

Research Article

A Wi-Fi Cluster Based Wireless Sensor Network Application and Deployment for Wildfire Detection

Alper Rifat Ulucinar,¹ Ibrahim Korpeoglu,¹ and A. Enis Cetin²

¹ Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey

² Department of Electrical and Electronics Engineering, Bilkent University, Bilkent, 06800 Ankara, Turkey

Correspondence should be addressed to Alper Rifat Ulucinar; ulucinar@cs.bilkent.edu.tr

Received 10 April 2014; Accepted 3 September 2014; Published 20 October 2014

Academic Editor: Wen-Hwa Liao

Copyright © 2014 Alper Rifat Ulucinar et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We introduce the wireless sensor network (WSN) data harvesting application we developed for wildfire detection and the experiments we have performed. The sensor nodes are equipped with temperature and relative humidity sensors. They are organized into clusters and they communicate with the cluster heads using 802.15.4/ZigBee wireless links. The cluster heads report the harvested data to the control center using 802.11/Wi-Fi links. We introduce the hardware and the software architecture of our deployment near Rhodiapolis, an ancient city raising on the outskirts of Kumluca county of Antalya, Turkey. We detail our technical insights into the deployment based on the real-world data collected from the site. We also propose a temperature-based fire detection algorithm and we evaluate its performance by performing experiments in our deployment site and also in our university. We observed that our WSN application can reliably report temperature data to the center quickly and our algorithms can detect fire events in an acceptable time frame with no or very few false positives.

1. Introduction

The Firesense (fire detection and management through a multisensor network for the protection of cultural heritage areas from the risk of fire and extreme weather conditions, FP7-ENV-2009-1244088-FIRESENSE) project is a 3-year-long STREP aiming to develop an *early warning* system to protect areas of archaeological and cultural interest from the risk of fire and extreme weather conditions [1]. Protecting archaeological and cultural sites that are surrounded by forests from wildfires is one of the most important goals of the developed system. The system provides reactive protection from wildfires by the early detection of the fire from the time of ignition. In order to achieve this goal, a wireless sensor network capable of sensing ambient temperature and relative humidity is used together with a network of both visible and infrared (IR) cameras. Data from the sensors and the smoke and fire detection software arrive at the data fusion module which decides whether there is a fire or not at the monitored area. One of the pilot sites where the developed system is deployed is the ancient city of Rhodiapolis. Rhodiapolis lies

on the outskirts of Kumluca, Antalya, Turkey. For Rhodiapolis deployment, we have developed WSN software to collect and process sensor data and make it available for external querying and analysis. In this paper, we will detail our WSN software together with the hardware components used. We will detail our network architecture and we will give temperature and humidity measurements from the site together with other measurements related to the lifetime of the WSN.

We also designed and implemented a fire detection algorithm that takes the sensed data as input and raises an alarm when a fire is detected. We tested our algorithms with the temperature data collected from Rhodiapolis and with the temperature data collected during outdoor fire experiments. We observed that the algorithm can successfully detect fires with no false positives.

The rest of the paper is organized as follows. In Section 2, we briefly discuss the related work on test-based fire monitoring with wireless sensor networks. In Section 3, we detail the network architecture used by our sensor network. In Section 4, we introduce the relevant hardware components used in Rhodiapolis for realizing the network architecture.

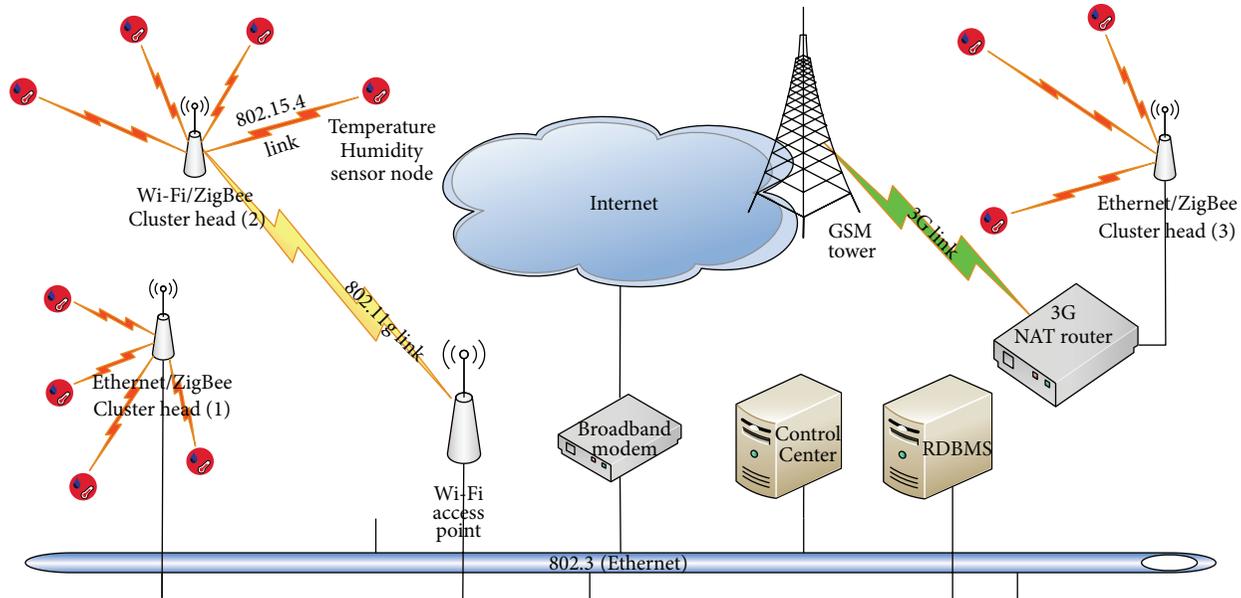


FIGURE 1: WSN network architecture.

In Section 5, we detail the software components of the system and explain the interaction between them. We also present our fire detection algorithm that is run at the control center. In Section 6, we report on some real-world data we collected from the site and we discuss the performance of our system and our fire detection algorithm. Finally in Section 7, we conclude the paper.

2. Related Work

In previous work, wireless sensor networks (WSN) have been considered as an efficient alternative to satellite based monitoring [2, 3] for the early detection of wildfires. The proximity of the sensors to the monitored fuel gives WSNs great advantage to meet the timeliness and accuracy constraints over other fire monitoring technologies, such as satellites.

In [4, 5], the authors propose a cluster based WSN paradigm for real-time detection of forest fires. They employ Chipcon CC2430 based ZigBee sensor nodes equipped with relative humidity and temperature sensors. Sensor data collected by the cluster heads is delivered via network coordinators to the gateway (router) nodes which are responsible for delivering the data to the monitoring computer.

In [6], Yu et al. propose a cluster based WSN paradigm with a neural networks based method for in-network data processing and evaluate the performance of the proposed system via simulations.

In [7], the authors describe the system architecture and the software and hardware framework of a WSN for wildfire monitoring. The sensor nodes used in this study are Mica2 motes with MTS400 sensor boards. The authors list field tests for the system among their plans of future work.

We have realized a very flexible WSN deployment for monitoring and early detection of wildfires using COTS

(commercial off-the-shelf) hardware components. The sensor data from the site and the derived alarm data are exposed to the outside world through an XML web service we have implemented. We have also designed and implemented a new temperature-based fire detection algorithm that can timely detect fire events. The parameters of the algorithm are optimized for low false positive rate and high probability of detecting fire events.

