each other. The maintenance protocol is in charge of creating the links between neighbor nodes and discovering subscribers and publishers. The routing module carries out the actual routing process based on the information collected by the former protocol. The middleware uses a directed acyclic graph based routing protocol that supports a many-to-many communication and can handle priority, deadline and reliability requirements in the communication between nodes. The protocols are fully distributed and multicasting techniques are used to improve communication between nodes. The API, on the other hand, provides a

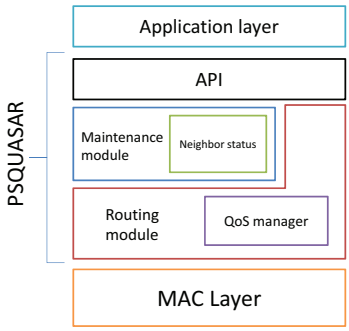


Fig. 3. PS-QUASAR module diagram

set of methods for developers to make use of the publish/subscribe programming model offered by PS-QUASAR.

In the context of our application scenario, the use of such high level abstraction significantly simplifies the task of collecting sensed data. At the same time, the middleware, if specified, provides reliable communication by means of retransmissions. More details on the middleware can be found in [Chen et al., 2013].

5 RAISE architecture

The general architecture of the application is depicted in Figure 2. The application scenario consists of a WSN deployed on a railway bridge (referred to simply as the bridge WSN for the rest of the paper) and sink nodes deployed on the trains, passing through, which will collect the information sensed by the bridge WSN. This WSN gathers important data about the structural health of the infrastructure such as vibrations and strain. An optional WSN could also be deployed inside the train to monitor abnormal situations as the train travels over the railway line or for the whole itinerary (train WSN). This information (vibration, temperature, material deformation,...) on the carriages' health can be tracked to detect problems in the train.

In the bridge WSN a set of different nodes are deployed along the railway infrastructure. Let us note that nodes are not only deployed on the railway tracks but also inside the structure itself so infrastructure aging and possible incidents can be detected. The goal of the network is to self organize to sense data whenever a train passes by and use the next train as a data mule to upload the sensed data. The data sensed at the bridge is transferred to the train by means of sink nodes, labelled S1 and S2 in Figure 2.

The first application prototype was developed with a single WSN that contained all the sensor nodes in the bridge. The tests carried out in this application prototype showed disappointing results in terms of reliability. In this first approach, a single node acts as head node of the network and collects the information that is sent by the rest of nodes. Since the reliability is significantly affected

by the distance between source and destination and by neighborhood traffic, in this scenario where a single WSN contains all nodes, collisions are frequent. As a result, reliability was shown to be around 70%-80% in our preliminary tests.

In order to increase performance clustering techniques need to be used. Nodes along the bridge are divided into independent sections (labeled as section a, b, c, ... in Figure 2). Consecutive sections operate on different channels, namely channel A and B, so there is no interference between them. In our prototype only two channels have been used, but a greater number of channels could be used if a higher throughput is desired in the data muling process as explained in Section 5.3. The use of separate sections reduces the maximum distance between nodes and the network traffic thereby improving network energy consumption and reliability.

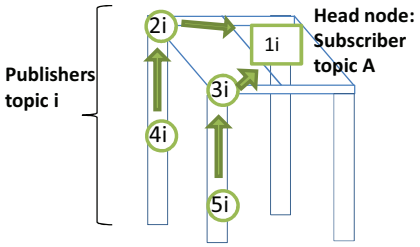


Fig. 4. Organization of a network section: a single head node collects the information sent from other nodes

For each Section i of the bridge WSN, node $1i$ subscribes to information on topic S_i. The rest of the nodes in Section i publish information on topic S_i. Figure 4 shows the connections between the nodes in a section (determined by the node range and the location where they are deployed). In the case study prototype, each section is composed of a total of 5 nodes. All these nodes (including the head node) participate in sensing data but only the head node communicates with the train to upload the sensed data. This organization of the section (tree-based) has been chosen because it minimizes neighborhood interference and therefore improves the reliability of each section. Application developers are not directly aware of the routing protocol, nor the network organization, that is, the middleware automatically delivers the information. This allows them to add or remove nodes from each section on-the-fly, even in other topologies distinct from the tree one used in this prototype. The sensed information collected in the head nodes is stored until the next train passes by. In that moment, the data muling protocol will start uploading the information to the train.

