TrainSense: A Novel Infrastructure to Support Mobility in Wireless Sensor Networks

Hugues Smeets, Chia-Yen Shih, Marco Zuniga, Tobias Hagemeier, and Pedro José Marrón

Universität Duisburg-Essen, Germany
{hugues.smeets,chia-yen.shih,marco.zuniga, tobias.hagemeier,pjmarron}@uni-due.de

Abstract. In this paper, we present *TrainSense*, a novel infrastructure to support the development and testing of mobile sensing applications. TrainSense merges a mote and a model train into a single mobile unit, and enhances the basic model train infrastructure with several important features required for the evaluation of mobile scenarios. First, we develop a *real-time* controller to send control packets to model trains and motes to manage the network topology. Second, we design and implement a positioning system with centimeter precision. Third, we use the power available on the tracks to provide unlimited energy to the motes. Fourth, we provide a way for the motes to dock into a custom USB port, for reprogramming and data download. We evaluate TrainSense in two ways: (i) we establish the correctness of the implementation and measure the performance of its components, and (ii) we demonstrate its practical use with two sample wireless sensor network application scenarios: self-deployment and data muling.

1 Introduction

Several studies highlight the fundamental role that mobility plays on the performance of Wireless Sensor Networks (WSNs). Mobile nodes can, for example, increase the communication capacity [12], enhance the sensing coverage [19] and facilitate network deployment [26]. But in order to achieve these benefits, the underlying mobile infrastructure has to provide some important features. Mobile WSNs need (i) advanced *topology management* capabilities, i.e. the ability to specify, simultaneously, the speed and direction of each individual node; (ii) the *ability to track and localize* nodes; (iii) a *reliable source of energy* to avoid unnecessary pauses or abrupt stops; and (iv) *automatic operations* for re-programming and data download, to minimize the amount of manual overhead.

Robotic ground vehicles can provide several of the features mentioned above, and hence, they have been the primary choice to enable WSN mobile applications ranging from mapping and navigation [17] to search and rescue [16]. However, before a robot-based infrastructure is "WSN-ready", some technical challenges need to be tackled such as, path scheduling [25, 13], localization [8], energy budgets, and deciding on suitable hardware platforms to sustain the particular demands of the application [23, 3]. Depending on the researcher's expertise, overcoming these challenges may not be an easy task.

Our aim is to propose an alternative platform to evaluate mobile WSNs applications. We believe that model trains have the necessary capabilities to support the development and evaluation of such applications. The state of the art in model trains provides sophisticated hardware for topology management. Train controllers can address multiple trains, through the tracks, to modify their individual speed and direction. This basic infrastructure has important characteristics that can be leveraged. The tracks can be used (i) to provide infinite energy to the mobile motes, (ii) to detect the position of nodes and (iii) as a back-up channel to send information to the motes. These features can help in testing various mobile scenarios. For example, a pre-deployment evaluation requiring a systematic study of the node positions could use TrainSense to move nodes sequentially to various locations and identify the deployment or mobility pattern that maximizes the metric of interest. The application could also use the tracks to simulate node failures by sending on/off commands to the motes, without interfering with the normal operation of the experiments.

In this paper, we propose TrainSense. The main idea of TrainSense is simple, but powerful: to integrate a model train and a mote into a single unit and to utilize the integrated mobile node along with the model train infrastructure for WSN applications. The main contribution of our work is the enhancement of the typical model train platform with the following new features to offer mobility support: *real-time train control*, *precise train positioning*, *energy management*, *automatic train-mote operations* and *back-channel information support.*

2 TrainSense Infrastructure

In this section, we describe the basic model train infrastructure, and then, we provide a general overview of TrainSense.

2.1 The Basic Model Train System

The technology to control model trains digitally is available since 1986 and it was introduced by Maerklin as the Maerklin Digital System. Figure 1 outlines a basic off-the-shelf train system. This basic system consists of five main components: (i) **the host computer**, which provides a user friendly interface to define the train movements, (ii) **the controller**, which sends packets through the tracks to control the speed and direction of locomotives, and to control the turnouts, (iii) **the detector**, which identifies the presence of trains when they reach a particular track section and reports these events to the controller, (iv) **the tracks**, which carry power and data to the trains, and (v) **the locomotives**, each one having a decoder to process the commands from the controller.

The user has significant freedom to design a track layout. The various track elements can be combined into practically any shape with the only constraint being a minimum curve radius of 36 cm. To make the system operational, some

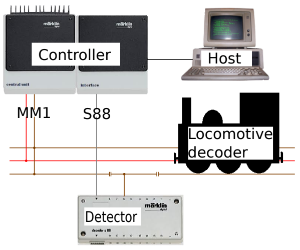

Fig. 1. The basic off-the-shelf train control system

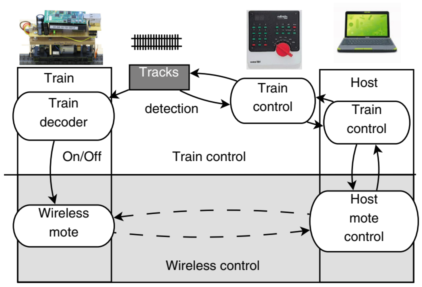

Fig. 2. The TrainSense architecture

simple wiring is required to connect the controller, tracks and detectors. The controller can be configured to handle simple automatic movements such as a train traveling between two points. If more complex mobility patterns are required, a computer with the appropriate software can be connected to the controller through an RS232 interface. A typical commercial system can control multiple trains and select their routes. It can also localize and position trains with a coarse precision, about 10cm. While these systems are well designed to reproduce the behavior of life-size trains, they miss important functions for the evaluation of WSNs. In the next section, we provide a broad description of the enhancements that we made to the basic system for supporting mobile WSNs.

