Dependable Over-the-Air Programming*

JENS-MATTHIAS BOHLI1, ALBAN HESSLER3, KLAUS MAIER1, OSMAN UGUS2 AND DIRK WESTHOFF2

1NEC Laboratories Europe, Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
E-mail: bohli@nw.neclab.eu
E-mail: klaus.maier@gmx.li

2Department of Computer Science, HAW Hamburg, Berliner Tor 7, 20099 Hamburg, Germany
E-mail: osman.ugus@haw-hamburg.de
E-mail: westhoff@informatik.haw-hamburg.de

3AGT Germany, Hilpertstrasse 20A, 64295 Darmstadt, Germany
E-mail: ahessler@agtgermany.com

Received: February 10, 2011. Accepted: May 30, 2011

The complexity of software running on wireless sensor networks has increased over the years, and the need for an over-the-air (OTA) programming tool has become prominent. The requirements for the network traffic generated by a code update and the security issues that arise from it are atypical for wireless sensor networks, thus requiring innovative solutions. In this article, we provide an integrated protocol suite for a secure and efficient code image propagation in multi-hop wireless sensor networks consisting of three main parts: i) An efficient data structure including a program memory efficient T-time signature based on Merkle’s one time signature; ii) A transmission efficient authenticated pagewise packet encoding using rateless erasure codes with security measures against denial-of-service-attacks; iii) An adaptive multi-hop propagation strategy which uses techniques from fuzzy control to mitigate the hidden terminal problem. Weaving means from fuzzy control into the propagation scheme enables exploiting the benefits of rateless erasure codes by efficiently reducing channel congestion and, thus, packet collisions.

Keywords: Fuzzy Logic, Network Coding, One-time Signature, Over-the-Air Programming, Security, Wireless Sensor Networks

*The paper was written while the author was employed at NEC Europe Labs.
1 INTRODUCTION

Wireless sensor networks (WSNs) represent an emerging set of technologies that will have profound effects across a wide range of industrial, scientific, and governmental applications. In the civilian sector, probably the largest WSNs currently discussed, prototypically implemented, and tested are WSNs for critical infrastructure protection (CIP) like for example oil pipelines, power distribution networks, or nuclear power plants. WSN technology supporting CIP applications is usually linked via remote terminal units to a supervisory control and data acquisition (SCADA) system, which allows monitoring, controlling, and acquiring resources in the critical infrastructure. Support of green technology or road-side deployments of WSN islands for car-to-infrastructure (C2I) public safety applications are other examples for deployed large scale WSNs. To ensure that the above WSN applications indeed gain momentum, a set of enabling technologies needs to be available. Once deployed, sensor nodes may need to be reprogrammable in situ either to remove bugs or to add new functionalities. Who ever has participated in a real-world deployment of a WSN, with e.g., some dozens of nodes or more, comes to the conclusion that a manual reprogramming is not a real option. The most efficient way to do this is over the wireless. However, over-the-air (OTA) programming is critical regarding various aspects: Firstly, a multicast transmission of a relatively large data stream over a noisy, unreliable channel causes re-transmission of lost packets and can easily build up to an undesirable energy-wasting process. Secondly, OTA programming is vulnerable to a multitude of attacks: In open, public, untrusted, or even hostile environments a new code image (CI) needs to be authenticated such that only authorized parties are able to upload it to the sensor nodes. This prevents a WSN from running a bogus code image. Furthermore, OTA programming is required to be resistant and robust against Denial of Service (DoS) attacks. An adversary may inject bogus packets during the dissemination phase and force sensor nodes to propagate a corrupted image through multiple hops to deplete their limited power. Ideally, dependable OTA programming shall also be robust against jamming attacks and other interferences. However, for the latter we argue that a proper choice of the radio technology is the pre-dominant factor.

Recent work on OTA programming for WSNs proposes to apply rateless erasure codes [27]. In [27] Rossi et al. showed that even for a relatively small data stream like a code image such coding schemes are very beneficial when transmitting to multiple receivers over a noisy, unreliable shared medium. However, the potential presence of an attacker is fully neglected. We will see that taking into account the presence of an attacker who maliciously modifies data is of particular importance when applying rateless erasure codes. Otherwise, the potential benefit of such a coding technique may turn out to be a boomerang resulting in energy-wasting re-transmissions of the full CI.
Our contribution  This work provides an integrated dependable multi-hop OTA programming protocol consisting of three complementary components:

1. an efficient security anchor consisting of a new structure for data stream authentication and an efficient stateful-verifier $T$-time signature scheme,
2. an algorithm for DoS resistant fountain code decoding that prevents bogus packets from hindering the overall decoding process,
3. a multi-hop propagation strategy that ensures a high packet transmission rate through mitigating the hidden terminal problem.

We introduce the above sketched components and their interaction after the introduction of preliminaries and models. Section 7 presents our testbed as well as simulation results in a realistic large-scale simulation using TOSSIM.

2 RELATED WORK

We first introduce prior publications on OTA upgrades in WSNs, and then focus our attention on the work related to the areas of our solutions.

Related work on OTA programming in general - Several OTA programming protocols [28, 25, 15] have been suggested. However, the most popular OTA programming protocol in the literature is Deluge [10], which is included in recent TinyOS [30] distributions. Deluge is an epidemic programming protocol that extends Trickle [19] by adding support for large object sizes. One feature of Trickle is the mechanism for sender selection through suppression that limits the number of senders in the same area, thus diminishing losses through packet collisions. An important concept introduced by Deluge is spatial multiplexing, which allows for parallel data transfer.

Rossi et al. proposed Synapse [27], a data dissemination protocol that applies rateless Luby Transform (LT) codes. Synapse shows improved efficiency over Deluge in a single-hop scenario. It applies the Gaussian elimination mechanism for decoding. An extension of Synapse with multi-hop support is recently presented in [26]. A similar approach is Rateless Deluge [8], which is an extension of Deluge using random linear codes. To solve the set of linear equations and retrieve the source data, Rateless Deluge applies Gaussian elimination with back substitution. This requires, however, a long decoding time due to the significant asymptotic runtime complexity. Finally, AdapCode [9] also applies network coding and Gaussian elimination.

Related work on secure OTA programming - Several secure OTA programming protocols have been proposed [6, 16, 4, 11]. Using a single digital signature to authenticate the root of a hash chain or a hash tree and thus the complete CI, is common to all these approaches. They differ mainly in the number of hash chains and hash trees employed, the granularity of data considered in the hash chains, and the amount of flexibility in verifying the packets
when arriving out-of-order. Perrig et al. [24] proposed a broadcast authentication scheme for resource-constrained environments. The sender broadcasts each packet including a message authentication code computed with a secret key known only to itself. The receiver stores the received packet without verifying its authenticity on-the-fly. After a certain delay, the sender broadcasts the secret key and all receiving nodes then verify the authenticity of the stored packet by using the disclosed key. The need for a loose time synchronization between the sensor nodes and the sender is the main drawback here.

