Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 4, April 2014, pg.190 – 198

SURVEY ARTICLE

Survey on Secure Updates using Over the Air Programming in Wireless Sensor Network

Naresh M. Bhagat[1], S. P. Akarte[2]

[1]ME (CSE), First Year, Department of CSE, Prof. Ram Meghe Institute of Technology and Research, Badnera, Amravati

Email ID: naresh.bhagat90@gmail.com

[2]Professor, Department of CSE, Prof. Ram Meghe Institute of Technology and Research, Badnera, Amravati

Email ID: s_akarte25@rediffmail.com

ABSTRACT: *Wireless Sensor Networks (WSNs) face many challenges including reliability, flexibility and security. When WSNs deployed in remote locations need to be reprogrammed, environmental conditions often make it impossible to physically retrieve them. Over the Air Programming (OAP) plays an important role in achieving this task. Over-the-air programming (OAP) is a fundamental service in sensor networks that relies upon reliable broadcast for efficient dissemination. SenSeOP Programming protocols provide a convenient way to update program images via wireless communication. In hostile environments where there may be malicious attacks against wireless sensor networks, the process of reprogramming faces threats from potentially compromised nodes. While existing solutions can provide authentication services, they are insufficient for a new generation of network coding-based reprogramming protocols in wireless sensor networks. We present a security approach that is able to defend pollution attack against reprogramming protocols based on network coding. It employs a homomorphic hashing function and an identity-based aggregate signature to allow sensor nodes to check packets on-the-fly before they accept incoming encoded packets, and introduces an efficient mechanism to reduce the computation overhead at each node and to eliminate bad packets quickly. In this paper, introduce SenSeOP, a selective and secure OTAP protocol for WSNs. For this purpose, the proposed protocol uses multicast transfer supported by cryptography. We evaluate the performance of our approach in real testbeds, compare it with state-of-the-art protocols, and show that this approach enables efficient and reliable wireless reprogramming.*

Keywords— *SenSeop; Security; Reprogramming*

I. INTRODUCTION

A Wireless Sensor Network (WSN) is composed of small and highly resource-constrained sensor nodes that monitor some measurable phenomenon in the environment, e.g., light, humidity, or temperature. WSNs are deployed in a steadily growing plethora of application areas.

These range from military (e.g., security perimeter surveillance) over civilian (e.g., disaster area monitoring) to industrial (e.g., industrial process control). Application scenarios of WSNs typically involve monitoring or surveillance of animals or humans, infrastructure, or territories. Their long-life and large-scale design, various deployment fields, and changing environments necessitate the feasibility of remote maintenance and in-situ reprogramming of sensor nodes using a so-called Over-The- Air Programming (OTAP) protocol. In particular, if sensor nodes are inaccessible after deployment, a reliable OTAP is crucial. We believe that in a plurality of WSNs, the network-wide dissemination of program code is not appropriate. Within a single WSN, the heterogeneity of sensor hardware, the deployment of manifold sensor technologies, the diversity of sensing and communication tasks, and possibly the event and location dependency of software require a flexible, group-wise selective OTAP approach in order to be able to efficiently reprogram a subset of nodes. Furthermore, securing the OTAP protocol is imperative in order to protect the OTAP from unauthorized reprogramming attempts, i.e., to prevent reprogram node attacks.

In this paper, present SenSeOP, a Selective and Secure over the Air Protocol which is integrated in our intrusion detection system and offers both, selective and secure reprogramming in WSNs. For our approach, assume infrequent and non-regular software updates. On the one hand, these updates are supposed to be time- and energy- efficient. On the other hand, it is essential that the primary application running on the sensor nodes is interrupted by the OTAP process as little as possible. However, short interruptions of several seconds are inevitable due to the usage of strong cryptographic operations and must be tolerated by the application. Additionally, we assume that confidentiality is not required in these scenarios since no trusted information, e.g., symmetric keys, are exchanged using our Senseop protocol. However, if in a specific scenario confidentiality is demanded, it may be achieved by link-layer encryption, Moreover, Senseop protocol in its current version is tailored to scenarios with all nodes residing in communication range to each other.