2.2 TrainSense Architecture

Our TrainSense architecture introduces changes to the basic system at two levels: train control and wireless control levels (see Figure 2).

At the train control level, TrainSense includes the basic system with the typical wiring required for standard components. There are, however, two important differences. First, the hardware and software of the controller and the detection mechanism have been re-designed to meet the requirements of mobility-enabled wireless sensor networks. Second, we developed new software for the host computer to allow more elaborated mobility patterns, as detailed in Section 3.1.

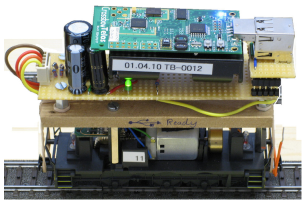

Fig. 3. A TrainSense mote

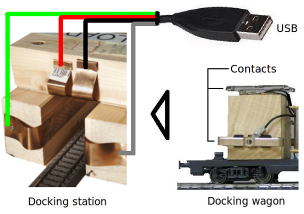

Fig. 4. The docking station

At the wireless control level, we introduced three new components for WSN applications: (i) the TrainSense mote, (ii) the host mote and (iii) the docking station. As illustrated in Figure 3, the TrainSense mote is composed of four main parts: a locomotive with its decoder, a mote, a custom decoder for the mote to receive data from the tracks, and the on-board power supply. To have a closed control loop over the train mobility, **the host-mote** connected to the host computer allows the mobile motes to send packets to change the speed and direction of other trains. Our infrastructure is operating system agnostic and can be used seamlessly with TinyOS or Contiki, for example.

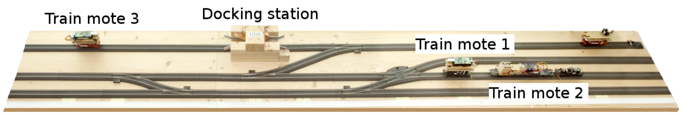

Fig. 5. The main segment of TrainSense

Finally, the **USB docking station** (see Figure 4) provides the same functionality as the wired USB connection, and has an effective bandwidth of about 97% of its wired counterpart. While connected to the docking station, a TrainSense mote can be reprogrammed or data can be downloaded automatically. Figure 5 illustrates the physical construction of a segment in TrainSense with the docking station and some motes.

3 TrainSense Features

TrainSense has several important features to support mobile WSNs. We now describe in detail their implementation and evaluation.

3.1 Real-Time Train Control

Real-time train control is the key feature to enable flexible, interactive topology management. Together with the host program, the TrainSense controller offers the possibility to communicate with motes through the tracks. It provides real-time guarantees for fine-grained topology management and accurate positioning.

Train Control Host Program. In WSNs, mobility patterns could be rather complex and a node may be required to make precise stops, or to change direction rapidly. For this purpose, we developed a new host software with various mobility primitives to support more complex mobile scenarios. The host program allows the user to specify commands for the train movement control and to define a handler action in response to detection events.

The Real-time TrainSense Controller. Several standards are available for model train communication. Our controller is based on the Maerklin/Motorola standard [1] because (i) it is well documented, (ii) the latest formats are backward compatible, (iii) the hardware for this format is readily available, and (iv) it is simple to implement. The controller can address 80 mobile trains and control 256 turnouts. The standard defines two main types of packets that can be sent through the tracks: packets containing addresses, speed information and direction commands (for trains), and packets for the turnouts to switch left or right. The train packets take less than 20 ms to reach a train. The data rate of the turnouts packets is twice as high, requiring only 10ms to reach a turnout.

The commercially available train controllers are not real-time systems, which is problematic since we need a short upper bounded time to detect events in order to position trains precisely. Therefore, we designed our TrainSense controller to provide real-time detection and train control. The control services that can be guaranteed to be performed in real-time include: setting locomotive speed, turnout direction, trigger upon a detection event, and getting the time between the last triggered action and the monitored event. To facilitate network management, our controller can send new types of packets to configure the motes.

3.2 Precise Positioning

TrainSense provides precise positioning of motes at user-specified locations. Conceptually, the basic detection mechanism works as follows: when a train crosses a point that has a detector, the wheels of the train create a short circuit between the two rails. This event is registered in a hardware register. The controller continuously polls the detector-registers and identifies if a train reached a detection point. In regular train model applications this method is used to stop the train or to switch the direction of a coming turnout.