**Related work on secure fountain codes** - Solutions to protect the authenticity of fountain codes involve public-key primitives such as homomorphic signatures [14, 2]. However, both methods are not applicable on a per packet basis in a WSN due to the high data overhead compared to the very short packages and the high computational complexity of public-key cryptography.

**Related work on fuzzy control for WSNs** - The use of fuzzy logic and fuzzy control has recently become popular also in WSNs. The work in [7] introduced a fuzzy approach for the problem of cluster-head election. FAIR [3] uses distributed fuzzy control systems for data aggregation in WSNs to provide robustness and quality of information in the presence of bogus aggregator nodes. Due to its proven real-time responsiveness distributed fuzzy control is the mean we have chosen also in this work to handle the multi-hop CI propagation.

### 3 ADVERSARY MODEL AND PROTOCOL REQUIREMENTS

We are interested in a setting where OTA programming takes place in a large scale, multi-hop WSN. More precisely, a base station (BS) intends to disseminate a CI to all sensor nodes in the network. The BS may be physically present in the network or it may be connected to it via a gateway. For both settings, it is assumed that the BS requires wireless multi-hop communication over several sensor nodes to reach all nodes. Thus, the nodes are not only receivers, but also, on the next hop, senders. Furthermore, we assume that the sensor nodes are not tamper-proof and are deployed in an untrusted or hostile environment. This is reflected in the adversarial model by allowing the adversary to take control over some nodes.

We consider a full adversary and a limited adversary. The full adversary can compromise sensor nodes and has full control over the network. This enables the adversary to eavesdrop, block, and insert packets at will. This adversary is also capable of modifying packets, as modifying can be seen as simultaneously blocking a packet and inserting a new one.

For the limited adversary, the control over the network is restricted. We assume that a significant amount of packets between honest nodes cannot be modified or blocked by the limited adversary. This is a realistic assumption and motivated by an adversary that is trying to remain undetected. We model
this by a success probability $0 < \gamma < 1$ for the adversary in trying to block a packet. The limited adversary can, however, eavesdrop and insert packets at will in the whole network. We identify two security requirements:

**Bogus code image protection** - No sensor node writes unauthenticated code into memory. This has to hold against the full adversary.

**DoS resistance** - If every honest node has a connection to the base station via honest nodes, a limited adversary should not be able to prevent the reprogramming, nor to let nodes waste an unreasonable amount of energy.

### 4 CODE IMAGE AUTHENTICATION

Signing a code image before its dissemination is essential to authenticate the identity of the sender device. Previous secure OTA programming solutions use digital signature schemes such as RSA or ECDSA to achieve this goal. So far the dominating evaluation criterion in measuring the feasibility of all those security solutions is the dissemination time. However, the very limited program memory available on sensor platforms did not receive the appropriate attention. Even efficient implementations of digital signature schemes still require a large amount of program memory. For instance, the implementation of ECDSA occupies 28.1% (13.5kB) of the available program memory on TelosB which is a widely-used sensor platform [20]. Other signature schemes such as RSA, NTRUSign, and XTR-DSA introduce 7.4kB, 11.3kB, and 24.3kB memory overhead on the same platform [5]. Such large memory footprints prohibit a shared implementation of application and OTA programming mechanism without cutting back on functionality or security. In the following, we describe an improved data structure for the code image authentication and an efficient signature mechanism with a very small memory footprint.

#### 4.1 Efficient Stream Authentication

For the authentication of the code image, we follow a hybrid approach where a root value is authenticated by a digital signature and the packets are verified via hash chains and hash trees against the root value. One benefit of this is that it allows for out-of-order packet verification of each received packet in a level-by-level fashion, while keeping the computational effort low.

A hybrid approach that combines hash trees with hash chains was first used by Deng et al. [4] to prevent DoS attacks. Hyun et al. [11] improved the scheme from Deng et. al., by storing a hash tree of hash chains, instead of a hash chain of hash trees. A new innovation in our authentication structure is the adoption of two hash functions $h_1$ and $h_2$ with output of different lengths, i.e., different security parameters. Two security parameters $\kappa$ and $\lambda$ denote the security against second pre-image attacks and denote the output lengths of the
hash functions \( h_1 \) and \( h_2 \). Collision-resistance if not necessary as in a OTA programming scenario the base station is naturally assumed to be honest.

We will choose \( \kappa < \lambda \). Security against denial of service attacks only requires a moderate security level, because this attack is only possible online, while the code update is in transmission. As the protection against DoS is done on a packet level, it is important to reduce the security overhead to the strict minimum. Hence, with two separate hash functions, the security level against DoS attacks can be reduced without weakening the security against the bogus code image attack, thus improving the efficiency while keeping the required practical security.

This work assumes an OTA programming protocol based on Deluge [10], which is widely in use for sensor networks. However, the solutions presented in this work may be applied for other OTA programming protocols with minor modifications as well. In Deluge, the code image is divided into fixed-size pages, such that a page can fit the available RAM memory on a sensor node. Each page is then split up into fixed-size packets depending on the network payload size. Finally, the code image is transmitted packet-wise in a page-by-page mode to all sensor nodes in the network. Let \( CI_{auth} = P_0 || P_1 || P_2 || \ldots || P_n \) denote the resulting code image after embedding the authentication message. Thereby, the first page \( P_0 \) represents the signature and hash tree page and is only used to bootstrap the security context for the code image. The pages \( P_1 \) to \( P_n \) carry the actual code image. We describe in the following how to construct and verify the encoded code image.

In the deployment phase, the administrator generates a public/private key pair \((pk, sk)\) by using the key generation algorithm \texttt{Gen} (see Section 4.2) and installs the \( pk \) on every sensor node. Next, we describe the necessary steps for OTA programming.

**Partitioning the plain CI** - The process starts by partitioning the plain CI into pages and the pages into packets. The page and packet sizes are given, which determine the number of packets per page. The page and packet sizes are mainly determined by the RAM available on the receiving nodes and the optimal packet size for the network protocol. Starting from \( P_1 \) the data of the plain CI is filled into the packets leaving \( \kappa \) bit in each packet for later addition of the hash chain and reserving \( \lambda \) bit of the last packet of all but the last page for an additional hash value. Depending on the CI size, the last page \( P_n \) will be shorter than pages \( P_1 \) to \( P_{n-1} \).