A).Over-The-Air-Programming Protocol

In the over the air programming protocol (OTAP) for WSNs. This protocol is described in two phases: infrastructure and reprogramming. The infrastructure is responsible to aggregate into the WSN the small world features. The reprogramming specifies the messages used to allow the code mobility. The problem assessed here is to reconfigure the program running on all sensor nodes of a WSN. Assuming the net- work is connected; all sensor nodes must receive the exact program image and be updated with the same version of code. Currently, only networks with stationary nodes are considered. A code dissemination protocol should meet the following requirements.[6]

a) Energy efficiency: The energy used during the re- configuration process should be as low as possible to not affect the network lifetime.

b) Time efficiency: The program should be propagated and installed as quickly as possible.

c) Reliability: The exact program image must be received by sensor nodes and eventually all sensors, in the network, are reprogrammed.

d) Autonomy: The reconfiguration process must be accomplished without human intervention. Although energy and time efficiency are not essential requirements for the correctness of code dissemination mechanisms they are important and cannot be overlooked for the practical use of any system. Since it is difficult to fulfill all the design goals in a system, the tradeoff has to be made to ensure the system's overall performance goals

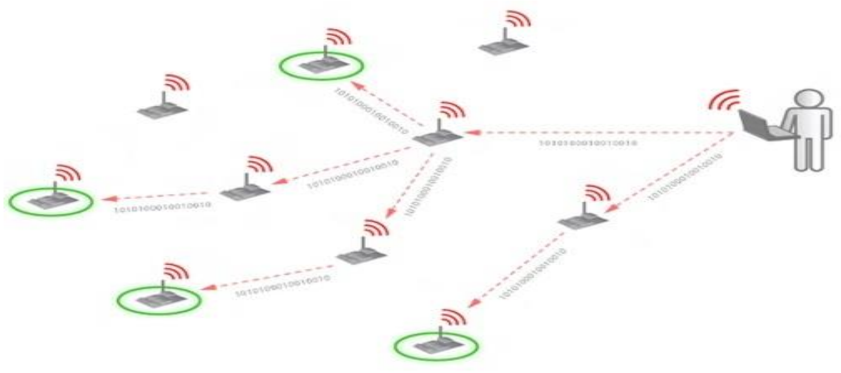


Figure 1. Over The Air Programming

II. Related Work

A) Validating the Model

Space does not allow for a full validation of the model against the many systems, protocols, and mechanisms reviewed in preparing the criteria and model. Instead, the model is validated here against three different systems, representing three different classes of software update: static/monolithic updating (MOAP), dynamic/mobile agent-based updating (Maté), and dynamic/component-based updating (Impala). In addition, the model is validated against a distribution protocol (Deluge).

B) Maté: Maté [1] is a communication-centric WSN middleware layer for TinyOS, based on a virtual machine (VM) architecture.

(a) Generation — Maté supports code in 24 instruction capsules; larger programs can be supported using subroutine capsules (currently limited to four). To generate the update, programs are assembled, and then insert takes place through the injection of the new modules (or new module versions) into the network using the capsule injector to a directly connected mote.

(b) Propagation — Every node broadcasts a summary of 32-bit capsule version numbers to its neighbors, based on a random timer, to advertize an update. Code capsules are flooded through the network via viral programming — code capsules can be marked as self-forwarding, and such capsules are flooded throughout the network. This flooding continues even after the entire network has been updated. Decreasing the advertisement frequency when an identical vector is heard — at density-aware transmission rates — can provide an efficient trade-off between response time and avoiding network saturation. Each node listens for broadcast version vectors. If it hears a version summary with older capsules than it has itself, then Decision triggers transfer/send to broadcast the newer modules. Complete capsules are transferred in each message; the target VM does a transfer/recv for these. No verify is required, as code capsules fit into single packets; the continuous flooding handles lost packets.

(c) Activation — When the target VM receives a new capsule version, it unloads the previous version, and loads the update. The new capsule version will be triggered for execute by an associated event. Neighbors effectively monitor the version update through the advertisements. Maté uses "viral code propagation" to deliver mobile code updates to all nodes; it focuses on the propagation and mobile-code related activation aspects. It also provides limited support for advanced generation and activation features.

