An Introduction to Over-the-Air Programming in Wireless Sensor Networks

A. S.A.Quadri, B. Othman Sidek

Collaborative Microelectronic Design Excellence Centre (CEDEC), USM, Malaysia
reachquadri@yahoo.com

Abstract
Over-the-air programming plays a vital role in wireless sensor networks (WSN). It is a standard for the transmission and reception of application-related information in a wireless communication system. In most of WSN applications, nodes are deployed once and are supposed to operate unattended for a long period. 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 WSN. 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 (OAP) is a fundamental service that serves this purpose. The paper provides an overview of over-the-air programming. It discusses state of art OAP protocols used in WSN applications.

Keywords: Wireless sensor network, Over-the-air programming, Reprogramming, Software, Protocol

I. Introduction
Over-the-air programming systems for communicating information to a remote or mobile device are known in the art. Often times the mobile device has a local computer that receives wireless over-the-air communication from a home device. The local computer specifically contains a microcontroller and memory systems to provide various functions for a user. Specifically, the local computer has a plurality of applications, systems or programs that provide different operating modes and functions for a user. In the art when an application, system or program needs to be updated, the home device sends a signal to the remote device providing the local computer with a plurality of computer code packets that comprise a computer code to update a system, program or application in the local computer. The protocol of dynamic program update is called as over-the-air programming. Presently, in the art when the computer code is received the local computer reboots, and during the rebooting process, checks the integrity of the code in order to determine whether the update is proper.
Wireless Sensor Networks (WSN) provides an effective means to monitor physical environments. The computing nodes in a WSN are resource-constrained devices whose resources need to be used sparingly. The main requirement of a WSN is to operate unattended in remote locations for extended periods. Physical conditions, environmental conditions, upgrades, user preferences, and errors within the code can all contribute to the need to modify currently running applications. Therefore, reprogramming of sensor nodes is required. A sensor network is
typically made up of hundreds or thousands of nodes, distributed over a vast area that often is not easily assessable (e.g., on or around a volcano or bridge). Manually reprogramming every node in the network by individually plugging each one to a laptop or a PDA is not a feasible option. In addition, harsh environmental conditions often make the retrieval of the nodes nearly impossible. The protocols that have been developed are all platform specific due to the differences in their architectures. In order to make one of these protocols work on a different hardware platform, it often requires numerous software modifications. Hence, reprogramming over the air is imperative for many applications.

The paper briefly discusses Over-the-air (OTA) programming in general and emphasis its significance in WSN applications and provides a comparison of various OAP protocols developed for reprogramming.

II. Terminology

Over-the-air programming (OTA) may refer to either free-to-air, digital television, or in the mobile content world, over-the-air service provisioning (OTASP), over-the-air provisioning (OTAP) or over-the-air parameter administration (OTAPA), methods of distributing new software updates to cell phones or provisioning handsets with the necessary settings with which to access services such as wireless application protocol (WAP) or multimedia messaging service (MMS). Some phones with this capability are labeled as being "OTA capable" (http://en.wikipedia.org).

The term OTA is also used in conjunction with the short messaging service (SMS), which allows the transfer of small text files using a mobile phone. In addition to short messages and small graphics, such files, which contain instructions for subscription activation, banking transactions, ringtones, and WAP settings can also be transferred. OTA messages can be encrypted to ensure user privacy and data security.

OTA is similar to firmware distribution methods used by other mass-produced consumer electronics, such as cable modems, which use trivial file transfer protocol (TFTP) (http://www.ietf.org) as a way to remotely receive new programs, thus reducing the amount of time spent by both the owner and the user of the device on maintenance. TFTP is a file transfer protocol known for its simplicity. It is generally used for automated transfer of configuration or boot files between machines in a local environment. Compared to file transfer protocol (FTP), TFTP is extremely limited, providing no authentication, and is rarely used interactively by a user.

When OTA is used to update a phone's operating firmware, it is sometimes called firmware over-the-air (FOTA). For service settings, the technology is often known as device configuration (DC). Various standardization bodies were established to help, develop, oversee, and manage OTA programming. One of them is the open mobile alliance (OMA) (http://www.openmobilealliance.org/Technical/htm).