**Construction of hash chains and hash tree** - First, the hash values have to be added to pages \( P_1 \) to \( P_{n-1} \). Then, the short special page \( P_0 \) will be constructed. The last page \( P_n \) does not contain hash values. Let \( p_{i,j} \) denote the \( j \)th packet of the \( i \)th page and \( h_1 \) and \( h_2 \) be two hash functions with output lengths \( \kappa \) and \( \lambda \), respectively. The hash of the \( j \)th packet from the \( i \)th page, \( h_1(p_{i,j}) \), is appended to the \( j \)th packet of the \( i-1 \)th page, \( p_{i-1,j} \). Moreover, the hash of the \( i \)th page, \( h_2(P_i) \), is appended to the previous page \( P_{i-1} \). This is done for all pages in the reverse order, namely \( P_n, \ldots, P_1 \). At
last, the hash tree in page $P_0$ is created. First, to authenticate the remaining page $P_1$ the needed hash values are computed. Those values constitute the leaves of the hash tree to be formed. The hash values of the hash tree are then partitioned into packets. The hash tree is formed such that a hash value at the $i$th level contains hash values of the packets of the hash values at level $i+1$. Hash values of a new level are computed and are again partitioned into packets.

**Signing the code image** - Let ‘CI header, $p_{root}, h_2(P_1)$’ be the message that is signed, where $p_{root}$ is the root of the hash tree, $h_2(P_1)$ the hash value of page $P_1$, and ‘CI Header’ is some header information that contains e.g., the version number of the new code image. The BS generates the signature $\sigma_{CI}$ for the message ‘CI header, $p_{root}, h_2(P_1)$’ by using the signing algorithm $\text{Sig}$ and updates its state (see Section 4.2). The message ‘CI header, $p_{root}, h_2(P_1)$’ and the signature are concatenated and distributed into packets.

**Verification** - The code image is started with transmitting page $P_0$ consisting of multiple packets. Page $P_0$ is the security anchor for the authentication. For this page there is no protection against the attacks for fountain codes possible. Thus, it should be transmitted traditionally packet-wise with the option of repeating requests if some packets were not received correctly. The page starts with ‘CI header, $p_{root}, h_2(P_1)$’ followed by the signature $\sigma_{CI}$ on this message. Those data cannot be verified on a per packet level, but only in full. Thus, if the signature verification fails, the full packets that carry the initial message and the signature need to be re-transmitted.

Once the signature verification succeeds, the hash tree root $p_{root}$ can be used to verify the following packets. This process continues until the hash tree leaves, which are the roots for the hash chains to authenticate the code image. Packets at higher tree levels authenticate the packets at lower tree levels. Hence, when packets from lower levels are received earlier than the packets from higher levels authenticating them, they are stored until the reception of authenticating packets. Once the hash tree is verified successfully, the values for the next page packets are set up. The packets of the remaining pages $P_i$ are transmitted with fountain code encoding (see Section 5).

### 4.2 Stateful-verifier $T$-time Signature Scheme

Let $h : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a second pre-image resistant hash function, and $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a one-way permutation. Based on these security primitives, we define the stateful-verifier $T$-time signature scheme. The $T$-time stateful-verifier signature scheme, $(\text{Gen}, \text{Sig}, \text{Vrfy})$, is defined as follows [31]:

**Gen**: On input of a security parameter $n$ and $T \in \mathbb{N}$, proceed as follows:

1. Compute $l$ random secret strings $k_i \leftarrow \{0, 1\}^n$, where $l = n + \lfloor \log(n) \rfloor + 1$;
2. Compute one-way chains $u_i = f^T(k_i)$ for all $1 \leq i \leq l$, where $f^T(k)$ denotes the $T$-fold composition of $f(k)$ with itself;

3. Output the public key $pk = (u_1, \ldots, u_l)$, the private key $sk = (k_1, \ldots, k_l)$, and the initial state for the signer $sts_1 = (T - 1)$, $1 \leq j \leq l$ and for the verifier $sts' = pk$.

Sig: On input of the private key $sk$, the state $sts_i$ as above, and a message $m \in \{0, 1\}^*$, proceed as follows: if $sts_{ij} \geq 0$ for all $1 \leq j \leq l$ then do the following. Otherwise stop (i.e. the private key $sk$ has expired).

1. Compute the $n$-bit hash $h(m)$ of the message $m$;
2. Compute the checksum as a binary string $\phi$ of length $\lceil \log(n) \rceil + 1$; representing the number of zeros in binary representation of $h(m)$;
3. Compute the signature $\sigma_m = (s_1, \ldots, s_l)$ such that $s_q = f^q(k_q)$ if the $q$th bit of $h(m)||\phi$ is 1 for all $1 \leq q \leq l$ and set the new state $sts_{i+1,q} = sts_{i,q} - 1$;
4. Output the signature $\sigma_m$.

Vrfy: On input of the public key $pk$ as above, a message $m \in \{0, 1\}^*$, and a signature $\sigma_m = (s_1, \ldots, s_l)$ proceed as follows:

1. Compute the $n$-bit hash of the message $h(m)$;
2. Compute the checksum $\phi$, representing the number of zeros in the binary representation of $h(m)$;
3. Do the following, iff for every $i$th bit of $h(m)||\phi$, which is 1, the signature contains a value $s_i$ such that $v_i = f^j(s_i)$ for some $j_i < T$.

Otherwise, output 0;
4. Update the state to $sts_{i+1}$ by replacing $v_i$ with $s_i$ for all $v_i = f^j(s_i)$;
5. Set the public key to $pk = sts_{i+1}$ and output 1.

Example  Let us assume that Bob has a message $m$ that Alice agrees to sign. Let $sk = (k_1, \ldots, k_l)$ be Alice’s private key, $pk = (u_1, \ldots, u_l)$ the public key, and $T = 3$. Alice signs the message $m$ with $h(m) = 1011$ as follows\(^1\). She computes the 3-bit (i.e., $\lceil \log(4) \rceil + 1$) checksum $\phi$ representing the number of 0’s in $h(m)$. That is 001. Then, she appends it to $h(m)$. The resulting string is $h'(m) = h(m)||\phi = 1011001$. Since Alice’s initial state is $sts = (2, 2, 2, 2, 2, 2)$, the signature of $m$ is $\sigma_m = (s_1, s_3, s_4, s_7) = (f^2(k_1), f^2(k_3), f^2(k_4), f^2(k_7))$. Alice sends the tuple $(m, \sigma_m)$ to Bob and updates her state to $sts' = (1, 2, 1, 1, 2, 2, 1)$.

