Tsimbalo, E., Fafoutis, X., & Piechocki, R. J. (2017). CRC Error Correction in IoT Applications. *IEEE Transactions on Industrial Informatics*, *13*(1), 361-369. https://doi.org/10.1109/TII.2016.2605628

CRC Error Correction in IoT Applications

Evgeny Tsimbalo, *Student Member, IEEE*, Xenofon Fafoutis, *Member, IEEE*, Robert J. Piechocki, *Member, IEEE*

***Abstract*—In this paper, error correction is introduced to the Bluetooth Low Energy (BLE) and IEEE 802.15.4 standards by utilising data redundancy provided by Cyclic Redundancy Check (CRC) codes used by both protocols to detect erroneous packets. A scenario with an energy-constrained transmitter and a constraint-free infrastructure is assumed which enables additional signal processing at the receiving side, keeping the transmitter intact. CRC error correction is achieved using a novel approach of applying iterative decoding techniques. The proposed methods are evaluated based both on simulated and real packets. It is shown that by enabling CRC error correction, up to 2.5 dB of the SNR gain can be achieved, while up to 35% of real corrupted packets can be corrected, at no extra cost for the transmitter. This results in potential range extension and longer battery life caused by a reduced number of retransmissions.**

***Keywords*-CRC; error correction; Bluetooth Low Energy; IEEE 802.15.4; short-range IoT**

I. Introduction

Internet-of-Things (IoT) emerging technologies enable multiple applications that aim at improving the quality of life of citizens of smart cities [1]. Energy-constrained communication is a fundamental challenge for the realisation of the IoT-enabling technologies. Bluetooth Low Energy (BLE) and IEEE 802.15.4 are two widely employed wireless standards in energy-constrained short-range IoT applications. Combined with long-range communication technologies, such as those proposed by Sigfox and the LoRa Alliance [2], BLE and IEEE 802.15.4 offer a full spectrum of wireless connectivity options for the IoT.

BLE is part of the Bluetooth 4 standard [3] that is aimed at very low power applications. As one of the *de facto* wireless communication options in modern smart phones, BLE has become a common choice for many manufacturers of commercial wearable gadgets. BLE is also the basis of the iBeacon technology [4] which enables proximity-based services and applications, such as indoor positioning. IEEE 802.15.4 [5] is a wireless standard that defines the Physical (PHY) and Medium Access Control (MAC) layers of the communications stack. It is the basis of several higher layer protocol suites such as Zigbee and 6LoWPAN that are widely used in various IoT applications, including smart grids [6]. It is also the basis of the recently-announced Thread [7] protocol suite that is targeting the integration of smart home applications with cloud-based services. Both BLE and IEEE 802.15.4 are considered as two of the main enabling wireless technologies for smart cities [1]. In BLE and IEEE 802.15.4, particular attention is paid to the energy efficiency of the transmitter. As a result, many traditional communication techniques, such as forward error correction, are disabled to avoid any additional energy spent at the transmitter. At the same time, to ensure data integrity, both standards employ Cyclic Redundancy Check (CRC) codes for error detection. To this end, prior to transmission each packet is encoded such that a number of redundant bits is generated and appended to the packet. CRC encoding consumes minimum amount of energy and does not compromise the energy efficiency of the transmitter. At the receiver side, a packet is forwarded to the upper layers of the stack only if it passes a CRC check, i.e., it does not have any bit errors. Otherwise, the packet is considered corrupted and is discarded.

In the application layer, packets discarded by the CRC check are experienced as performance degradation, whose nature depends on the mechanics of the MAC layer used. More specifically, both BLE and IEEE 802.15.4 can operate in either a unidirectional broadcasting mode or a connection-oriented acknowledge-based mode. In the former case, packet loss imposes limits on the operational range. In the latter case, discarded packets cause retransmissions resulting in additional energy consumption of the transmitter.

While CRC codes are traditionally used for error detection only, they have an inherent error correction potential due to redundancy they introduce to transmitted data. If some additional signal processing was added at the receiver to utilise this redundancy and actually correct some errors, it would reduce the packet loss experienced by the upper layers. When broadcasting, this could extend the operational range or decrease the required transmit power. In the connection-oriented mode, the number of retransmissions would be reduced. In both cases, the energy efficiency can be improved at no extra cost for the transmitter. In other words, CRC error correction is a compelling choice for applications where an energy-constrained transmitter communicates with a constraint-free receiver, such as wearable sensors streaming data to a smart infrastructure.

In this paper, the error correction potential of the CRC codes employed in BLE and IEEE 802.15.4 is investigated in a real environment. Using the ideas proposed in [8][9] for BLE only, two iterative decoding techniques are applied to correct errors in both standards. These ideas are further investigated by considering how the frame size, and thus the level of redundancy, influences the error correction potential. For benchmarking, a simple look-up algorithm that is able to correct corrupted packets with no more than a single bit error is implemented. By comparing the techniques, different situations are identified where one of the techniques is more beneficial than the others.

This work was performed under the SPHERE IRC funded by the UK Engineering and Physical Sciences Research Council (EPSRC), Grant EP/K031910/1.

E. Tsimbalo, X. Fafoutis and R. J. Piechocki are with the Department of Electrical and Electronic Engineering, University of Bristol, Bristol, BS8 1UB, U.K. (e-mail: e.tsimbalo@bristol.ac.uk, xenofon.fafoutis@bristol.ac.uk, r.j.piechocki@bristol.ac.uk).

The proposed error correction algorithms are evaluated via simulation and using datasets of real corrupted data acquired using a commercial off-the-shelf radio that supports both BLE and IEEE 802.15.4.

The remainder of the paper is organised as follows. In Section II, a brief overview of the background and related work is provided. Following that, in Section III basic concepts behind CRC codes are introduced. In Section IV, the description of two proposed error correction methods is given. In Section V, the correction potential of the aforementioned algorithms is evaluated through simulation. In Section VI, the performance is further investigated in a practical scenario, both in terms of correction potential and complexity, based on datasets of real corrupted packets. Finally, Section VII concludes the paper.

II. Background and Related Work

Some error correction techniques for CRC codes have been proposed over the years. A simple look-up algorithm correcting all single errors was described by the inventors of CRC codes in [10]. This algorithm is based on the fact that each possible single-error position in a packet gives a unique remainder after polynomial division of the polynomial corresponding to the packet by the generator polynomial, regardless of the actual bits transmitted in the packet. As a result, a look-up table of all possible remainders and the corresponding error bit positions can be calculated in advance, using packets with one in error positions and zeros elsewhere. When an erroneous packet with a single bit error arrives, the corresponding remainder is calculated and checked against the loop-up table, thus the error position is identified. This method can correct 100% of all single-error packets.