3. Network Architecture

A cluster based topology is used by the WSN. There are two types of nodes: cluster heads and sensor nodes. Sensor nodes are connected to cluster heads with ZigBee links. The cluster heads can be heterogeneous in hardware. Since the sensor nodes communicate with ZigBee radios, each cluster head must have at least one ZigBee radio used to talk to sensor nodes in its cluster. In order to communicate with the control center (CC), a cluster head can use any appropriate interface such as WiFi, Ethernet, 3G modem, or a combination of these as depicted in Figure 1. There are no intracluster or intercluster multihop communications in the WSN network we deployed in Rhodiapolis. All sensors report sensor data to their designated cluster heads in one hop and all cluster heads report their gathered data to the control center in one hop. Therefore, one can expect minimal network delays in this architecture. However, multihop communication can be supported for some other cases and for other deployment sites with multihop routing protocols such as OLSR [8, 9] or AODV [10, 11].

Figure 1 depicts our deployment in Rhodiapolis. There are three clusters each consisting of a cluster head and associated sensor nodes. Sensor nodes are capable of measuring ambient temperature and relative humidity. A cluster head operates as the ZigBee *network coordinator* and each of the sensor nodes

in the cluster operates as a ZigBee *end device*. The sensor nodes of a cluster report data to their respective cluster head in one hop. The cluster head for each sensor is determined at deployment time. We use ZigBee PAN ids to discriminate the clusters from each other. Each cluster head uses a distinct PAN id. And each sensor is configured, at deployment time, with the PAN id of the cluster it is planned to join.

As indicated in Figure 1, cluster head 1 is equipped with an Ethernet network interface card (NIC). It resides on the same LAN as the control center and it sends the gathered sensor data to the control center via the Ethernet bus. The second cluster is located far away from the control center and so cannot reside on the same Ethernet LAN as the control center. Hence, cluster head 2 uses a Wi-Fi link to send sensor data to the control center via an access point (AP). The AP is configured in bridge mode, so it basically extends the wired LAN with the wireless LAN (WLAN). Finally, cluster head 3 is equipped with an Ethernet NIC which it uses to communicate with a 3G GSM NAT router. The 3G NAT router is connected to the Internet via a 3G GSM link. On the control center side, there exists a broadband modem which connects the control center to the Internet. Hence, cluster head 3 is able to send the gathered sensor data to the control center via Internet.

The WSN does not currently support queries. The sensor nodes are programmed to report sensor data periodically to their cluster heads. However, control center can issue other commands to the sensor nodes. For instance, it can specify the transmit period (TXT) which determines the interval between two successive reports of the sensor node. Or it can specify the measure interval (MSI) which determines the interval between two successive samplings of the sensor node. The control center can *set* or *get* any of the parameters specified in the ZBS-120/121 interface control document [12] including TXT and MSI values. Hence, data flow is bidirectional between the sensors and the control center. This issue will be detailed further in Section 5.

4. Hardware Architecture

The ZigBee sensor nodes deployed on the site are Pikkerton ZBS-120/RH sensors [13]. These modules are ruggedized for outdoor deployment. They operate on the 2.4 GHz ISM band and have an outdoor line-of-sight transmission range of up to 120 m. Some of the Pikkerton nodes are equipped with both temperature and humidity sensors whereas others are equipped with only a temperature sensor. The ZigBee radio modules used on the Pikkerton sensor nodes are Digi's XBee S2 OEM RF modules [14]. These are low power radio modules designed to prolong battery lifetimes. When the *boost mode* is enabled, the transmit power output for these modules is 3 dBm and when the boost mode is disabled, it is 1 dBm.

As part of the cluster-head hardware, we use Digi's XStick ZB USB adapters [15]. These adapters have an outdoor line-of-sight range of 50 m. The RF data rate is 250 Kbps and the serial interface data rate is 115.2 Kbps. This adapter interfaces the ZigBee network on the cluster head. The XStick ZB adapters we are using employ FTDI USB converter chips and basically they are virtual serial ports to the cluster-head modules.

A cluster-head module communicates with the ZigBee network via this virtual serial port.

XBee S2 modules support three operational roles in the ZigBee network: coordinator, router, and end device. Since we use the XStick ZB USB adapters in the cluster heads as our *coordinators*, we configure the XBee S2 modules as *end devices* and no ZigBee radio is configured as a router. This forms the star topology ZigBee network that we have discussed in Section 3.

The cluster-head hardware consists of, apart from the XStick ZB USB adapter, a PC Engines Alix2d2 system board [16]. The Alix.2 series system boards have 2 Ethernet/2 mini-PCI or 3 Ethernet/1 mini-PCI ports. The Alix2d2 boards we are using are equipped with 2 Ethernet interfaces, 2 mini-PCI ports, 1 CompactFlash socket, and 2 USB ports, with a 500 MHz AMD Geode LX800 CPU and 256 MB of RAM. The operating system image is stored on a 1 GB CompactFlash card. We are using Debian 6 on these cards. If Wi-Fi connectivity is needed on a cluster head, then one Wistron CM9 mini-PCI 802.11a/b/g card [17] is installed with two high gain omnidirectional antennas to the Alix2d2 board.

5. Our Software Architecture, Components, and Fire Detection Algorithm

Our WSN software consists of two main modules: the cluster-head module and the control center module. Both modules are implemented using the Python programming language. The cluster-head module, referred to as *CHm* from now on, is responsible for gathering the sensor data from the sensor nodes in its cluster and for making the sensor data available as documents formatted in extensible markup language (XML). *CHm* is also responsible for issuing commands to the sensor nodes in its cluster. The control center module, referred to as *CCm* from now on, is responsible for gathering cluster data from the cluster heads and making the data from all of the clusters available from a single source as a document formatted in XML. Clients can request the data of all clusters deployed at a site from *CCm*. Both *CHm* and *CCm* implement a RESTful [18] web service for WSN data gathering and management. We now detail *CHm* and *CCm* in the following subsections.

5.1. The Cluster-Head Module. *CHm* is responsible for gathering periodic sensor data from each of the sensors in its cluster and for sending management messages to them. The sensor nodes run an application that periodically collects sensor data and sends the collected data to the network coordinator (cluster head). This application is part of the firmware of the sensor nodes. The sensor data that is periodically sent by a sensor node consists of the following items:

- (i) the long address associated with the sensor,
- (ii) last temperature measurement value,
- (iii) last relative humidity measurement value (if the related sensing module is available),
- (iv) last battery voltage measurement value,
- (v) last battery state (OK or LOW).

CHm receives the sensor data made available by a sensor node and puts this data in its in-memory cache or updates the data associated with the sensor in its cache. When a sensor data message is received, CHm uses the sender's long address as a key to index the sensor data in its cache. Each measurement value in CHm's cache is labeled with a timestamp value. Timestamps are generated with accuracies in seconds. Later when a client requests the cluster data, CHm formats the sensor data in its cache as an XML document and responds with this document. Since CHm only maintains an in-memory cache of sensor data and no persistent cache, when the in-memory cache is destroyed as a result of the restart of the CHm software, there will be no data available for a sensor till the sensor sends its next data message to the cluster head.

