IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 23, 2018, accepted March 20, 2018, date of publication April 4, 2018, date of current version April 23, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2818740

Low Complexity Parity Check Code for Futuristic Wireless Networks Applications

SALAH ABDULGHANI ALABADY[1], AND FADI AL-TURJMAN[2], (Member, IEEE)
[1]Computer Engineering Department, University of Mosul, Mosul 41002, Iraq
[2]Computer Engineering Department, Antalya Bilim University, 07190 Antalya, Turkey

Corresponding author: Salah Abdulghani Alabady (eng.salah@uomosul.edu.iq)

ABSTRACT The Internet of Things refers to the aptitude of remotely connecting and monitoring anything, anytime, and anywhere via futuristic wireless networks. Due to the unreliable wireless links, broadcast nature of wireless transmissions, interference and noisy transmission channels, frequent topology changes, and the various quality of wireless channel, there are challenges in providing high data rate service, high throughput, high packet delivery ratio, low end-to-end delay, and reliable services. In wireless network and real time application systems, low complexity and shorter codeword length in channel coding scheme are preferred. Consequently, in order to address these challenges, we propose a novel error detection and correction codes called the low-complexity parity-check (LCPC) codes with short codeword lengths for futuristic wireless networks applications. The proposed codes have less complexity and lower memory requirement in comparison to turbo and low-density parity-check (LDPC) codes. Simulation results demonstrated that the proposed LCPC codes outperform the Hamming and Reed–Solomon codes, in addition to the renowned LDPC codes. It offers up to 3-dB coding gain.

INDEX TERMS Error detection and correction, LCPC, LDPC, short codeword, Internet of Things.

I. INTRODUCTION

Smart environments are advancing in various domains in our daily life. It tackles various problems in different fields related to populated cities' constructions, medications, transportation, and the industries of agriculture, forestry, fishery, etc. Futuristic wireless networks, which are fault-tolerant and satisfy competitive Quality of Service (QoS) standards, are expected to be the essential backbone for monitoring, visualizing, analyzing and optimizing flows of resources, energy, information, and people in these smart environments. Obviously, there is an urgent need for new wireless communication paradigms, which are fault-tolerant enough to bridge the existing gap between theory and practice. The implementation of futuristic networks in smart environments have recently been significantly affected by the emerging phenomena called Internet of Things (IoT). IoT and Wireless sensor networks (WSNs) have attracted a huge attention of researchers and have been applied in most aspects of our lives in various fields of technology [1]. The main idea of this concept is the ubiquitous presence around us with variety of things or objects that are connected to the Internet such as mobile phones, laptops, daily used objects like refrigerators, televisions and smart sensors [2].

Mainly, due to the rapid proliferation of wearable devices, smart sensors and smartphones, the IoT enabled technology is evolving from conventional hub based system to more personalized systems. Efficient IoT-enabled systems can be realized by providing fault-tolerant access to rich information with unobtrusive monitoring. Wireless communication links which are rapidly prone to failures are at the heart of this concept, and their development is a key issue if such concept is to achieve its potential [3], [4]. A major concern in IoT and WSNs is the energy conservation and consumption. The fundamental challenge for the realization of the IoT enabling technologies is energy-constrained communication [5]. Therefore, avoiding or reducing the number of retransmitting the error packets is very significant. Depending on the channel condition, there are different schemes to reduce the energy consumption. One method is to find the optimal frame size. If the channel is good, bigger frames make more sense, as they will all go through with less overhead. While the channel conditions are not as good, smaller frames are better, as the probability of having a frame in error is lower with smaller frames. Another strategy to reduce the energy consumption is using Forward Error Correction mechanisms. Hence, we focus in this article on error correction aspects of

wireless connections in the IoT era [2]. A Low Complexity Parity Check (LCPC) codes that detect and correct single and double bit errors is propose.

During the past decade, many error detection and correction code schemes (Turbo codes, RS, BCH, and LDPC codes) have been investigated to increase the reliability of the wireless network systems, in order to fulfill the quality of the data in a high data rate wireless network. Each of the designed codes has its own advantage to be used as the channel coding scheme in a communication system. Recently, there have been wide spread of research works on LDPC codes since it offers near Shannon limit performance [5].