More sophisticated look-up techniques correcting some of double-error codewords were also developed [11]. However, all these techniques aim at correcting a particular number of errors for certain packet sizes and CRC codes. To the best of knowledge of the authors, no unified approach has been proposed that can be applied to any CRC code and packet size to correct an arbitrary number of errors limited only by the error-correction capabilities of the code itself.

Many state-of-the-art error correction codes employ iterative decoding algorithms. One of those algorithms, known as Belief Propagation (BP), was originally developed for Low Density Parity Check (LDPC) codes [12], a special type of linear codes that have a sparse parity check matrix. In general, BP provides good decoding performance when applied to any linear code, as long as the parity check matrix of the code is sparse and the corresponding Tanner graph contains no cycles of length four [13].

In [13] and [14], the authors proposed methods to eliminate four-cycles on the Tanner graph of an arbitrary linear code, demonstrating the results on Hamming and Reed-Solomon codes. The same techniques can be applied to CRC codes, making them suitable for BP-based error correction.

As an alternative to BP, the decoding of a linear code can be viewed as a linear program (LP), the idea that was first introduced in [15]. This approach resulted in an algorithm based on the Alternating Direction Method of Multipliers (ADMM) initially proposed in [16] and applied to LDPC codes in [17]. Practical and computationally simple modifications were further developed in [18], [19] and [20]. While the ADMM-based algorithm has only been investigated in the context of LDPC codes, it can also be applied to correct errors for any linear code, such as the CRC.

III. Preliminaries

In a CRC-based system, each packet of data is encoded by a systematic CRC encoder which adds redundant bits and forms a codeword. This operation is efficiently implemented in hardware using a linear feedback shift register (LFSR) circuit that is defined by the generator polynomial of the code in question. In BLE, the CRC-24 code is employed, with 24 redundant bits being added [3]. Similarly, the CRC-16 code that uses 16 redundant bits is implemented in IEEE 802.15.4 [5]. The generator polynomials of the codes can be expressed as follows:

$$\text{BLE}: \; g(x) = x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1; \quad (1)$$

$$\text{IEEE}\,802.15.4: \; g(x) = x^{16} + x^{12} + x^5 + 1. \quad (2)$$

At the receiver, after the signal is demodulated and converted to the binary form, CRC check is performed by calculating the remainder after division of the polynomial corresponding to a received codeword by the generator polynomial. Again, this operation is efficiently implemented in hardware using an LFSR. If the remainder is zero, the packet is assumed to be correct; otherwise, an error occurred and the packet is discarded.

A CRC code can be described using the matrix notation. Let $\mathbf{x} \in \{0,1\}^N$ denote a transmitted codeword vector corresponding to a CRC-encoded packet, and let $\mathbf{r} \in \{0,1\}^N$ denote the received codeword vector after demodulation before the CRC check. Here, N is the total number of bits in the codeword including the redundant ones; denote M as the number of redundant bits. For a given generator polynomial, one can construct a parity check matrix $\mathbf{H} \in \{0,1\}^{M \times N}$ that relates each redundant bit with the original systematic bits. The parity check matrix can be visualised as the Tanner graph, whereby each column of $\mathbf{H}$ is represented by a variable node and each row of $\mathbf{H}$ is represented by a check node [21].

Using the parity check matrix concept, the error *detection* problem (or CRC check) can be reformulated as finding whether $\mathbf{Hr} = \mathbf{0}$. However, in this work the focus is on the error *correction* problem that decodes the transmitted codeword based on $\mathbf{r}$. The optimum decoder is based on the maximum *a posteriori* (MAP) rule [21]:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}:\mathbf{Hx}=\mathbf{0}}{\arg\max} P(\mathbf{x}|\mathbf{r}). \quad (3)$$

The decoders presented in this work attempt to efficiently solve (3) in two different ways.

IV. Decoding

A. ADMM

The ADMM attempts to solve the decoding problem (3) by converting it to an LP. To see how, (3) can be reformulated

as a maximum likelihood (ML) decision rule by assuming equiprobable codewords and employing Bayes' rule:

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}:\mathbf{Hx}=\mathbf{0}} p(\mathbf{r}|\mathbf{x}). \tag{4}$$

In the natural logarithm domain, the probability density function in (4) can be expressed as

$$\ln p(\mathbf{r}|\mathbf{x}) = \ln \prod_{i=1}^{N} p(r_i|x_i) = \sum_{i=1}^{N} \ln p(r_i|x_i). \tag{5}$$

Here, it is assumed that individual bits propagate through the channel independently. For each bit, the likelihoods of the bit being 1 and 0 are related as follows:

$$\ln p(r_i|x_i = 1) = \gamma_i + \ln p(r_i|x_i = 0),$$

where γ_i is the log-likelihood ratio LLR for the i-th bit. The ML problem can now be restated as

$$\arg\max_{\mathbf{x}:\mathbf{Hx}=\mathbf{0}} \left(\sum_{i=1}^{N} [\gamma_i x_i + \ln p(r_i|x_i = 0)] \right) = \arg\max_{\mathbf{x}:\mathbf{Hx}=\mathbf{0}} \left(\sum_{i=1}^{N} \gamma_i x_i \right). \tag{6}$$

Finally, by introducing the negative LLR $\bar{\gamma}_i \triangleq -\gamma_i$, ML decoding can be converted to a minimisation problem

$$\text{minimise } \bar{\gamma}^T \mathbf{x}, \text{ subject to } \mathbf{Hx} = \mathbf{0}. \tag{7}$$

In [15], it was shown how the minimisation problem (7) can be formulated as an LP over the convex hull of all codewords. Using an LDPC code as an example, it was demonstrated that LP decoding performs similarly to the state-of-the-art decoder based on BP. In addition, it was shown that LP decoding is guaranteed to produce an ML codeword. However, the computational complexity of the original LP decoder is much higher than that of BP, and as a result in [17] the authors introduced a faster algorithm based on the ADMM which was originally developed in [16]. With this modification, the ADMM for ML decoding can be formulated as follows:

$$\text{minimise } \bar{\gamma}^T \mathbf{x}, \tag{8}$$

$$\text{subject to } \forall j,\ \mathbf{P}_j \mathbf{x} = \mathbf{z}_j, \mathbf{z}_j \in \mathbb{PP}_{d_j}. \tag{9}$$

