International Journal of Computer Engineering Science (IJCES)
Volume 3 Issue 2 (February 2013) ISSN : 2250:3439
https://sites.google.com/site/ijcesjournal http://www.ijces.com/

Software Maintenance of Deployed Wireless Sensor Nodes for Structural Health Monitoring Systems

S.A.Quadri[1], Othman Sidek[2]

[1] Collaborative Microelectronic Design Excellence Centre (CEDEC), Universiti Sains Malaysia, Engineering Campus, 14300 Nibong Tebal, Malaysia
Email: reachquadri@yahoo.com

[2] Collaborative Microelectronic Design Excellence Centre (CEDEC), Universiti Sains Malaysia, Engineering Campus, 14300 Nibong Tebal, Malaysia

Abstract. The decreasing cost of sensors is resulting in an increase in the use of wireless sensor networks for structural health monitoring. In most applications, nodes are deployed once and are supposed to operate unattended for a long period of time. Due to the deployment of a large number of sensor nodes, it is not uncommon for sensor nodes to become faulty and unreliable. Faults may arise from hardware or software failure. Software failure causes non-deterministic behavior of the node, thus resulting in the acquisition of inaccurate data. Consequently, there exists a need to modify the system software and correct the faults in a wireless sensor node (WSN) network. Once the nodes are deployed, it is impractical at best to reach each individual node. Moreover, it is highly cumbersome to detach the sensor node and attach data transfer cables for software updates. Over-the-air programming is a fundamental service that serves this purpose. This paper discusses maintenance issues related to software for sensor nodes deployed for monitoring structural health and provides a comparison of various protocols developed for reprogramming.

Keywords: software maintenance, over-the-air programming, wireless sensor network, node deployment, structural health monitoring

1 Introduction

Recent developments in the miniaturization of sensing, computing, and communication technology have made it possible to use a large number of sensors within a wireless sensor network. Their low cost makes it feasible to deploy them in significant numbers across large areas, and consequently, these devices have become a promising candidate for structural health monitoring (SHM) systems. Most monitored structures have sensors that measure several types of parameters. Some examples of applications are the Gotaalvbron bridge in Sweden, where more than 70,000 sensors are installed in a single bridge (55,000 strain sensors and 11,000 temperature sensors) [1], and the new I35 bridge in Minneapolis, which has more than 350 sensors (150 strain gauge vibrating wires, 150 thermocouples, 10 potentiometers, 20 accelerometers, 4 corrosions, and 12 long gauge optical fibers) [2, 3]. In the United States, 61 of California's long span bridges have been instrumented with over 900 sensing channels [4]. Small structures may require 10 to 15 sensor channels, while large bridges may need as many as 350 sensor channels [5].

In most applications, sensor networks are deployed once and expected to operate unattended for a long period of time. The real challenge is how to manage and maintain this large scale network of wireless sensor nodes (WSN). Beutel et al. [6] discuss the real time problems encountered during the deployment of nodes; 14 different projects were reviewed with different goals, requirements and levels of success in deploying the sensor network. The study analyzes each network's behavior in detail. While concerning the issue of faults using WSN networks in SHM systems, a software crash was reported, which resulted in a complete loss of data during the monitoring of a building structure under study. In the deployment of wireless sensor nodes in a highway bridge over Big Sucker Brook, Waddington, NY, Whelan et al. [7] state that the only issue encountered was related to software.

The limited computational resources available on a node impose some restrictions on the amount of processing that can be successfully performed at the node. If this limit is exceeded, processing tasks may not run to completion causing non-deterministic behavior and various kinds of failures. The software embedded in the node may be corrupted. Pointers and memory locations may become corrupted, message buffers may be overwritten, and certain sensing and processing events might get lost. The node might even be forced into deadlock or livelock states from which it cannot recover on its own. Obtaining accurate data is the primary objective of the node. Even when any single node fails, resulting in non contribution to the aggregated data acquisition, it obviously increases the margin of error. The false sensor readings may lead to serious consequences for the analyzers.

Mainstream software systems have repeatedly proven the need for adaptability and extensibility. While the nature and frequency of changes may differ in WSN networks, software artifacts will change after they have been deployed in the field, and a careless approach to software evolution will result in short-lived applications. In conventional systems, software maintenance accounts for 60-70% of software costs [8]. About 50% of this effort is perfective, 21% corrective, 25% adaptive and 4% preventive [9]. Thus, a variety of workarounds such as wrappers and patches are used

to enable modifications [10]. Therefore, there exists a need to maintain the system software in the WSN for long-term, efficient and reliable performance of the nodes.

Over-the-air (OTA) programming is a fundamental service that is based on reliable broadcast communication. Over-the-air programming (OAP) eliminates the need for detaching the sensor nodes and attaching data transfer cables when updating the sensor software. Although existing OAP protocols have many merits, they suffer from fundamental limitations that can significantly impair their use in future systems. Mainly, the performance of existing OAP protocols quickly degrades as the network size and density increase, and even more so when packet loss is high. Unlike other protocols, these have to be designed very carefully; furthermore, the WSN has its own design and resource constraints. Resource constraints include a limited amount of energy, short communication range, low bandwidth and limited processing and storage space in each sensor node.

This paper provides a brief overview of software maintenance, node deployment and maintenance issues, and provides a comparison of various OAP protocols developed for reprogramming nodes.

2 Software Maintenance

The IEEE definition for software maintenance is [11]: "Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment." This definition reflects the common view that software maintenance is a post-delivery activity: it starts when a system is released to the customer or user and encompasses all activities that keep the system operational and meet the user's needs. This paper considers post deployment activity of sensor nodes in a WSN network.

Lientz and Swanson [12] divide maintenance into three components: corrective, adaptive, and perfective maintenance. Corrective maintenance includes all the changes made to remove actual faults in the software. Adaptive maintenance encompasses the changes needed as a consequence of some mutation in the environment in which the system must operate, for instance, altering a system to make it run on a new hardware platform, operating system, DBMS, TP monitor, or network. Finally, perfective maintenance refers to changes that originate from user requests; examples include inserting, deleting, extending, and modifying functions, rewriting documentation, improving performances, or improving ease of use. Pigoski et al. [13] suggest joining the adaptive and perfective categories and calling them enhancements, as these types of changes are not corrective in nature, but are improvements. As a matter of fact, some organizations use the term software maintenance to refer to the implementation of small changes, whereas software development is used to refer to all other modifications.

IEEE [14] redefines the Lientz and Swanson [12] categories of corrective, adaptive, and perfective maintenance, by adding emergency maintenance as a fourth category. The redefined IEEE definition is [14]: "Corrective maintenance: reactive modification of a software product performed after delivery to correct discovered

faults. Adaptive maintenance: modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment. Perfective maintenance: modification of a software product performed after delivery to improve performance or maintainability. Emergency maintenance: unscheduled corrective maintenance performed to keep a system operational."

WSN networks may operate in remote and harsh environments and, in this case, applications need to operate in an unattended way for long periods of time. These challenges introduce some difficulties. Firstly, the environment can evolve over time, so it is very difficult to anticipate how a sensor node must operate during its lifetime. Secondly, the application requirements can change. For example, due to technological advances or even a better understanding of the environment, it would be interesting to have different application behaviors. Thirdly, software bugs often result in node reboots; for example, bugs can cause failure to restart the watchdog timer of the micro controller. Furthermore, it has been observed that software bugs result in hanging or killing threads, such that only part of the sensor node software continues to operate [15].

The aforementioned difficulties make WSN management and maintenance challenging tasks. Software maintenance that involves updating is a necessary requirement for a reliable service in order to maintain stability and enable modification of the characteristics of wireless sensor networks [16-19]. The following sections discuss node deployment issues and available techniques for efficient maintenance of system software.

3 Node Deployment and Maintenance Issues

As sensor networks move from research to deployment, from laboratory to the real world, issues of management and reconfiguration will grow in importance. A sensor network is composed of a large number of sensor nodes that are densely deployed either inside the phenomenon or very close to it. The position of sensor nodes need not be engineered or predetermined. This allows random deployment in inaccessible terrains or disaster relief operations. On the other hand, this also means that sensor network protocols and algorithms must possess self-organizing capabilities. Basically, operating system (OS) support is important to facilitate the development and maintenance of WSN networks [20-28]. As the number of devices in the monitoring systems increases, the amount of attention paid to networking, processing and maintenance also increases sharply. Various issues related to nodes are discussed below.

3.1 Features of Sensor Nodes

A sensor node, also known as a 'mote' is a node in a wireless sensor network that is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network. Sensor nodes are fitted with an onboard processor. Instead of sending the raw data to the nodes responsible for the fusion, they use their processing abilities to locally carry out simple computations and transmit only the required and partially processed data.