One of the earliest error detection and correction codes available are the Hamming codes, which are able to detect up to double bit errors but just able to correct single bit error. This constraint to correct double bit errors can be attributed to limited of number of syndromes available. The codes consist of 3 rows and 7 columns which account for 8 values of syndrome [5], [6]. In case of single bit error, there are seven possibilities of error pattern when the codeword length equals 7 bits. In this case each error pattern is assigned to one syndrome vector. On the contrary, double bit errors throw 21 possibilities of error pattern for the same codeword length equal to 7 bits. The Hamming code does not have the requisite number of syndrome (i.e., 21) which results in each 3 error pattern being assigned by one syndrome vector. This makes the correction operation very difficult and impossible, and it fails to decide the correct error pattern from the three possible error patterns. In addition, Hamming code cannot detect more than two bit errors (e.g., burst error) [5], [7], [8].

RS [9] code is another famous error correction code and is the subset of the BCH code [9]. For, a particular RS code specified as RS (n, k) with s-bit symbols, the number and type of errors that can be corrected in RS code depends on the characteristics of that code [10]. An RS decoder can correct up to t symbols that contain errors in a code word, where $2t = n - k$. To increase the capability of error correction, the number of the parity code must also increase. This means that the value of t in RS code must be very large.

Likewise, the error correction capability of the LDPC code also depends on the code word length and the characteristic of the parity check matrix [5], [6]. The error correction capability of the LDPC codes depend on the codeword length and the characteristics of the parity check matrix [7], [8]. The decoder gives a better performance with a larger codeword and with good parity-check matrices. In practice, to achieve a better BER performance with LDPC codes close to the channel capacity, the length of the LDPC codeword used should be in the order of thousands of bits [9], [10]. The matrix multiplication for that big codeword size demands huge memory, computational requirements and more complex decoding [9], [11]–[13]. The LDPC codes fail to correct errors if the number of errors occurred is greater than the error correction capability of the decoder. Furthermore, LDPC codes require iteration in the detection and correction error processes around 10 to 50 times of iteration [14], [15]. Therefore, the need of efficient channel codes with lower encoding and decoding complexity, and lower memory size requirement, which do not require any iteration in the decoding process, is quite obvious. For short codeword length coding, there have been several attempts in the literature. For example, authors in [10]–[12] and [16]–[21] take into consideration the short codeword length for LDPC code. On the other hand, authors in [22]–[25] are interested in the short codeword length for turbo codes.

This paper considers various code rates for the LCPC code (i.e., 0.428, 0.375 and 0.444), LCPC (7, 3), LCPC (8, 3) and LCPC (9, 4). With the intention of producing a simple error correction code, the capability of Hamming code that able to correct up to double bit errors is extend. The proposed LCPC codes offer lower encoding and decoding computational loads as compared to the Turbo code [25], RS, BCH, and LDPC codes since the former does not involve iteration process and requires very low memory and low complexity. Each LCPC code is represented as a short-length codeword, which makes the proposed codes particularly attractive for low-latency and real time applications of IoT and futuristic wireless networks applications.

The simulation results show that the proposed LCPC codes outperforms other code types such as Hamming, RS and LDPC codes. The LCPC (9, 4) code produces 3 dB coding gain as compared with LDPC (8, 4) code with decodes Bit Flip Decoding algorithm at BER $= 10^{-5}$, and 1 dB at BER $= 10^{-5}$ in case of Log Domain decoding algorithms. On the other hand, LCPC (9, 4) code produces 2.1 dB coding gain as compared with the RS (7, 4) and 1.7 dB with the Hamming (7, 4) code at BER equal to 10^{-5}. The main difference between LCPC code and Hamming code is that the latter can correct single bit error and detect double bit error. Whereas, the LCPC codes can detect and correct single and in many cases of double bit errors (i.e., 21 in case LCPC (9, 4)).

This paper is organized as follows. Section II provides the description of the proposed approach to LCPC codes. Section III presents the complexity analysis of the LCPC code. Section IV presents the performance of LCPC codes and analysis of simulation results. Section V contains the conclusions.