CHm dynamically learns about the sensors in its cluster as the sensors report to the cluster head. This approach makes the addition of new sensors to a cluster very easy. A new sensor can be added to a WSN cluster by just configuring the ZigBee PAN id of the sensor to the PAN id used by the network coordinator of the cluster. As the new sensor node reports to its cluster head, CHm will learn about it and put its data in CHm cache.

CHm has two main modes of operation. Either it periodically pushes (HTTP POSTs) cluster data to CCm as XML documents or the CCm pulls (HTTP GETs) the cluster data. In Rhodiapolis site, cluster heads push data to the control center. This approach makes the addition of new clusters to the site very easy. As depicted in Figure 1, it is possible to add a new cluster to the site via the Internet. This greatly reduces deployment costs by making it possible to use the existing network infrastructure. It is even possible to easily build a wireless sensor network covering geographically distributed clusters.

CHm is also responsible for broadcasting commands to the sensor nodes in its cluster and receiving ACK/NACK messages from the sensors in response to these commands. Currently, CHm only supports broadcasting commands and it is not possible to address a single sensor as the recipient of a command message. As part of the CHm RESTful service interface, a timeout value has to be specified with a command invocation that determines the interval for which the cluster head should listen for ACK/NACK messages from sensors in response to the broadcasted command message. After the end of this interval, CHm reports with an XML document the long addresses of the sensors that have sent an ACK and the long addresses of the sensors that have sent a NACK message in response to the command message. Sensors that did not receive the broadcast command message or sensors that have failed to respond within the specified timeout or sensors whose responses simply got lost are missing in CHm's report. A client issuing a command to the WSN cluster shall need a priori knowledge about the long addresses of the sensors of the cluster in order to be able to identify such missing sensors.

CHm is able to generate temperature threshold alarms. If a *SETHITEM* command [12] is issued to the cluster via CHm's web service API (by the end user or by another software module), CHm records the temperature threshold value. Later, if a sensor node reports a temperature value higher

than or equal to the configured threshold, CHm adds alarm state to its cache. Subsequently when the sensor node that has caused the temperature threshold alarm to be raised reports a temperature value lower than the configured threshold, CHm clears the cached alarm state.

5.2. The Control Center Module and Our Fire Detection Algorithm. CCm gathers sensor data from all the clusters and builds an in-memory cache of the data from all of the sensors deployed on a site. A client can retrieve site data as an XML document from CCm. It is also possible for a client to request a specific cluster's data from CCm. As explained in Section 5.1, cluster heads push data to CCm and CCm puts or updates the cluster data in its in-memory cache. Each CHm module is configured with a unique numeric cluster identifier which is sent with the cluster data to CCm. CCm uses this identifier as a key to index the cluster data in its cache.

CCm is using a *sigma-threshold* (standard deviation) based fire detection algorithm we designed, which generates fire alarms based on the temperature values. The algorithm requires a training phase to reduce generation of false alarms. Hence, at startup, CCm is trained for each sensor node at the site using the available temperature data in the RDBMS. Ideally, the training data should cover a whole day. If whole day's data is available in the training set, then a separate sigma-threshold is calculated for each hour of the day for a sensor (using Algorithm 1). The standard deviation (σ_{it}) of the measured temperature values is calculated iteratively over sliding windows of *wnd_size* samples using Algorithm 2. The sigma-threshold is calculated separately for each sensor node because temperature values change in different magnitudes for each sensor due to the unique location of each sensor. After the sigma-thresholds have been initialized, the standard deviation of the sequence of fresh samples arriving from a sensor is calculated iteratively over a sliding window and compared to the calculated sigma-threshold for that sensor. If the change (δ) in the last reported temperature value of a sensor node from the current window's mean (μ) is higher than the sigma-threshold of that sensor, CCm raises a sigma-threshold alarm for that sensor node.

sigma.coeff, *wnd_size*, and the size of the training data set are three important parameters affecting the performance of the proposed system. As mentioned above, the training data set should cover at least 24 hours of measurement data because the changes in temperature vary greatly during the course of the day due to the angle of the sunlight beams and shades of the landforms surrounding the sensors. As discussed in Section 6, if the training data spans multiple days, the rate of false positives is greatly reduced.

The selection of *wnd_size* depends on the transmit period of the sensors. The greater the *wnd_size* is, the older the historical data taken into account while calculating the δ is. And the system gets more sensitive to rapid temperature increases. However, this also increases the rate of false positives. For the purposes of our system, a time frame of 80 minutes is a good balance between sensitivity and rate of false positives.

The *sigma.coeff* controls how much increase in temperature is allowed before the system generates an alarm. It allows

Global: σ_{thres} , sigma-thresholds dictionary. Keys will be the hours of the day.
Global: sigma_coeff , sigma coefficient for threshold calculation.
Global: wnd_size , window size.
Global: $s_{\sigma_{it}}$, dictionary of sum of σ_{it} . Keys will be the hours of the day.
Global: $s_{\text{var}_{it}}$, dictionary of sum of var_{it} . Keys will be the hours of the day.
Global: wnd_count , the number of windows processed so far.
Input: train_samples , training data set covering 24 hours. Each sample in the training set is timestamped. $\text{train_samples}_i.ts$ yields this timestamp and $\text{train_samples}_i.val$ yields the actual temperature measurement for the i th element.

- (1) $\text{wnd_count} \leftarrow 0$
- (2) **for** $h = 0$ to 23 **do**
- (3) $\text{hour_samples} \leftarrow \{\text{train_samples}_i.val : \text{train_samples}_i.ts.hour = h\}$
- (4) $s_{\sigma_{it}}[h] \leftarrow 0$
- (5) $s_{\text{var}_{it}}[h] \leftarrow 0$
- (6) **for** $s = \text{wnd_size}$ to $\text{len}[\text{hour_samples}] - 1$ **do**
- (7) $\text{new_sample} \leftarrow \text{hour_samples}_s$
- (8) $\text{samples} \leftarrow \{\text{hour_samples}_i : s - \text{wnd_size} \leq i < s\}$
- (9) $(\sigma_{it}, \text{var}_{it}) \leftarrow$ Calculate iterative deviation using Algorithm 2
- (10) **end for**
- (11) $\sigma_{\text{thres}}[h] \leftarrow \sigma_{it} + \text{sigma_coeff} * \sqrt{\text{var}_{it}}$
- (12) **end for**

ALGORITHM 1: Training.

Global: wnd_count
Global: $s_{\sigma_{it}}$
Global: $s_{\text{var}_{it}}$
Input: h , hour of the day.
Input: new_sample , the fresh measured value.
Input: samples , array of measurements of size wnd_size prior to new_sample .
Output: $(\sigma_{it}, \text{var}_{it}, \delta)$, calculated iterative deviance, iterative variance and delta from the mean of samples

- (1) $\mu \leftarrow \sum_i \text{samples}_i / \text{len}[\text{samples}]$
- (2) $\delta \leftarrow \text{new_sample} - \mu$
- (3) $s_{\sigma_{it}}[h] \leftarrow s_{\sigma_{it}}[h] + |\delta|$
- (4) $s_{\text{var}_{it}}[h] \leftarrow s_{\text{var}_{it}}[h] + \delta^2$
- (5) $\text{wnd_count} \leftarrow \text{wnd_count} + 1$
- (6) $\sigma_{it} \leftarrow s_{\sigma_{it}}[h] / \text{wnd_count}$
- (7) $\text{var}_{it} \leftarrow s_{\text{var}_{it}}[h] / \text{wnd_count}$

ALGORITHM 2: Iterative Deviation Calculation.

us to fine-tune our fire detection system according to the characteristics of the common environment of the sensors. In a region where daily rapid temperature changes are common, a high sigma_coeff should be used. Section 6 gives an insight for the effects of sigma_coeff on the system performance.