- The number of sensor nodes in a sensor network can be several orders of magnitude.
- Sensor nodes are densely deployed.
- Sensor nodes are prone to failures.
- The topology of a sensor network changes very frequently.
- Sensor nodes mainly use a broadcast communication paradigm.
- Sensor nodes are limited in power, computational capacities, and memory.
- Sensor nodes may not have global identification (ID) because of the large amount of overhead sensors.

3.2 Hardware Constraints

A sensor node is made up of four basic components, a sensing unit, a processing unit, a transceiver unit, and a power unit. They may also have additional application-dependent components such as a location finding system, power generator, and mobilizer. Sensing units are usually composed of two subunits: sensors and analog-to-digital converters (ADCs). The analog signals produced by the sensors based on the observed phenomenon are converted to digital signals by the ADC, and then fed into the processing unit. The processing unit, which is generally associated with a small storage unit, manages the procedures that make the sensor node collaborate with the other nodes to carry out the assigned sensing tasks. A transceiver unit connects the node to the network. One of the most important components of a sensor node is the power unit.

All of these subunits may need to fit into a matchbox-sized module [29].The required size, however, may be smaller than even a cubic centimeter [30], which is light enough to remain suspended in the air. Apart from size, there are some other stringent constraints for sensor nodes. These nodes must consume extremely low power, operate in high volumetric densities, have low production cost, be dispensable and autonomous, operate unattended, and be adaptive to the environment [31].

3.3 Sensor Network Topology

Hundreds to several thousands of nodes may be deployed throughout the sensor field. They are deployed within tens of feet of each other [29]. The node densities may be as high as 20 nodes/m3 [32]. Deploying a high number of nodes densely requires careful

handling of topology maintenance. The issues related to topology maintenance can be examined in three phases [33]:

- Predeployment and deployment phase: Sensor nodes can be either thrown in as a mass or placed one by one in the sensor field. They can be deployed by dropping from a plane, delivered in an artillery shell, rocket, or missile, and placed one by one by either a human or a robot.
- Post-deployment phase: After deployment, topology changes are usually due to change in a sensor node's position, reachability (due to jamming, noise, moving obstacles, etc.), available energy, malfunction, and task details.
- Redeployment of additional nodes phase: Additional sensor nodes can be redeployed at any time to replace malfunctioning nodes or due to changes in task dynamics.

Cheng et al. [34] have discussed various deployment strategies for large scale deployment considering extension of the network lifetime. Various models and algorithms on node deployment can be found in [35-39].

3.4 Power Supply and Consumption

A major challenge impeding the deployment of wireless sensor networks for structural health monitoring (SHM) is the development of a means to supply power to the sensor nodes in an efficient manner. The various schemes can be studied in [40-45]. A wireless sensor node, being a microelectronic device, can only be equipped with a limited power source (< 0.5 Ah, 1.2 V). In some application scenarios, replenishment of power resources might be impossible. The sensor node lifetime, therefore, shows a strong dependence on battery life. In a multihop ad-hoc sensor network, each node plays the dual role of data originator and data router.

The malfunctioning of a few nodes can cause significant topological changes and might require rerouting of packets and reorganization of the network. Hence, power conservation and power management take on additional importance. It is for these reasons that researchers are currently focusing on the design of power-aware protocols and algorithms for sensor networks. The main task of a sensor node in a sensor field is to detect events, perform quick local data processing, and then transmit the data. Power consumption can hence be divided into three domains: sensing, communication, and data processing [33].

Aggressive research is being conducted into the development of self powered sensors for SHM applications, which could be a potential solution to the problem and enable permanent unattended sensor installations. Sazonov et al. [46] have presented a prototype of a novel self-powered wireless system for applications in SHM of bridges, which relies on harvesting the energy of bridge displacements created by passing traffic to produce power.

3.5 Node Localization

Wireless sensor networks (WSN) owe their success mainly to the modern technological advancements in recent years that have enabled the production of low-cost, low-power and minute sensor nodes. When deployed at random, the wireless sensor networks can be treated as a class of ad-hoc self configuring networks which operate without the help of any dedicated base station. One of the main challenges in such an ad-hoc configuration is the localization of sensor nodes upon deployment in some area of interest. Because of the deployment of a large number of sensor nodes, it is often not possible to hand place these sensor nodes.

Localization of sensor nodes is important in many respects in a wireless sensor network; first, it informs the remote end user of the precise location within a network where the specific "event' of interest took place. Second, in case of node failure, identification of the affected area within the sensor network can be made. Similarly, the known location of sensor nodes helps in determining efficient routing paths to the sink, either mobile or static, thus conserving energy and time. In wireless sensor networks, the capabilities of individual sensor nodes are extremely limited, and thus, collaboration is required with minimum energy expense. Localization, therefore, plays an important role. The estimation of localization in wireless sensor networks can be further studied in [47-50].

3.6 Node Maintenance

A wireless sensor network is a system of small, wirelessly communicating nodes, where each node is equipped with multiple components. Due to the deployment of a large number of sensor nodes in uncontrolled or even harsh or hostile environments, it is not uncommon for the sensor nodes to become faulty and unreliable. Fault is an incorrect state of hardware or a software program resulting in failure of a component [51]. Ko et al.[52], while sharing their experience in long-term bridge monitoring systems, express that the utmost care must be taken to protect the data acquisition unit (DAU). At least two health monitoring systems out of the 20 that were under study were found with malfunctions due to improper DAU protection. DAUs must be designed to resist a variety of environmental conditions, such as temperature, humidity, lightning and electromagnetic interference.

The failure of sensor nodes, however, should not affect the overall task of the sensor network. This is the reliability or fault tolerance issue. Fault tolerance is the ability to sustain sensor network functionalities without any interruptions due to sensor node failures [53, 54]. Some of the faults result from system or communication hardware failure and the fault state is continuous in time. For example, a node may die due to battery depletion. Faults occurring due to energy depletion are continuous and, as time progresses, these faults may increase, resulting in a non-uniform network topology [55, 56].

Problems that can occur due to sensor node failure are: loss in connectivity, delay due to the loss in connection, and partitioning of the network due to the gap created by the failed sensors. Therefore, to overcome sensor node failure and to guarantee

system reliability, faulty nodes should be detected and appropriate measures to recover connectivity must be taken in order to compensate for the faulty nodes. Coverage maintenance arises when sensor nodes die due to battery drain or environmental causes. The area that was covered by the dead node may be left uncovered, which is a potential point of vulnerability in the field. Hence, coverage maintenance is required to make up for the death of the node [57-59].

As far as the issue of energy conservation is concerned, many protocols have been developed. Clustering is one such attempt to control energy dissipation for sensor data dissemination in a multihop fashion [60-67]. Also, the power supply on each sensor node is limited, and frequent replacement of the batteries is often not practical due to the large number of nodes in the network. The issue related to maintenance or replacement of batteries, whether primary or secondary, can be studied in detail in [68, 69]. The worst case, in which the nodes have gone out of order, is that they have to be replaced, which is known as redeployment [70, 71].

There are two widely different node failure models used to study the fault tolerance of nodes: the isolated failure model and the geographically correlated failure model [72, 73]. In order to diagnose and detect faults, Huang et al. [74] proposed a technique called Sympathy, which is a diagnostic tool for detecting and debugging failures in sensor networks. It is specifically designed for data-collection applications, where nodes periodically send data back to a centralized base station or sink. Sympathy detects failures in a system by selecting metrics such as connectivity, data flow, node's neighbor and next hops. Connectivity metrics provide connectivity information from every node in the network. Sympathy collects every node's current routing table with information for the next hop and path quality. Flow metrics provide the network's traffic load as well as its connectivity. Sympathy collects packet level information transmitted and received from each node. In addition, Sympathy also maintains information for packets transmitted from the sink to the nodes. Based on these metrics, Sympathy detects when nodes are not delivering sufficient data to the sink and locates the cause of the failure.

Sympathy can identify three types of failures: self, path, and sink. In self failure, the node itself has failed due to a crash, re-boot, bug in software code, or connectivity issue. In path failure, a node along the path fails causing other nodes to fail, or there are collisions along the path. In sink (base station) failure, the whole network appears to be failing. Failure at the sink may be due to bad sink placement, changes in the environment after deployment, or connectivity issues. The software in the nodes is a necessary requirement for reliable service. It is also necessary to maintain the stability and enable modification of the characteristics of wireless sensor networks [16-19]. Once the nodes are deployed, however, it is impractical to reach each individual node. Moreover, it is highly cumbersome to detach a sensor node and attach data transfer cables for software updates. Thus, a scheme is required to wirelessly reprogramme the nodes.