Over-the-air programming can be viewed as a method for programming computer code at a remote platform having local computer, the local computer including a flash memory. The method includes receiving a plurality of computer code packets wherein the computer code packets are provided by wireless transmission. The next step involves storing the plurality of code packets in an alternate area of flash memory of the local computer wherein the plurality of computer code packets comprise a computer code. The local computer then recognizes the reception of the complete copy of the computer code and checks the integrity of the received
computer code before the rebooting process. In response to checking the integrity of the computer code, a boot address of the local computer is changed such that at the convenience of a user the local computer may be rebooted after the boot address is changed. Once the local computer is rebooted, the computer restarts with the alternate computer code wherein there is not a need to check the integrity of the computer code during the rebooting process.

In general, the term OTA implies the use of wireless mechanisms to send provisioning data or update packages for firmware or software updates to a mobile device, so that the user does not have to go to a store or a service center to have applications provisioned, parameters changed or firmware or software updated.

III. OTA for Mobile Phone Communication

In general, terminal devices for a wireless communication network are manufactured and distributed with predetermined communication capabilities and a minimal operation capability. Each unit includes a substantial memory for plug-in feature programming, which is initially empty. A terminal user selects desired features or set of features and contacts a service provider. The provider's equipment downloads software programming corresponding to the desired feature into the memory of the terminal device. The software takes the form of a plug-in module, written to the program interface specification of the core software of the terminal device. When loaded into memory and interfaced through the core software, the module allows the terminal device to implement the desired feature. Features are developed in packages. Features can be added, upgraded or replaced at any time by downloading new feature modules into the memory.

As mobile phones accumulate new applications and become more advanced, OTA configuration has become increasingly important as new updates and services come on stream. OTA via SMS optimizes the configuration data updates in subscriber identification module (SIM) cards and handsets and enables the distribution of new software updates to mobile phones or provisioning handsets with the necessary settings with which to access services such as WAP or MMS. OTA messaging provides remote control of mobile phones for service and subscription activation, personalization and programming of a new service for mobile operators and Telecommunication parties.

The OTA mechanism requires the existing software and hardware of the target device to support the feature, namely the receipt and installation of new software received via the wireless network from the provider. New software is transferred to the phone, installed, and put into use. It is often necessary to turn the phone off and back on for the new programming to take effect, though many phones will automatically perform this action.

Depending on implementation, OTA software delivery can be initiated upon action, such as a call to the provider's support system or can be performed automatically. Typically, it is done via the former method to avoid service disruption at an inconvenient time, but this requires subscribers to manually call the provider. Often, a carrier will send a broadcast SMS text message to all subscribers (or those using a particular model of phone) asking them to dial a service number to receive a software update. This solution allows operators to manage wireless device functionality including renumbering handsets, updating phone settings, applications and subscriber data and adjusting preferred roaming list (PRL) to manage cost structures. PRL is a database residing in a wireless device, such as a cell phone, that contains information used during the system selection and acquisition process. To provision parameters in a mobile device OTA, the device needs to have a provisioning client capable of receiving, processing and setting
the parameters. For example, a device management client (DMC) in a device may be capable of receiving and provisioning applications, or connectivity parameters.

IV. Significance of Over-the-air Programming in WSN

Wireless sensor networks have been proposed for a wide variety of application areas (Akyildiz et al., 2002; Nakamura et al., 2007; Boukerche, 2008). WSNs may have to operate in remote and harsh environments; in this case, applications need to be operated in an unattended way for long periods. These challenges introduce some difficulties. First, the environment can evolve over time, so it is very difficult to anticipate how a sensor node must operate during its lifetime. Second, the application requirements may change. For example, due to technological advances or even a better understanding of the environment, it would be interesting to have different application behaviors. The aforementioned difficulties make WSN management and maintenance challenging tasks. Within this context, OAP protocols play a key role as an enabling technology for WSN applications (Chen, 2008).

In traditional networks, the data communication typically happens between arbitrary communicating entities, whereas in WSN data communication tends to happen in such a way that the sink node is either the destination or origin of a message. Usually, WSNs are composed of a set of homogeneous nodes; however, Gupta and Kumar (2000) and Yarvis et al. (2005) showed that homogeneous ad-hoc networks suffer from poor fundamental limits and network performance. Consequently, heterogeneous sensor networks are gaining wide acceptance. Such a network assumes the existence of a large number of low-end sensors and a small number of powerful high-end sensors. The greater communication capacity of high-end sensors leads to a non-trivial network topology structure.