On receipt of the message $m$ and its signature, Bob first computes $h'(m) = h(m)||\phi = 1011001$. He accepts the message as authentic, since for each

\(^1\) We assume that $m$ is the first message which Alice signs for Bob.
ith bit of \( h'(m) \) that is 1, the signature \( \sigma_m \) contains a parameter \( s_i \) such that 
\[
f(f^2(k_i)) = f(s_i) = u_i.
\]
Bob changes his public key (his new state \( s't \)) to 
\( (s_1, u_2, s_3, s_4, u_5, u_6, s_7) \). In case of an invalid signature message, Bob drops it
and keeps his old state as it was before the verification process.

4.3 Security Analysis

**Bogus code image protection** - The code image is protected by the \( T \)-time signature that is transmitted in the initial page \( P_0 \). The private key is only known to the trusted base station and cannot be accessed by the adversary. The value signed contains a hash value of the first data page \( P_1 \), and every page contains a hash value of the subsequent page, thus, building a hash chain over the code image. The hash function for the hash chain over the pages \( P_i \) of CI is truncated to \( \lambda \) bit. We assume security of the hash function against a second pre-image attack so that the effort for the attack is in the range of \( 2^\lambda \). Finding a second pre-image is required to break the authentication with a known message attack. The standard notion for the security of signature schemes is existential unforgeability under adaptive chosen-message-attacks.

Simplified, in that definition the adversary may obtain an arbitrary number of signatures on self-chosen messages from an oracle and then has to output a message and a valid signature that was not obtained by the oracle. If the adversary does not succeed in this experiment the signature scheme is secure according to this notion. The given \( T \)-time signature scheme does not offer this high security. For \( T = 1 \) the scheme is mainly identical to Merkle’s one-time signature that offers high security guarantees. In fact, as it is a one-time signature scheme, learning old signatures is of no help as after one use the key is discarded. However, with the \( T \)-time signature scheme, learning signed messages will help the adversary.

We evaluate the security of the signature scheme against a known-message attack. With a known-message-attack, the adversary learns a message signature pair without being able to choose the message. Security against known-message-attacks is clearly weaker than security against chosen-message-attacks, but a justifiable notion for OTA programming as we assume that the adversary has no influence on the code image update being disseminated.

We introduce some notation used in the following. Given a public key 
\( pk = (u_1, \ldots, u_l) \), a private key \( sk = (k_1, \ldots, k_l) \), and a signature message \( \sigma = (s_1, \ldots, s_l) \), we call \( u_i \) the public key parameter, \( k_i \) the private key parameter, and \( s_j = k_i \) the signature parameter.

**Proposition 4.1.** Given a \( T \)-time signature scheme with a \( \lambda \)-bit hash function \( h \). The probability that a signature for a randomly selected message can be forged after having obtained \( t \) signatures for uniformly at random chosen messages is \( \leq \frac{s^t}{2^\lambda} \), where \( s \leq \lambda \) denotes the number of distinct signature
parameters (i.e., private key parameters) disclosed with those \( t \) signatures excluding the checksum part\(^2\).

Given the \( T \)-time signature scheme with an \( \lambda \)-bit hash function \( h \). Considered only the \( h(m) \) part of the signatures, after receiving \( t \) signatures, the adversary knows on average \( \lambda - \frac{\lambda}{2t} \) distinct private key parameters.

Due to space restrictions we do not provide the proof here and refer to [31]. From Proposition 4.1 we can see that our scheme provides strong security of \( \lambda \) bit if at most 1 signature is known to the adversary. This reduces to \( \frac{\lambda}{2t} \)-bit security when the strong adversary obtains \( t > 1 \) signatures.

We conclude that the security offered by the \( T \)-time signature scheme is high enough for the application of OTA programming. The adversary can trivially obtain 1 message signature pair while it is in transmission, but we argue that it is unlikely that a second one is obtained. OTA programming will occur infrequently. Furthermore, it is plausible that it will be detected if some nodes failed to update the code and therefore did not update their public key. In such a situation it can be assumed that all nodes have received code image \( i \) before dissemination of code image \( i + 1 \) starts. Thus, all nodes will update their public key and the adversary cannot obtain multiple valid known message signature pairs for a public key that is held by one node. For 1 known signature, the \( T \)-time signature scheme offers strong security. The signature scheme fails gracefully; for the situation that some nodes fail to receive code update \( i \) when code update \( i + 1 \) is in transmission, the scheme still offers a certain weaker security.

**DoS resistance** - The packet-wise hash values are for DoS resistance which allows to reduce the length of the hash values to \( \kappa < \lambda \) bit. A lower security level is acceptable as a DoS attack is an online attack, i.e. it has to be executed while the programming is ongoing, in contrast to the bogus code image attacks that can be prepared in the interval between two code disseminations. The DoS protection is highly relevant for the encoding with a fountain code and is in detail analyzed in Section 5.

5 ON-THE-FLY VERIFIABLE FOUNTAIN CODES

5.1 Fountain Codes

For the efficient and robust page-wise CI propagation, we use rateless erasure codes, or fountain codes [22]. The idea of a fountain code is that a sender is able to generate a potentially unlimited sequence of code words while allowing the receiver to decode the source data from any subset of encoded packets equal or slightly larger than the number of source packets. Fountain codes

\(^2\)If \( \lambda = 2^m - 1 \), the checksum bits are uniformly distributed and the signature including the checksum part can be treated as a hash function that outputs values of length \( \lambda + \lfloor \log \lambda \rfloor + 1 \).
offer the advantage that they do not require the receivers to inform the sender about individual missing packets. If packets get lost due to the unreliable medium, it is not required at the sender side to resubmit exactly the missing packet. Instead, due to the fountain characteristic, any new encoded packet $X_{j+k}$ can be transmitted by the sender. Taking that into account and assuming that all parties finally would like to receive the same CI, fountain codes allow using the wireless broadcast channel more efficiently in an OTA programming scenario. The same encoded packet may, based on their potential different pre-existing knowledge about previously successfully received encoded packets, allow different receivers to filter out different information relevant to them. The first efficient realizations of fountain codes are Luby Transform (LT) Codes [21]. After splitting a data block, or CI, into $m$ packets, an encoded packet is computed in two steps.

1. A packet degree $d$ is randomly chosen according to a given distribution. The choice of the weight distribution is the key parameter with respect to the performance and the efficiency of the coding scheme $\rho(d)$.