Whelan et al. [75] conducted a study on the integral abutment of a highway bridge. Their test results describe a small loss of data because of a software bug, which crept into the embedded software code. Stajano et al [76] discuss 3 sites where WSN monitoring was deployed. The first site is the north anchorage chambers of the Humber Bridge, which is a major suspension bridge that crosses a river estuary in

East Yorkshire. Each of the four underground anchorage chambers have dehumidification units to ensure that the exposed unprotected steel strands of the main suspension cables of the bridge do not corrode while exposed to the air in the chamber. A 12-node WSN was installed in the north chambers to verify whether these dehumidification units were operating correctly. The second site is a reinforced concrete bridge, known as the Ferriby Road Bridge, located several hundred meters to the north of the Humber Bridge. A seven-node WSN was installed, with three nodes measuring changes in crack widths, three nodes measuring changes in bearing inclination, and a final node measuring temperature. The third site is a London underground tunnel on the Jubilee Line. There, a network of 26 nodes was set up to measure changes in displacement, inclination, temperature, and relative humidity in the 180 m-long stretch of concrete-lined tunnel. The study discusses the practical difficulties of node deployment and suggests a technique in relation to the issue of software maintenance, known as over-the-air (OTA) programming technique, by virtue of which a whole network of nodes can be reprogrammed by uploading a new code image into them.

Joel Koshy et al [18] discussed the reasons why it is necessary to remotely reprogram sensor nodes. First, the scale and distributed nature of WSN applications makes it difficult to get things right the first time. Bug fixes, run-time application adaptation and other software maintenance concerns can be addressed only by employing a reprogramming mechanism. Secondly, application specialization to optimize energy usage and performance for network longevity may be possible only during run-time. WSNs tend to be highly sensitive computing environments in which small local changes can result in significant global consequences. Normally, an ideal configuration can only be attained empirically at run-time, because it is not possible to anticipate every application scenario. For example, it may be necessary to choose a routing protocol from a protocol suite at run-time, suited to prevalent conditions [77]. Similarly, knobs such as power management, radio frequency modulation and dynamic voltage scaling algorithms may need to be adjusted at run-time. Thirdly, it is likely that some WSNs will be deployed for long periods of time and provide different services at different times. Due to storage constraints, it is unfeasible to load all these services into the nodes prior to deployment. Instead, applications and services could be swapped in and out, depending on contexts of use, such as current time, location, user input, and environmental stimuli. Run-time service composition is also useful in context-aware pervasive computing environments with small computing devices [78].

3 Over-The-Air (OTA) Programming

Over-the-air programming (OTA) is a fundamental service that depends on reliable broadcast communication. OTA programming eliminates the need for detaching the sensor nodes and attaching data transfer cables when updating the sensor software. The desirable characteristics of OTA protocols for reprogramming are firstly, the time and space complexity of algorithms in reprogramming should be well fitted to the

capacity profile of a sensor node, since they are generally small in size with limited hardware capacities. Secondly, reprogramming should be energy-efficient; sensor nodes are usually battery powered and can hold/gain limited amounts of energy. Among computing, communication, and sensing functions, communication consumes a large portion of the energy. Thirdly, reprogramming requires the program code to be delivered in its entirety. However, wireless communication is unreliable due to possible signal collisions, interferences, and packet contentions. Fourthly, scalability is crucial for large-scale sensor network deployment. Scalability has two requirements for a widely applicable reprogramming service: scale for number of nodes, from tens up to hundreds or even thousands of nodes, and scale for varying node density, from sparse to dense networks. Fifthly, there are several programming support limitations in current TinyOS [79].

Ideally, maintenance operations should not degrade the reliability and the structure of the subject system; neither should they degrade its maintainability. Moreover, the activity of maintenance should be effective, convenient, and as far as possible should incur the least overhead costs. Several protocols have been designed and studied in the past few years; a brief review is given below.

In contrast to the traditional way of reprogramming microcontrollers using In-System Programming (ISP), sensor networks need a way to update the node's firmware without human intervention. Consequently, the University of California at Berkeley developed XNP [80], a one-hop protocol that offers firmware updates through a wireless link. XNP (Crossbow Network Programming) is the network programming implementation for TinyOS that was introduced with a 1.1 release version. The XNP implementation provides the basic capability of network programming; it delivers the program code to the sensor nodes remotely. XNP provides single hop solution; however, it has some limitations. First, XNP does not scale to a large sensor network. It disseminates program code only to the nodes that can be directly reached by the host machine. Thus, the nodes outside the single hop boundary cannot be programmed. Second, XNP has low effective bandwidth compared to ISP; when XNP updates the program code with another version, it sends the whole program code rather than the difference. This incurs the same programming time even when the difference is small. If the sensor nodes can build program code images incrementally using the previous code image, the overall programming time can be reduced [81].

Reijers et al. [82] proposed an energy-efficient code distribution scheme to wirelessly update the code running in a sensor network. Energy is saved by distributing only the changes to the currently running code. The new code image is built using an edit script of commands that are easy to process by the nodes. A small change to the program code can cause many changes to the binary code because the addresses of functions and data change. The scheme is resilient to missing packets in that it can continue processing the following packets and start a recovery procedure in a later phase. Updating all software on the nodes is possible, including the operating system and the code distribution scheme itself. The scheme distributes binary native code, so the programmer is not bound to a virtual machine, but can do all low level optimizations necessary when programming for wireless sensor networks. The scheme significantly reduces the amount of communication compared to simply

transferring the binary code. Nevertheless, the algorithms lack speed and a lot of overhead is incurred when splitting the script.

MOAP is a multihop network programming mechanism developed by Stathopoulos et al. [83]. The main contributions of MOAP are its code dissemination and buffer management. One of the challenges of multihop network programming is propagating program codes over multiple sensor nodes without saturating the network; so they used ripple dissemination protocol to regulate the network traffic. Ripple protocol disseminates the program code packets to a selective number of nodes without flooding the network with packets. For buffer management, they used a sliding window scheme. Sliding window schemes maintains a window of program codes and allows lost packets within the window to be retransmitted. Sliding windows take small footprints so that packets can be processed efficiently in on-chip RAM. While MOAP advances the data dissemination problem, it still ignores many design decisions. MOAP requires nodes to receive the entire code image before making advertisements. It does not allow the use of spatial multiplexing to leverage the full capabilities of the network. Methods for intelligent sender selection are not considered. While the authors mention the possibility of using forward error-correction techniques, no evaluation is provided to show their usefulness.

Jeong et al. [84] proposed an incremental approach in which the host program generates the difference of the two program images using the Rsync algorithm [85], and then sends the difference to the sensor nodes as script messages. The sensor nodes rebuild the program image based on the previous program version and the received script messages. The Rsync algorithm compares the two binary files and finds the matching blocks even though they are located in an arbitrary location within the files. This approach speeds up the transmission time. It assumes no prior knowledge of the program code structure (hardware independent), and overhead is incurred only in calculating the rolling checksum in the Rsync algorithm.

Levis [86] proposed an algorithm called Trickle, for propagating and maintaining code updates in wireless sensor networks. Trickle uses a “polite gossip” policy, where motes periodically broadcast a code summary to local neighbors but stay quiet if they have recently heard a summary identical to theirs. When a mote hears an older summary than its own, it broadcasts an update. Instead of flooding a network with packets, the algorithm controls the send rate so each mote hears a small trickle of packets, just enough to stay up to date. With this simple mechanism, Trickle can scale to thousand-fold changes in network density, propagate new codes in the order of seconds, and impose a maintenance cost on the order of a few sends an hour. The behavior of trickle is almost the inverse of protocols such as SPIN [87], which transmits metadata freely but controls data transmission. One limitation of Trickle is that it currently assumes motes are always on. To conserve energy, long-term mote deployments often have very low duty cycles (1%). Correspondingly, motes are rarely awake, and rarely able to receive messages.

Chlipala et al. [88] proposed Deluge, a reliable data dissemination protocol for propagating large amounts of data (more than can fit in RAM) from one or more source nodes to all other nodes over a multihop, wireless sensor network. To achieve robustness against lossy communication and node failures, an epidemic approach was adopted. Representing the data object as a set of fixed-sized pages provides a

manageable unit of transfer, which supports spatial multiplexing and provisions for incremental upgrades. Due to the large data size, it identifies a set of possible optimizations and evaluates their effectiveness. Deluge algorithm reliably distributes data across an increasingly sized multi-hop network, while maintaining a constant amount of local state. And the energy required to distribute this data is within the allowable per-mote energy budget. Deluge is similar to SPIN-RL [89] in that it makes use of a three-stage (advertisement-request-data) handshaking protocol. SPIN-RL is designed for broadcast network models and provides reliability in lossy networks by allowing nodes to make additional requests for lost data. The interaction between nodes is kept strictly local and avoids the need to maintain neighbor tables. This property allows Deluge to be robust to widely varying connectivity scenarios. Since there is no need to maintain state about all neighboring nodes, nodes may move and connectivity can vary without requiring nodes to adapt to such changes. Enabling spatial multiplexing through per-page pipelining decreases the time to complete and transmit messages.