Algorithm 3 details the above procedure for a single sensor. Before Algorithm 3 can be used, CCm should be trained using Algorithm 1. The training data is obtained from the RDBMS and covers 24 hours data. At Rhodiapolis site, we use 50 for wnd_size and 2.5 for sigma_coeff .

CCm has to be configured with a static IP address whereas cluster heads do not need to be configured with static IP addresses. As CHm pushes cluster data to CCm, CCm learns the IP address of the cluster head. CCm can later send management messages to the cluster head using this IP address.

CCm also logs sensor data to a relational database. However, CCm uses only its in-memory cache while servicing site and cluster data requests. When the in-memory cache is destroyed (when, for instance, the CCm software is restarted), CCm will be unaware of a cluster till the cluster head sends its next message to CCm.

Figure 2 summarizes the sensor data flow. Each sensor node in a cluster pushes data to the cluster head (CHm) periodically. The cluster head pushes cached cluster data to the control center (CCm). A sensor data client pulls cached cluster or cached site data from CCm in XML format. In Figure 2, red arrows represent a PUSH operation which corresponds to an HTTP POST request. Blue arrows represent a PULL operation which corresponds to an HTTP GET request.

```

Global:  $\sigma_{thres}$ 
Global:  $\sigma_{coeff}$ 
Input:  $new\_sample$ , the fresh measured value.
Input:  $ts$ , the timestamp of  $new\_sample$ .
Input:  $samples$ , array of measurements of size  $wnd\_size$  just prior to  $new\_sample$ .
(1)  $h \leftarrow ts.hour$ 
(2)  $(\sigma_{it}, var_{it}, \delta) \leftarrow$  Calculate iterative deviation using Algorithm 2
(3) if  $\delta > \sigma_{thres}[h]$  then
(4)   Raise sigma-threshold alarm
(5) else
(6)   Clear sigma-threshold alarm
(7) end if
(8)  $\sigma_{thres}[h] \leftarrow \sigma_{it} + \sigma_{coeff} * \sqrt{var_{it}}$ 

```

ALGORITHM 3: Sigma Threshold Classification for a Single Sensor.

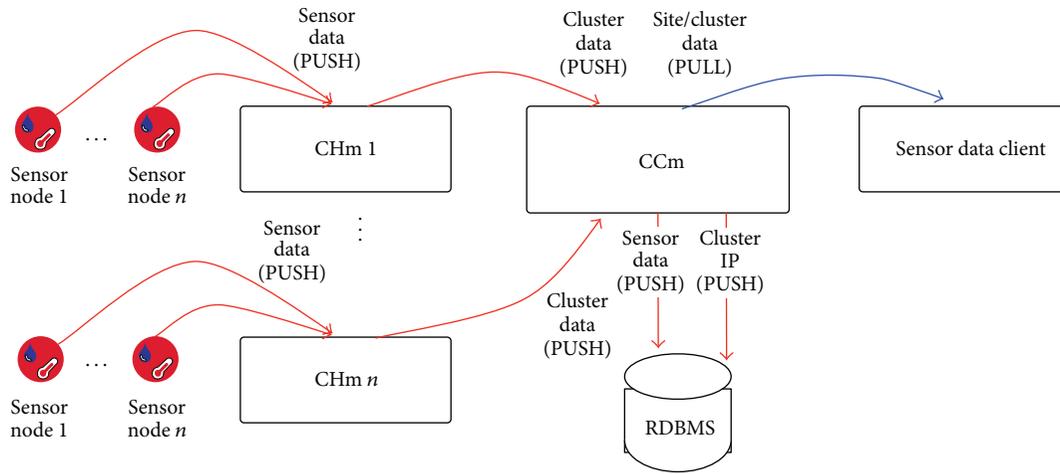


FIGURE 2: Sensor data flow diagram.

In Figure 3, we summarize the command data flow. Again, red arrows represent PUSH operations and blue arrows represent PULL operations. A management client queries the RDBMS for cluster head IP addresses and sends unicast commands to selected cluster-head modules. Each CHm that has received a command broadcasts that command in its cluster. Each receiving sensor node in the cluster processes the command and *pushes* an ACK or a NACK message back to CHm. CHm accepts these ACK/NACK messages within a specified interval, prepares a report in XML at the end of this interval, and responds back to the management client that has issued the command Figure 3.

6. Measurements, Experiments, and Results

After deploying our sensor network and data harvesting system in Rhodiapolis, we collected and recorded sensor data over an extended period of time. To illustrate typical summer weather conditions under which the sensors operate in Rhodiapolis, we give temperature and relative humidity plots, respectively, in Figures 4 and 5, for one of the sensor nodes (node with id 17), for a period of 9 days. Also in

Figure 6, we report the normalized temperature values and the relative humidity values on the same plot. The temperature values are normalized with respect to the maximum temperature measured in the course of 9 days, which is 37.7°C . The oscillation of the temperature and the relative humidity values, as seen in Figures 4 and 5, are typical due to the rotation of the Earth around its own axis. As it can be seen in Figure 6, generally speaking, temperature and relative humidity are inversely proportional with each other. When other atmospheric parameters stay the same, increasing temperature implies decreasing levels of relative humidity, which in turn implies increasing probability of fire.

Using the sensor data from Rhodiapolis, we investigated the performance of the sigma-threshold based algorithm for the temperature values given in Figure 4. If the system is trained with 24 hours of data from August 28, 2012, then we get 42 false positives for the rest of the observation interval with a wnd_size of 50 and a σ_{coeff} of 2.5. The total number of samples in the rest of the observation interval is 6956, which corresponds to a false positive ratio of 0.006. The plot of the false positives is given in Figure 7. The temperature readings that have caused a false positive are marked with