2. The encoded packet is obtained by choosing $d$ out of the $m$ source packets at random from a uniform distribution, namely $\{p_{i_1}, \ldots, p_{i_d}\}$, and successively XORing them to compute

$$X = \bigoplus_{i=1}^{d} p_{i_i}. \quad (1)$$

This is done for at least $N > m$ encoded packets $X$. The information which packets $p_{i_i}$ have been considered for a concrete encoded packet $X_j$ is represented in a coding vector $C_j$ of size $1 \times m$. The degree of the encoded packet $X_j$ equals the Hamming weight of its coefficient vector. We denote the weight of a coefficient vector $C$ with $D(C)$ and its $i$-th bit with $C[i]$. The receiver decoding procedure is equivalent to solving the linear equation system $t = Gs$ for $s$. The $m \times m$ matrix $G$ consists of $m$ linear independent coefficient vectors of successfully received packets, whereas the vector $t$ contains the corresponding incoming encoded packets $X$. The vector $s$ is the vector of all $m$ plaintext packets which shall be computed. Solving the linear equation system requiring in the standard way roughly $m^3$ computation steps is not very efficient. The decoding effort can be reduced by choosing a sub-optimal decoding process, the so called LT decoding process, and an adapted degree distribution $\rho(d)$.

LT decoding process - The LT decoding process is depicted in Figure 1. It uses two buffers $A$ and $B$, in which encoded and decoded packets are stored, respectively. Encoded packets $X, C$ received over the radio interface are decoded in accordance to the information in the coding vector $C$. Those plaintext packets $p_i$ which are already stored in the buffer $B$, and which are relevant according to the currently processed packet $(X, C)$, will be
applied to the actual decoding. The processing is illustrated in the Algorithm 1 Decode.

**Algorithm 1 Decode.**

**Require:** $X, C, p_i \in B$

**Ensure:** $X', C'$

1: **for all** $p_i$ in $B$ **do**
2:   **if** $C[i] = 1$ **then**
3:     $X' = X \oplus p_i$
4:   **end if**
5: **end for**
6: $C' = C$

If after passing the algorithm **Decode** the remaining encoded packet $(X', C')$ is of degree $D(C') = 1$, it is actually a source packet $p$, and is added to the list of decoded packets in buffer $B$. If the degree remains larger than one ($D(C') > 1$), it is inserted in the buffer $A$. The buffer $B$ stores all the actually decoded plaintext packets $p_i$. If a new element is added to $B$, all $(X, C)$ in buffer $A$ will again be applied.

Since the decoding is mainly based on XOR operations, the LT decoding process is extremely efficient. However, the limiting factors on a sensor node are the data overhead and the buffer sizes for buffers $A$ and $B$.

The efficiency of LT Codes depends largely on the degree distribution of the encoded packets. Due to the decoding, a high number of packets with low weight needs to be present, i.e. the decoding process cannot start before a packet with weight 1 is received. On the other hand, the redundancy should
be minimized such that a set of slightly more than \( m \) packets is enough to fully decode the information.

5.2 Network Coding and Security Challenges
While in the single-hop OTA programming scenario, all packets are encoded at the base station, in a multi-hop programming scenario, intermediate sensor nodes have to encode packets themselves. Pure forwarding of encodings from the base station would use the benefits of fountain codes only at the first hop. Encoding of information inside the WSN is a form of network coding causing new threats by poisoning attacks thus requiring new security solutions.

An attack to network coding systems is the so-called poisoning attack, a special form of a DoS attack where modification of a single bogus packet can stop the whole programming process. This is caused by the transmission of encoded packets that for themselves do not carry usable data. Only if enough well-formed packets are received, the decoding process can be started to recover the code image. By modifying only a few blocks, a limited adversary now has the power to prevent the receiver from decoding any block. Even worse, if the receiver is a node that performs network coding, i.e., combining the malicious packet with well-formed packets, the node produces new malicious packets that spread the error throughout the network. In the end, this may prevent successful decoding for all receivers.

5.3 Security Enhanced Fountain Codes
Our security enhancements for LT codes for OTA programming ensure:

1. a bogus code image will always be detected before loading it into the program memory;
2. on a per packet basis nearly on-the-fly verification of incoming data is possible;
3. a filter criterion ensures buffering of only those incoming encoded packets, which are likely to be decoded in one of the next decoding rounds.

To clarify our security enhancement 2, we define the term nearly on-the-fly verification. We term on-the-fly verification that a packet can be verified on reception, and does not need to be stored in buffer \( A \). A packet \((X, C)\) that circulates between the process Decode and buffer \( A \) is called to be in the state atomic decoding. We call a packet to be nearly on-the-fly verified if it can be verified after it was for a certain time in the state atomic decoding, but can still be autonomously verified.

Weaving security into LT codes
The basic idea for the security enhancements is to authenticate the source packets, as only for those an end-to-end relationship between the base station
and all sensor nodes exist. As we argued, this is not possible in general, but it is possible if encoded packets can be verified individually. The optimal situation would be for the receiver to be able to verify each incoming packet on-the-fly. This is possible with homomorphic signatures, albeit not feasible in WSNs. The use of LT codes, however, together with our security extensions, allows a nearly on-the-fly verification and makes it therefore possible to authenticate the source packets. Only verified information is used to decode a yet unverified encoded packet. If the decoding fails, the error can be localized to be in exactly the currently decoded packet.

If all but one packets corresponding to the ‘1’-positions of the corresponding coefficient vector $C_i$ have already been successfully verified, the actual packet $(X_i, C_i)$ can be verified on-the-fly. Otherwise the packet has to be buffered in $A$ until this precondition is fulfilled.

The weight distribution of LT codes [21] assumes a high number of packets, say $m > 10000$. Thus, it is no surprise that this distribution is not energy optimal for pages of size $m = 32$ or $m = 64$. Namely, the transmission overhead is relatively high, while the saved complexity in decoding is relatively low. The optimal distribution in terms of data overhead for a small number of packets is identified in [12, 27].

Remark: Rossi et al. [27] propose an alternative decoding algorithm by Gaussian elimination with a high probability of packets with a high degree. This contradicts however, our strategy of on-the-fly, or, better, nearly on-the-fly verification. With Gaussian elimination, the feedback gives only the information, that one of the lines that were used to extract one packet was faulty. To identify the packet, e.g. by decoding attempts with different subsets of packets, is computationally extremely expensive.

Security enhanced encoding
The encoding is performed as described in Section 4.1.

Security enhanced decoding
On the receiver side, we enrich the decoding process of LT codes with i) an initial $T$-time signature verification, ii) nearly on-the-fly packet verification, and iii) intelligent packet filtering in case of buffer overflow at $A$. Since topic i) has already been described in Section 4, we continue with ii) and iii).

Decoding of the fountain code - The security enhanced LT decoding process is depicted in Figure 2. The security enhancements for an incoming packet $(X, C)$ with degree $1 \leq D(C) \leq K$ are as follows: When leaving the status atomic decoding and before a decoded packet $X'$ is eventually written into the buffer $B$, the $j$-th hash value $h_1(p_{ij})$ of the relevant hash list for $P_i$ is compared with the hash value computed by hashing $h_1(X')$ to check whether

$$h_1(p_{ij}) \overset{?}{=} h_1(X') \quad (2)$$
is true. This verification is done for each completely atomically decoded packet $X', C'$ with actual degree $D(C') = 1$ and $C'[j] = 1$. Our nearly on-the-fly verification ensures with a reasonable high probability that no bogus packet is stored in the buffer $B$ with the consequence that the remaining decoding process can be performed purely on correctly decoded packets $p_{ij}$.