Kulkarni and Wang [90] proposed a multihop network reprogramming protocol (MNP), which provides a reliable service to propagate new program codes to all sensor nodes in the network over radio. In multihop reprogramming, any node that has the new code image is a potential sender. It is likely that too many senders are transmitting at the same time. Sender selection mechanism is proposed, in which source nodes compete with each other based on the number of distinct requests they have received. This effectively solves the concurrent sender problem. One of the problems in reprogramming is the issue of message collision. To reduce the problem of collision, a sender selection algorithm is proposed that attempts to guarantee that in a neighborhood there is at most one source transmitting the program at a time. The sender selection is greedy in that it selects the sender that is expected to have the most impact. It uses pipelining which enables fast data propagation. It is energy efficient because it reduces the active radio time of a sensor node by putting the node into "sleep" state when its neighbors are transmitting a segment that is not of interest. In MNP, sensor nodes do not need to have any location information or maintain neighbor status. Sensor nodes make local decisions independently and, hence making the protocol scalable.

Arumugam et al. [91] proposed Infuse, a time division multiple access-based (TDMA) reliable data dissemination protocol. Infuse takes two input parameters: the choice of the recovery algorithm to deal with unexpected channel errors (e.g., message corruption, varying signal strength), and whether a sensor should listen only to a subset of its neighbors to reduce the amount of active radio time. He considered two recovery algorithms based on the sliding window protocols that use implicit acknowledgments: go-back-N and selective retransmission. Since Infuse uses a TDMA-based MAC protocol, sensors need to listen to the radio only in the slots assigned to their neighbors. In the remaining slots, sensors can turn off their radio. Moreover he proposed an algorithm to reduce messages receptions and the active radio time further by using the notion of preferred predecessors.

Naik [92] presented Sprinkler, a reliable data dissemination service for wireless embedded devices that are constrained in energy, processing speed, and memory. Sprinkler embeds a virtual grid over the network whereby it can locally compute a

connected dominating set of the devices to avoid redundant transmissions and a transmission schedule to avoid collisions. To reduce energy consumption, Sprinkler computes a subset of nodes as senders. The subset is connected and every node in the network has a neighbor in the subset. The problem of selecting the minimum number of senders is computing a minimum connected dominating set (MCDS) of the graph induced by the wireless network, which is known to be NP hard even for a unit disk graph [93]. Sprinkler effectively manages the latency by computing a near optimal schedule using a local D-2 coloring algorithm [94]. As a cluster-based approach, Sprinkler divides the whole WSN area into square-shaped clusters, and one node is selected in each cluster as the cluster head. It maintains hierarchy, which is the concept of super nodes or cluster head nodes. A connected dominating set (CDS) is calculated from the cluster head set. The nodes in CDS will be selected to receive and rebroadcast the data in the first phase in Sprinkler. In the second phase data will be transmitted from CDS nodes to all non-CDS nodes. Compared to the other schemes, the CDS algorithm is centralized and causes extra overhead.

Levis et al. [95] proposed the Firecracker protocol for data dissemination in wireless sensor networks. Firecracker uses a combination of routing and broadcasts to rapidly deliver a piece of data to every node in a network. To start dissemination, the data source sends data to distant points in the network. Once the data reaches its destinations, broadcast-based dissemination begins along the paths, like a string of firecrackers. By using an initial routing phase, Firecracker can disseminate at a faster rate than scalable broadcasts, while sending fewer packets. The selection of points to route to has a large effect on performance, indicating possible requirements for any-to-any routing protocols in wireless sensor networks. It maintains hierarchy, which is the concept of super nodes or cluster head nodes. Super nodes in Firecracker are nodes in each corner, or are randomly selected. With this hierarchy approach, Firecracker achieves threefold speedup, using one-third the transmission cost of Trickle [86].

Phillips [96] presented Aqueduct, which establishes “aqueducts” of intermediate nodes between source nodes and target nodes. Data is only propagated along these aqueducts. Aqueduct adheres to four primary design principles: (1) dynamic network reprogramming for heterogeneous WSNs with diverse hardware, software, and application roles, (2) efficiency in terms of reduced code overhead by limiting involvement in forwarding, (3) robustness to spatially irregular, time-varying RF links by constructing symmetric links, (4) and incorporation of new capabilities into the framework provided by Deluge [88]. Aqueduct adds new capabilities to Deluge, modifying its state machine and state transitions, while also adding a new forwarding state. Aqueduct offers a practically useful code propagation protocol for heterogeneous WSNs that has been evaluated over a real test bed and has shown to incur significantly lower overhead than Deluge [88]. While reprogramming a partial network, scope selection approach of aqueduct saves energy across the whole network.

Marron et al. [97] proposed the TinyCubus project in order to solve three key issues to provide efficient management and configuration of applications and system software in sensor networks: the distribution and management of roles within the network, efficient code distribution algorithms, and efficient on-the-fly code update

algorithms for sensor networks. The first issue is motivated by the increasing heterogeneity of sensor network applications and their need for more complex (non-homogeneous) network topologies and structures. The second one is motivated by the intrinsic energy constraint issues and, in general, the resource limitation of sensor networks. Finally, the third one is needed due to the nature of monitoring applications and optimization needs from applications that should be able to efficiently incorporate code updates so that the network can adapt to its surroundings on the fly.

However, most current code update algorithms always transmit the complete code image (including the operating system), which usually amounts to several kilobytes, or blindly divide the code image into blocks without considering the structure of the code. Examples of these two approaches are XNP [80] and Deluge [88]. XNP is included in TinyOS 1.1. It lacks the ability to forward code in a multi-hop network and simply broadcasts the complete code image in a single-hop network. Deluge has been included in more recent TinyOS releases to replace XNP. It allows for incremental updates by dividing the code into fixed-sized pages. In addition, it includes functionality to disseminate the update in a multi-hop network, while keeping the number of network packets low.

Reijers and Langendoen [82] use a diff like approach to compute an edit script that transforms the installed code image into a new one. Likewise, the incremental network programming protocol presented by Jeong and Culler [84] uses the Rsync algorithm [85] to find variable-sized blocks that exist in both code images and then only transmits the differences. However, both of these approaches just compare the bytes of the code without using knowledge about the application structure. In some cases this leads to inefficient behavior.

In all of these approaches, all nodes will eventually have installed the same code image without support for adaptation. However, in order to reduce the number of packets, it might be desirable to install the required components only on those nodes that need it and store the other ones, if they are received at all, for later adaptation in a free part of flash memory, which typically is less constrained than program memory. Therefore, the solution uses knowledge about the application structure by grouping codes into packages of components. It offers more flexibility than simply replacing arbitrary pieces of codes because it makes it possible to dynamically change the current set of installed packages through adaptation. That way the sensor nodes can possess several components even though they only need one of them for their current role. When the role changes or other factors make it necessary, the node can easily exchange the currently used component.

TinyCubus proposes a flexible description language and uses a role concept in scope selection. It reprograms nodes with a particular role via nodes in a specific role (e.g., reprogramming all temperature sensors via vibration sensors). However, TinyCubus faces efficient code dissemination problems because it cannot guarantee that all target nodes can be reached and reprogrammed.

Lee et al. [98] proposed a transaction-based approach to solve the version inconsistency problem of OTA programming. A multinode OTA programming procedure is modeled as a transaction, such as a database transaction, and the programming of each individual node is modeled as a sub transaction. When the update is finished, the system and the manager could commit or abort the transaction

according to the result of the sub transaction. The commit and abort actions synchronize the software version and thus the version inconsistency problem would never occur. The system was implemented on a zigbee [99] platform developed by Industrial Technology Research Institute (ITRI) Hsinchu, Taiwan. The transaction-based update system is managed by command line and a TCL scripting interface [100,101]. Most computation power required by the system relies on the base station, thus a more powerful base station is required.

Hagedorn et al. [102] proposed a scheme known as Rateless Deluge, based on rateless codes, which significantly improves OAP in such environments by drastically reducing the need for packet rebroadcasting. Rateless codes provide an efficient means of addressing channel contention in sensor networks, while at the same time minimizing control messages, such as those contributing to the ACK/NACK implosion problem. Two rateless OAP protocols were designed and implemented, rateless Deluge and ACKless Deluge, both of which replace the data transfer mechanism of the established OAP Deluge protocol with rateless analogs. Compared to Deluge [88], one of the most widely used OAP protocols at present, these implementations (i) reduce communication on both the data and control planes, (ii) reduce latency at moderate levels of packet loss, (iii) are more scalable to dense networks, and (iv) generally consume far less energy, a premium resource in wireless sensor networks. Although ACKless Deluge adds communication on the data plane, it is particularly efficient on the control plane as it almost completely eliminates the needs for retransmission requests by receiving nodes and packet retransmissions by sources. Since it is unlikely that nodes will request packets belonging to a previous page, ACKless Deluge is able to take full advantage of pre-coding and speed-up data transfer. A simple mathematical approach is provided to determine the number of extra packets needed by ACKless Deluge in order to guarantee with high probability, such that all the nodes receive enough packets to decode a page. Overall, the rateless Deluge, augmented with FEC mechanism, achieves excellent performance with respect to almost all the metrics relevant to wireless sensor networks.