We extended this detection method with a dead-reckoning[1] technique to position nodes anywhere on the track: if a node crosses detector x at time t with speed s, the node can be positioned at $x + \Delta x$, by waiting $\frac{\Delta x}{s}$ seconds and then sending a "stop" packet to the train. However, for dead reckoning to be accurate, the controller must provide short and bounded delays for register reading and for data packet delivery.

To provide the required real-time guarantees, the TrainSense controller assigns the highest priority to data packet transmissions, and the second priority to register reading. Reading each register takes 200 μs, hence if there are n detectors, the controller requires $n \times 200\mu s$ to read all the registers. Usually, data transmission operations take 10% of the CPU time. Considering a train moving at the maximum speed of 0.5 m/s, the maximum positioning error introduced by the controller is $[(0.2n)1.1 + 20]ms \times 0.5m/s$. Assuming 50 detectors, the maximum error is $\approx$ 1.5 cm.

[1] In the rest of this paper, "dead reckoning" refers to the process of estimating the position of a mote by having it moving at a constant speed when it crosses the initial position and letting it run for a determined time to position it at a determined distance from that initial position. This requires starting the mote ahead enough of the initial position.

One issue with dead reckoning is the potential of cumulative errors, that is, the longer a mote travels before re-calibrating its location to a ground truth, the higher the positioning and localization error can be. The detectors and the tracks play the role of re-calibrating the dead reckoning estimation. Depending on the availability of a detector near the train, there are two types of positioning scenarios in TrainSense: detector-based or dead-start[2].

Detector-based Scenario. In dead reckoning, it is assumed that the train speed is constant, which is not always true. When a train starts from a still position, it takes approximately 1 to 2 seconds for the train to reach its nominal speed. This initial period introduces an error in dead reckoning. We eliminate this error if we can make the train reach its nominal speed before it reaches the detector used for the dead reackoning. To evaluate the positioning precision of that scenario, we performed the following set of experiments: a train was positioned 20 cm before a detector, which allowed sufficient time for the train to reach its nominal speed. After the controller detected that the train had crossed the detector, a stop packet was sent to position the train at $i \times s$ cm, where s represents the nominal speed of the train in cm/s and i represents a period of time that goes from 1 sec to 8 sec in steps of 1 sec. The tested nominal speeds are: 5, 7, 10, 15 which were translated to 12, 19, 30 and 37 cm/s. At each nominal speed, each point was measured 30 times.

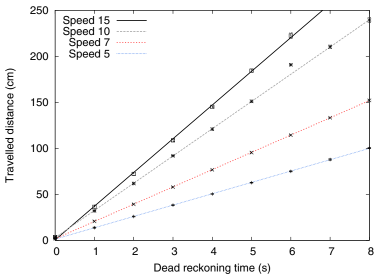

Fig. 6. Positioning the mote using the detectors and dead reckoning.

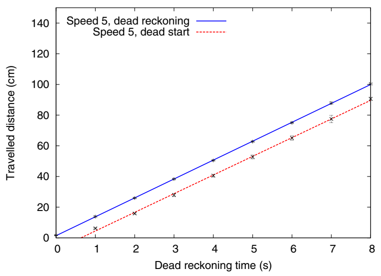

Fig. 7. Positioning the mote using dead start versus dead reckoning

Figure 6 shows the accuracy of the positioning method. Each point represents the average of the reached positions. Each line shows a linear regression on all the points corresponding to a given speed setting. There are two important points to highlight: first, for most points, the margin of error is below 2cm, and the maximum deviation, a bit more than 2cm, was obtained at the maximum distance traveled, due to cumulative errors of dead reckoning. Second, occasionally, the speed of the train changes and causes larger errors than expected (see

[2] When dead reckoning is not possible, the mote starts from zero speed, reaches constant speed after a certain time and is left moving at that speed for a determined time. We call this technique "dead start".

point 6 for speed 10). We found that this happens due to manufacturing errors, in some occasions the motor stutters and changes its nominal speed. This is not a common event, but it occurs and its effect on the accuracy of dead reckoning is within 10cm (for the distances and speeds measured). The solution is to have several detectors in the infrastructure to correct inaccuracies.

Dead-start Scenario. If there is no detector before the target point, or if the detectors are not far enough to allow the mote to reach its nominal speed, then TrainSense switches to a dead start scenario. The likelihood of dead-start events depends on the density of detectors in the infrastructure, the more detectors, the less frequent the event. Figure 7 shows a comparison between two scenarios for speed 5. We found that trains had more difficulty in overcoming their inertia at lower speeds. The evaluation process was the same as the one for dead reckoning but without using a detector. The mote was placed at position 0 and control packets were used to start and stop the train. We observe that dead start had a **constant** negative offset of about 10 cm, with respect to dead reckoning. This effect can be canceled by letting the train run for a longer period when no detector is found during the positioning trip (a 1 sec delay for this speed). The precision is therefore also 2cm, but the resolution is limited to 10cm. Dead reckoning is needed for the best resolution.

3.3 Energy Management

Energy consumption has been the main limitation of mobile networks. Model trains allow us to leverage the tracks to provide a constant energy source to the motes. However, several issues need to be solved. First, to extract energy from the tracks, we built a classic AC to DC converter to convert the voltage to 5V from an alternative voltage of ± 18V. The converter resides on the **on-board power supply** unit. It takes the track voltage as input, and regulates and buffers the output to power the mote through the mote's USB interface. The schematic of the whole on-board electronics (power supply, mote control and mote) is shown in figure 8.