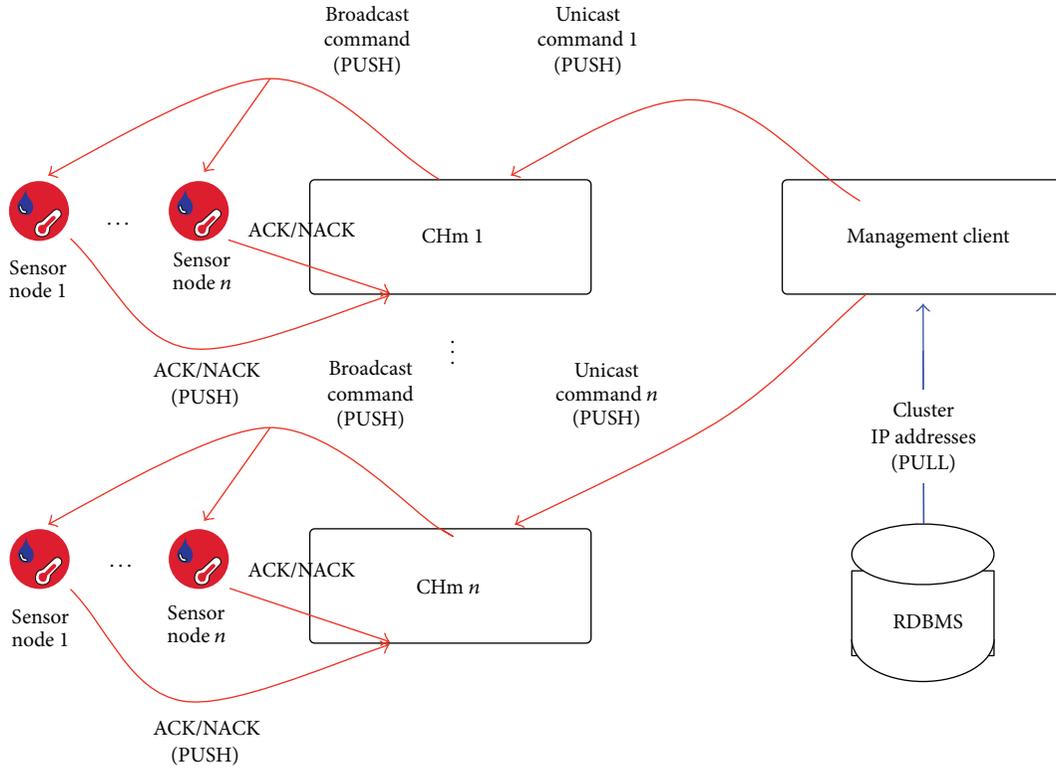


FIGURE 3: Command data flow diagram.

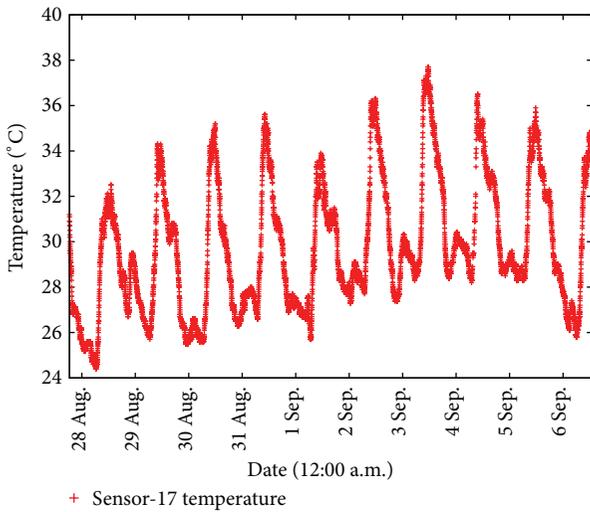


FIGURE 4: Temperature plot for sensor-17.

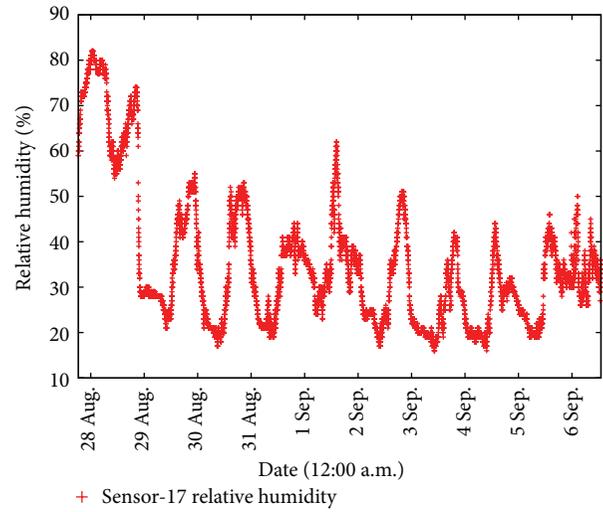


FIGURE 5: Relative humidity plot for sensor-17.

an arrow. As it can be seen in Figure 7, majority of the false positive incidents (about 67%) occur on August 29 and 30, the two days immediately following the training day.

However, using the same *wnd_size* and *sigma_coeff* values, if the system is trained with 48 hours of data from the days August 28 and 29, 2012, then the number of false positives decreases to 2 for the rest of the observation interval, which consists of a total of 6106 measurements. The false positive

ratio decreases to about 0.0003. The plot of false positives with two days of training is given in Figure 8.

If we increase *sigma_coeff* from 2.5 to 3.0 and the system is trained with the one-day data from August 28, then we obtain 23 false alarms, all on August 29 as shown in Figure 9. The false positive rate for the rest of the observation period is now about 0.003. And if the system is trained with 48 hours of data from the days August 28 and 29, 2012, and a *sigma_coeff*

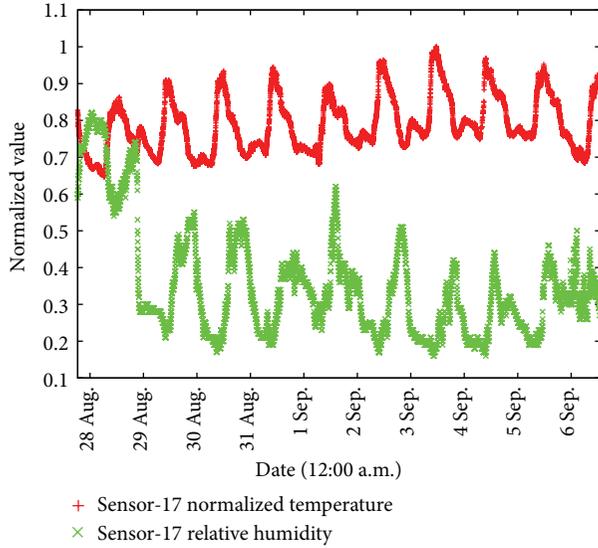


FIGURE 6: Normalized temperature and relative humidity plot for sensor-17.

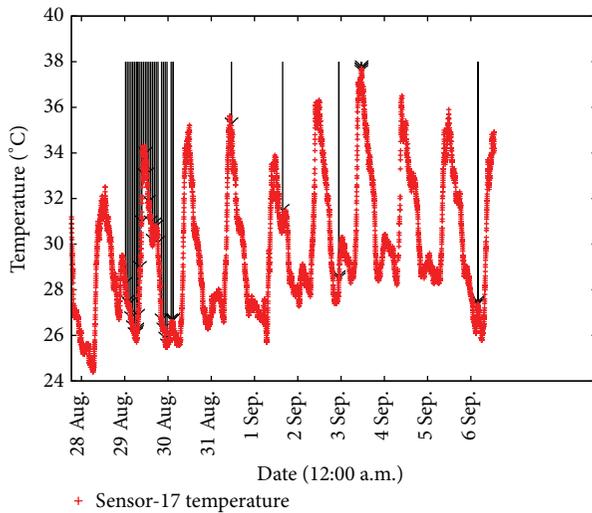


FIGURE 7: False positives for sigma-threshold alarm if the system is trained only with the data from August 28, 2012.

of 3.0 is used, then the false positive rate for the rest of the observation period drops to 0.0.

We did not have any wildfires after the WSN deployment in Rhodiapolis. So, in order to evaluate the performance of our fire detection algorithms in case of a fire, we conducted a fire experiment at the parking lot of the Department of the Electrical and Electronics Engineering (EE parking lot) at Bilkent University. Figure 10 shows the placement of the sensor nodes. The fire was ignited at 15:01 at the parking lot and a total of five sensors were placed around the ignition point. The sampling interval (MSI) and the transmit interval (TXT) of each sensor are set to 1s.