C). MOAP: MOAP [2] is a multihop, over-the-air code distribution mechanism specifically targeted at MICA2 motes running TinyOS.

(a) Generation — Only static/monolithic image updates are sup- ported. To build the update, a packetizer converts the soft-ware image to 16-byte segments. Insert is supported by delivering the software update to a base station, along with the software version number.

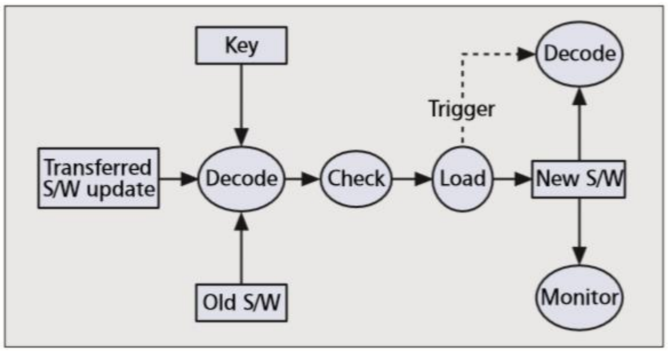


Figure 2. Activation.

(b) Propagation--To advertize the update, the base station (and, subsequently, secondary sources) broadcast publish messages advertising the version number of the new code. This ripple approach explicitly increases latency in order to reduce trans-mission energy. Network partitioning is handled by a late join- er mechanism — nodes continuously send publish messages to advertise their version. Receivers listen for a period to receive all subscriptions, in order to optimize the selection of sender. Decide is implemented by receiving nodes checking the advertised version number. Transfer/client requests the update with subscribe messages. A link-statistics mechanism is used to try to avoid unreliable links. If transfer/server receives no sub- scribe messages within a timeout period, it "commits" by triggering activation. The store-and-forward approach provides a "ripple"

pattern of updates. Once a node has received an entire image, it becomes a sender in turn. The verify performance is enhanced by using a bitmap of received segments to reduce EEPROM reads. Missing segments are identified by the receiver using a sliding window, and are re-requested directly from the receiver using a unicast message to prevent duplication. NACK-based re-requests reduce both energy usage and memory space requirements for the sender. Re- requests are prioritized to reduce delay by quick recovery. Sliding window acknowledgments reduce power consumption (reduced EEPROM reads) at the cost of reduced out-of-order message tolerance. If unicast retransmission requests are not responded to, broadcasts are used.

(c) Activation-- A decode function transforms the downloaded text records into binary data. The check is provided by the underlying link mechanisms on a per-packet/segment basis. A bootloader performs load and execute by transferring the new image to program memory from EPROM, and then rebooting. The version advertisements provide a limited monitor, which gives version feedback to neighbors. MOAP focuses on the performance of the propagation protocol, and provides limited support for advanced generation or activation features.

D). Impala/ZebraNet : Impala is the event-based middleware layer of ZebraNet [3], which uses wildlife tracking as a target application in the development of mobile wireless sensor networks.

(a) Generation — Impala supports wide range of updates (from bug fixes, through updates, to adding and deleting entire applications). It uses version numbering to ensure compatibility of updates with existing modules. In generate applications can be formed of multiple modules (with an 8 KB module limit). Linking takes place on the target nodes, so unlinked binaries are prepared for injection. To insert the update, it is sent to the base station.

(b) Propagation — Nodes advertize the version numbers of both complete applications and their constituent modules. Nodes listen to their neighbors' advertisements, and a decide triggers advertize to increase a backoff timer exponentially if all neighbors have the same software versions this significantly reduces management traffic, but can delay updates if network connectivity to nodes with earlier software versions is lost and recovered. In transfer/recv, nodes make unicast requests (using node ID as a tie-breaker) to trigger transmission of newer modules only, thus saving on network bandwidth. Transfer/send nodes respond to requests from other nodes, by transmitting the code for the first requested modules. In transfer/recv, if memory space is exhausted, then older incomplete applications are deleted. Multiple contemporaneous updates (different versions of the same module) can be processed. To verify updates, lists are used to keep track of which modules have been received; partially received modules are re-requested on the next advertisement through co-operation with a decide. When all the modules in a particular update have been received, activation is triggered.