Here, $\mathbf{P}_j$ is the operation of selecting those bits of $\mathbf{x}$ that participate in the j-th check; $\mathbf{z}_j$ is a replica vector for the j-th check; d_j, $j = 1, ..., M$ is the degree of the j-th check node; $\mathbb{PP}_{d_j}$ is the parity polytope of dimension d_j [17]. Denoting $y(\mathbf{x})$ the objective function to be minimised, the augmented Lagrangian in the unscaled form [16] for (8) can be written as

$$\mathcal{L}_\mu(\mathbf{x}, \mathbf{z}, \lambda) = y(\mathbf{x}) + \sum_j \lambda_j (\mathbf{P}_j \mathbf{x} - \mathbf{z}_j) + \frac{\mu}{2} \sum_j \|\mathbf{P}_j \mathbf{x} - \mathbf{z}_j\|_2^2, \tag{10}$$

where $\mu > 0$ is the augmented Lagrangian parameter and λ_j is an auxiliary variable. The solution to (8) is an iterative algorithm with the k-th iteration being

$$\mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} \mathcal{L}_\mu(\mathbf{x}, \mathbf{z}^{[k]}, \lambda^{[k]}), \tag{11}$$

$$\mathbf{z}^{[k+1]} = \arg\min_{\mathbf{z}} \mathcal{L}_\mu(\mathbf{x}^{[k+1]}, \mathbf{z}, \lambda^{[k]}), \tag{12}$$

$$\lambda^{[k+1]} = \lambda_j^{[k+1]} + \mu(\mathbf{P}_j \mathbf{x}^{[k+1]} - \mathbf{z}^{[k+1]}). \tag{13}$$

In (11) and (12), the two primal variables - $\mathbf{x}$ and $\mathbf{z}$ - are updated in an alternating fashion. Therefore, the ADMM can be viewed as a message passing algorithm on a graph, with x_i, $i = 1, ..., N$ and z_j, $j = 1, ..., M$ being variable and check nodes respectively.

To improve the performance of the algorithm at low SNRs and to avoid error floors at high SNRs, a penalty function was introduced in [18] such that the modified objective function $y(\mathbf{x})$ can be rewritten as

$$\bar{\gamma}^T \mathbf{x} + \sum_i f(x_i), \tag{14}$$

where $f : [0, 1] \mapsto \mathbb{R} \cup \{\pm\infty\}$ is a penalty function. Two penalty functions are proposed in [18]:

$$f_1(x) = -\alpha_1 \|x - 0.5\|_1, \tag{15}$$

$$f_2(x) = -\alpha_2 \|x - 0.5\|_2^2. \tag{16}$$

They are called l_1 and l_2 penalty functions respectively, with $\alpha_1, \alpha_2 > 0$ being the penalty coefficients. As shown in [18], the l_2 penalty function provides better PER performance than the l_1 penalty function.

To finalise the algorithm, the over-relaxation technique advocated in [16] can be adopted to improve decoding convergence. Denoting $\rho > 1$ the over-relaxation parameter and c_i, $i = 1, .., N$, the degree of the i-th variable node, the ADMM-PD algorithm with with the l_2 penalty function is summarised in Algorithm 1.

In the update for $\mathbf{z}_j$ in Algorithm 1, $\Pi_{\mathbb{PP}_{d_j}}(\cdot)$ is the projection onto the parity polytope $\mathbb{PP}_{d_j}$ [17]. In this work, the original projection algorithm proposed by [17] is employed. More computationally effective techniques were derived in [19] and [20].

As discussed in [8], the selected ADMM algorithm has several parameters: the augmented Lagrangian parameter μ, the penalty coefficient α (for a given penalty function), the over-relaxation parameter ρ and the maximum number of iterations T_{max}. Investigation into the selection of these parameters for some LDPC codes and the AWGN (additive white Gaussian noise) channel was carried out in [18], where it was shown that the algorithm is rather sensitive to parameters settings. In [8][9], a slightly different set of optimum parameters for the BSC channel was identified as follows:

$$\mu = 3, \alpha_2 = 1, \rho = 1.8. \tag{17}$$

As for T_{max}, it is clear that increasing the maximum number of iterations improves the performance but leads to longer decoding time. This compromise will be investigated in the results sections.

Algorithm 1 ADMM-PD with over-relaxation.

Input: Vector of negative LLRs $\bar{\gamma}$ and parity check matrix $\mathbf{H}$.
Output: Decoded vector $\mathbf{x}$.
1: **Initialisation**: Construct the selection matrix $\mathbf{P}_j$ for each check node j based on $\mathbf{H}$. Initialise λ_j as the all zeros vector and $\mathbf{z}_j$ as the all 0.5 vector. Set $k = 0$.
2: **Variable node update**: For each variable node i, do:
Calculate $\bar{\mathbf{z}}_j = \mathbf{P}_j^T \mathbf{z}_j^{[k]}$, $\bar{\lambda}_j = \mathbf{P}_j^T \lambda_j^{[k]}$, $\forall j$.
Calculate $t_i = \sum_j \left(\bar{\mathbf{z}}_j - \frac{\bar{\lambda}_j}{\mu} \right) - \frac{\bar{\gamma}_i}{\mu}$.
Update

$$x_i^{[k+1]} \leftarrow \frac{1}{c_i - 2\alpha_2/\mu}(t_i - \frac{\alpha_2}{\mu}).$$

Project $x_i^{[k+1]}$ onto $[0, 1]$: $x_i^{[k+1]} \leftarrow \Pi_{[0,1]} x_i^{[k+1]}$.
3: **Check node update**: For each check node j, do:
Calculate

$$\mathbf{v}_j^{[k+1]} \leftarrow \rho \mathbf{P}_j \mathbf{x}^{[k+1]} + (1-\rho)\mathbf{z}_j^{[k]} + \frac{\lambda_j^{[k]}}{\mu}.$$

Update $\mathbf{z}_j^{[k+1]} \leftarrow \Pi_{\mathbb{PP}_{d_j}}(\mathbf{v}_j^{[k+1]})$.
Update

$$\lambda_j^{[k+1]} \leftarrow \lambda_j^{[k]} + \mu \left[\rho \mathbf{P}_j \mathbf{x}^{[k+1]} + (1-\rho)\mathbf{z}_j^{[k]} - \mathbf{z}_j^{[k+1]} \right].$$

4: Make a tentative hard decision on $\mathbf{x}^{[k+1]}$: if $x_i^{[k+1]} \geq 0.5$, $\hat{x}_i = 1$; otherwise $\hat{x}_i = 0$.
5: If $\mathbf{H}\hat{\mathbf{x}} = \mathbf{0}$, then **return** $\mathbf{x} = \hat{\mathbf{x}}$. Otherwise, if $k+1$ is smaller then the maximum number of iterations T_{max}, do $k \leftarrow k+1$ and loop to **Variable node update**. Otherwise, declare decoding failure and **Stop**.