Figure 11 shows the temperature data for all of the five sensors. Amongst the five sensors, Sensor-11 has the highest

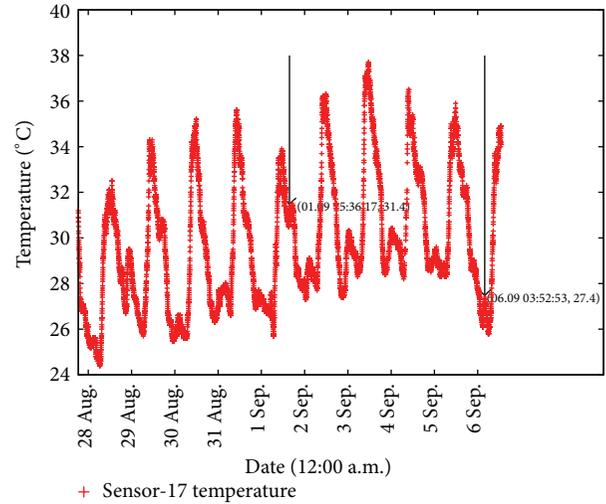


FIGURE 8: False positives for sigma-threshold alarm if the system is trained with the data from August 28/29, 2012.

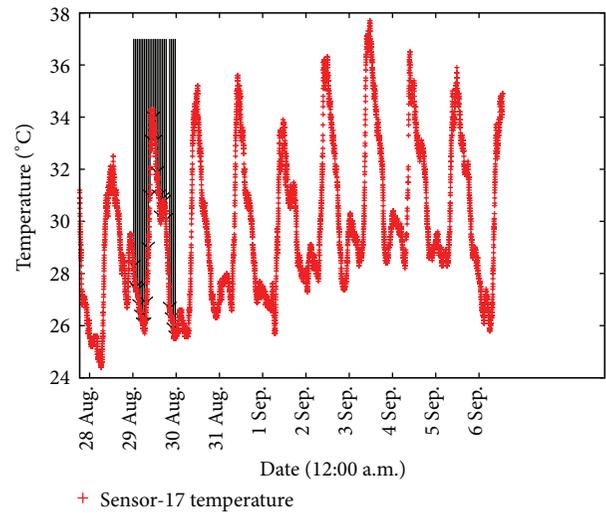


FIGURE 9: False positives for sigma-threshold alarm if the system is trained only with the data from August 28, 2012, and a sigma coefficient of 3.0 is used.

increase in temperature due to the wind, as the wind was blowing towards Sensor-11 and carrying the heat energy released by the fire. This phenomenon can be clearly observed in Figure 11, where the temperature readings of Sensor-11 reach as high as 40°C.

Figure 12 shows the sigma-threshold alarms given for Sensor-11 during the EE parking lot experiment. The system has been trained with the temperature data of August 28 and 29 given in Figure 4. The σ_{coeff} is 3.0. As explained in the previous paragraphs, the system in Rhodiapolis generates no false positives with these parameters. For the EE parking lot experiment, the system generated 14 sigma-threshold alarms for Sensor-11, all during the starting phase of the fire as seen in Figure 12, where each alarm is marked with an arrow. Hence, the system was able to successfully detect the fire in

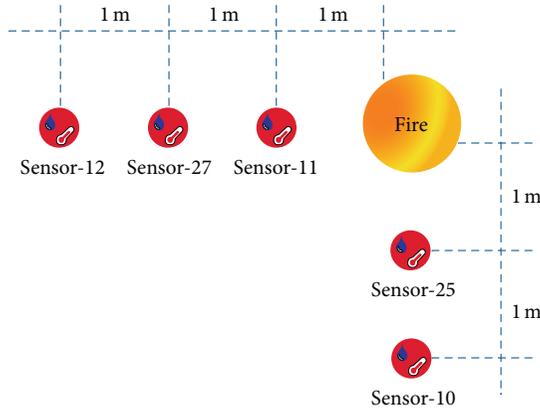


FIGURE 10: Placement of the nodes in the EE parking lot experiment.

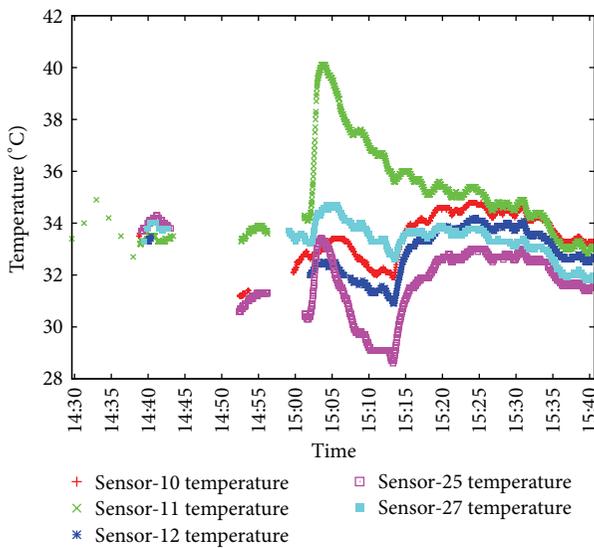


FIGURE 11: Temperature data for all of the five sensors deployed for the EE parking lot experiment. MSI and TXT of all sensors are 1 s.

this experiment by generating multiple fire alarms (one would be enough to warn the firefighting personnel) using the data arriving from Sensor-11.

A second outdoor experiment was performed at a parking lot of the North Campus of the Bogazici University (another partner in the Firesense Project). A total of four sensors were deployed around the fire ignition point. The sampling interval (MSI) and the transmit interval (TXT) of each sensor are set to 100 s. Figure 13 shows the temperature measurements of all of the four sensors for the duration of the experiment. Sensor-22 was closest to the fire and measured a maximum of 47.2°C, the highest temperature among the four sensors. In Figure 14, the sigma-threshold alarms given for Sensor-22 by the system after being trained with the temperature data of August 28 and 29 (Figure 4) are shown. The *sigma_coeff* is again 3.0. There are a total of 6 alarms; hence the fire was detected successfully.

We also tested the system's indoor fire detection capabilities using artificial heat sources. After training the system

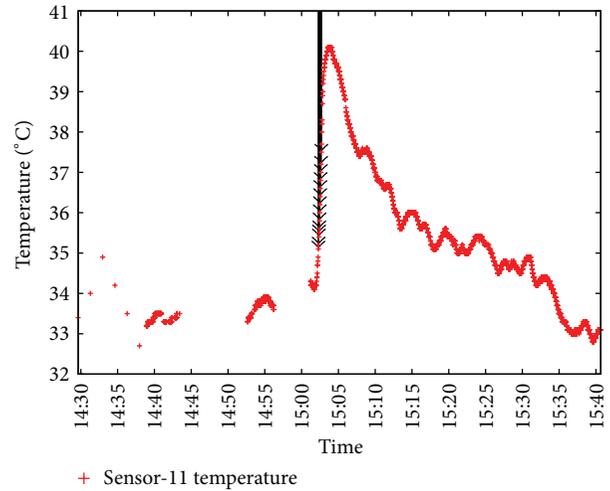


FIGURE 12: Sigma-threshold alarms given for Sensor-11 during the EE parking lot experiment.