(c) Activation — Simple sanity checks are performed (e.g., only valid memory accesses, and a return statement). To load an update, the old version application is terminated, the modules in the new version are linked in, and the pointers in the application activation table are updated. Code memory management is provided in 2 K blocks to cope with multiple applications each consisting of multiple modules. To execute after loading, the new application is first initialized. Applications can continue running while updates are being download- ed. Monitor is effectively provided to neighbors through the advertisements. Impala concentrates primarily on the propagation of the update, but also provides somewhat more comprehensive sup- port for generation and activation. The explicit support for modules can significantly decrease overhead.

E). Deluge : Deluge [4] is a data dissemination protocol for reliably propagating large amounts of data throughout a WSN using incremental upgrades for enhanced performance.

(a) Generation — Deluge does not itself support this.

(b) Propagation — A program image is split into fixed size pages that can be 'reasonably' buffered in RAM, and each page is split into fixed size packets so that a packet can be sent without fragmentation by the TinyOS network stack. Nodes advertize using a broadcast containing a version number and a page bit vector, using a variable period based on updating activity. Nodes listen to their neighbors' advertisements, and if the decide determines that it needs to upgrade part of its image to match a newer version, then, after listening to further advertisements for a time, trans- fer/recv sends a request to the selected neighbor for the lowest page number required and the packets required within that page. After listening for further requests, trans- fer/send selects a page, and broadcasts every requested packet in that page. Verify triggers transfer/recv to either request new pages, or re-request missing packets from a previous page. When a node receives the last packet required to complete a page, verify triggers advertize to broadcast an advertisement before requesting further pages to allow parallelization of the update within the network. Heuristics are used by transfer/send to try and select relatively remote senders (to minimize radio contention). Incremental updates are supported by advertize, indicating which pages have changed since the previous image version; the decide then triggers transfer/recv to request just the changed pages.

(c) Activation — Deluge itself does not support this: an associated bootloader, TOSBoot, allows Deluge objects to be invoked at boot time. Deluge concentrates on the efficient propagation of the update, and includes many optimizations to minimize the reprogramming time and power used.