Panta et al. [103] presented a protocol called Stream that greatly reduces the number of bytes transmitted over the wireless medium for reprogramming. The application image together with the reprogramming protocol image is transferred. Using the facility of having multiple code images on a node and switching between them, Stream pre-installs the reprogramming protocol as one image and the application program equipped with the ability to listen to new code updates as the second image. This consequently, reduces the reprogramming time, the number of bytes transferred, the energy expended, and the usage of program memory.

Krasniewski et al. [104] proposed a protocol called Freshet for optimizing the energy for code upload and speeding up the dissemination if multiple sources of codes are available. A fundamental insight used in Freshet is that nodes can be put to sleep by making the advertisement-request-data handshake happen only at certain points in time. The energy optimization is achieved by equipping each node with limited nonlocal topology information, which it uses to determine the time when it can go to sleep since code is not being distributed in its vicinity. The protocol to handle multiple sources provides a loose coupling of nodes to a source and disseminates code in waves, each originating at a source with a mechanism to handle collisions when the

waves meet. Freshet functions in three phases for each new code image: Blitzkrieg, Distribution, and Quiescent. It aggressively conserves energy by putting nodes to sleep between the blitzkrieg and the distribution phases as well as the quiescent phase. Freshet introduces a scheme to disseminate code from multiple originators, use location information, and reduce control message overhead. Freshet uses spatial multiplexing to transfer the code. This implies that a node can transfer the code to a neighbor before it has received all the pages for a given version. In effect, the node can initiate transfer once it has the first page for the version. This makes the delay proportional to the sum of the network diameter and the code size rather than the product of the two.

Rossi et al. [105] presented a reprogramming system for WSNs called SYNAPSE, which was designed to improve the efficiency of the error recovery phase. SYNAPSE features a hybrid ARQ (HARQ) solution where data is encoded prior to transmission and incremental redundancy is used to recover from losses, thus considerably reducing the transmission overhead. For the coding, digital Fountain Codes were selected. In fact, a Fountain Code (FC) [106], specifically designed to meet the needs of sensor network reprogramming, is used at the heart of the data dissemination/recovery process. This code is designed to maintain high efficiency, in terms of overhead, in the face of small packet sizes and typical program lengths. These codes were selected due to their desirable properties: FCs are rateless and have a low computational complexity, as encoding and decoding are performed efficiently through XOR operations.

State of the art protocols, such as Deluge [88], implement error recovery through the adaptation of standard Automatic Repeat reQuest (ARQ) techniques. These, however, do not scale well in the presence of channel errors and multiple receivers. They use three-way handshakes as per the ADV-REQ-CODE paradigm. They implement randomization when sending advertisements, exploit broadcast transmissions for the code, NACKs to request missing data and implement the method proposed in Stream [103].

Heo et al. [107] proposed a novel program updating mechanism considering resource constraints of sensor nodes. The proposed mechanism was designed for sensor nodes with the NOR flash memory. This is generally used to store program images. It was designed to minimize the number of flash write/erase operations, which consume a great deal of energy, and to provide wear-leveling for the NOR flash memory, by setting a function as the basic unit of program updating, and partition a function into fixed-sized blocks that can be separately relocated in memory. A paragraph was defined as a basic unit of writing. The mechanism provides a method similar to the paging technique for sensor nodes without MMU (Memory Management Unit), which is necessary for implementing the paging technique in general operating systems. Jeong et al. [84] proposed the Incremental approach in which the host program generates the difference of the two program images but did not consider wear-leveling for the NOR flash memory. Consequently, there was the memory wastage because of the number of write/erase operations.

Kim et al. [108] proposed a new Commissioning and Deployment Method for WSNs, by introducing a partial download procedure of node program for efficient node commissioning. The basic idea is the same as that for Efficient Partial Node

Update for Wireless Sensor Networks Using a Simulated Virtual Node [109], where the concept of a virtual node is used for finding near optimal smallest partial updating way. When a newer sensor node is out-of-the-box and turns on the power switch, the sensor node sends the initial node information including attached sensor types, processor type, program memory size, and RF capability, which is called pre-commissioning. The commissioning server also receives the operating network information from a profile server, which manages sensor networks.

ZigBee alliance provides some procedure for initial procedure of commissioning for small wireless nodes [99]. The commissioning server decides what the best role of this sensor node will be. ZigBee alliance suggests three role types for nodes, coordinators, routers, and end devices. The four major steps are: step 1- commissioning server receives the specification of a newer sensor node, and analyzes the best role for this node, considering the management information of the operation network, step 2- commissioning server optimizes a source code for the assigned role of this sensor node, compiles the source code; then the newly compiled executable code is compared with the pre-programmed code in the node, and the difference is generated as a partial node updating rule, Step 3- commissioning server sends the partial updating rule to the node and the received node reflects the rule into the program memory, Step 4- ZigBee commissioning procedure is optionally performed for setup of the network parameters after resetting the node. Experimental results show that the partial update method can dramatically reduce the communication overhead for the node update by up to 72.5%.

Maia et al. [110] used small world features to improve over-the-air programming. The small world-based protocol takes into account the communication workflow of sensor networks to create shortcuts toward the sink, thus improving the reprogramming process. The endpoints of these shortcuts are nodes with more powerful hardware, resulting in a heterogeneous wireless sensor network. The goal was to improve and assess the OAP Deluge protocol [88] applied to a network with small world features. Therefore, this work presents a new in-network algorithm called OAP-SW that combines the shortcut creation with OAP to improve network reprogramming. It enabled pipelining by dividing a program into fixed-sized segments, which in turn are divided into packets. Such an approach results in a desired characteristic for network reprogramming, known as spatial multiplexing, which enables different parts of the network to do the reconfiguration process at the same time. The OAP-SW protocol transmits fewer messages when compared with the Deluge [88]. This occurs because the small world infrastructure, provided by OAPSW, reduces the minimal average path length of the network.

Shaikh et al. [111] proposed a protocol that divides the code image into application and reprogramming support. It pre-installs the reprogramming protocol as one image and the application program is equipped with the ability to listen to new code updates as the second image. Three substantially more sophisticated protocols are: Deluge [88], MNP [90], and Freshet [104]; they focus on transferring the image of the entire reprogramming protocol together with the minimally necessary part. As a point of optimization, the stream AS-RS approach keeps the basic mode of transfer the same as in Deluge; that is it transfers just what is needed, which is the application code (or the code of the updates to the application). It transfers close to the minimally

required image size by segmenting the total program image into an application image and the reprogramming image. Application image refers to the user application and reprogramming image refers to the protocol component for protocol, such as MNP [90], Deluge [88] or Freshet [104]. An application is modified by linking it to a small component called Application Support (AS) while Reprogramming Support (RS) is pre-installed in each node. Overall, the design principle is to limit the size of the AS and to provide it the facility to switch to RS when triggered by a code update related message.

In Freshet [104], to save energy the sleeping time of the node is estimated beforehand and this estimation if often found inaccurate due to the variability of the wireless channel; however, the stream AS-RS Approach protocol achieves this goal by rebooting the node from Stream-RS only when a new node arrives at one of its neighbors. Thus the user application running on the node can put the node to sleep until the time to reboots. This opportunistic sleeping feature conserves energy in resource-constrained sensor networks.

In Deluge [88], once a node's reprogramming is over, it keeps on advertising the code image it has; hence radio resources are continuously used in the steady state. In the stream AS-RS Approach, Stream-AS does not advertise the data it has. The benefit of this protocol is that a lower number of bytes are transferred over the wireless medium leading to increased energy savings and reduced delays for reprogramming

The reprogramming system can be classified according to several, criteria, such as single-hop and multihop. Single hop disseminates codes only within the radio communication range of a base station. Examples are XNP [80], Reijers approach [82] and Incremental approach [84]. Multihop code dissemination protocols are epidemic in nature. Almost all recent protocols are developed to support multihop communication. During encoding, most reprogramming systems disseminate the compiled program image across the network. The overhead is usually large in cases when only minor changes occur between the new and old versions. Some use the delta patches like Incremental [84]; however, Mate virtual machine can be used instead of native nesC compiled codes since Mate scripts are much smaller and simpler to write. An example is Trickle [86]. Since carrier sense multiple access medium access control (CSMA MAC) is in the TinyOS release, most reprogramming systems use CSMA. TDMA demands careful scheduling of time slots, and its implementation on a sensor platform is much more complex than CSMA. Sprinkler [92] and Infuse [91] use TDMA.

The approach of sending the codes in which super nodes/head nodes reprogram other nodes in their local areas is thought to be maintaining hierarchy. Firecracker [95] and Sprinkler [92] maintain such hierarchy. Pipelining is done through segmentation; a program is divided into several segments (called pages) each of which contains a fixed number of packets. Instead of completely receiving a whole program before forwarding it, a node becomes a source node after it receives only one complete segment. Most reprogramming systems only disseminate one program to the whole network. Therefore, the scope selection function allows administrators or the network to dynamically select any particular nodes to be reprogrammed. Approaches that employ such selection are Aqueduct [96] and TinyCubus [97].