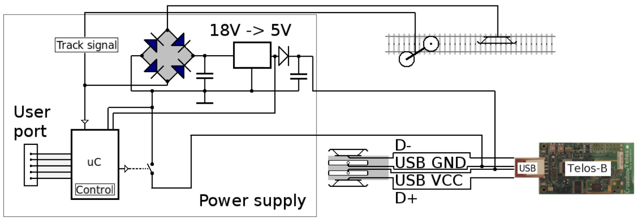

Fig. 8. The schematic of the power supply

The second issue is the lack of constant connectivity, i.e., if the tracks become dirty, the power supply is temporarily cut off, reseting the mote. To guarantee a constant power supply, we use gold-caps, which are small capacitors with very high capacity to make the power supply resilient to contact problems. We must ensure that the power supply is not affected by the short disruption period caused by dirty tracks. To assess the reliability of the power supply, we measured the duration of operation without power from the tracks during which we can still provide a stable voltage to the connected mote. Given that communication is the most energy-consuming activity in a mote [28], the evaluation is done with the radio continuously on. We programmed a mote to send packets constantly at the minimum and maximum power levels: -20dBm and 0dBm, respectively. The mote was placed at approximately 2 meters from the host-mote. The host-mote measured the RSSIs of the received packets. During the experiments, we measured the output voltage of our on-board power supply with an oscilloscope. Once the mote was positioned on the tracks, we charged the gold capacitors for a minute using the baseline mode (i.e., a constant 18V in the track with no activity on the tracks). Then, we disconnected the power supply from the tracks. The results are shown in figure 9. As we can see, even at the maximum transmission power level, the capacitors provided a lifetime of 1.5 minutes after disconnection (the point at which the RSSI drops drastically). For the lowest power level, the lifetime was almost 3 minutes. Usually, power disconnections caused by dirty tracks occur for at most fractions of a second, and hence, the on-board power supply guarantees continuous power to the mote.

Another useful feature is the **on-board mote control**, which allows turning the motes on and off remotely. This feature can simulate a full or an empty battery. An 89C51 Atmel controller decodes the address field in the track signal. If it matches its own, it reads the function bit: a one/zero turning the mote on/off, respectively. Our controller allows making use of the data fields to include other commands e.g., measuring the mote's power consumption.

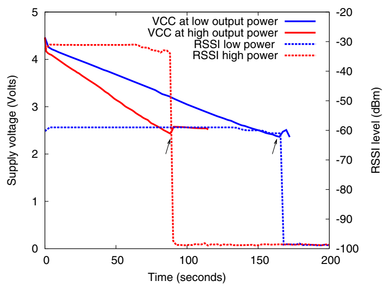

Fig. 9. The power supply voltage

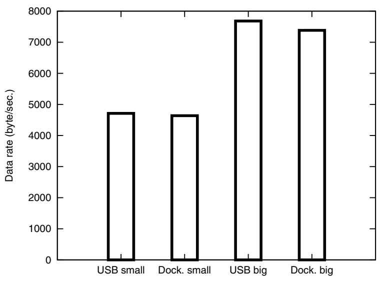

Fig. 10. Docking vs. Direct USB

3.4 Automatic Reprogramming and Data Download

To facilitate re-programming and data downloading, we developed USB docking stations for the TrainSense mote as described in Figure 4. The physical docking station is simple and inexpensive. It has four contacts, which are made of "chrysoscale", an alloy whose electric conductivity is as good as copper, but it does not rust. The two contacts on the top are used for the power (USB VCC and ground) and the two on the side are the data lines. The goal behind the design of the USB docking station was to provide the reliability, bandwidth and zero-interference effect of a wired USB port. However, since the trains do not have the power to dock a mote directly into a regular female USB port, the docking station was developed to allow easier, automatic insertion.

To evaluate the throughput of our docking station, we positioned a TrainSense mote 10cm from the station and used the controller to dock in the mote. The mote was then instructed to send 1000 packets of 10 bytes and 1000 packets of 50 bytes. After that, the mote was moved to the original un-docked position, and the docking was repeated 10 times. We compared the transmission taking place through our docking station with transmissions on a standard 1.5m long USB cable. Figure 10 shows our test results.

For the docking station, the data rate of the small packets was 98% compared to the regular port, for the 50-byte packets, 96%. It is important to note that no packets were lost: the failed transmissions were successfully corrected at the USB level, which has a robust error correction mechanism with retransmission. The lost packets simply result in a reduced transmission rate. This indicates that the connections of our docking station are not exactly as good as a standard cable, but they have a negligible impact on performance.

3.5 Back-Channel Information

As mentioned before, an important feature of model trains is that the tracks are used to convey control data. Therefore, each train has a decoder that can be extended to communicate with the motes. To send a command to a node, the controller sends a packet to the address of the locomotive that carries the mote. Once the mote-decoder identifies its address, it performs the required action. Currently, we have implemented a mote on/off command to control the topology dynamically.