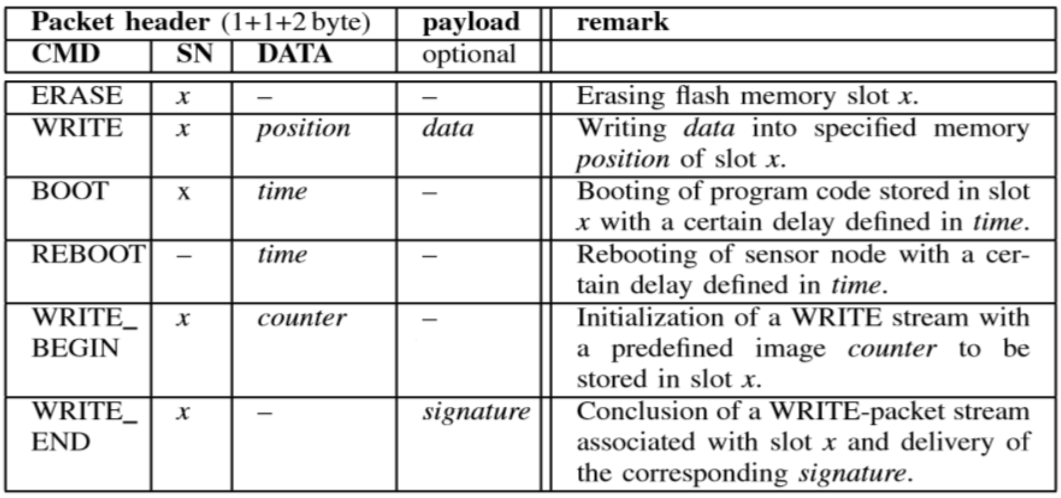
III. GENERAL INTRODUCTION TO ENCRYPTION ALGORITHM

Many encryption algorithms are widely available and used in information security. They can be categorized into Symmetric (private) and Asymmetric (public) keys encryption. In Symmetric keys encryption or secret key encryption, only one key is used to encrypt and decrypt data. In Asymmetric keys, two keys are used; private and public keys. Public key is used for encryption and private key is used for decryption (e.g. RSA and ECC). Public key encryption is based on mathematical functions, computationally intensive and is not very efficient for small mobile devices . There are many examples of strong and weak keys of cryptography algorithms like RC2, DES, 3DES, RC6, Blowfish, and AES. RC2 uses one 64-bit key. DES uses one 64-bits key. Triple DES (3DES) uses three 64-bits keys while AES uses various (128,192,256) bits keys. Blowfish uses various (32-448); default 128bits while RC6 is used various (128,192,256) bits keys . The most common classification of encryption techniques can be shown in Figure 1. Here we provide a method for evaluating performance of selected symmetric encryption of various algorithms. Encryption algorithms consume a significant amount of computing resources such as CPU time, memory, and battery power. Battery power is subjected to the problem of energy consumption due to encryption algorithms. Battery technology is increasing at a slower rate than other technologies. This causes a "battery gap". We need a way to make decisions about energy consumption and security to reduce the consumption of battery powered devices. This study evaluates six different encryption algorithms namely; AES, DES, 3DES, RC6, Blowfish, and RC2. The performance measure of encryption schemes will be conducted in terms of energy, changing data types - such as text or document, Audio data and video data- power consumption, changing packet size and changing key size for the selected cryptographic algorithms.[14]

IV. DESIGN AND IMPLEMENTATION

I approach adopts several features of Deluge, e.g., the fragmentation of the image into a stream of several pages subdivided into packets, the insertion of metadata to the program image, the error detection mechanism, and the flash memory layout using multiple slots. For the packet-based delivery of the program image, we adapted the message format from NWProg and extended the defined commands in order to enable the intended security mechanisms presented below.

Request packets are generated by the OTAP operator using a python script running on the server. These packets contain a specific command (CMD), a slot number (SN), additional information (DATA), and optional payload. For instance, a WRITE command packet is provoking the receiver to write the data block included in the payload to the specified position (DATA) of the image slot defined in the SN field. An overview of all commands and the corresponding allocations is provided in Table 1.



Packet header (1+1+2 byte)			**payload**	**remark**
CMD	**SN**	**DATA**	optional	
ERASE	*x*	–	–	Erasing flash memory slot *x*.
WRITE	*x*	*position*	*data*	Writing *data* into specified memory *position* of slot *x*.
BOOT	x	*time*	–	Booting of program code stored in slot *x* with a certain delay defined in *time*.
REBOOT	–	*time*	–	Rebooting of sensor node with a certain delay defined in *time*.
WRITE_ BEGIN	*x*	*counter*	–	Initialization of a WRITE stream with a predefined image *counter* to be stored in slot *x*.
WRITE_ END	*x*	–	*signature*	Conclusion of a WRITE-packet stream associated with slot *x* and delivery of the corresponding *signature*.

Table 1: Senseop Protocol Commands And The Corresponding Request Packet Format.

Beside the request packets, reply packets (6bytes) are used as acknowledgements (ACKs). Containing the header of the corresponding request packet (4bytes), these packets signal the success of requests with an error field. Since we apply an Automatic Repeat reQuest (ARQ) mechanism with stop-and- wait strategy, sequence numbers are required, in particular for WRITE packets. The memory position stored in the DATA field of WRITE packets in combination with the slot number offers a unique identification and, thus, is leveraged as an implicit sequence number.