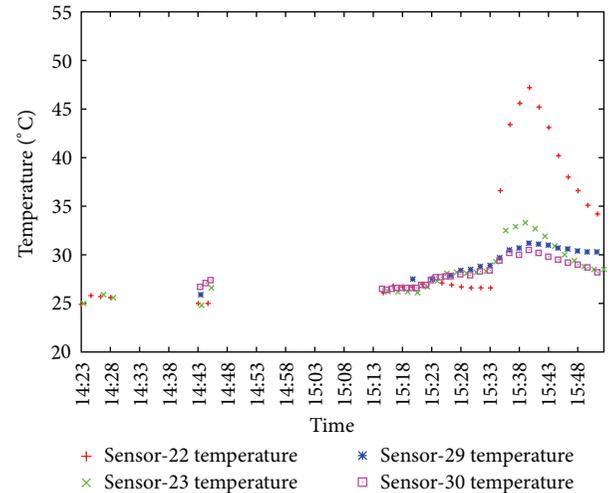


FIGURE 13: Temperature data for all of the four sensors deployed for the Bogazici University parking lot experiment. MSI and TXT of all sensors are 100 s.

consisting of a single sensor with the temperature data from August 28 and 29 (Figure 4) and using a *sigma_coeff* of 3.0, we generated heat using a lighter to make the sensor heat up. The sensor's TXT and MSI periods are 1 s. Figure 15 shows the sigma-threshold alarms given during this experiment together with the temperature data recorded. The lighter was lit three times as the figure reveals and a total of 66 alarms were given at these times by the system, all of which were given when the lighter was on and no alarm was given when the lighter was off. The maximum temperature value recorded by the sensor during this experiment is 38.5°C.

We have also observed the battery voltage levels of the sensor nodes deployed in Rhodiapolis. In our deployment, all sensor nodes have been configured with exactly the same parameters, including the transmission period (TXT = 100 s) and the sampling period (MSI = 100 s). However, after two

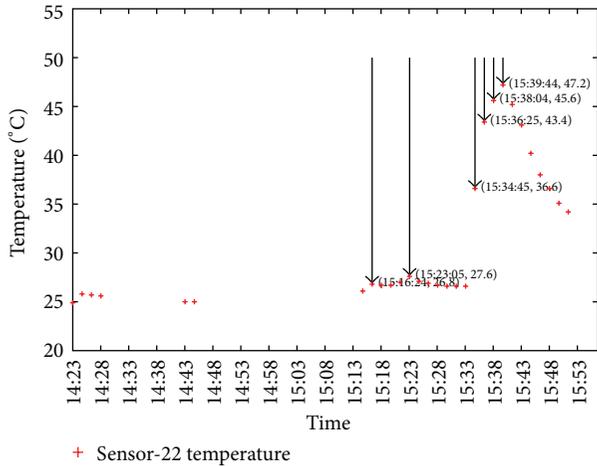


FIGURE 14: Sigma-threshold alarms given for Sensor-22 during the Bogazici University parking lot experiment.

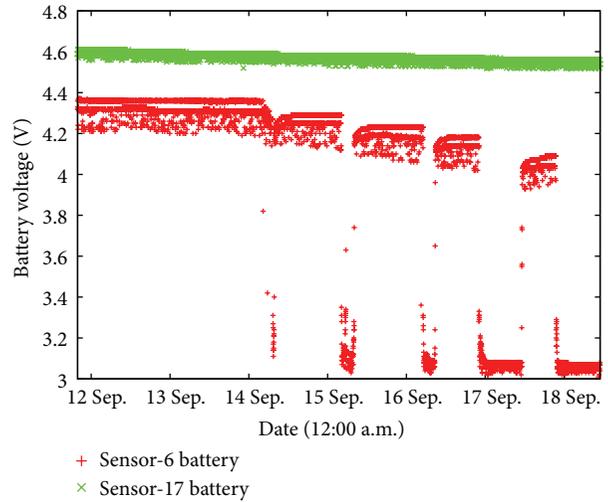


FIGURE 16: Battery voltage levels of Sensor-6 and Sensor-17.

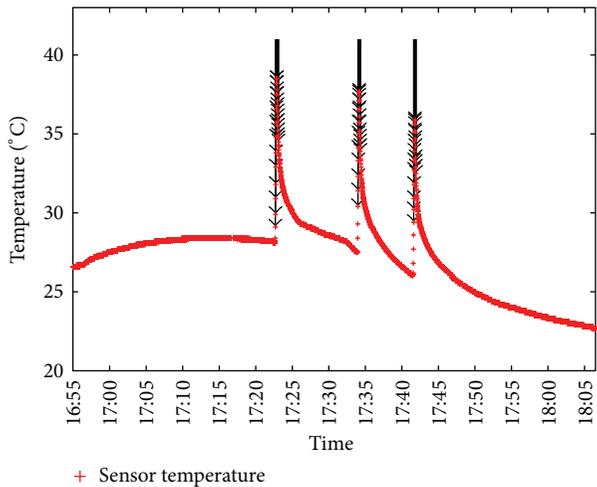


FIGURE 15: Sigma-threshold alarms given during the indoor artificial fire experiment. MSI and TXT of the sensor are both 1 s.

weeks of operation, two sensor nodes in one cluster and one sensor node in another cluster started reporting low battery levels.

Figure 16 shows the change in the battery voltage levels of Sensor-6 and Sensor-17 from the September 12 till the September 18. Sensor-6 is amongst the quick battery drainers whereas Sensor-17 is a normal (relatively slower) drainer. As it can be seen in Figure 16, the voltage levels of Sensor-17 are rather stable at about 4.6 V and drop rather slowly as expected. However, the voltage level of Sensor-6 drops from 4.27 V to 3.31 V on the September 14 in about 100 s, stays below 3.41 V for about 1300 s, and rises back to 4.16 V. Sensor-6 voltage levels oscillate as much as 22%, progressively spending more time below 3.4 V during the observation interval. The standard deviation of the measured battery voltage levels of Sensor-17 between September 12, 2012, and September 18, 2012, is about 0.019 V, whereas the standard

deviation of the measured battery voltage levels of Sensor-6 for the same period is about 0.489 V.

Based on the battery voltage readings of Sensor-6 and Sensor-17 in Figure 16, it should be noted that even though the ZigBee radio module parameters and the application parameters of the sensor nodes are kept the same, the power consumption of individual nodes might differ significantly. One important factor that may cause such differences is the location of the sensor nodes. The XBee S2 radio modules employed by the sensor nodes for ZigBee connectivity retransmit a unicast packet if they do not receive a network-layer acknowledgement for the packet from its destination within specific timeout [14]. If a sensor node is located far away from the ZigBee network coordinator (the cluster head) or if the wireless medium between the sensor node and the coordinator is obstructed, for instance, by trees, then the sensor node may have to retransmit packets frequently. This would imply more power consumption for this sensor node when compared to an unobstructed or nearer node.

We further investigate the maximum battery lifetime of a sensor node by conducting an indoor experiment with a single sensor. The sensor node is placed 1 m apart from its cluster head and there are no other clusters or sensor nodes or known ZigBee transmitters nearby. Since there are no other sensor nodes or ZigBee transmitters, we expect no packet collisions and we expect that the number of retransmissions will be at a minimum for the single sensor. The sensor node is configured to sample its sensing units once every second (1 Hz) and to send the sensor readings once every second to the cluster head. We start the experiment with new, unused batteries and leave the system running till the batteries are completely depleted and the sensor is no longer able to transmit packets.