### 5.4 Evaluation of Fountain Codes

The functionality of the encoder and decoder has been implemented with Python in the Sage mathematics environment [29]. The simulations were done by averaging from 10000 executions with the parameters described below.

**Overhead and performance**

In this section, we analyze the overhead of fountain encoded data transmission in terms of required buffer size at the receiving node and the data overhead on a binary symmetric channel. We consider two cases, a code image with 32 packets per page and one with 64 packets per page. The optimal probability distribution for those cases as well as the expected number of packets that is required to decode successfully was investigated in [12, 27] and is shown in Table 1. It shows that with fountain codes, 43.8 and 82.8 packets need to be received on average instead of 32 and 64, respectively, in unencoded transmission. The overhead caused by this is 36.9% in case of 32 source packets and 29.4% in case of 64 source packets. In terms of efficiency ratio, this means 1.369 and 1.294, respectively. An ideal code would be able to decode the source data with any 32 or 64 packets out of the data stream. For a fixed rate such a code is called Maximum Distance Separable (MDS) code. An example for such a code is a Reed-Solomon-Code. However, those code are
TABLE 1
Optimal degree distribution for \( m = 32, 64 \) packets.

<table>
<thead>
<tr>
<th>( D(C) )</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 )</td>
<td>0.212</td>
<td>0.161</td>
</tr>
<tr>
<td>( 2 )</td>
<td>0.351</td>
<td>0.400</td>
</tr>
<tr>
<td>( 4 )</td>
<td>0.288</td>
<td>0.256</td>
</tr>
<tr>
<td>( 8 )</td>
<td>0.101</td>
<td>0.101</td>
</tr>
<tr>
<td>( 16 )</td>
<td>0.048</td>
<td>0.045</td>
</tr>
<tr>
<td>( 32 )</td>
<td>—</td>
<td>0.037</td>
</tr>
</tbody>
</table>

packets needed (avg) 43.8 82.8
standard deviation 6.8 9.3

in practice only decodable at very high data rates where they do not have the fountain property than virtually unlimited packets can be constructed, i.e. in the case of a practical Reed-Solomon-Code only a slightly higher number of packets than needed can be constructed. On restricted devices, the decoding process can generally be considered too expensive.

The analyses done previously assume that arbitrary buffer space is available at the receiving node. While it is clear that the buffer \( B \) of decoded packets has a maximum size of \( m \) packets, the actually required size for the buffer \( A \) is not obvious. We assume that buffers \( A \) and \( B \) share the same memory, which makes sense as with progressing decoding of a page buffer \( A \) will empty while buffer \( B \) fills up. Concerning the total buffer sizes of buffers \( A \) and \( B \), we simulated random decoding processes and sketched the maximum total size of buffers \( A \) and \( B \) for \( m = 32 \) and \( m = 64 \) packets in Figure 3 and Figure 4. Luckily, the total size of \( A \) and \( B \) is not significantly higher than the minimum required size of \( m \) packets. This is due to the fact that buffer \( A \) is needed in the beginning when many packets cannot be successfully decoded, while once buffer \( B \) contains enough packets, many incoming packets will be on-the-fly decodable and do not need to be stored in buffer \( A \). Concretely, we suggest a buffer size for \( A \) and \( B \) together of 44 packets for \( m = 32 \) packets per page and of 83 packets for \( m = 64 \) packets. This buffer size is sufficient in 95% of all transmissions. In the remaining cases, packets have to be dropped in the decoder and the number of packets that need to be sent increases.

Concerning the data transmission overhead that is introduced by fountain codes compared to non-encoded transmission, we analyze the number of encoded packets that are necessary when using fountain codes in the absence of an attack dependent on the packet loss rate and compare it to the number of packets required with a broadcast transmission and selective repeat requests. We assume a binary symmetric channel with a packet loss probability \( p \). This
channel is the ideal situation for fountain codes, as it assumes that packet failures are independent between any pair of nodes. Results with a more realistic network are provided in section 7. Let $X$ denote the random variable that describes the number of transmissions needed until the packet is successfully
received. The probability that one packet needs to be transmitted more than \( k \) times until it is received by one receiver is \( \Pr(X > k) = p^k \). The maximum for \( n \) receivers is therefore \( \Pr(\max(X_1, \ldots, X_n) > k) = 1 - (1 - p^k)^n \) and the expected duration for one packet is

\[
E_{\max}(X_1, \ldots, X_n) = \sum_{k \geq 0} \left( k \cdot (1 - (1 - p^{k-1})^n - 1 + (1 - p^k)^n) \right)
\]

\[
= \sum_{k \geq 0} \left( k \cdot ((1 - p^k)^n - (1 - p^{k-1})^n) \right)
\]

\[
= \sum_{k \geq 0} \left( 1 - (1 - p^k)^n \right).
\]

As in non-encoded transmission, every packet has to be received, and this value has to be multiplied by \( m \) to obtain the expected number of needed packets to receive one page consisting of \( m \) packets. For fountain encoded transmission any \( m \) packets can be received. The required number of packets follows the maximum of negative binomial distributed random variables. The probability that one receiver needs to have more than \( k \) packets transmitted until \( m \) packets are received is \( \Pr(X > k) = \sum_{j=0}^{N-1} \binom{k-1}{j} (1-p)^j \cdot (1 - p)^{k-j} \). In the same way as above, the expected maximum duration for \( n \) receivers is then

\[
E_{\max}(X_1, \ldots, X_n) = \sum_{k \geq 0} \left( 1 - (1 - \Pr(X > k))^n \right). \tag{3}
\]