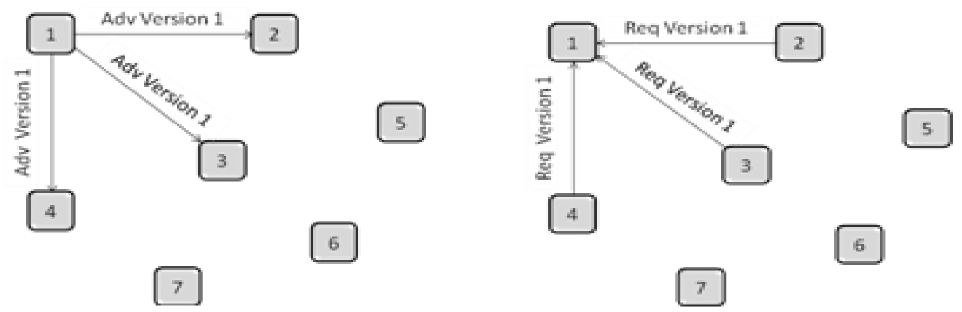
A). Reprogramming Protocol Phase

The OAP-SW protocol is a NACK based protocol and it relies on periodic advertisements to keep nodes informed of their neighbors states. Its basic cycle of communication is depicted in Figure 2 and detailed in the following steps: [6]

Step 1 Every node in the network periodically broadcast advertisements (Figure 2(a)). Each advertisement contains a version number and a bit vector describing which pages of that version the advertiser has received completely. Faster advertisements are used in periods of active updating.

Step 2 The node determines the lowest numbered page it requires and sends a request to the first sender that advertised that page, indicating the page and packets within that page are needed (Figure 2(b)).

Step 3 The node starts a period of waiting. When this period ends, the sender broadcasts a data packet for every requested packet of the selected page (Figure 2(c)).



Fig(a) **Fig(b)**

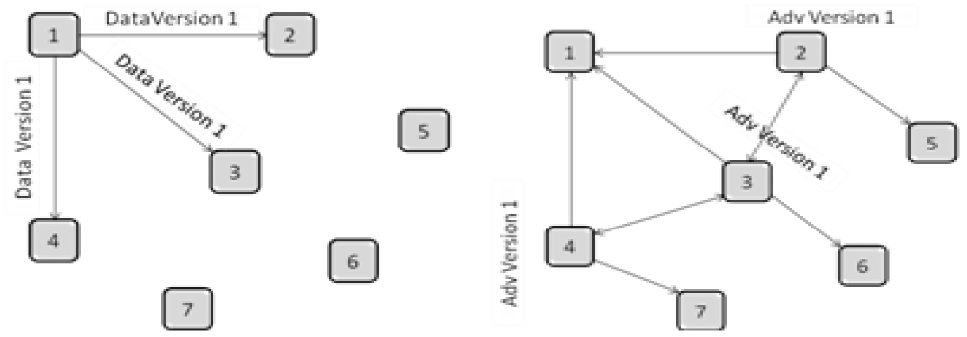


Fig (c) **Fig(d)**

Figure 3: Basic steps of protocol messages

Step 4 When a node receives the last packet it needs to complete a page, it broadcasts an advertisement before attempting to request any new packets to still needs, thus enabling spatial multiplexing (Figure 2(d)).

It should be observed that the protocol still avoids maintaining any kind of neighbor table or any other information on the states of other nodes, resulting in a robust and simple implementation.

V. THEAT MODEL AND SECURITY GOALS

Corresponding to define the following generic threat model and security goals demanded by our approach.

1). Sensor nodes deployed in the WSN are non-tamper-proof and may be compromised by an attacker.

2).The base station acting as the OTAP gateway is assumed to be a trusted device which cannot be compromised.

3).Assume an inside attacker who is able to eavesdrop on, inject, and manipulate packets in the WSN. Arising from the threat model, the security goals stated below represent the essential requirements for secure OTAP protocols.

A).Reprogram node attacks resilience: The main goal of the SenSeOP protocol is the resilience against reprogram node attacks with malicious code. Thus, the authenticity of each received program image has to be verified. It must be guaranteed that all devices solely reprogram authenticated images of authorized entities. Additionally, prior to reprogramming, the integrity of received images must be guaranteed since the assumed attacker may be able to manipulate data. Replay attacks resilience: A secondary goal is the resilience against replay attacks. It must be guaranteed that an attacker cannot replay eavesdropped program images. Furthermore, the attacker must not be able to exploit eavesdropped images to reprogram uninvolved nodes which are not addressed by this image.

B). Mitigation of DoS attacks: An additional goal is the mitigation of DoS attacks . Using cost-intensive ECC in terms of computational power, it is possible that an attacker aims for a DoS attack by forcing the verification of malicious or faked images. This may lead to disruptions of the primary application and to the depletion of constrained energy resources.