Overall the application has three different modules: sensing module, collecting module and data muling module. Figure 5 shows the relationship between each of the modules and the trains's schedule. Sensing and collecting modules run on all nodes in each section whereas the data muling module is only used in

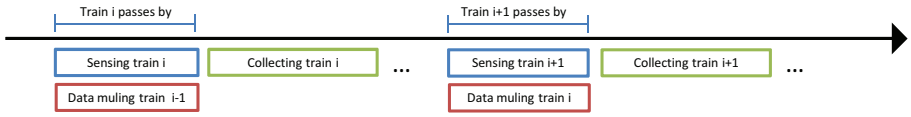
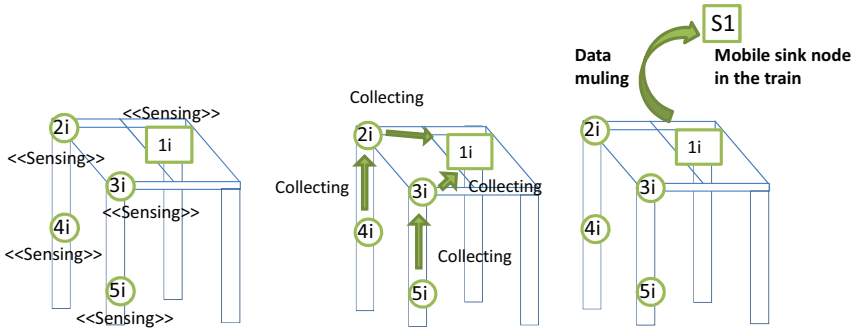


Fig. 5. Train schedule and the execution of the different modules

head nodes. These three modules are explained in Sections 5.1, 5.2 and 5.3 and depicted in Figures 6(a), 6(b) and 6(c), respectively.



(a) Sensing module (b) Collecting module (c) Data muling module

Fig. 6. Different modules of the application running in the nodes

5.1 Sensing

The sensing module retrieves data as trains pass by. It is far more useful to gather the data when the train is passing through as this provides real information on how the infrastructure behaves when it is actually in use. This can be used to detect abnormal vibrations or material deformation which indicates that the health of the infrastructure has been compromised. The frequency rate at which to sample depends on the information to be collected. Since the information gathered by each sensor is simulated in our application prototype we have assumed a sampling rate of 2Hz and a sample size of 2 bytes. Nodes in the bridge WSN are instructed to start sensing whenever a train passes by. To do that, the application needs to identify whenever a train is approaching the bridge in order to start collecting data. In tests presented in Section 6, the command to start sensing is given by the simulation script each time a train passes by. In an actual deployment there are several alternatives that can be used to detect a nearing train. BriMon [Chebrolu et al., 2008] for example, suggests the use of frontier nodes which are nodes placed upstream of the sensor network to detect nearing trains in time to notify the rest the network to start sensing. Another option would be to use accelerometers to detect vibrations coming from approaching

trains. In the same way, when a train leaves the bridge, nodes are instructed to stop sensing.

All information sensed in each node is stored in a data cache. This data cache will be accessed by the collecting module.

5.2 Collecting data

The application collects data by default where there are no trains on the bridge. The collecting module in each node sends the information stored in the local data cache to the head node of the section. For example, in Figure 4 all nodes send data packets, containing the sensor readings, to node 1 whenever there is data in the local data cache. To do this, sensor nodes call the *publish* primitive of the PS-QUASAR middleware and it automatically handles the delivery. Packet payload is filled with as much data as possible, from the local data cache in order to minimize the number of packets to be sent. In order to be sure that data is delivered, communication is reliably configured. This is achievable by using an additional parameter in the *publish* primitive that accepts QoS requirements. The collection module is programmed to use retransmissions with ACKs in order to confirm that data has been delivered. When the module is enabled, data cache is periodically checked and if it contains something then a packet is sent to the head node. The data collected by each section will not be transferred until the next train arrives so packet delay is not a concern. Therefore, data in the cache does not need to be sent immediately to the head node. For each node, a collecting period of 1 second has been chosen between consecutive transmissions of data to the head node. Figure 7 shows the pseudocode of the sensing and collecting data modules, respectively, using the API shown in Figure 1. Head nodes call the *ps_subscribe* method to express their desire to receive all the information associated with a topic. Nodes in the same section use the *ps_publish* method to send the information on that same topic. The third parameter of the method establishes that the information needs to be sent reliably. The middleware automatically delivers the information to the corresponding subscribers.