The back channel design is a good example of the modularity of the TrainSense architecture. Since we use the standard packet format, a commercial controller could also send information to the nodes. For example, if an application does not require the real-time characteristics of TrainSense (i.e., no fine grained positioning), a user could just purchase a commercial system and connect the on-board card (Figure 3) to the train and mote.

3.6 Zero Electromagnetic Interferences

As described previously, the tracks are used to provide power to the trains and to activate the turnouts. When the trains are moving or turnouts are activated, they draw currents whose flow in the rails causes an "antenna effect", i.e., generating electromagnetic fields that can cause interference to radio frequency communication. It is important to asses the impact of such an "antenna effect". We show in this section that the electromagnetic fields generated by the tracks do not cause any degradation in the communication among motes.

We evaluate four operational modes. **Mode A–baseline**: the tracks are powered using the on-board power supply with 18 volts DC and no interference signal is present. This condition resembles a mote operating with ideal batteries (3V) without TrainSense. **Mode B–heavy traffic**: the controller sends a continuous stream of packets to the trains. This condition resembles a heavy traffic operation of TrainSense. **Mode C–frequent turnouts changes**: the turnouts are switched back and forth as fast as possible. Frequently switching turnouts is the harshest condition of all, because the coils within the turnouts consume more than one ampere. Hence, they can produce high electromagnetic pulses. **Mode D–dirty tracks**: the power on the tracks is turned on and off every second. This condition is very unlikely to happen during normal operation. It resembles extremely dirty tracks causing repeated loss of electric contact.

In order to evaluate the interference level, we performed two sets of experiments. In both experiments, we made sure that no other source of interference is present, e.g., WiFi. First, we positioned a TrainSense mote nearby four turnouts and measured the noise floor while the controller activated the four different modes. The noise floor was sampled every 500 μs (at 2KHz) and the four modes were tested in a round robin fashion with 10 seconds for each mode. A total of 1000 rounds were measured. Figure 11 shows the distribution of the measured noise floor values for each mode. Each distribution contains 20 million noise floor samples. We can observe no significant differences.

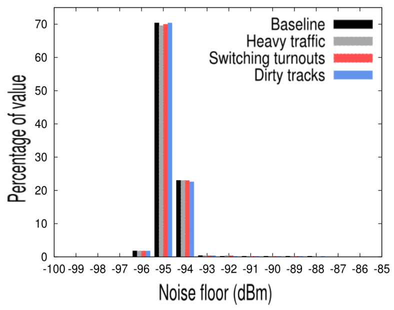

Fig. 11. The noise floor measurement

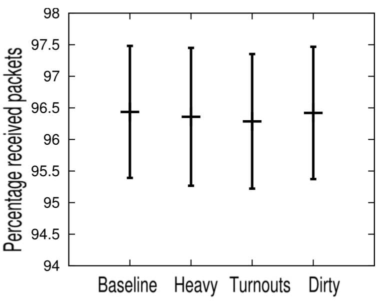

Fig. 12. Packet loss measurement

The noise floor measurement is a good indicator that the sampled noise floor remains constant during the different modes, but this is not sufficient. The motors of the trains, the coils of the turnouts and the sudden changes on the DC voltage of the tracks may cause pulses of short duration (in the order of a few microseconds) that are missed by the relative coarse sampling rate of 2KHz. The second set of experiments measures the effect of these short pulses.

Increasing the sampling rate is not the best option to capture short interference pulses. Even at the highest noise sampling rate achieved for motes, the short pulses would be missed (60KHz in [5]). Thus, we used actual packet transmissions to quantify the effect of such interferences. The hypothesis is the following: short interference pulses can corrupt a few bits of an ongoing packet transmission. Hence, when they are present we should see a decrease of the packet reception rate. To test this, we used the same setup as in the previous experiment, but made the mote send packets continuously. We used a controlled scenario to make sure that the packet reception rate was not affected by dynamics in the environment. We used the lowest output power (-20dBm) and selected a distance that lead to a low RSSI at the receiver, so any variation in the noise floor could be detected via a packet loss. Short pulses would damage some bits of a packet, and hence, cause a packet loss. Figure 12 shows the mean and standard deviation of the packet delivery rate for the four operation modes. We observe that there is no statistical difference, the difference between the best and worst modes being negligible (less than 0.2% of the reception rate).

4 Supporting Mobile WSN Applications

In this section we demonstrate the support offered by TrainSense to implement two sample mobile WSN applications: self-deployment and data mules. Both applications use the positioning system and the constant energy features. In the data mule case, we also use the USB docking station. Both scenarios are implemented using TelosB motes and TinyOS 2.1.

4.1 Self-deployment Application

In self-deployment applications, nodes move autonomously in the target area to identify the locations that maximize a network metric, e.g., coverage or connectivity. Our application aims to evaluate a route-repair scenario (connectivity) using self-deploying motes to repair the route between two disconnected nodes.