C). Compromise-tolerant: Also relevant in terms of security is compromise tolerance. A single compromised device must not allow the attacker to compromise other devices in the WSN.

D). Light-weight security solution: Finally, the proposed security solution has to be light-weight in terms of required computational power and memory.

VI. MEMORY FOOTPRINT

The code size resulting from our modular SenSeOP protocol integrated into an exemplary core application is summarized in usage is visualized, separated into the basic OTAP component and optional extensions which are successively added to the core application. One can observe that the optional but efficient multicast extension yields only minimal additional cost. In contrast, the OTAP component as well as the security extensions causes an inevitable memory demand. However, in particular for the G-Node platform, which offers more program memory (ROM), the achieved code size is appropriate for a deployment in practice since there is enough memory left for a reasonable application.

A). Setup

In the evaluation, two one-hop topology testbeds are deployed. One testbed is composed of five TelosB nodes in communication range of each other, whereas the second testbed consists of G-Nodes. In each testbed, one specific node attached to the PC and running the TinyOS BaseStation application is acting as an OTAP gateway. The other four nodes represent the clients which are intended to be reprogrammed. Therefore, a test application is preinstalled on these nodes. This application contains the realized SenSeOP module with all extensions and a simple blink component. The purpose of the blink application is to be able to distinguish between the current and the new application which is supposed to be installed via OTAP. Therefore, there are two alternative versions of this test application compiled in advance. Each version using a different LED color for its blinking. Using SenSeOP, in each step of the evaluation, the current version is replaced by its counterpart and vice versa. As performance metrics, we consider the reliability of the SenSeOP operation as well as the latency and the overhead introduced by this protocol:

a). Reliability: Aside from the robustness against reprogramming node attacks. the reliability of the SenSeOP protocol is an important requirement. The reliability is defined as the success probability of each reprogramming operation, i.e., whether the reprogramming operation was successful or not. In particular, a successful reprogramming operation includes the complete delivery of the new program image and its signature as well as the verification of its integrity and its authenticity.

b). Latency: The latency of the reprogramming operation is defined as the period of time between the initial WRITE BEGIN packet sent by the OTAP base station and the reply of the concluding WRITE END packet sent by the receiver. As a result, the time required for the verification and the rebooting of the corresponding sensor node(s) is not considered by the latency metric.

c).Overhead: In order to determine the protocol overhead, the overall traffic on application layer induced by reprogramming operations is considered. This is done by summarizing the size of all packets sent during these operations. This includes all request packets of the base station and the replies of every receiver. Furthermore, it is distinguished between control overhead, which is the traffic resulting from control packets and protocol headers, and the payload, which is associated with the program image. The latter includes all possible retransmissions belonging to the program image as well.

B).Comparison to related work

In order to compare SenSeOP with NWProg and Deluge, we consider the overhead and the latency of the reprogramming again. However, instead of measuring the overhead on application layer, the measurement is carried out on the physical layer taking the link layer ACKs used by NWProg into account. We apply the same setup described in SectionV-B, the maximum payload size, and the default configuration of the protocols. Using NWProg and SenSeOP (unicast) , we reprogram one node, whereas all nodes are reprogrammed by Deluge and SenSeOP (multicast) . Due to space limitations, we just present results for the TelosB testbed. The results are depicted in Figure 4 and show that in the unicast case, our secure approach achieves better performance than NWProg which uses link layer ACKs, resulting in a higher overhead and a higher latency in this scenario. Since our approach cannot guarantee the success of the reprogramming using broadcast transmission, we use the multicast mode in the comparison with Deluge. Due to the usage of reply messages instead of selective negative ACKs, a higher overhead as well as an increased latency must be tolerated with our approach, as can be seen in Figure 4. However, considering that our SenSeOP is secure and offers a selective group-wise reprogramming, the additional costs are acceptable.