The various protocols studied above are summarized in Table I. [79].

Table 1. Summary of Various Reprogramming Protocols

Name	Encoding/decoding	MAC	Hop	Scope	Hierarchy	Pipelining
XNP	Complete program	CSMA	Single hop	Whole network	no	no
Reijers	Platform-dependent patch	CSMA	Single hop	Whole network	no	no
Incremental	Platform-independent patch	CSMA	Single hop	Whole network	no	no
Trickle	Mate script	CSMA	Multihop	Whole network	no	no
MOAP	Complete program	CSMA	Multihop	Whole network	no	no
Deluge	Complete program	CSMA	Multihop	Whole network	no	yes
MNP	Complete program	CSMA	Multihop	Whole network	no	yes
Sprinkler	Complete program	TDMA	Multihop	Whole network	yes	no
Firecracker	Complete program	CSMA	Multihop	Whole network	yes	no
Infuse	Complete program	TDMA	Multihop	Whole network	no	yes
Aqueduct	Complete program	CSMA	Multihop	Selected nodes	no	yes
TinyCubus	Modular update	CSMA	Multihop	Selected nodes	no	no
Transaction-based	TCL script	CSMA	Multihop	Whole network	no	no
Rateless	Complete program	CSMA	Multihop	Whole network	no	yes
Stream	Complete program	CSMA	Multihop	Whole network	no	yes
Freshet	Complete program	CSMA	Multihop	Whole network	no	yes
Synapse	Complete program	CSMA	Multihop	Whole network	no	yes
Flash Memory	Modular update	CSMA	Single hop	Whole network	no	no
Commissioning	Modular update	CSMA	Multihop	Whole network	no	no
Small World Concept	Complete program	CSMA	Multihop	Whole network	no	yes
Stream AS-RS*	Complete program	CSMA	Multihop	Whole network	no	yes

**(AS) Application Support (RS) Reprogramming Support*

5 Conclusion

As sensor networks move from research to deployment, from laboratory to the real world, from small scale to large scale, issues of maintenance will be challenging. It is not easy to predict all the problems that may arise while installing the sensor nodes before deploying a sensor network. Reprogramming is necessary to fix bugs, update codes, and manage application requirement changes. When nodes are densely deployed in a hostile area, such as a tunnel or bridge structure, it is highly cumbersome to physically reach all nodes and provide necessary maintenance. Software maintenance is a major phase in the software development cycle that plays a significant role in reliable performance. Thus, with respect to software maintenance issues, robust, efficient and tested codes should be installed in the sensor network before deployment. Corrective, adaptive, and perfective maintenance should be provided by relevant software updating at required times. Ideally, maintenance operations should not degrade the reliability and the structure of the subject system, and neither should they degrade its maintainability.

Reprogramming is important in facilitating the management and maintenance of WSNs, as well as enabling adaptive sensor applications. It becomes a crucial service to the success of currently employed WSNs. There are many new hardware platforms, new operating systems (TinyOS, SOS), and new applications (well-controlled bridge monitoring, randomly deployed nodes in tunnel monitoring) that keep on emerging in the process of the WSN revolution. However, many problems need further investigation to make reprogramming highly usable and efficient. Aggressive research is being done on code dissemination. However, design trade-offs and impact factors have not been fully understood. Design and implementation of energy-efficient routing and one-to-many communication protocols for WSN are a continuing focus of research.

Over-the-air (OTA) programming eliminates the need of detaching sensor nodes and attaching data transfer cables when updating the sensor software. Many protocols have been designed for efficient software maintenance in the deployed area. Various protocols have been discussed; however, unlike other protocols, WSN has its own design and resource constraints. Aggressive research is going on to make reprogramming highly usable and efficient. Such light weight and scalable protocols are required that can embed intelligence in sensor node system software, which can sustain the capability of the system to provide a reliable service with least over head under all real time constraints.

References

1. Branko, G., Daniele, P., Daniele, I.: Integrity monitoring of an old steel bridge using fiber optic distributed sensors based on Brillouin scattering, Proceedings of SPIE on Nondestructive characterization for composite

materials, aerospace engineering, civil infrastructure and homeland security, San Diego, California, USA, 1-8 (2007)
2. Glisic, B., Inaudi, D.: Fiber optic methods for structural health monitoring, John Wiley & Sons, New York (2008)
3. FIGG Bridge Engineers, Judith Dupre (eds.) Bridging the Mississippi; The new I-35W bridge (1st edition), Publisher: FIGG Bridge Engineers, UK (2008)
4. Hipley, P.: Caltrans current state-of-practice, Proceedings of Instrumental Systems for Diagnostics of Seismic Response of Bridges and Dams, Richmond, CA, 3-7 (2001)
5. Ni, Y. Q., Wang, B.S., Ko, J.M.: Simulation studies of damage location in Tsing Ma Bridge deck, Proceedings of Nondestructive Evaluation of Highways, Utilities, and Pipelines, San Diego, CA, 312- 323 (2001)
6. Jan, B., Kay, R., Matthias, W., Matthias, R.: Deployment Techniques for Sensor Networks, Sensor Networks, Springer, 219-248 (2009)
7. Whelan, M. J., Gangone, M.V., Janoyan, K. D.: Highway Bridge Assessment Using an Adaptive Real-Time Wireless Sensor Network, IEEE Sensors Journal, 9:1405–1413 (2009)
8. Fjeldstad, R., Hamlen, W.: Application program maintenance study: Report to our correspondents, Tutorial on Software Maintenance, IEEE Computer Press Society (1983)
9. Lientz, B., Swanson, E.: Problems in application software maintenance. Communications of the ACM, 24(11): 763-769 (1981)
10. Denys, G., Piessens, F., Matthijs, F.: A survey of customizability in operating systems research, ACM Computing Surveys, 34(4) : 450-468 (2002)
11. IEEE Std. 610.12-1990, Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, Los Alamitos, CA (1990)
12. Lientz, B.P., Swanson, B.E.: Software Maintenance Management, Addison-Wesley: Reading, MA, (1980)
13. Pigoski, T.M.: Practical Software Maintenance – Best Practices for Managing Your Software Investment ,John Wiley & Sons, New York, NY (1997)
14. IEEE Std. 1219-1998, Standard for Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA (1998)
15. Langendoen, K., Baggio, Visser.: Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture, Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS), Piscataway, NJ, 8–15 (2006)
16. Dunkels, A., Finne, N., Eriksson, J., Voigt, T.: Run-time dynamic linking for reprogramming wireless sensor networks, Proceedings of the 4th International conference on Embedded networked sensor systems, Colorado, USA, 15-28 (2006)
17. Kogekar, S., Neema, S., Koutsoukos, X.: Dynamic software reconfiguration in sensor networks, Proceedings of the 2005 Systems Communications (ICW '05), Montreal, Canada, 413 – 420 (2005)

18. Koshy, J., Pandey, R.: Remote incremental linking for energy-efficient reprogramming of sensor networks, Proceedings of Second European Workshop on Sensor Networks, Istanbul, Turkey, 354-365 (2005)
19. Yi, S., Min, H., Cho, Y., Hong, J.: Molecule: An adaptive dynamic reconfiguration scheme for sensor operating systems, Computer Communications, 31(4): 699-707 (2008)
20. Dong, W., Chen, C., Liu, X., Bu, J.: Providing OS Support for Wireless Sensor Networks: Challenges and Approaches, Communications Surveys and Tutorials, IEEE 12(4) 519 – 530 (2010)
21. Tiny OS. [Online] Available: http://www.tinyos.net
22. Dunkels, A., Gronvall, B., Voigt, T.: Contiki—a lightweight and flexible operating system for tiny networked sensors, Proceedings of 29th conference on local computer networks, Tampa, Florida, 455 – 462 (2004)
23. Han, C.C., Kumar, R., Shea, R., Kohler, E., Srivastava, M.: A dynamic operating system for sensor nodes, Proceedings of the 3rd international conference on Mobile systems, applications, and services (MobiSys '05), Seattle, WA, 1-14 (2005)
24. Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A.: MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms, Mobile Networks and Applications, 10(4): 563–579 (2005)
25. Eswaran, A., Rowe, A., Rajkumar, R.: Nano-RK: An energy-aware resource-centric RTOS for sensor networks, Proceedings of 26th IEEE International Real-Time Systems Symposium, Miami, Florida, 255-265 (2005)
26. Cha, H., Choi, S., Jung, I., Kim, H., Shin, H.: RETOS: Resilient, expandable, and threaded operating system for wireless sensor networks. Proceeding of sixth IEEE International Symposium on Information processing in sensor network, Massachusetts, USA, 148 – 157 (2007)
27. Cao, Q., Adbelzaher, T.F., Stankovic, J.A.: The LiteOS operating system: Towards Unix-like abstractions for wireless sensor networks, Proceedings of IEEE International Conference on Information Processing in Sensor Networks (IPSN), St. Louis, MO, USA, 233 – 244 (2008)
28. Gu, L., Stankovic, J. A.: t-kernel: Providing reliable OS support to wireless sensor networks, Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys), Boulder, Colorado, USA, 1-14 (2006)
29. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks, Proceedings of the sixth annual international conference on Mobile computing and networking, Boston, MA, USA, 56–67 (2000)
30. Pottie, G.J., Kaiser, W.J.: Wireless Integrated Network Sensors, ACM Communication 43(5): 551-562 (2000)
31. Kahn, J.M., Katz, R.H., Pister, S.J.: Next Century Challenges: Mobile Networking for Smart Dust, Proceedings of Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '99), Seattle, Washington, USA, 271–278 (1999)