B. BP

BP attempts to solve the decoding problem (3) by considering the posterior probability for each individual bit, $P(x_i|\mathbf{r})$, $i = 1, ..., N$. In this way, decoding is turned into the marginalisation problem [21] which is efficiently tackled by BP. While several implementations of BP are available, the sum-product algorithm in its log-likelihood form [21] is employed in this work. For the purpose of completeness, it is presented Algorithm 2.

C. Calculating inputs for decoding

It can be observed that both ADMM and BP decoders require LLRs as their inputs. As was mentioned above, it is assumed that the demodulator produces hard outputs. Therefore, the channel between the CRC encoder and decoder can be modelled as a binary symmetric channel (BSC) which is characterised by the crossover probability that the bit is flipped. Denoting this probability as χ, the LLR for the i-th received bit r_i can be calculated as

$$\gamma_i = (2r_i - 1) \ln \left(\frac{1-\chi}{\chi} \right). \tag{18}$$

In practice, the crossover probability can be estimated by sending known data bits and calculating a bit error rate (BER) at the receiver. When transmitted bits are unknown, some reliability indication can be used to estimate χ. For example, the received signal strength indication (RSSI) is usually available in practical receivers. Practical recommendations will be discussed in later sections.

Algorithm 2 Log-likelihood sum-product implementation of BP.

Input: Vector of LLRs γ and parity check matrix $\mathbf{H}$.
Output: Decoded vector $\mathbf{x}$.
1: **Initialisation**: Construct sparse representation of $\mathbf{H}$:
$\mathcal{N}_m = \{n : H_{mn} = 1\}$, $m = 1, ..., M$;
$\mathcal{M}_n = \{m : H_{mn} = 1\}$, $n = 1, ..., N$.
Set $\eta_{m,n}^{[0]} = 0$ for all $(m, n) : H_{mn} = 1$;
$\lambda_n^{[0]} = \gamma_n$, $n = 1, ..., N$; $k = 0$.
2: **Check node update**: For each $(m, n) : H_{mn} = 1$, compute:

$$\eta_{m,n}^{[k+1]} \leftarrow -2 \tanh^{-1} \left(\prod_{j \in \mathcal{N}_m \setminus n} \tanh \left[-\frac{\lambda_j^{[k]} - \eta_{m,n}^{[k]}}{2} \right] \right).$$

3: **Variable node update**: For $n = 1, ..., N$, compute:

$$\lambda_n^{[k+1]} \leftarrow \gamma_n + \sum_{m \in \mathcal{M}_n} \eta_{m,n}^{[k+1]}.$$

4: Make a tentative hard decision: if $\lambda_n^{[k+1]} \geq 0$, $\hat{x}_n = 1$; otherwise $\hat{x}_n = 0$, $n = 1, ..., N$.
5: If $\mathbf{H}\hat{\mathbf{x}} = \mathbf{0}$, then **return** $\mathbf{x} = \hat{\mathbf{x}}$. Otherwise, if $k+1$ is smaller then the maximum number of iterations T_{max}, do $k \leftarrow k+1$ and loop to **Check node update**. Otherwise, declare decoding failure and **Stop**.

D. Removing cycles from the parity check matrix

It is known that the performance of iterative decoding techniques such as BP applied to a general linear code is affected by the presence of short cycles on the Tanner graph of the code [13]. In particular, cycles of length four, or four-cycles, should be avoided. As shown in [14], the total number of four-cycles for an $M \times N$ parity check matrix $\mathbf{H}$ can be calculated as

$$\sum_{i=1}^{M} \sum_{j=i+1}^{M} \binom{\left(\mathbf{H}\mathbf{H}^T\right)_{ij}}{2}. \tag{19}$$

For example, for the packet size of 39 bytes (312 bits) supported by both BLE and IEEE 802.15.4, there are 813816 four-cycles in the case of the CRC-24 code and 758300 four-cycles in the case of the CRC-16 code. Therefore, direct application of algorithms such as BP would result in extremely poor decoding performance.

The maximum cycle strategy (MCS) algorithm to sparsify the parity check matrix of a general linear code by removing all four-length cycles was presented in [13]. In this algorithm, auxiliary variable and check nodes are added to the Tanner graph for every four-cycle such that all parity check equations remain intact. Compared with the original approach given in [14], the MCS algorithm significantly reduces the number of auxiliary nodes. When applied to the 24×336 $\mathbf{H}$ matrix of the CRC-24 code, the MCS algorithm results in only 276 additional variable and check nodes, making the size of the modified matrix 300×612. Similarly, in the case of the 16×328 matrix of the CRC-16 code, 268 additional variable and check nodes are added, increasing the size of the matrix to 292×596.

The resulting matrices do not have any cycles of length four and therefore are suitable for BP-based decoding. When it comes to the ADMM, the algorithm does not explicitly require the absence of short cycles and therefore may be used with

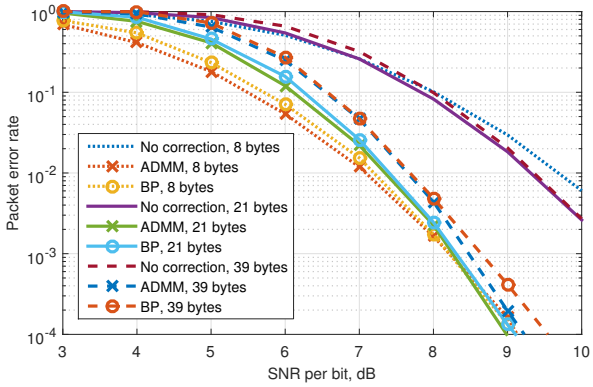


Fig. 1. Simulated PER for three different BLE packet sizes (8, 21 and 39 bytes), with and without CRC error correction.

the original parity check matrix. At the same time, it was observed empirically that the ADMM converges significantly slower when applied to the original matrix. Based on that, and for the sake of fare comparison, in this paper both ADMM and BP are applied to the modified parity check matrix with all four-cycles removed.

V. Performance Evaluation: Simulation

The performance of the proposed decoding methods is first demonstrated via Monte Carlo simulation. To this end, a BSC channel is simulated with the crossover probability calculated as

$$\chi = Q\left(\sqrt{2RE_b/N_0}\right), \tag{20}$$

where E_b/N_0 is the equivalent signal to noise ratio (SNR) calculated per bit, Q is the Q-function [21] and R is the code rate.