In most applications where sensor networks are deployed once and expected to operate unattended for a long period, the challenge is how to manage and maintain this large-scale network of wireless sensor nodes. Beutel et al. (2009) discussed 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 analyzed 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. (2009) stated 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. Thus, there exists a need to provide efficient and feasible software maintenance strategy to deployed sensor nodes.
Over-the-air programming is a fundamental service that is based on reliable broadcast communication. Over-the-air programming eliminates the need for detaching the sensor nodes and attaching data transfer cables when updating the sensor software. In OAP, the program binary image is generally fragmented to be transported from base station to sensor nodes. Figure 1 shows over-the-air programming in WSN, application programmer transmits code updates over-the-air from pc to sensor nodes mounted on bridge.

Figure 1. Over-the-air programming for sensor nodes mounted on bridge structure

V. Literature Review

Various OTA protocols have been proposed for WSN in the past few years. The first breakthrough on the contrary to the traditional way of reprogramming microcontrollers using In-System Programming (ISP) sensor networks that explored a way to update the node’s firmware without human intervention was developed at Berkeley, the University of California. Popular known as XNP (Crossbow Network Programming) is a one-hop protocol that offers firmware updates through a wireless link (http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/NetworkReprogramming).

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 (http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/xnp). Reijers and Langendoen (2003) proposed an energy-efficient code distribution scheme to wireless 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 was built using an edit script of commands that are easy to process by the nodes. A small change to the program code could cause many changes to the binary code because the addresses of functions and data change. The scheme was 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 was possible, including the operating system and the code distribution scheme itself. The scheme distributed 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 reduced the amount of communication compared to simply transferring the binary code. Nevertheless, the algorithms lack speed and a lot of overhead was incurred when splitting the script.

MOAP is a multihop network programming mechanism developed by Stathopoulos et al. (2003). 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 was provided to show their usefulness.

Jeong and culler (2004) proposed an incremental approach in which the host program generates the difference of the two program images using the Rsync algorithm (Tridgell, 1999), 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 (2004) 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 (Heinzelman et al., 1999),
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. (2003) 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. Moreover, the energy required to distribute this data was within the allowable per-mote energy budget. Deluge is similar to SPIN-RL (Kulik, 2002) 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 exists 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 (2005) 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 (2004) 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, Arumugam (2004)
also proposed an algorithm to reduce messages receptions and the active radio time further by using the notion of preferred predecessors.

Naik et al. (2007) 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 (Clark et al., 1990). Sprinkler effectively manages the latency by computing a near optimal schedule using a local D-2 coloring algorithm (Krumke et al. 2001). 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 re-broadcast 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 and Culler (2004) 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 (Levis, 2004).

Phillips (2005) 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. 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 (Chlipala et al., 2003). While reprogramming a partial network, scope selection approach of aqueduct saves energy across the whole network.

Marron et al. (2005) 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
was motivated by the increasing heterogeneity of sensor network applications and their need for more complex (non-homogeneous) network topologies and structures. The second one, motivated by the intrinsic energy constraint issues and, in general, the resource limitation of sensor networks. Finally, the third one was 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 and Deluge. 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 (2003) used 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 (2004) used the Rsync algorithm (Tridgell, 1999) to find variable-sized blocks that exist in both code images and then only transmits the differences. However, both of these approaches just compared 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 and Shen (2007) 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 (http://www.ZigBee.org) platform developed by Industrial Technology Research Institute (ITRI) Hsinchu, Taiwan. The transaction-based update system was managed by command line and a TCL scripting interface (Ousterhout,
1994; Welch et al., 2003). Most computation power required by the system relies on the base station, thus a more powerful base station is required.

Hagedorn et al. (2008) 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 replaced the data transfer mechanism of the established OAP Deluge protocol with rateless analogs. Compared to Deluge, 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 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. (2007) 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, reduced the reprogramming time, the number of bytes transferred, the energy expended, and the usage of program memory.