32. Shih, E.: Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks, Proceedings of the seventh annual international conference on Mobile computing and networking (MOBICOM 2001), Rome, Italy, 272–286 (2001)
33. Akyildiz, I.F., Su, W., Yogesh, S., Erdal, C.: A Survey on Sensor Networks, IEEE Communications Magazine 40(8):102–114 (2002)
34. Cheng, Z., Mark, P., Wendi, B.H.: General network lifetime and cost models for evaluating sensor network deployment strategies, IEEE transactions on mobile computing 7(4): 484 – 497 (2008)
35. Zhang, J., Ting, Y., Sang, H.S.: Deployment Strategies for Differentiated Detection in Wireless Sensor Networks, Proceedings of Third Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2006), Reston, VA, USA, 316 – 325 (2006)
36. Zou, Y., Chakrabarty, K.: Uncertainty-Aware and Coverage-Oriented Deployment for Sensor Networks, Journal of Parallel and Distributed Computing 64(7): 788–798 (2004)
37. Dhillon, S., Chakrabarty, K.: Sensor Placement for Grid Coverage under Imprecise Detections, Proceedings of Fifth International IEEE Conference on Information Fusion (FUSION 2002), Maryland, USA, 2:1581 – 1587 (2002)
38. Meguerdichian, S., Koushanfar, F., Potkonjak, M., Srivastava, M.: Coverage Problems in Wireless Ad-hoc Sensor Networks, Proceedings of IEEE Conference on Computer Communications (Infocom 2001), Anchorage, Alaska USA, 3:1380 –1387 (2001)
39. Zou, Y., Chakrabarty, K.: Sensor Deployment and Target Localization in Distributed Sensor Networks, ACM Transactions on Embedded Computing Systems 3(1): 61–91 (2004)
40. David, L.M., Eric, B.F., Michael, D.T., Timothy, G., Overly, K.: Experimental studies of using wireless energy transmission for powering embedded sensor nodes, Journal of Sound and Vibration 329(12): 2421–2433 (2010)
41. Farrar, C.R., Park, G., Allen, D.W., Todd, M.D.: Sensor network paradigms for structural health monitoring, Structural Control and Health Monitoring 13: 210–255 (2006)
42. Todd, M.D., Mascarenas, D.L., Flynn, E.B., Rosing, T., Lee, B., Musiani, D., Farrar, C.R.: A different approach to sensor networking for SHM: remote powering and interrogation with unmanned aerial vehicles, Proceedings of Sixth International Workshop on Structural Health Monitoring, Stanford, CA, 1-16 (2007)
43. Brown, W.C.: The history of wireless power transmission, Solar Energy, 56(1):3–21 (1996)
44. Maryniak, G.E.: Status of international experimentation in wireless power transmission, Solar Energy, 56(1):87–91 (1996)

45. Choi, S., Song, K., Golembiewskii, W., Chu, S.H., King, G.: Microwave powers for smart material actuators. Smart Materials and Structures 13(1): 38–48 (2004)
46. Edward, S., Haodong, L., Darrell, C., Pragasen, P.: Self-powered sensors for monitoring of highway bridges, IEEE Sensors, 9(11):1422-1429 (2009)
47. Munir, S.A., Yu, W.B., Ma, J.: Efficient Minimum Cost Area Localization for Wireless Sensor Network with a Mobile Sink State Key Laboratory of Networking and Switching, Proceedings of 21st International Conference on Advanced Networking and Applications (AINA 2007), Ontario, Canada, 533 – 538 (2007)
48. Rappaport, T.S., Reed, J.H., Woerner, B.D.: Position Location using Wireless Communications on Highways of the Future, IEEE Communications Magazine, 34(10): 33-43 (1996)
49. McGuire, M., Plataniotis, K.N., Venetsanopoulos, A.N.: Location of Mobile Terminals using Time Measurements and Survey Points, IEEE Transactions on Vehicular Technology, 52(4): 999-1011 (2003)
50. Langendoen, K., Reijers, N.: Distributed Localization in Wireless Sensor Networks: A Quantitative Comparison, Computer Networks: The International Journal of Computer and Telecommunications Networking, 43(4):499-518 (2003)
51. Koushanfar, F., Potkonjak, M., Sangiovanni-Vincentelli, A.: Fault tolerance in wireless ad hoc sensor networks, Proceedings of First IEEE International Conference on Sensors (SENSORS 2002), Florida, USA, 2:1491–1496 (2002)
52. Ko, J.M., Ni, Y.Q.: Technology developments in structural health monitoring of Large-scale bridges, Engineering Structures, 27(12): 1715–1725 (2005)
53. Hoblos, G., Staroswiecki, M., Aitouche, A.: Optimal Design of Fault Tolerant Sensor Networks, Proceedings of the IEEE international conference on control applications, Anchorage, Alaska, 467 – 472 (2000)
54. Shen, C., Srisathapornphat, C., Jaikaeo.: Sensor Information Networking Architecture and Applications, IEEE Personal Communications, 8(4): 52 – 59 (2001)
55. Venkataraman, G., Emmanuel, S., Thambipillai, S.: A Cluster-Based Approach to Fault Detection and Recovery in Wireless Sensor Networks, Proceedings of 4th IEEE International Symposium on Wireless Communication Systems (ISWCS'07), Trondheim, Norway, 35 – 39 (2007)
56. Arora, A., Dutta, P., Bapat, S., Kulathumani, V., Zhang, H.: A line in the sand: a wireless sensor network for target detection, classification, and tracking. International Journal of Computer Networks, 46(5):605–634 (2004)
57. Sekhar, A., Manoj, B.S., Siva, C., Ram, M.: Dynamic Coverage Maintenance Algorithms for Sensor Networks with Limited Mobility, Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PERCOM'2005), Kauai Island, Hawaii, 51–60 (2005)

58. Gage, D.W.: Command Control for Many-Robot Systems, Proceedings of the Nineteenth Annual AUVS Technical Symposium (AUVS-92), Huntsville AL, 10(4): 28-34 (1992)
59. Howard, A., Mataric, M.J., Sukhatme, G.S.: Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem, Proceedings of the 7th International Symposium on Distributed Autonomous Robotics Systems (DARS02), Fukuoka, Japan, 299-308 (2002)
60. Heinzelman, W.R, Chandrakasan, A., Balakrishnan, H.: Energy efficient communication protocol for wireless micro sensor networks, Proceedings of Hawaii International Conference on System Sciences, Hawaii, USA, 1:3005–3014 (2000)
61. Soro, S., Heinzelman, W.B.: Prolonging the lifetime of wireless sensor networks via unequal clustering, Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005), Denver, Colorado, 13:236–243 (2005)
62. Xiang, M., ShiWei-rena, J., Chang-jianga, Z.Y.: Energy efficient clustering algorithm form maximizing lifetime of wireless sensor networks, International Journal of Electronics and Communications, 64(4): 289-298 (2010)
63. Selvakennedy, S.: An Adaptive Data Dissemination Strategy for Wireless Sensor Networks, International Journal of Distributed Sensor Networks, 3(1): 23–40 (2007)
64. Fengjun, S.: A Single-Hop Active Clustering Algorithm for Wireless Sensor Networks. Advances in Soft Computing, Springer Berlin, 116:397-406 (2009)
65. Yean, F.W., Yeong-Sung, L.; Energy-Efficient Data Aggregation Routing and Duty-Cycle Scheduling in Cluster-based Sensor Networks, Proceedings of Fourth Annual IEEE Consumer Communications and Networking Conference (CCNC 2007), Las Vegas, Nevada, 95 – 99 (2007)
66. Tal, A., Danny, B., Danny, D., Bracha, H.: Efficient Clustering for Improving Network Performance in Wireless Sensor Networks, EWSN'08, Lecture Notes in Computer Science, vol. 4913, Springer: Verlag, 221-236 (2008)
67. Younis, O., Fahmy, S.: A scalable framework for distributed time synchronization in multi-hop sensor networks, Proceedings of Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, IEEE SECON 2005, Santa Clara, California USA, 13–23 (2005)
68. Eric, M.Y.: Energy Scavenging for Wireless Sensor Nodes, Proceedings of 2nd IEEE International Workshop on Advances in Sensors and Interfaces (IWASI 2007), Bari, Italy, 1–4 (2007)
69. Cian, O.M., Terence, O., Rafael, V.M., James, R., Brendan, O.: Energy scavenging for long-term deployable wireless sensor networks, Talanta, 75(3):613-623 (2008)