5.3 Data muling

Once the information has been collected in the head node the next train that passes by will be used as the data mule to get the information from it. This module only runs on head nodes which are the only nodes that communicate with the train. The module is executed concurrently together with the sensing module in the head nodes. The module basically starts sending data packets from the data muling cache to the train whenever a passing train is detected.

Two issues need to be tackled in the mobile data transfer. The first one is to reliably send the information to the train and the second to send it at a speed that allows all sensed data to be uploaded to a single train. The first one is solved by using reliable transmission based on ACKs. The second depends on different parameters such as the train speed and the hardware used to transmit

```
Sensing module

Every SENSING_PERIOD:
    Sample s = get_sample_from_sensors();
    save_to_data_cache( s );

Collecting module – head node
At initialization
   Ps_subscribe( topic_i, handle_data_listener );

Function handle_data_listener( Address src_address, Sample data){
   save_to_data_muling_cache( data );
}

Collecting module – sensor node
Every COLLECTING_PERIOD:
   if( data_cache_not_empty ){
       Sample sample_collection[] = get_samples_from_data_cache();
       ps_publish( topic_i, sample_collection, qos_reliable );
   }
```

Fig. 7. Pseudocode of collecting and sensing modules for Section i

the data (node radio range, data rate, ...). In order to further increase the throughput of the proposed data muling protocol multiples sink nodes are used. In our prototype two sink nodes have been used in order to double the transfer rate of the protocol but a higher number of sink nodes could be used if necessary, for example if the sampling rate required is higher. The radio range of each head node is not assumed to be higher than the one for normal sensor nodes. In Figure 2, for example, sink node S1 collects data from head nodes 1a, 1c and 1e while sink node S2 does the same from head nodes 1b, 1d and 1f. The results presented in Section 6.2 show that the proposed data muling protocol is feasible.

6 Evaluation

The complete case study application has been implemented with the settings described in Section 6.1. The results in terms of reliability and quantity of data generated by the WSN are shown in Section 6.2. The mobile data transfer results are discussed in Section 6.3. Finally the power consumption is presented in Section 6.4.

6.1 Environment set-up and scenario settings

The application scenario has been implemented in C programming language for the Tmote-sky motes running the Contiki operating system [Dunkels et al., 2004]. Table 1 shows the main features of these motes. Power consumption in the table and in the rest of the tests has been calculated using the energest module [Dunkels et al., 2007] provided by Contiki OS. The resulting code has been simulated using the Cooja simulator [Osterlind et al., 2006]. The Cooja simulator emulates Tmote-sky motes at machine code instruction set level. The communication model takes into account packet loss when nodes are transmitting at

Attribute	Value
Processor	MSP430 8MHz
Radio	CC2420 802.15.4 compliant
Battery	2 AA batteries
Power consumption	Sending: 59.1 mW Receiving: 52.2 mW CPU: 5.4 mW LPM: 0.1635 mW
Operating system	Contiki OS

Table 1. Tmote-sky specifications

the same time, namely collisions are taken into account in the simulation. The Contiki test editor plugin has been used to control the simulation and to actually simulate the movement of the train. This plugin allows users to control many different settings of the scenario, such as node position, at different instants of time. This feature has been used to actually recreate the movement of the train passing through the bridge WSN. Nodes have been deployed as depicted in Figure 2, that is 30 nodes divided into 6 sections of 5 nodes each. The script simulates 20 trains passing over the bridge, one every 60 seconds. Each train moves at a speed such that the sink nodes on the train are in range of each head node for around 7 seconds. For example, this means that if both head nodes and sink nodes in the train have a radio range of 50 metres the train is travelling at a speed of 100 km/h (assuming ideal conditions).