II. THE PROPOSED LCPC CODES

In this section, a general proposed method for the LCPC codes is presented. The LCPC code is defined a block code (n, k), where n is the codeword length and k is the information length, respectively. In this paper, three different kinds of LCPC codes are presented. In order to explain how the LCPC codes work, we present in details the encoding and decoding of the LCPC (9, 4) code in the next subsections. The encoding/decoding of the LCPC (8, 3) and LCPC (7, 3) codes is identical with the LCPC (9, 4) code. The **G** and **H** matrices of the LCPC (8, 3) and LCPC (7, 3) codes are presented in the Appendix.

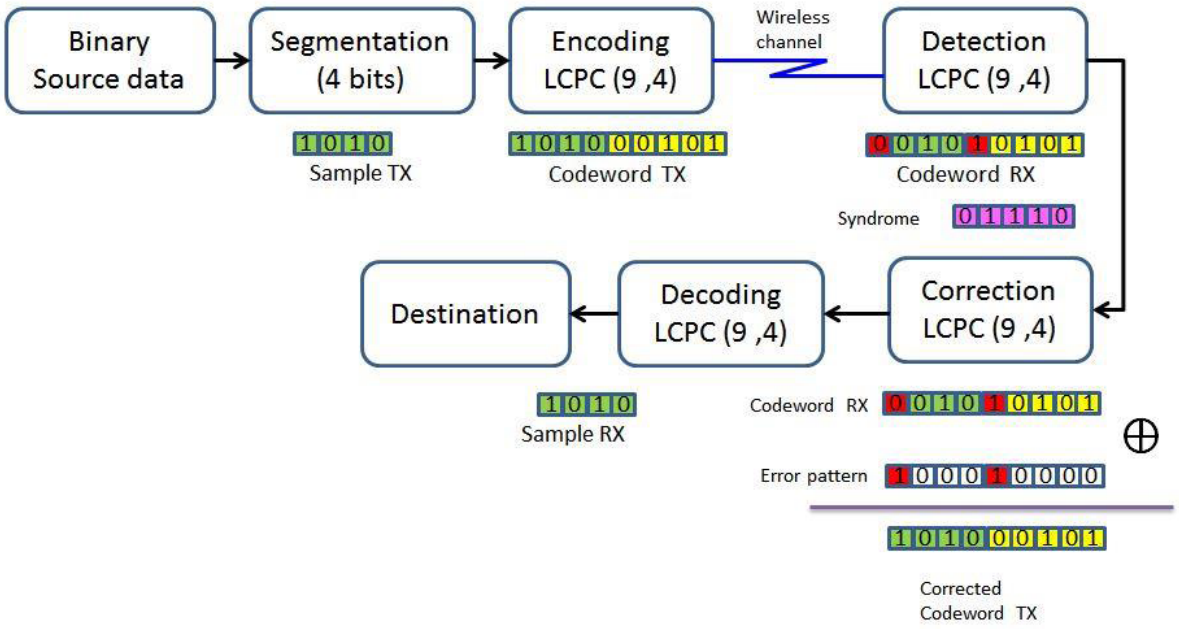

FIGURE 1. Components of encoding and decoding processes in LCPC codes.

A. LCPC CODE ENCODING

A general block diagram of encoding and decoding processes in LCPC code is shown in Fig. 1. The first step in the encoding process of LCPC code is to segment the source data sequence into symbols of equal length (i.e., k bits). Subsequently, we take each symbol (k bits) and map it into a codeword c of n bits, where $n > k$. The $n - k$ additional parity-check bits are the redundancies added, which are used for error detection and correction. The LCPC code is a block code which takes the data stream from the source encoder, divides it into four-bit symbol (i.e., k), and then encodes each four-bit symbol (depending on the number of rows in $\mathbf{G}$ matrix) into a nine-bit codeword (i.e., n)(depending on the number of columns in $\mathbf{G}$ matrix), before the transmission. The symbol of source data is denoted as $SD_i = (v_1, v_2, \ldots v_k)$, where $1 \leq i \leq j$, and j is the number of symbols of the source data, v is a binary bit, and $k = 4$ is the length of the symbol. Each k-bits symbol is then encoded into an n-bits codeword before the transmission. The encoding process is implemented using the generator matrix $\mathbf{G}$, that could be expressed as Eq. 1. The four left columns represents the Identity (I) matrix and the five right columns represents the parity (P) matrix. The parity (P) matrix is computed using Eq. (4).