VII. CONCLUSION

This review identifies the large number of different dissemination protocols have been developed for WSN software updates. Many of these have features designed for particular platforms or execution environments, or for particular types of network configuration. This review identifies three different classes of how software updates are executed (or made available for execution): static/monolithic updates, dynamic/modular updates, and dynamic/mobile agent- based updates. This review also identifies two classes of update format: incremental updates (which are dependent on the current version), and full updates (which are not). Finally this review has identified two connectivity solutions: using the sensor network itself, and using a parallel maintenance network.

Overall this progress through discrete solutions reveals the fundamental research challenge in WSN software updating: to bring all this together into a cohesive framework. Most of the discrete solutions are optimal in some situations, but not in others, and thus a framework needs the flexibility to react to the current state of the WSN in order to operate dependably, efficiently, effectively, and securely. The inaccessibility and scale planned for sensor networks argue for an autonomic approach [9][10] to solving the software update problem and an approach that is able to defend attack against reprogramming protocols based on network coding. This approach employs a hashing function and an identity-based aggregate signature to allow sensor nodes to check packets on-the-fly before they accept incoming encoded packets, and introduces an efficient mechanism to reduce the computation overhead at each node and to eliminate bad packets quickly.[11]

REFERENCES

[1] P. Levis and D. Culler, “Maté: A Tiny Virtual Machine for Sensor Net- works,” Proc. 10th Int’l. Conf. Architectural Support for Programming Languages and Op. Sys., San Jose, CA, Oct. 2002, pp. 85–95.

[2] T. Stathopoulos, J. Heidemann, and D. Estrin, “A Remote Code Update Mechanism for Wireless Sensor Networks,” CENS tech. rep. #30, Dept. Comp. Sci., UCLA, Nov. 2003.

[3] T. Liu et al., “Implementing Software on Resource-Constrained Mobile Sen- sors: Experiences with Impala and ZebraNet,” Proc. 2nd Int’l. Conf. Mobile Sys., Apps. and Svcs., Boston, MA, June 2004, pp. 256–69.

[4] J. Hui and D. Culler, “The Dynamic Behavior of a Data Dissemination Proto- col for Network Reprogramming at Scale,” Proc. 2nd Int’l. Conf. Embedded Networked Sensor Sys., Baltimore, MD, Nov. 2004, pp. 81–94.

[5] P. Dutta, J. Hui, D. Chu, and D. Culler, “Securing the Deluge Network

[6] Patrick E. Lanigan, Rajeeve Gandhi, Priya Narasimhan, “Sluice: Secure Dissemination of Code Updates in Sensor Networks,” in Proc. of the 26th Int. Conference on Distributed Computing Systems (ICDCS ’06) , Lisboa, Portugal, 2006, pp. 53–53.

[7] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms, “Explicit Multicast (Xcast) Concepts and Options (RFC 5058),” http://tools.ietf

[8] Guilherme Maia, Daniel L.Guidoni, Andre L. L. Aquino, Antonio A. F. Loureiro, “Improving an Over-the-Air Programming Protocol for Wireless Sensor Networks Based on Small World Concepts” MSWIM’09, October 26–30, 2009,

[9] Diaa Salama Abd Elminaam, Hatem Mohamed Abdual Kader, and Mohiy Mohamed Hadhoud “Evaluating The Performance of Symmetric Encryption Algorithms”, International Journal of Network Security, Vol.10, No.3, PP.216–222, May 2010I

[10] Kephart, J.O., Chess, D.M.: “The Vision of Autonomic Computing”. In: IEEE Computer, vol. 36, iss. 1, Jan. 2003. IEEE (2003) 41-50

[11] Xing She Zhou, Yee Wei Law and Marimuthu Palaniswami, "A Secure Method for Network Coding-based Reprogramming Protocols in Wireless Sensor Networks", I.J.Computer Network and Information Security, 2011, 2, 34-40

AUTHORS

Mr. Naresh M. Bhagat, ME (CSE) ,First Year,Department of CSE,Prof. Ram Meghe Institute Of Technology and Research, Badnera, Amravati. Sant Gadgebaba Amravati University, Amravati, Maharashtra, India – 444701

Prof. S. P. Akarte, Assitantant Professor, Department of CSE, Prof. Ram Meghe Institute Of Technology and Research, Badnera,Amravati. Sant Gadgebaba Amravati University, Amarvati, Maharashtra, India - 444701