6.2 Reliability and data generated by the bridge WSN

Attribute	Value
Sensing rate	2Hz
Sample size	2 bytes
Mean data generated for each train in one section	706 bytes
Data collection reliablity achieved	100%
Data collection mean number of retransmissions	1.076
Data collection maximum number of retransmissions	7

Table 2. Data generated and reliability

Results obtained in the tests are summarized in Table 2. One section generates on average 706 bytes everytime a train crosses the bridge. The total amount of sensed data has been received by each head node which gives a reliability of 100%. Although the maximum number of retransmissions carried out by a sensor is 7 the mean number of retransmissions is 1.076 which means that almost no

retransmissions have been carried out. Also, it shows that even in networks with low traffic it is really difficult to obtain 100% reliability without using techniques such as retransmissions. This leads us to believe that simulators which do not take collisions into account do not produce realistic results.

6.3 Data muling

In the tests carried out each of the trains receives the readings sensed when the previous train was crossing the bridge. Each train is in range with each head node for approximately 7 seconds. The information sensed for each of the 20 trains in the test has successfully been received by the sink nodes S1 and S2. The reliability achieved is 100% because ACKs have been used to confirm the reception of data packets. To do that the runicast library provided by Contiki OS has been used. The data muling transfer rate from the bridge WSN to nodes S1 and S2 for each train is 0.668 Kbps and 0.665 Kbps. That means that the mean data muling transfer rate for the whole system is 1.334 Kbps. Although in the application scenario all packets have been successfully transferred to the train the data muling transfer rate is really low compared to the maximum data rate of the mote (around 45 Kbps). Several factors may have influenced this data rate drop. First, the head node also carries out the sensing task at a rate of 2Hz which slows down the data muling process. Head nodes can be programmed not to carry out sensing if a higher data rate is needed in the head nodes. Also, the operating system and the retransmission mechanism introduces some latency, especially when ACKs have not been received (the radio has to wait a predefined time if no ACK has been received before sending a retransmission). Finally, the radio range of the head nodes is assumed to be relatively short (i.e. 50 metres if the train moves at a speed of 100 Km/h). By extending the radio range of head nodes the data muling transfer rate can be easily increased.

6.4 Power consumption

	Head node	**Normal node**	**Sink node**
Sensing and data muling	9.392	0.971	1.812
Collecting	1.950	1.832	

Table 3. Power consumption (mW)

The power consumption of each kind of node has been measured and is shown in Table 3. This power consumption can be compared to that presented in Table 1 for the different modes of the mote. Power consumption during the data collection is relatively low, 1.950 mW and 1.832 for head nodes and normal nodes, respectively. Power consumption of the sink nodes is also low, although energy consumption in sink nodes is not a concern because they can be powered as they are located on the train. During the sensing and data muling processes, head

nodes have the highest energy consumption since all the information gathered by the network needs to be transmitted by them. However, this only happens when trains are crossing the bridge which constitutes a really short amount of time compared to the amount of time the head nodes are collecting information.

7 FUTURE WORK

The results obtained in the test with the application prototype suggest that the application scenario is actually feasible. However, there are still open questions that need to be tackled, such as which specific sensors to use in the sensor nodes and how the way in which they are deployed can affect the accuracy of the readings. There are also several issues and behaviors that have not been captured by the simulators such as the influence of the bridge's infrastructure or the speed of the train on the performance of the sensor radio that require further consideration. This paper, however, can be used as a starting point from which to consider all these unanswered questions. In addition, the use of PS-QUASAR has proven to be invaluable as it automatically handles the QoS requirements specified at the application layer and substantially simplifies the task of programming WSN applications. Based on the results obtained, we believe PS-QUASAR is suitable for a wide range of applications in the context of CIP.

8 CONCLUSIONS

A railway infrastructure health monitoring application that uses WSNs has been presented in this paper. The WSN collects information about the structural health and behavior of the infrastructure when a train travels along it and relays the readings to a base station. The base station then uses the next train(s) as a data mule to upload the information. The WSN makes use of a publish/subscribe based middleware called PS-QUASAR to significantly simplify the task of developing the application and to allow new nodes to be added on-the-fly. Other techniques used to minimize packet loss, mainly due to collisions, are packet caching, data fusion and clustering. The evaluation carried out shows that the mobile data transfer is actually feasible and that the results obtained are satisfactory, both in terms of reliability and power consumption.