Krasniewski et al. (2008) 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 could be put to sleep by making the advertisement-request-data handshake happen only at certain points in time. The energy optimization was 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 introduced 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 made the delay proportional to the sum of the network diameter and the code size rather than the product of the two.
Rossi et al. (2008) 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) (MacKay, 2005), specifically designed to meet the needs of sensor network reprogramming, was used at the heart of the data dissemination/recovery process. This code was 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 (Chlipala, 2003) implemented 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 used 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 (Panta et al., 2007).

Heo et al. (2010) 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. That 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 and Culler (2004) 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 and Joo (2009) 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 was the same as that for Efficient Partial Node Update for Wireless Sensor Networks Using a Simulated Virtual Node (Kim et al., 2008), where the concept of a virtual node was 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 (http://www.ZigBee.org) provides some procedure for initial procedure of commissioning for small wireless nodes. 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. (2009) 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 applied to a network with small world features. Therefore, this work presents a new in-network algorithm called OAP-SW that combined the shortcut creation with OAP to improve network reprogramming. It enabled pipelining by dividing a program into fixed-sized segments, which in turn were divided into packets. Such an approach resulted 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. This occurred because the small world infrastructure, provided by OAPSW, reduces the minimal average path length of the network.

Shaikh et al. (2010) 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. It focused 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, Deluge or Freshet. 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 (Krasniewski, 2008), to save energy the sleeping time of the node is estimated beforehand and this estimation is 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 (Chlipala, 2003), 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 (http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/NetworkReprogramming), Reijers approach (Reijers and Langendoen, 2003) and Incremental approach (Jeong and culler, 2004). 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; 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 (Levis, 2004). 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 (Naik et al., 2007) and Infuse (Arumugam, 2004) 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 (Levis and Culler, 2004) and Sprinkler (Naik et al. 2007) 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 (Phillips, 2005) and TinyCubus (Marron et al., 2005). The various protocols studied above are summarized in Table I (Quadri and Sidek, 2013).

VI. Over-the-air Programming Protocols in WSN

As WSN is a versatile platform for many applications like structural health monitoring, aggressive research is being carried out in this domain and we often come across many new protocols. In most of the cases, the necessity of specific application is the basis for protocol development.

The desirable characteristics of OTA protocols for reprogramming are (a) 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 with limited hardware capacities; (b) 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; (c) 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; (d) 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; (e) there are several programming support limitations in current TinyOS (Wang et al., 2006).

Ideally, software 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. 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. Several protocols have been designed and studied in the past few years; the summary of various protocols discussed in literature review section is presented in Table 1.

**TABLE 1**
**SUMMARY OF VARIOUS REPROGRAMMING PROTOCOLS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Encoding/decoding</th>
<th>MAC</th>
<th>Hop</th>
<th>Scope</th>
<th>Hierarchical</th>
<th>Pipelining</th>
</tr>
</thead>
<tbody>
<tr>
<td>XNP</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Single hop</td>
<td>Whole network</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Reijers</td>
<td>Platform-dependent patch</td>
<td>CSMA</td>
<td>Single hop</td>
<td>Whole network</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Incremental</td>
<td>Platform-independent patch</td>
<td>CSMA</td>
<td>Single hop</td>
<td>Whole network</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Trickle</td>
<td>Mate script</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>MOAP</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Deluge</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>MNP</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Sprinkler</td>
<td>Complete program</td>
<td>TDMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Firecracker</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Infuse</td>
<td>Complete program</td>
<td>TDMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Aqueduct</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Selected nodes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>TinyCubus</td>
<td>Modular update</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Selected nodes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Transaction-based</td>
<td>TCL script</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Rateless</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Stream</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Freshet</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Synapse</td>
<td>Complete program</td>
<td>CSMA</td>
<td>Multihop</td>
<td>Whole network</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
VII. Conclusion

Over-the-air programming is a fundamental service based on reliable broadcast communication. 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. Over-the-air programming is necessary to fix bugs, update codes, and manage application requirement changes. It 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. It eliminates the need for detaching the 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. Compare and contrast of various protocols presented in this paper serve as basic stuff for the beginners. 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 overhead under all real time constraints.

References