The diagram in Figure 5 compares fountain codes with unencoded transmission as e.g. in Deluge for the case of 64 data packets per page. The required packets are shown dependent on the packet-loss probability for 4, 8, 16 and 100 receivers that are in the transmission range of a sender. All receivers are assumed to have a mutual independent packet-loss rate of \( p \). The scenarios with 4, 8, and 16 neighbors are well motivated in a WSN. The scenario with 100 receivers might apply for a single-hop scenario from a powerful base station. The simulation takes into account that only 64 packets are needed for automatic repeat request data transmission, while on average 83 fountain encoded packets are needed. This shows the fountain code approach is more efficient than a selective re-transmission as used in Deluge in a broadcast medium. For example assuming 8 receivers under the above introduced setting the break-even point for a fountain code based transmission compared to a purely Deluge based one is at a packet loss probability of approximately 0.075. Another advantage of fountain codes is that at the receiver’s side, even the packets from multiple senders complement one another. Thus if all receivers are in the range of multiple senders, the number of packets per sender can be considerably reduced. More details on this are provided in Section 6.
**DoS protection** - For the protection against DoS attacks, we solely consider the introduced limited adversary. With this adversary, packets arrive unmodified at the receiver side with a probability of $0 < \gamma \leq 1$ during a given time window. The adversary has the possibility to modify packets or to drop packets in transmission. Assume the adversary modifies an encoded packet $(X, C)$ to $(\hat{X}, \hat{C})$. This will delay the decoding process by inserting bogus packets that will block the buffer until they are discarded. As we decode packets only using verified information, the bogus packet will not disrupt decoding of correct packets. Thus, the most harmful attack is to block the limited buffer space, e.g. by modifying the packet to have a high degree vector $\hat{C}$. To mitigate the effect of this attack, we implemented a filter algorithm that chooses which packet to drop in case the buffer space is exhausted [1].

The other option an attacker has is to drop packets. As long as they are randomly performed by the attacker, at the receiver side packet dropping attacks perform equivalent to a lossy wireless channel and are therefore handled by the LT coding scheme itself as analyzed in Figure 5. More serious threats to OTA programming are selective dropping attacks in which the attacker drops exactly all those packets, which are required to start or progress the decoding process at the receiver side. The fountain code offers protection also for that case, because there is not one packet which is explicitly needed, but almost all are helpful for decoding. However, against selective jamming at a high rate, no protection on protocol level is possible.
6 MULTI-HOP PROPAGATION WITH FUZZY CONTROL

To minimize the energy/power consumption, the dissemination of the code image should be achieved in a minimum of time. The design of Deluge [10] was also tailored to achieve this objective, as the authors observed that the energy consumption grows largely proportional with the required time for code update completion. Because OTA programming generates a high traffic on the wireless, the time between two packets is too short to set the radio to sleeping mode and the radio stays in listening mode for the whole programming operation. For a meaningful evaluation of the overall energy/power consumption, other costs such as computation overhead for coding should be considered. However, the LT decoder relies on very simple operations such as XOR and bit operations, and thus we argue it has little impact on the energy budget.

The approach taken here, in contrast to most of the related work, does not adopt any techniques that limit the number of senders in a local area in order avoid packet loss through collision. The rateless property of the fountain codes allows that any nodes that possess the requested page can transmit it concurrently without the need to coordinate with other senders. With the current sensor radio technology, sending at full power costs about as much as listening, hence having more senders does not cost additional energy. This of course increases the risk for the hidden terminal problem. If remained uncontrolled without a proper mechanism, this can severely deteriorate the performance of the dissemination protocol.

To decrease the number of packet collisions and mitigate the hidden terminal problem, we apply means from fuzzy logic to dynamically adapt the transmission rates of sending nodes in accordance with the local congestion of the radio channel. It is worth noting that reducing the number of packet losses alleviates the increased overhead from the security enhancements. Fuzzy control theory was recognized as a promising approach to control the level of channel utilization as it is well suited for resource constrained sensors. The conceptual design of the applied fuzzy control system closely follows the work from Kahlert and Frank [13]. We designed a fuzzy controller that throttles the transmission rate of a node based on the number of NACK and data packets overheard, scaled with the dynamic number of neighboring nodes [23].

**Protocol State Machines** - Before the OTA programming, the BS broadcasts the summary of the code image, which it wants to propagate. Nodes that do not possess the code image get into a receiving state, whereas the other nodes start disseminating the code image. The programming of a deployed node is realized in two steps. Firstly, the dedicated page $P_0$ is received containing the signature of the root packet of the hash tree (see Section 4). After verifying the signature, the subsequent pages can be received and authenticated using security enhanced fountain codes.
The dissemination of a page $P_i$, $i \geq 1$ is shown in Figure 6. At each point in time, a deployed node can either be a sender or a receiver, but not both at the same time. In state $P_{iR}$ nodes request data from neighboring nodes. Requests are actually NACK messages in our protocol. The reception of an encoded packet triggers the transition into the $Decode$ state, in which the security enhanced sub-optimal decoding is applied. After the complete page was successfully received and decoded, it needs to be authenticated, which is done in the $P_{iAT}$ state. Successful verification triggers both a copy process of the page into the flash memory and a transition into the $P_{iS}$ state. The state $P_{iS}$ allows deployed nodes to act as senders on the next hop. To satisfy its neighbors demand, the node executes the following steps: (i) transition into the $Fuzzy$ state, (ii) invocation of the fuzzy controller, and (iii) broadcasting of encoded packets. After some waiting period, during which no further requests are received, a sending node requests the next page ($P_{iR}$). It should be noted that requests for lower numbered pages are always answered immediately, which is essential for sustaining reliability of the reprogramming protocol. This propagation results in page-wise spatial multiplexing, as exhibited in Figure 7.
Simulation Environment and Protocol Fine-Tuning - In order to evaluate the propagation scheme and for fine-tuning its parameters, a comprehensive simulation was performed with the discrete event simulator TOSSIM [18].

The simulator has the advantages to run with the same TinyOS [30] code that runs on the mote and scales to hundreds of nodes. The communication model of the simulator supports a path loss model and asymmetric channel gains; it also supports interferences generated by simultaneous communications in the WSN. Additionally, Lee et al. demonstrated that using a signal-to-noise ratio (SNR) curve with a closest-fit pattern matching (CPM) noise model increases the accuracy of wireless packet delivery simulation [17]. This is the reason why in TOSSIM environmental noise is modeled with CPM. CPM can capture bursts of interference and other correlated phenomena. It generates statistical models from noise traces, so that the simulator reflects real world noise environments. The tuning simulations were based on a noise trace that was sampled during a demo session at the fourth TinyOS Technology Exchange, which is characterized by a relatively low noise floor.

The main weaknesses of the simulator are the lack of a fading model, the fact that packet length is not taken into account for packet losses other than those caused by collisions, and finally that the external Flash is not supported. To cope with the last issue, we created a simple file system, which, however, neglects timing or resource issues that happen on the real node.

Table 2 lists the default parameters used for our simulations. They were used as input for a Java TinyOS tool that generates link gains between pair of nodes. We opted for a grid topology, with the base station in one of its corners. We set distance, signal decay exponent, and signal power such that on average nodes had only a few neighbors with a good channel quality.