Set-up. We place two nodes, a source and a sink, statically at the opposite extremes of the TrainSense prototype as shown in Figure 5. The sink node is connected to the host computer as the host-mote. Recall that the host-mote can interact with the controller to control the train movement. Given the short distance between the sink and source (around 3m), we use a pseudo-signal attenuation to achieve multi-hop transmissions. A link is considered valid if its RSSI is above -75dBm. For an output power of 0dBm, this threshold leads to a transmission range between a few tens of cm up to 1m. On the source side, we place six mote-trains. The mission of these motes is to establish a multi-hop path between the source and the sink.

We use a greedy approach for data forwarding. The data is forwarded to the mote that is closest to the sink. We use the mote ids to represent sink-proximity: the lower the id, the closer the mote is to the sink. The six mote-trains are aligned with monotonically decreasing ids (towards the sink). Before sending a packet, the source node sends a neighbor discovery message. Each neighbor replies with its id and the RSSI of its incoming link. To avoid collisions, the motes spread their transmissions using a simple TDMA-MAC based on their node-ids. Upon receiving the reply packets from its neighbors, the source forwards the information to the neighbor that is closest to the sink and that has a link above the specified threshold. All intermediate motes function the same way.

Deployment. Initially, the sink detects that no packets arrive from the source and, thus, instructs the train controller to deploy the first mote-train with the lowest id. The first train moves towards the sink sending continuous discovery beacons. When the RSSI between the mobile mote and the sink reaches a value above the required threshold, the link is established. Then the sink instructs the controller to stop the train. If the mote does not repair the route, i.e., the sink still receives no packets from the source, the sink instructs the controller to deploy the next train, and this process is repeated until the route is repaired.

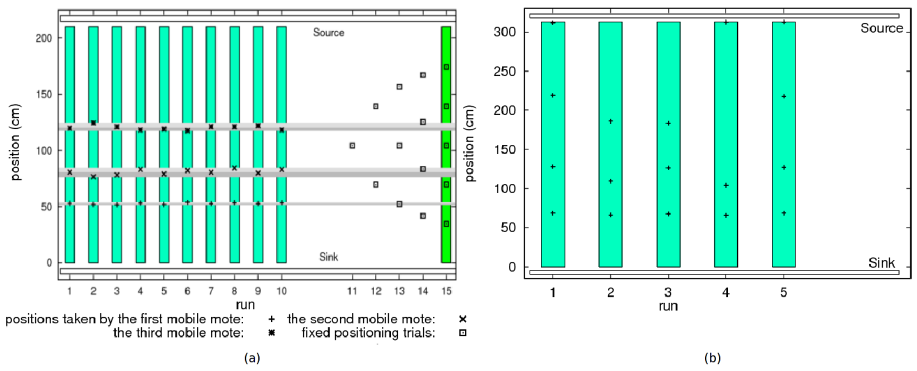

Fig. 13. (a) A comparison of automatic and fixed deployment, (b) Self-deployment in an uncontrolled environment

Results. We perform our evaluations in two types of environments: controlled and uncontrolled. In the controlled scenario, to obtain consistent results, we remove potential sources of interference: leaving the lab unoccupied and closing the door to avoid dynamics in the wireless channel. The experiments are performed remotely through the host computer. We perform ten runs of the experiment, and the results are presented in the first ten points of the x-axis in Figure 13(a). The figure shows that three motes are required to create a connected chain. The shaded areas indicate the minimum, maximum and average positions of these motes. The low variance in the final positions reflects the low channel dynamics. For completeness, we also perform uniform deployments (shown in the last five x-axis points of the same figure). The aim is to test the other end of the self-deployment spectrum, where nodes are deployed in a uniform way (same inter-node distance). In this case, the controller instructs n nodes to deploy in a uniform manner. The uniform deployment is achieved by using the positioning feature of TrainSense. The value of n varies from 1 to 5, in steps of 1. For each n, we perform ten trials. The path is repaired with at least five motes.

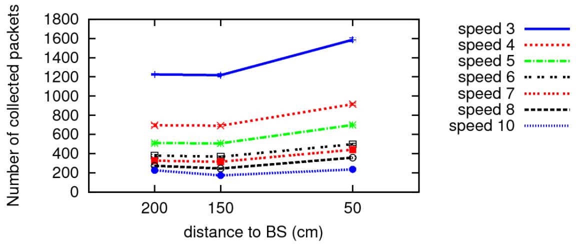

Fig. 14. Comparing data collection at various constant speeds

In order to assess the dynamics of the environment, we also perform the self deployment experiment in an uncontrolled scenario, where people move across the room. The same experimental setting is used except for the distance between the source and the sink (3.10 meters instead of 2.2). Figure 13(b) shows the significant effect of the room conditions on the number of nodes required to repair the path and their locations. This behavior is somehow expected due to the well-known fading effects in wireless communication, where shifts of a few centimeters (half the wavelength) can change the signal strength by tens of dBm.

4.2 Data Muling

We present now a data-mule application that makes use of the docking station offered by TrainSense. The aim is to have the data mule to move along the track, collect data from some static nodes placed on the side of the tracks and then to offload the collected data at the USB docking station.