Fig. 1 illustrates packet error rate (PER) performance of CRC error correction based on ADMM and BP applied to BLE for three different packet sizes - 8, 21 and 39 bytes (the corresponding code rates are 0.73, 0.88 and 0.93). The maximum number of decoding iterations is limited to 1000 for both decoders. The performance curves without error correction are also presented for comparison. It can be immediately seen that for all packet sizes, error correction enables a significant gain in the equivalent SNR: 2.5, 2 and 1.8 dB at the PER of 10^{-2} for the packet size of 8, 21 and 39 bytes respectively. This can potentially improve the sensitivity threshold of a BLE receiver and, as a result, the distance between the transmitter and receiver. It is also clear that shorter packets benefit more from error correction due to the lower code rate. In line with previous works [8][9], ADMM slightly outperforms BP in both scenarios.

Fig. 2 shows the correction rate of ADMM and BP for all three packet sizes and for different numbers of errors per packet. Here, the correction rate is defined as the number of corrected packets divided to the number of erroneous packets. It can be observed that both decoders mostly fail to correct more than three errors per packet regardless of the packet size. To see why, it is worth estimating the theoretical error correction potential of the CRC code in question. It can be

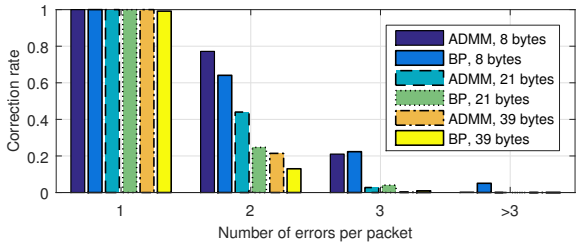


Fig. 2. Simulated correction rate for two BLE packet sizes (8, 21 and 39 bytes) as a function of number of errors per packet.

shown that the minimum Hamming distance (MHD) d_{min} of a cyclic code cannot be larger than the weight of the code's generator polynomial. Hence, for the CRC-24 code, $d_{min} \leq 8$. The *guaranteed* maximum number of corrected errors in the case of a hard-input optimum, nearest-neighbour decoder can be calculated as $\lfloor (d_{min}-1)/2 \rfloor$ [21], which is equal to 3 for the CRC-24 code. Therefore, the simulation results are fully in line with the code's error correction potential.

From Fig. 2, ADMM can be seen to significantly outperform BP in terms of the correction rate for all packet sizes and when there are no more than two errors per packet. The benefit of using smaller packets and a somewhat lower code rate is also clearly illustrated: the correction rate of both decoders rises significantly when the packet size is reduced from 39 to 8 bytes. Overall, it can be predicted that for real environments where single-, double-, and triple-error packets constitute a dominant portion of erroneous packets, CRC error correction can provide substantial benefits.

Figs. 3 and 4 depict simulation results for IEEE 802.15.4 for the same packet sizes as in the case of BLE. From Fig. 4 it can be observed that the correction rate for multiple-error packets is much lower for 802.15.4 than for BLE, and both decoding algorithms fail to correct more than two errors per packet. This can be explained using the same argument about the error correction potential as before: the MHD of the CRC-16 code does not exceed 4, hence if optimum hard-input decoding was performed, only single-error packets would be corrected for certain. Interestingly, both decoders can still correct some double-error packet. Despite the differences, the PER performance of IEEE 802.15.4 is similar to that of BLE, as can be observed from Fig. 3, with a slightly smaller SNR gain due inferior error correction capabilities: 2.2 dB for 8-byte packets compared with 2.5 dB for BLE. Both ADMM and BP perform almost identically in terms of the PER.

VI. Performance Evaluation: Real Data

In this section, the performance of ADMM and BP decoding is evaluated for real corrupted BLE and IEEE 802.15.4 packets. A discussion on the implementation aspects and complexity of decoding is also included.

The corrupted packets were collected using the TI CC2650 (CC2650EM-7ID) evaluation module which is the first commercial off-the-shelf radio that supports both BLE and IEEE 802.15.4. The module was interfaced to the SmartRF06 evaluation board controlled by SmartRF Studio 7 software that allows all received packets to be logged along with an RSSI level

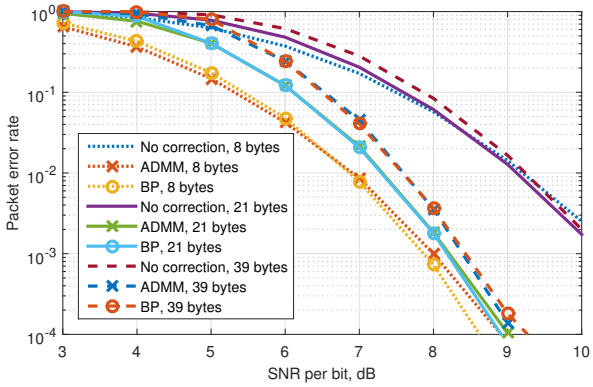


Fig. 3. Simulated PER for two IEEE 802.15.4 packet sizes (8, 21 and 39 bytes), with and without CRC error correction.

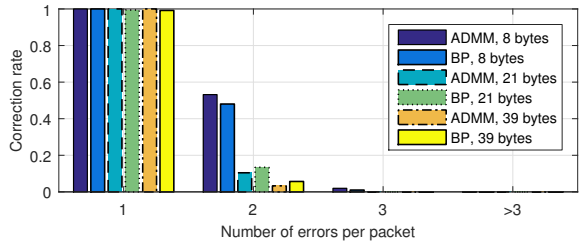


Fig. 4. Simulated correction rate for two IEEE 802.15.4 packet sizes (8, 21 and 39 bytes) as a function of number of errors per packet.

for each packet. To emulate real practical deployments, the experiments were carried out in a laboratory environment consisting of workbenches with testing equipment, office furniture and computers. For each test scenario, multiple transmitter locations were used. In addition, dynamic channel conditions were tested by collecting packets when the transmitter was moving. On top of that, some interference was present due to the radio operating in the same 2.4 GHz ISM band as a few active WiFi networks in the same area. Four datasets were collected corresponding to the two radio configuration (BLE and IEEE 802.15.4) and two packets sizes (21 and 39 bytes). For the remainder of the paper they will be referred to as BLE-21, BLE-39, 802.15.4-21 and 802.15.4-39 respectively. Each dataset consists of at least 1000 corrupted packets.

To start with, the statistics of the number of bit errors per packet for the collected datasets can be analysed, as summarised in Table I. First, it can be observed that double-error packets are the most common case in all four scenarios, comprising up to 30%. In the case of BLE, single-error packets are the second most common, making up 18% and 16% for BLE-21 and BLE-39 respectively. In the case of IEEE 802.15.4, the proportion of single-error packets is somewhat lower, with triple-error packets being the second most popular. All in all, packets with less than four errors make up the majority of all corrupted packets in all scenarios. This implies that in a real environment there is considerable potential for error correction.