Among other parameters, an optimal NACK interval was identified. It was in our empirical study the parameter that had the most impact on the protocol performance. A NACK interval defines the minimum waiting time
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path loss exponent</td>
<td>3.0</td>
<td>Exponent for the signal decay in relation with distance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exponent for free-space path loss is 2</td>
</tr>
<tr>
<td>Signal power at 1 meter</td>
<td>-52 dBm</td>
<td>Set to achieve a dozen of neighbors per node</td>
</tr>
<tr>
<td>Grid distance</td>
<td>10 m</td>
<td>Typical line of sight range is 50 m</td>
</tr>
<tr>
<td># nodes</td>
<td>225</td>
<td>A rather large WSN</td>
</tr>
<tr>
<td>MAC payload size</td>
<td>28 B</td>
<td>TinyOS default</td>
</tr>
<tr>
<td>Page size</td>
<td>32 packets</td>
<td>Encoding according to Table 1</td>
</tr>
<tr>
<td># simulations</td>
<td>10</td>
<td>Number of simulations for each data point</td>
</tr>
</tbody>
</table>

TABLE 2
TOSSIM default topology and channel parameters to simulate an office indoor environment.

before sending a NACK or retransmitting one. The NACK interval was varied between 30 and 150 milliseconds. The corresponding results are shown in Figure 8. We counted packets collisions, which are the packets that the simulator should have delivered, but which are finally dropped because of a

FIGURE 8
Effects of varying the NACK interval length on the completion time, number of page loads, number of packet collisions, and number of packets sent. The error bars indicate standard deviations.
simultaneous transmission with too high power. Since our protocol is based on broadcasting, one packet can be lost due to collisions at multiple receivers, hence increasing multiple times the count. We also took into account the number of page loads, which counts how many times in average a node needs to load a previously stored page to serve a node that is lagging behind. This has considerable impact on the performance, as the current buffer must be dumped to load the requested page.

7 IMPLEMENTATION RESULTS

We implemented the proposed OTA programming solutions in nesC for tinyOS 2.x [30]. SHA-1 was selected as the main cryptographic building block for all security components including the realization of the one-way function required in the $T$-Time signature verifications.

Simulations Evaluation - To compare the outlined reprogramming approach with the current de-facto standard Deluge, simulations were run for different network sizes. Note that the losses generated by the simulator due to noise or interference are not independent from each other, hence fountain coding is not advantaged by the simulator. Figure 9 shows the simulation results corresponding to the dissemination of a 11kB code image. An important finding was that our reprogramming protocol performed substantially faster than its competitor Deluge, regardless of whether fuzzy control was applied or not. It should be noted that the faster performance allows nodes

FIGURE 9
Dissemination of a 11kB code image. The simulations were run in differently-sized grid networks. The error bars indicate standard deviations.

---

3 SHA-1 provides the pre-image resilience required by the one-way function $f$. We evaluated this implementation both in TOSSIM and on a small testbed.
to switch off their radios sooner, which reduces energy consumption significantly [10]. We observe that increasing the page size improves the LT codes performance at the expense of RAM resource. The application of fuzzy control effectively decreased the number of packet collisions and the total number of packets sent. On the one hand, the fuzzy control approach mitigated the hidden terminal problem. On the other hand, it resulted in slightly slower performance. Hence, fuzzy control is more valuable in scenarios where the transmitting power is much higher than the listening energy consumption, or where an application is still needed to run in the background.

**Testbed Results** - We ran our multi-hop dissemination protocol, without the use of fuzzy control or security in a small testbed consisting of 10 TelosB nodes in an office environment as depicted in Figure 10. We believe that comparing a secure Deluge such as Seluge with our secure solution would yield about the same network performance results, as the security overhead are similar. Our testbed offered characteristic fading aspects due to metallic structure and objects. We disseminated a 41kB code image to 9 nodes, and measured how much time elapsed between the first packet received at the first node, and the reception of the last code image packet at the last node. Due to the spatial multiplexing of both protocols, the difference of completion time between the first and last node is not high, at most a few seconds in the measurements we made. We used standard TinyOS settings (e.g. payload size of 28 bytes). The transmission power was varied to force multi-hop transmission. For each set of parameters, we made 4 measurements. We do not want to draw strong conclusions from this testbed due to its limitations, but want to show that our solution performs well in a real environment.

In Table 3 we show the results from the testbed. First, we observe that for transmitting to only one node with a good link, the fountain code overhead is decisive and our solution is obviously less advantageous. Since more packets need to be sent to transmit successfully a complete page due to the fountain

![FIGURE 10](image)

*Node positions for the office environment testbed. The dimensions of the place are about 25x14 meters.*
TABLE 3
Code update completion time [s].

<table>
<thead>
<tr>
<th>Setting</th>
<th>Deluge (Avg., [Min., Max.])</th>
<th>Our solution (Avg., [Min., Max.])</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 dBm, 1 node</td>
<td>69 [68, 71]</td>
<td>99 [96, 104]</td>
</tr>
<tr>
<td>0 dBm</td>
<td>102 [88, 122]</td>
<td>112 [107, 121]</td>
</tr>
<tr>
<td>-10 dBm</td>
<td>146 [139, 151]</td>
<td>129 [118, 139]</td>
</tr>
<tr>
<td>-20 dBm</td>
<td>245 [226, 262]</td>
<td>179 [169, 186]</td>
</tr>
</tbody>
</table>

encoding our solution is about 43% slower for such a scenario. The testbed also confirmed that Deluge prevails in one-hop scenarios. Indeed, when the BS sends with full power (0 dBm), the network is single-hop due to the limited size of our testbed, although nodes 8 and 9 do not share a good link with the BS. With lesser channel quality, and higher multi-hop network characteristic, our solution becomes much more competitive, and outclasses Deluge as can be seen with -10 dBm and -20 dBm transmission power. As confirmed by our simulations, our transmission technique becomes advantageous in large or/and lossy networks.

8 CONCLUSIONS

We provided and analyzed protocols for a dependable OTA programming in multi-hop wireless sensor networks consisting of restricted devices. By using $T$-time signatures and by combining it with a hash tree and a hash chain, we provide an efficient and footprint optimized security for the code image authentication. We enhanced the dependability of the code image propagation process by applying fountain codes with a packet degree distribution that allows nearly on-the-fly decoding and verification. In addition and in particular for dense multi-hop WSNs, fuzzy control has been applied to ensure that intermediate nodes forward only a limited but relevant degree of encoded data. Our results are promising for various real-world OTA programming settings.

9 ACKNOWLEDGEMENT

The work of authors from NEC presented in this paper was partly funded by the European Commission within the STREP WSAN4CIP project. The work of authors from HAW is partly funded by the German BMBF project SKIMS. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsement of the WSAN4CIP project or the SKIMS project or the European Commission or the BMBF.
REFERENCES