Setup. Three static motes are placed at 50, 150 and 200cm from the docking station located at one end of the TrainSense track. These static nodes are positioned 20cm away from the tracks and they broadcast their data with power level 1 (chip CC2420, TelosB motes). The data mule process is started by the host using the train controller. The data mule makes one round trip, toward the opposite end of the infrastructure and back to the docking station. Power level 1 is not very reliable, but it is appropriate for this scenario because we do not test repeatability and the transmission range is about 30cm.

Results. Figure 14 depicts the results of the data mule experiment with various speed levels. The x axis indicates the location of the static nodes and the y axis the number of packets collected. All collected packets were correctly delivered through the USB docking station. The figure shows the expected trade-off between speed and data collection, the lower the speed the higher the collected packets. For example, for speed 5 (12cm/s), the number of collected packets per node is approximately 500, while for speed 10 (30cm/s), the number is approximately 200. When normalized with respect to the round trip time, both speeds lead to the same delivery performance. In practical settings however lower speeds are preferred because they would reduce the number of docking operations, which allows more time for data-gathering.

5 Related Work

In this section, we discuss related work with respect to mobility support, existing testbeds and mobility-assisted network deployment for WSN applications.

Exploiting node mobility is a good way to improve important network performance such as lifetime and throughput [27, 21, 4]. Node mobility can be classified into *uncontrolled* and *controlled mobility* [4]. With uncontrolled mobility, a mobile node can move randomly and freely. In contrast, with controlled mobility, a mobile node only moves along the pre-defined trajectory. Several studies have shown that controlling the mobility of the node (e.g., a mobile sink) can achieve remarkable performance improvement, especially on the WSN lifetime. The concept of controlled mobility was first presented by Gandham et al. [11]. The author presented an ILP (integer linear program) approach to deploy multiple mobile stations to prolong the WSN lifetime. Much work has been conducted on WSNs with mobile nodes moving along fixed paths [15, 18, 21, 4]. In [20], the author proposes a perimeter-based solution, in which the mobile sink cyclically moves along the perimeter of the monitored area. In [24], Somasundara et al. proposed a mobile infrastructure to reduce the communication energy consumption of low-energy embedded devices. TrainSense offers a model-train-based infrastructure that can support these controlled mobility scenarios.

Since TrainSense can also serve as a testing environment, we include related work on the mobility testbeds. MiNT-m [6] is a well-known testbed that provides physical mobility support. The mobile platform is based on Roomba [2] robots fitted with unique multi-color tags that allow localization. With respect to MiNT-m, the advantages of TrainSense are constant energy supply, cost (a train is approximately 3 times less expensive than the most affordable robots). The main comparative disadvantage of TrainSense is the restricted movement and the initial setup. TrueMobile [14] is a testbed where a few mica2 nodes are carried by robots. The robots are tracked from above using a vision system. On a similar line of work, Foerster et al. [9] present an architecture where nodes are carried by robots moving on the ground. A camera keeps track of the robot positions. These robotics testbeds share advantages and disadvantages that are similar to MiNT-m, when compared to TrainSense.

TrainSense is a unique mobility infrastructure. To the best of our knowledge, there is no prior effort that takes a model train setup and enhances it to the extent presented in this study. There are, however, studies that exploit the basic functionality of integrating a mote with a train. Roziers et. al. provide a testbed where a single node is located on a train that circles around a 3m×3m area [7]. Similarly, Rensfelt et. al. developed a simple robot that can follow a line painted on the floor [22]. In SmartHop [10], the authors use model trains to evaluate hand-off mechanisms in sensor networks. TrainSense also focuses on constrained paths, but provides significantly more advantages such as continuous power, complex topology management and positioning, and a USB docking station.

6 Conclusion and Future Work

The main goal of our study is to bring to the attention of the community a new platform to test and debug mobile WSNs. The new platform, TrainSense, is based on off-the-shelf model trains. The inherent characteristics of model trains – controllers, powered tracks and detectors – provide a natural configuration to support mobile WSNs applications. We took this basic infrastructure and added important features to create a comprehensive mobility platform. TrainSense provides continuous and unlimited energy to the motes, allows the set up of different mobility patterns, permits accurate positioning and automates mote reprogramming and data downloading via the use of USB docking stations.

We believe that TrainSense can help in fostering the study of mobility in sensor network applications. In particular, TrainSense fits well the research areas in controlled mobility, which include important scenarios such as vehicular networks, barrier coverage and pipe-line monitoring. TrainSense can also help in the systematic study of sensor networks, e.g., nodes can be moved sequentially and repeatably to various locations to identify the deployment or mobility pattern that maximizes the metric of interest.

Model trains are certainly not a panacea to add mobility to WSNs. Free robots provide a quicker setup time and more path freedom. But by design, a model train setup guarantees the reproducibility of the deployment and the repeatability of the positioning and traffic patterns. A train testbed can also be installed as a permanent setup at various heights, saving space and allowing the effect of height on the RF propagation to be measured. TrainSense is an ongoing effort and there are several items planned for the near future: a modular infrastructure to allow more complex experiments, a GUI to simplify the design of mobility patterns and opening TrainSense to the community.