Figure 17 shows the change in the voltage levels of the sensor during the course of the experiment. The experiment has been run uninterrupted from 23:27:55, October 1, 2012, till 21:53:26, October 24. This corresponds to a battery lifetime of 1981531 seconds (about 23 days).

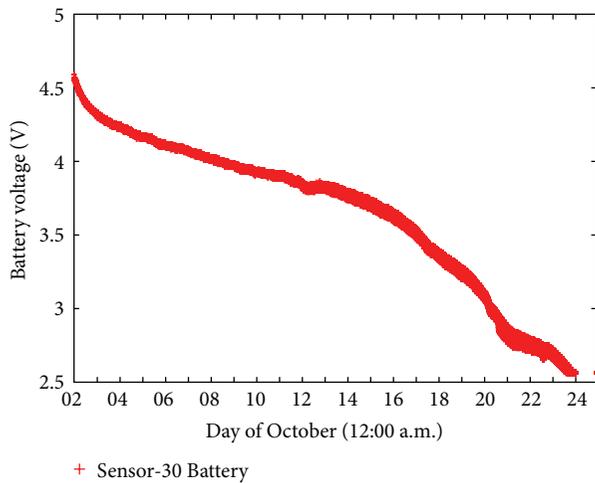


FIGURE 17: Battery voltage levels of Sensor-30 in the battery lifetime experiment.

In the ideal conditions where there are no packet collisions or packet retransmissions and the transmit power of the ZigBee radio is kept at minimum, we may expect a battery lifetime of about 2293 days (about 6.3 years) for a sensor node in our deployment, where the transmission period is set to 100 s. This is a loose upper bound on the battery lifetime of a sensor node deployed in Rhodiapolis because the sensors are placed tens of meters away from their cluster heads and they share the wireless medium. The increased distance between the sensors and their cluster heads causes the sensors to spend more power for transmitting packets than the sensor of the battery lifetime experiment. Also since there are multiple sensor nodes in Rhodiapolis sharing the wireless medium, the increased number of packet collisions and retransmissions will yield shorter battery lifetimes.

7. Conclusions and Discussion

In this paper, we introduced and described our WSN deployment in Rhodiapolis, which has been realized as part of the Firesense EU FP7 project. We explained our network, hardware, and software architectures in detail with the hope that they will be useful to other researchers and networking practitioners with a similar goal of developing and deploying a WSN for habitat monitoring. We gave insights from the operation of the WSN. We also presented a temperature variation based fire detection algorithm and provided some measurement results. From the results discussed in Section 6 it can be concluded that it is better to train the system with at least two days of measurements, and using a sigma coefficient (Section 5.2) of 3.0 should eliminate most of the false positive alarms for the Rhodiapolis deployment.

Also between August 25, 2012, and September 21, 2012, the maximum temperature value recorded by a sensor is 47.5°C. This should be considered when determining the temperature threshold above which an alarm will be issued.

In our outdoor fire experiments we have observed the importance of wind for successfully and timely detecting

wildfires with ambient temperature sensors. As the wind blows from the fire towards a nearby sensor, the sensor's temperature readings rise more rapidly because the wind carries some of the heat energy released by combustion to the sensor. Wind blowing from the fire to the sensors makes the convective heat transfer more efficient.

An important direction for future research is to equip our sensor nodes with a variety of other sensing units such as ambient light or CO units to collect data in more dimensions. Utilizing these data in our sigma-threshold based detection algorithm can increase the system performance and decrease the time needed to detect forest fires.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported in part by the European Union FP7 Programme with Project Firesense, FP7-ENV-2009-1-244088-FIRESENSE. The authors thank Dr. Dimitris Syrivelis and Mr. Dimosthenis Delimpasis of ITI-CERTH for providing them valuable support in the sensor and cluster head hardware. This work was performed when Alper Rifat Ulucinar was a Ph.D. student in the Department of Computer Engineering, Bilkent University, Ankara, Turkey.

References

- [1] "Firesense project home page," 2014, <http://www.firesense.eu/>.
- [2] Z. Li, S. Nadon, and J. Cihlar, "Satellite-based detection of Canadian Boreal forest fires: development and application of the algorithm," *International Journal of Remote Sensing*, vol. 21, no. 16, pp. 3057–3069, 2000.
- [3] Y. E. Aslan, I. Korpeoglu, and Ö. Ulusoy, "A framework for use of wireless sensor networks in forest fire detection and monitoring," *Computers, Environment and Urban Systems*, vol. 36, no. 6, pp. 614–625, 2012.
- [4] J. Zhang, W. Li, Z. Yin, S. Liu, and X. Guo, "Forest fire detection system based on wireless sensor network," in *Proceedings of the 4th IEEE Conference on Industrial Electronics and Applications (ICIEA '09)*, pp. 520–523, Xi'an, China, May 2009.
- [5] J. Zhang, W. Li, N. Han, and J. Kan, "Forest fire detection system based on a ZigBee wireless sensor network," *Frontiers of Forestry in China*, vol. 3, no. 3, pp. 369–374, 2008.
- [6] L. Yu, N. Wang, and X. Meng, "Real-time forest fire detection with wireless sensor networks," in *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing (WCNM '05)*, vol. 2, pp. 1214–1217, IEEE, September 2005.
- [7] Y. Li, Z. Wang, and Y. Song, "Wireless sensor network design for wildfire monitoring," in *Proceedings of the 6th World Congress on Intelligent Control and Automation (WCICA '06)*, vol. 1, pp. 109–113, IEEE, Dalian, China, June 2006.
- [8] "Rfc3626: optimized link state routing protocol (olsr)," 2003, <http://dl.acm.org/citation.cfm?id=RFC3626>.
- [9] "olsrd project home page," 2014, <http://www.olsr.org/>.

- [10] "Rfc3561: Ad hoc on-demand distance vector (aodv) routing," 2003, <http://portal.acm.org/citation.cfm?id=RFC3561>.
- [11] "kernel aodv," 2014, http://w3.antd.nist.gov/wctg/aodv_kernel/index.html.
- [12] "Pikkerton zbs-120/121 interface control document," 2012, http://www.pikkerton.com/_mediafiles/62-interface-control-document-zbs-x2x.pdf.
- [13] "Zbs-120/220-r (outdoor/ruggedized) datasheet," 2009, http://www.pikkerton.com/_mediafiles/37-ds_zbs-x20-r_v1_01_en.pdf.
- [14] "Xbee/xbee-pro zb rf modules manual," 2014, http://ftp1.digi.com/support/documentation/90000976_T.pdf.
- [15] "Xstick usb to xbee wireless pan adapter for laptops & pcs datasheet," 2010, http://www.digi.com/pdf/ds_xstick.pdf.
- [16] "Pcengines alix.2/alix.3/alix.6 series system boards," 2009, <http://www.pcengines.ch/pdf/alix2.pdf>.
- [17] "Wistron cm9 minipci card product home page," 2014, <http://www.mini-box.com/s.nl/it.A/sc.8/category.1557/.f?%20id=387>.
- [18] R. T. Fielding, *Architectural styles and the design of network-based software architectures [Ph .D. dissertation]*, University of California, Irvine, Calif, USA, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.