TABLE I
DISTRIBUTION OF PACKETS WITH DIFFERENT NUMBERS OF ERRORS IN A REAL ENVIRONMENT FOR BLE AND IEEE 802.15.4 AND TWO PACKET SIZES (21 AND 39 BYTES).

# of bit errors	BLE		IEEE 802.15.4	
	21 bytes	39 bytes	21 bytes	39 bytes
1	18%	16%	11%	10%
2	28%	27%	30%	27%
3	12%	11%	15%	16%
>3	42%	46%	44%	47%

A. Calculating decoder inputs

In Section IV-C, it was already highlighted that both ADMM and BP decoders require reliability information about bits in the form of the LLRs. In a situation when only binary information is available for the decoder, the crossover (or bit-flipping) probability needs to be estimated so that the LLRs can be calculated according to (18). In practice, the RSSI can serve as a measure of the reliability of received data. For instance, the TI CC2650 platform used in this work provides an RSSI level for every received packet. To map the RSSI level to the crossover probability χ, a look-up table can be calculated in advance and stored at the receiver. To avoid additional computations, it is beneficial to store $\psi \triangleq \ln[(1-\chi)/\chi]$ instead of χ such that the LLR can be directly calculated as $\pm\psi$, depending on the value of the bit in question, according to (18). To construct the look-up table, one can resort either to simulation or real measurements. In this work, ψ is calculated by transmitting known bits, measuring an average BER for each possible RSSI level and then using it as the crossover probability χ.

Naturally, the constructed look-up table may contain sub-optimum values from the point of view of decoding. For instance, the real channel might be different from the one used to construct the look-up table. To optimise error correction performance, the look-up table can be calibrated to match the real environment. As a figure of merit, the average correction rate for each RSSI level can be used. It should be noted that to compute the average correction rate, no knowledge of transmitted data is required, since a simple syndrome calculation can be performed to check whether a packet has been corrected or not. In this work, ψ is calibrated at each RSSI level as follows:

$$\psi = \psi_0 + \triangle,\ \triangle \in \{-2, -1, 0, 1, 2\},$$

where ψ_0 is the initial value from the look-up table for a given RSSI obtained via BER measurements and $\triangle$ is the calibration term. The values of $\triangle$ are chosen based on the fact that ψ_0 mostly takes values between -2 to 12 (corresponding to the BER of 0.5 and 10^{-5}). For each RSSI level, the optimum value of ψ is identified as the one that gives the best correction rate.

To provide an even finer level of calibration, the process can be performed on-the-fly, for each packet individually. It is clear that this method implies additional processing time, so it is important to understand potential benefits. Fig. 5 illustrate the correction rate performance of the ADMM decoder for BLE-21 packets for three cases: calibration for each RSSI

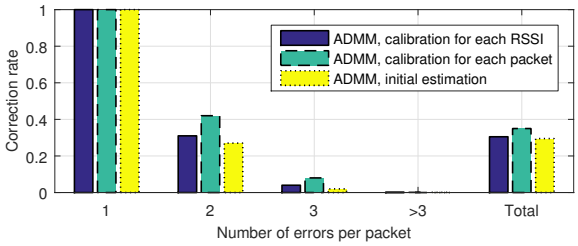


Fig. 5. Correction rate of ADMM for BLE-21 packets for different calibration methods.

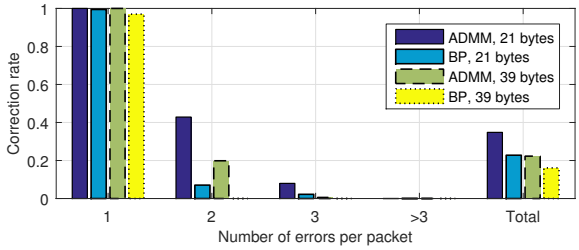


Fig. 6. Correction rate for BLE and different packet sizes as a function of number of errors per packet.

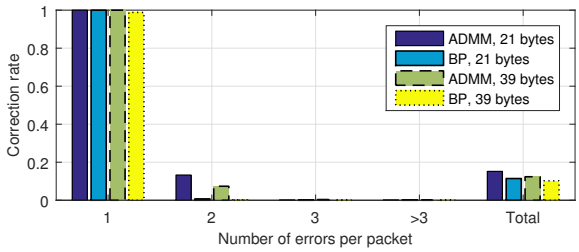


Fig. 7. Correction rate for IEEE 802.15.4 and different packet sizes as a function of number of errors per packet.

level based on previous statistics, calibration for each packet and no calibration at all. It can be seen that the per-RSSI calibration method introduces only a marginal benefit for the dataset in question, increasing an overall correction rate by 1%. The reason is that the same environment was used for the measurements and calibration. In practice, the benefits may be more substantial, especially if the initial values are based on approximation. In contrast, the per-packet calibration improves the overall correction rate by 6%, being especially advantageous for double-error packets. Therefore, depending on processing time constraints, the system designer can select one of the three approaches and trade off complexity for better performance.

B. BLE and 802.15.4 performance

In this section, the performance of ADMM and BP decoders applied to BLE and IEEE 802.15.4 is evaluated in terms of the correction rate. To highlight error correction potential, the per-packet LLR calibration method described before is used. Fig. 6 illustrates the results for for BLE-21 and BLE-39 packets. Compared with the simulation results presented in Fig. 2, the overall correction rate ('Total' bar) that can serve as an ultimate figure of merit is also included. It can be seen that when it comes to real packets, the ADMM performance is similar to the simulated one, with the correction rate for triple-error packets being even higher (7% against 3%). In contrast, BP demonstrates poor decoding performance in practice, correcting only 7% of double-error BLE-21 packets as opposed to 20% for simulated data, and almost not being able to correct double-error 39-byte packets. The total correction rate of ADMM is 12% higher than that of BP for BLE-21 packets and 6% higher for BLE-39 packets. Overall, up to 35% of all corrupted BLE-21 packets and up to 22% of all corrupted BLE-39 packets can be corrected. Compared with the look-up method mentioned in Section II that is able to correct all single-error packets, ADMM can correct twice as many BLE-21 packets and 38% more BLE-39 packets.

Fig. 7 demonstrates the correction rates in the case of IEEE 802.15.4. Again, ADMM outperforms BP by correcting 13% real 802.15.4-21 packets and 7% 802.15.4-39 packets, as opposed to 1% and 0% for BP. Overall, the total correction rate for IEEE 802.15.4 is much lower than for BLE: only up to 15% of 802.15.4-21 packets and up to 12% 802.15.4-39 packets can be corrected. As discussed before, this is due to inherently lower error correction potential of the CRC-16 code compared to the CRC-24. While the advantage of ADMM over the look-up method is only marginal, it can still be important in situations when there is no alternative way to recover corrupted packets.