Acknowledgments. This work was supported by the Spanish Project TIN2011-23795 WiCMaS:Wireless based Critical Information Management Systems.

References

Aboelela et al., 2006. Aboelela, E., Edberg, W., Papakonstantinou, C., and Vokkarane, V. (2006). Wireless sensor network based model for secure railway operations. *Performance, Computing, and Communications Conference, 2002. 21st IEEE International*, 0:83.

Akyildiz et al., 2002. Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer Networks*, 38:393–422.

Bischoff et al., 2009. Bischoff, R., Meyer, J., Enochsson, O., Feltrin, G., and Elfgren, L. (2009). Event-based strain monitoring on a railway bridge with a wireless sensor network. In *4th International Conference on Structural Health Monitoring of Intelligent Infrastructure (SHMII-4)*.

Chebrolu et al., 2008. Chebrolu, K., Raman, B., Mishra, N., Valiveti, P. K., and Kumar, R. (2008). BriMon: A Sensor Network System for Railway Bridge Monitoring. In *The 6th Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*.

Chen et al., 2011. Chen, J., Díaz, M., Llopis, L., Rubio, B., and Troya, J. M. (2011). A survey on quality of service support in wireless sensor and actor networks: Requirements and challenges in the context of critical infrastructure protection. *Journal of Network and Computer Applications*, 34(4):1225 – 1239.

Chen et al., 2013. Chen, J., Daz, M., Rubio, B., and Troya, J. M. (2013). Ps-quasar: A publish/subscribe qos aware middleware for wireless sensor and actor networks. *Journal of Systems and Software*, 86(6):1650 – 1662.

Dunkels et al., 2004. Dunkels, A., Grnvall, B., and Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA.

Dunkels et al., 2007. Dunkels, A., Osterlind, F., Tsiftes, N., and He, Z. (2007). Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th workshop on Embedded networked sensors*, EmNets '07, pages 28–32, New York, NY, USA. ACM.

Gaura et al., 2010. Gaura, E., Girod, L., Brusey, J., Allen, M., and Challen, G. (2010). *Wireless Sensor Networks, Deployments and Design Frameworks*. Springer.

Kim et al., 2007. Kim, S., Pakzad, S., Culler, D., Demmel, J., Fenves, G., Glaser, S., and Turon, M. (2007). Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 254–263, New York, NY, USA. ACM.

Kundu et al., 2008. Kundu, S., Roy, S., and Pal, A. (2008). A power-aware wireless sensor network based bridge monitoring system. In *Networks, 2008. ICON 2008. 16th IEEE International Conference on*, pages 1 –7.

Lee et al., 2007. Lee, R.-G., Chen, K.-C., Lai, C.-C., Chiang, S.-S., Liu, H.-S., and Wei, M.-S. (2007). A backup routing with wireless sensor network for bridge monitoring system. *Measurement*, 40(1):55 – 63.

Lynch et al., 2006. Lynch, J. P., Wang, Y., Loh, K. J., Yi, J.-H., and Yun, C.-B. (2006). Performance monitoring of the geumdang bridge using a dense network of high-resolution wireless sensors. *Smart Materials and Structures*, 15(6):1561.

Mottola and Picco, 2011. Mottola, L. and Picco, G. P. (2011). Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, 43(3):19:1–19:51.

Osterlind et al., 2006. Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006). Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641 –648.

Spanish Official Bulletin of the State (BOE), 2005. Spanish Official Bulletin of the State (BOE) (2005). Instrucción sobre las inspecciones técnicas en los puentes de ferrocarril (itpf-05). fom/1951/2005.

Whelan et al., 2007. Whelan, M. J., Fuchs, M., Gangone, M. V., and Janoyan, K. D. (2007). Development of a wireless bridge monitoring system for condition assessment using hybrid techniques. In *Proceedings of SPIE, the International Society for Optical Engineering*, pages 28–32.