70. Pi, X., Yu, H.: Redeployment Problem for Wireless Sensor Networks, Proceedings of International Conference on Communication Technology, ICCT '06, China, 1 – 4 (2006)
71. Wang, C., Ding, J.: The optimum sensor redeployment scheme using the most frangible clusters set, Computer Communications, 31(14): 3492-3502 (2008)
72. Mitali, S., Viktor, K. P.: Energy-efficient and Fault-tolerant Resolution of Topographic Queries in Networked Sensor Systems, Proceedings of the 12th International Conference on Parallel and Distributed Systems, ICPADS-06, Minneapolis, 10 -15 (2006)
73. Ganesan, D., Govindan, R., Shenker, S., Estrin, D.: Highly resilient, energy-efficient multipath routing in wireless sensor networks, Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing, Long Beach, CA, USA, 11-25 (2001)
74. Huang, J.H., Amjad, S., Mishra, S.: CenWits- A sensor-based loosely coupled search and rescue system using witnesses, Proceedings of the Third International Conference on Embedded Networked Sensor Systems (Sensys'05), San Diego, CA, 180-191 (2005)
75. Matthew, J.W., Michael, V.G., Kerop, D.J., Ratneshwar, J.: Real-time wireless vibration monitoring for operational modal analysis of an Integral abutment highway bridge, Engineering Structures, 31(10): 2224-2235 (2009)
76. Frank, S., Neil, H., Ian, W., Peter, B., Campbell, M., Kenichi, S.: Smart bridges, smart tunnels - Transforming wireless sensor networks from research prototypes into robust engineering infrastructure, Ad Hoc Networks, 8(8): 872- 888 (2010)
77. He, Y., Raghavendra, C., Berson, S., Braden, B.: A programmable routing framework for autonomic sensor networks, Proceedings of the Fifth International Workshop on Active Middleware Services, WA, USA, 60–68 (2003)
78. Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J., Zukowski, D.: Challenges: An application model for pervasive computing, Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking, Boston, Massachusetts, USA, 266.274 (2000)
79. Qiang, W., Yaoyao, Z., Liang, C.: Reprogramming Wireless Sensor Networks: Challenges and Approaches, IEEE Network, 20(3):48–55 (2006)
80. Jeong, J., Kim, S.: A Broad Network reprogramming, US Berkeley Online Library (2003)
81. Mote in-network programming user reference, version number 20030315, Crossbow Technology, Berkeley Online Library (2003)
82. Niels, R., Langendoen, K.: Efficient Code Distribution in WSN, Proceedings of the 2nd ACM International Workshop on Wireless Sensor Networks and Applications, San Diego, CA, USA, 60 – 67 (2003)
83. Thanos, S., John, H., Deborah, E.: A Remote Code Up-date Mechanism for Wireless Sensor Networks, CENS Technical Report number 30, (2003)
84. Jeong, J., Culler, D.: Incremental Network Programming for Wireless Sensors, Proceedings of 1st Annual IEEE Communications Society

Conference on Sensor and Ad Hoc Communications and Networks, Santa Clara, California, USA, 25–33 (2004)
85. Andrew, T.: Efficient Algorithms for Sorting and Synchronization, PhD thesis, Australian National University, 47-58 (1999)
86. Philip, L.: Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks, Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation, California, USA, 1:15-28 (2004)
87. Heinzelman, W.R., Kulik, J., Balakrishnan, H.: Adaptive protocols for information dissemination in wireless sensor networks, Proceedings of the fifth annual ACM/IEEE International Conference on Mobile computing and networking, Washington, USA, 174–185 (1999)
88. Adam, C., Jonathan, H., Gilman, T.: Deluge Data Dissemination for Network Reprogramming at Scale, Technical report, University of California at Berkeley, (2003)
89. Kulik, J., Heinzelman, W.R., Balakrishnan, H.: Negotiation-based protocols for disseminating information in wireless sensor networks, Wireless Networks 8(2):169-185 (2002)
90. Kulkarni, S.S., Wang, L.: MNP: Multihop Network Reprogramming Service for Sensor Networks, Proceedings of 25th IEEE International Conference on Distributed Computing Systems (IEEE ICDCS 2005), Columbus, Ohio, USA, 7–16 (2005)
91. Arumugam, M.: Infuse - A TDMA Based Reprogramming Service for Sensor networks, Proceedings of Second International conference on Embedded Networked Sensor Systems (SenSys '04), Baltimore, USA, 281-282 (2004)
92. Naik, V., Arora, A., Sinha, P., Hongwei, Z.: Sprinkler: A Reliable and Energy Efficient Data Dissemination Service for Wireless Embedded Devices, IEEE Transactions on Mobile Computing, 6(7): 777 – 789 (2007)
93. Clark, B., Colbourn, C., Johnson, D.: Unit Disk Graphs, Discrete Mathematics, 86: 165-177 (1990)
94. Krumke, S., Marathe, M., Ravi, S.: Models and Approximation Algorithms for Channel Assignment in Radio Networks, Wireless Networks, 7(6):575-584 (2001)
95. Philip, L., David, C.: The firecracker protocol, Proceedings of the 11th workshop on ACM SIGOPS European workshop (EW 11), Leuven, Belgium, 1-5 (2004)
96. Phillips, L.A.: Aqueduct: Robust and efficient code propagation in heterogeneous wireless sensor networks, Master's thesis, University of Colorado, Boulder (2005)
97. Marron, P.J., Lachenmann, A., Minder, D.: Management and Configuration Issues for Sensor Networks, International Journal of Network Management, 15(4):235–253 (2005)
98. Lee, Y.F., Shen, C.C.: A transaction-based approach to over-the-air programming in wireless sensor networks, Proceedings of International

Symposium on Communications and Information Technologies (ISCIT 2007), Australia, 1377–1382 (2007)
99. ZigBee Alliance (http://www.ZigBee.org)
100. Ousterhout, J.K.: TCL and Tk tool kit, Addison-Wesley: Reading MA, (1994)
101. Welch, B.B., Jones, K., Hobbs, J.: Practical programming in TCL and TK, 4th Edition, Prentice-Hall: Upper saddle River, NJ, (2003)
102. Andrew, H., Starobinski, .D, Trachtenberg, A.: Rateless Deluge - Over-the-Air Programming of Wireless Sensor Networks Using Random Linear Codes, Proceedings of International Conference on Information Processing in Sensor Networks (IPSN 2008),ST. Louis, Missouri, 457–466 (2008)
103. Panta, R.K, Khalil, I., Bagchi, S.: Stream: Low Overhead Wireless Reprogramming for Sensor Networks, Proceedings of 26th IEEE International Conference on Computer Communications (INFOCOM 2007), Alaska, USA, 928–936 (2007)
104. Krasniewski, M.D., Panta, R.K., Bagchi, S., Yang, C.L., Chappell, W.J.: Energy-efficient On-demand Reprogramming of Large-scale Sensor Networks, ACM Transactions on Sensor Networks, 4(1):1-38 (2008)
105. Michele, R., Zanca, G., Stabellini, L., Crepaldi, R., Harris, A.F., Zorzi, M.: SYNAPSE: A Network Reprogramming Protocol for Wireless Sensor Networks Using Fountain Codes Sensor, Proceedings of 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, San Francisco, California USA, 188–196 (2008)
106. MacKay, D.J.C.: Fountain Codes, IEE Proceedings Communications, 152(6): 1062–1068 (2005)
107. Junyoung, H., Boncheol, G., Sang, I.E., Pankoo, K., Gwangil, J.: Energy Efficient Program Updating for Sensor Nodes with flash memory, Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 194-200 (2010)
108. Bong, W.K., Seong-Soon, J.: A New Commissioning and Deployment Method for Wireless Sensor Networks, Proceedings of Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2009), Sliema, Malta, 232 – 237 (2009)
109. Kim, B.W., Ryu, J.H., Joo, S.: A New Efficient Partial Node Update for Wireless Sensor Networks Using a Simulated Virtual Node, Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Sydney, Australia, 179–182 (2008)
110. Guilherme, M., Daniel, L.G., Andre, L.L., Aquino, A., Loureiro, A F.: Improving an over-the-air programming protocol for wireless sensor networks based on small world concepts, Proceedings of the 12th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems, Tenerife, Canary Islands, Spain, 261-267 (2009)
111. Shaikh, R.P., Thakare, V.M., Dharaskar, R.V.: Efficient Code Dissemination Reprogramming Protocol for WSN, International Journal of Computer and Network Security (IJCNS), 2(2): 116-122 (2010)