C. Complexity analysis

To evaluate the complexity of the proposed decoding algorithms, it can be first estimated how the overall correction rate depends on the maximum number of decoding iterations T_{max}, which in turn directly affects the average time delay introduced by error correction. Fig. 8 illustrates the results for various maximum numbers of iterations for the BLE-21 case. As a reference, the correction rate of the look-up method mentioned in Section II that is able to correct all single-error packets is shown. No LLR calibration is performed, but in line with the previous results and to highlight error correction potential the optimum values on a per-packet basis are used. It can be observed that the benefit from the higher number of iterations for the BP decoder is only marginal: the correction rate increases only by 28% by changing T_{max} from 10 to 1000. By contrast, the correction rate of the ADMM decoder is boosted by seven times for the same scenario. It can also be seen that BP outperforms ADMM when the maximum number of iterations is small. These results confirm the fact that the convergence rate of BP is in general faster than that of ADMM, while the probability of successful decoding of the latter is higher in asymptotic time. At the same time, neither of the decoders is better than the look-up method when $T_{max} = 10$, hence the maximum number of iterations should be at least 100 for the decoders to be efficient.

In practice, the performance of error correction is also characterised by the time delay it introduces at the receiver. To estimate such delay, both the ADMM and BP decoders

TABLE II
COMPLEXITY OF ADMM AND BP IN TERMS OF THE AVERAGE DECODING TIME (MS) AND THE AVERAGE NUMBER OF FLOPS PER PACKET.

Max. # of iterations	ADMM		BP	
	Time, ms	Gflops	Time, ms	Gflops
10	1.1	0.8	2.4	1.7
100	8.5	6.1	18.9	13.5
500	42.5	30.4	94.5	67.5
1000	85.0	60.7	189.0	135.5

were implemented in C++[1] as per Algorithms 1 and 2, with the average decoding time per packet being measured on a desktop machine with Intel i7 3.1 GHz CPU, 8 GB RAM and a Microsoft Visual C++ 2010 compiler. It was observed empirically that the average decoding time grows linearly with T_{max}, with each 10 iterations requiring approximately 1.1 ms for ADMM and 2.4 ms for BP. The compromise between the correction rate and the average decoding time for the BLE-21 dataset is shown in Fig. 9 for both decoders, where the decoding time was measured for the same values of T_{max} as in Fig. 8. The intersection point, corresponding to the average decoding time of 5.5 ms, can be clearly observed. Above that point, ADMM is superior to BP both in terms of correction rates and speed, exhibiting the average delay of up to 80 ms per corrupted packet. It should be noted that below the intersection point, the correction rate of both ADMM and BP becomes inferior to that of the look-up method correcting all single errors.

To provide an implementation-independent complexity estimation of the proposed algorithms, a simple repetitive operation consisting of an addition of two integer numbers and multiplication of two floating-point numbers was implemented, using the same compiler and hardware platform as before. For convenience, let one *flop* define a combined operation of such addition and multiplication. By measuring the time required to perform different numbers of flops for the configuration mentioned above, a linear relationship was observed between the two, with one flop corresponding to 1.4 ns. Based on this value, the average decoding time can be expressed in terms of a number of flops, which gives an approximate platform-independent complexity estimation. Table II summarises the complexity estimation for the two algorithms in terms of both the average decoding time and the average number of flops per packet, for different values of the maximum number of decoding iterations.

Based on the presented analysis, it can be concluded that the maximum number of iterations in the case of ADMM could be configured to match the application requirements. For applications where data is transmitted only occasionally, such as smart metering systems where a single packet is sent once in 24 hours [23], T_{max} can be set to a value much larger than 1000, potentially increasing the probability of packet correction.

[1]The projection algorithm implementation for the ADMM decoder was taken from [22].

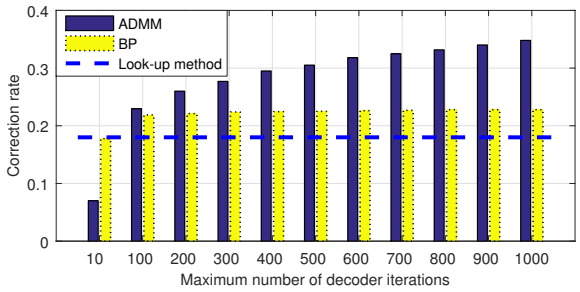


Fig. 8. Correction rate for BLE-21 for ADMM (dark) and BP (light) decoders as a function of the maximum number of decoder iterations.

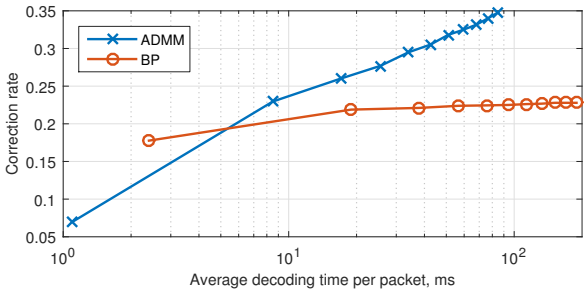


Fig. 9. Average decoding time per packet (ms) for BLE-21 ADMM/BP as a function of the maximum number of decoder iterations.

VII. CONCLUSIONS

In this work, error correction techniques are investigated in the context of energy-efficient communication standards for the IoT lacking traditional forward error control methods due to transmitter constraints. The proposed error correction is based on utilising the existing redundancy added to transmitted data by CRC codes, ubiquitously used to detect errors in received data. No additional overhead or signal processing is imposed by the proposed techniques, which preserves the energy efficiency of the transmitter and compliance with an existing physical layer. The correction is performed by post-processing at the receiver. Two iterative decoding algorithms, ADMM and BP, traditionally used for state-of-the-art error control codes, are employed. Based on two widely deployed IoT standards, BLE and IEEE 802.15.4, the proposed methods are first evaluated through simulations. It is shown that an SNR gain of up to 2.5 dB can be obtained by introducing error correction, with no extra cost for the transmitter. Potential benefits include longer operational range and battery life. It is also demonstrated that a significant portion of packets containing up to 3 bit errors can be corrected, with smaller packets having higher correction rates.