References

[1] `http://spazioinwind.libero.it/scorzoni/motorola.htm`
[2] `http://www.irobot.com/`
[3] `http://www.mobilerobots.com/researchrobots/researchrobots/p3at.aspx`
[4] Basagni, S., Carosi, A., Petrioli, C.: Controlled vs. uncontrolled mobility in wireless sensor networks: Some performance insights. In: IEEE VTC (2007)
[5] Boano, C.A., Voigt, T., Noda, C., Römer, K., Zuniga, M.: Jamlab: Augmenting sensornet testbeds with realistic and controlled interference generation. In: IPSN 2011 (2011)
[6] De, P., Raniwala, A., Krishnan, R., Tatavarthi, K., Modi, J., Syed, N.A., Sharma, S., Chiueh, T.: Mint-m: An autonomous mobile wireless experimentation platform. In: MobiSys 2006 (2006)
[7] des Roziers, C.B., Chelius, G., Ducrocq, T., Fleury, E., Fraboulet, A., Gallais, A., Mitton, N., Noel, T., Valentin, E., Vandaele, J.: Two demos using senslab: Very large scale open wsn testbed. In: DCOSS 2011(2011)
[8] Evans, L.H.D.: Localization for mobile sensor networks. In: MobiCom 2004 (2004)
[9] Förster, A., Förster, A., Leidi, T., Garg, K., Puccinelli, D., Frederick Ducatelle, S.G., Gambardella, L.M.: Motel: Towards flexible mobile wireless sensor network testbeds. In: EWSN 2011(2011)

[10] Fotouhi, H., Zuniga, M., Alves, M., Koubaa, A., Marrón, P.: Smart-hop: A reliable handoff mechanism for mobile wireless sensor networks. In: Wireless Sensor Networks, pp. 131–146 (2012)

[11] Gandham, S., Dawande, M., Prakash, R., Venkatesan, S.: Energy efficient schemes for wireless sensor networks with multiple mobile base stations. In: IEEE GLOBECOM 2003 (2003)

[12] Grossglauser, M., Tse, D.N.C.: Mobility increases the capacity of ad hoc wireless networks. IEEE/ACM Trans. Netw. 10(4) (2002)

[13] Gu, Y., Bozda, D., Brewer, R.W., Ekici, E.: Data harvesting with mobile elements in wireless sensor networks. Comput. Netw. 50(17) (2006)

[14] Johnson, D., Stack, T., Fish, R., Flickinger, D., Ricci, R., Lepreau, J.: Truemobile: A mobile robotic wireless and sensor network testbed. University of Utah Flux Group Technical Note FTN-2005-02 April 8 (2005)

[15] Kansal, A., Rahimi, M., Estrin, D., Kaiser, W., Pottie, G., Srivastava, M.: Controlled mobility for sustainable wireless sensor networks. In: IEEE SECON (2004)

[16] Ko, A., Lau, H.Y.K.: Robot assisted emergency search and rescue system with a wireless sensor network. Int. Jour. of Advanced Science and Technology 3 (2009)

[17] Kotay, K., Peterson, R., Rus, D.: Experiments with robots and sensor networks for mapping and navigation. In: FSRR 2005 (2005)

[18] Liang, W., Luo, J., Xu, X.: Prolonging network lifetime via a controlled mobile sink in wireless sensor networks. In: IEEE GLOBECOM 2010 (2010)

[19] Liu, B., Brass, P., Douss, O., Nain, P., Towsley, D.: Mobility improves coverage of sensor networks. In: MobiHoc 2005 (2005)

[20] Luo, J., Hubaux, J.-P.: Joint mobility and routing for lifetime elongation in wireless sensor networks. In: IEEE INFOCOM 2005 (2005)

[21] Luo, J., Panchard, J., Piórkowski, M., Grossglauser, M., Pierre Hubaux, J.: Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In: IEEE/ACM DCOSS 2006 (2006)

[22] Rensfelt, O., Hermans, F., Gunningberg, P., Larzon, L.-Å., Björnemo, E.: Repeatable experiments with mobile nodes in a relocatable wsn testbed. Computer Journal 54(12), 1973–1986 (2011)

[23] Sibley, G., Rahimi, M.H., Sukhatme, G.S.: Robomote: A tiny mobile robot platform for large-scale ad-hoc sensor networks. In: ICRA, pp. 1143–1148 (2002)

[24] Somasundara, A., Kansal, A., Jea, D., Estrin, D., Srivastava, M.: Controllably mobile infrastructure for low energy embedded networks. IEEE Transactions on Mobile Computing 5(8), 958–973 (2006)

[25] Sugihara, R., Gupta, R.K.: Path planning of data mules in sensor networks. ACM Trans. Sen. Netw. 8(1), 1–27 (2011)

[26] Wang, G., Cao, G., Berman, P., Porta, T.F.L.: Bidding protocols for deploying mobile sensors. IEEE Trans. Mob. Comput. 6(5), 563–576 (2007)

[27] Wang, Z., Basagni, S., Melachrinoudis, E., Petrioli, C.: Exploiting sink mobility for maximizing sensor networks lifetime. In: HICSS 2005 (2005)

[28] Ye, W., Heidemann, J., Estrin, D.: An energy-efficient mac protocol for wireless sensor networks. In: IEEE INFOCOM 2002, vol. 3 (2002)