The techniques are then verified in a practical scenario using the TI CC2650 platform that supports both standards. To start with, datasets of corrupted packets were collected for both standards and different packet sizes in an indoor environment and it was shown that the majority of all real erroneous packets have no more than 3 errors, indicating significant error correction potential. When the error correction methods were applied, it was demonstrated that totally up to 35% of corrupted BLE packets and up to 15% of corrupted IEEE

802.15.4 packets were corrected. It was shown that ADMM significantly outperforms BP when in comes to practice in both scenarios.

Finally, the complexity of the proposed methods was analysed, where it was shown that the performance of ADMM improves steadily with the number of iterations, in contrast with BP. The processing delay introduced by both algorithms was measured based on a desktop PC implementation. Depending on the limit on the number of iterations, ADMM exhibited the average delay per corrupted packet from 5.5 to 80 ms, providing the correction rates from 0.18 to 0.35 respectively. It should be emphasized that the proposed methods introduce complexity only at the receiver side, without affecting the transmitter.

REFERENCES

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, 2014.

[2] G. Margelis, R. Piechocki, D. Kaleshi, and P. Thomas, "Low throughput networks for the iot: Lessons learned from industrial implementations," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, Dec 2015, pp. 181–186.

[3] *Specification of the Bluetooth system. Core Version 4.1*, Bluetooth SIG, 2013. [Online]. Available: http://www.bluetooth.com

[4] N. Newman, "Apple iBeacon technology briefing," *J. Direct, Data Digit. Mark. Pract.*, vol. 15, no. 3, pp. 222–225, Jan. 2014.

[5] "IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2006*, 2006.

[6] V. C. Güngör, D. Sahin, T. Kocak, S. Ergüt, C. Buccella, C. Cecati, and G. P. Hancke, "Smart grid technologies: Communication technologies and standards," *IEEE Trans. Ind. Informatics*, vol. 7, no. 4, pp. 529–539, 2011.

[7] Thread Group, "Thread Stack Fundamentals V2," 2015.

[8] E. Tsimbalo, X. Fafoutis, and R. Piechocki, "CRC Error Correction for Energy-Constrained Transmission," in *Proc. 26th IEEE Int. Symp. Personal, Indoor and Mobile Radio Commun. (PIMRC)*, 2015.

[9] E. Tsimbalo, X. Fafoutis, and R. Piechocki, "Fix It, Dont Bin It! - CRC Error Correction in Bluetooth Low Energy," in *Proc. 2nd IEEE World Forum on Internet of Things (WF-IoT)*, 2015.

[10] W. Peterson and D. Brown, "Cyclic Codes for Error Detection," *Proc. IRE*, vol. 49, no. 1, pp. 228–235, Jan. 1961.

[11] S. Babaie, A. K. Zadeh, S. H. Es-hagi, and N. J. Navimipour, "Double Bits Error Correction Using CRC Method," in *2009 Fifth Int. Conf. Semant. Knowl. Grid.* IEEE, 2009, pp. 254–257.

[12] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[13] V. Kumar and O. Milenkovic, "On graphical representations of algebraic codes suitable for iterative decoding," *IEEE Commun. Lett.*, vol. 9, no. 8, pp. 729–731, 2005.

[14] S. Sankaranarayanan and B. Vasic, "Iterative Decoding of Linear Block Codes: A Parity-Check Orthogonalization Approach," *IEEE Trans. Inf. Theory*, vol. 51, no. 9, pp. 3347–3353, Sep. 2005.

[15] J. Feldman, M. Wainwright, and D. Karger, "Using Linear Programming to Decode Binary Linear Codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.

[16] S. Boyd, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2010.

[17] S. Barman, X. Liu, S. Draper, and B. Recht, "Decomposition methods for large scale LP decoding," *2011 49th Annu. Allert. Conf. Commun. Control. Comput.*, vol. 59, no. 12, pp. 253–260, 2011.

[18] X. Liu, S. C. Draper, and B. Recht, "Suppressing pseudocodewords by penalizing the objective of LP decoding," in *2012 IEEE Inf. Theory Work.* IEEE, Sep. 2012, pp. 367–371.

[19] X. Zhang and P. H. Siegel, "Efficient iterative LP decoding of LDPC codes with alternating direction method of multipliers," in *2013 IEEE Int. Symp. Inf. Theory*. IEEE, Jul. 2013, pp. 1501–1505.

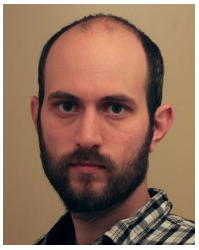
[20] G. Zhang, R. Heusdens, and W. B. Kleijn, "Large Scale LP Decoding with Low Complexity," *IEEE Commun. Lett.*, vol. 17, no. 11, pp. 2152–2155, Nov. 2013.

[21] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.

[22] X. Liu, "ADMM Decoder. Projection algorithm in C++," available online at https://sites.google.com/site/xishuoliu/codes, accessed: June 2015.

[23] M. D. Vithanage, X. Fafoutis, C. B. Andersen, and N. Dragoni, "Medium access control for thermal energy harvesting in advanced metering infrastructures," in *Proc. IEEE Eurocon*, 2013, pp. 291–299.

Evgeny Tsimbalo (S'14) received a Diploma in Electrical and Electronic Engineering (with distinction) from the Ural State University, Russia in 2005 and an MSc degree in Wireless Communications and Signal Processing (with distinction) from the University of Bristol, UK in 2012. He spent several years in industry before returning to academia in 2013. Since then, Evgeny has been a researcher with the University of Bristol, where he is also pursuing a PhD degree in Wireless Communications. Since 2014, Evgeny has also been part of the EPSRC SPHERE project. His interests include communication theory, forward error correction, information theory and space-time signal processing.



Xenofon Fafoutis (S'09-M'14) holds a PhD degree from the Embedded Systems Engineering section of the Technical University of Denmark in 2014; an MSc degree in Computer Science from the University of Crete in 2010; and a BSc in Informatics and Telecommunications from the University of Athens in 2007. He was also associated with the Foundation for Research and Technology - Hellas (FORTH). Since 2014, he is a researcher with the University of Bristol and a member of the EPSRC SPHERE interdisciplinary research collaboration. His interests lie in the fields of wireless embedded systems, wireless sensor networks and the internet of things.

Robert Piechocki (M'06) received an MSc degree from Technical University of Wroclaw (Poland) in 1997 and a PhD degree from the University of Bristol in 2002. Robert is currently a reader in Advanced Wireless Access and a member of Communications Systems and Networks group. His research interests span the areas of Statistical Signal Processing, Information and Communication Theory, Wireless Networking, Body and ad-hoc networks, Ultra Low Power Communications and Vehicular Communications. He has published over 100 papers in international journals and conferences and holds 13 patents in these areas. For the SPHERE project Robert is leading the development of wearable and wireless technologies.