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (1)$$

In the encoding unit the redundant bits r is then added to each symbol to make the length of the codeword equal to n, where $n = k + r$, and $r = 5$. The codeword of the symbol corresponds to $CD_{Ti} = (\beta_1, \beta_2 \ldots \beta_n)$, where $n = 9$, and β_i is a binary bit.

A codeword CD_{Ti} given by Eq. (2), that is used for encoding the symbols data which is defined as a multiplication between SD_i and $\mathbf{G}$:

$$CD_{Ti} = SD_i \times G \quad (2)$$

Where, SD_i is an information symbol, CD_{Ti} is the transmitted codeword and $\mathbf{G}$ is the proposed generator matrix.

Eqs. (3) and (4) show the symbol information bits and parity bits respectively for LCPC (9, 4) code. In LCPC (9, 4) code, the number of redundant parity bits is 5, so the maximum number of syndrome vector obtained is 32 (i.e., 2^5) and this makes the LCPC (9, 4) code can detect and correct single and many cases of double bit errors. While, in the Hamming (7, 4) code, the number of redundant parity bits is 3, so the maximum number of syndrome vector obtained is 8 (i.e., 2^3). In addition, Hamming code cannot detect more than two bit errors (e.g., burst error).

$$\begin{aligned} \beta_1 &= v_1 \\ \beta_2 &= v_2 \\ \beta_3 &= v_3 \\ \beta_4 &= v_4 \\ \beta_5 &= \gamma_1 = v_1 \oplus v_2 \oplus v_3 \oplus v_4 \\ \beta_6 &= \gamma_2 = v_1 \oplus v_2 \oplus v_3 \\ \beta_7 &= \gamma_3 = v_1 \oplus v_2 \oplus v_4 \\ \beta_8 &= \gamma_4 = v_1 \oplus v_3 \oplus v_4 \\ \beta_9 &= \gamma_5 = v_2 \oplus v_3 \oplus v_4 \end{aligned} \quad (3)\ (4)$$

Where, the five bits ($\gamma_1, \gamma_2, \ldots \gamma_5$) are the parity bits, and the four bits ($v_1, v_2, \ldots v_4$) are the symbol bits.

The proposed parity check matrix **H** of the LCPC (9, 4) code is given by Eq. (5), which can be used in error detection.

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

B. LCPC DECODING

The decoding algorithm is of three stages. Firstly, is the error detection which is implemented by compute the syndrome vector. Secondly, is the determine of the error pattern. Thirdly, is the error correction.

1) LCPC ERROR DETECTION

The detection process detects errors in the received codeword (CD_{Ri}), that is defined as the transmitted codeword (CD_{Ti}) with errors pattern (EP), as shown in Eq. (6). The parity check matrix **H** of LCPC code is used for this purpose. After the codeword is received, the syndrome vectors (SY) are obtained from the (CD_{Ri}).

$$CD_{Ri} = CD_{Ti} + EP \quad (6)$$

$$SY = H \times CD_{Ri}^{T} \quad (7)$$

Where, SY $= (\gamma_1, \gamma_2, \ldots \gamma_r)$ is the syndromes binary vector. Therefore, Eq. (7), can be written as:

$$SY = H \times (CD_{Ti} + EP)^{T} \quad (8)$$

$$SY = H \times CD_{Ti}^{T} + H \times EP^{T} \quad (9)$$

Since any row in the **H** matrix is orthogonal to the rows of the **G** matrix, and the inner product of a row in **G** with a row in **H** will be zero, the result of multiplication **H** by the CD_{Ti}^{T} is zero if there are no bit errors in the codeword as shown in Eq. (10), where the CD_{Ti}^{T} is the transpose of the transmitted codeword.

$$H \times (CD_{Ti})^{T} = 0 \quad (10)$$

From Eqs. (9) and (10), the SY can be expressed as:

$$SY = H \times (EP)^{T} \quad (11)$$

Here, EP^{T} is the transpose of error pattern. From Eq. (11), it can be noticed that the syndrome vector SY depends only on the error pattern EP (i.e., number of error bits, and the position of the error bits). The error detection is implemented by calculating the SY value. For any received codeword, if a SY is the null vector, (i.e., SY = 0), it indicates that the received codeword (CD_{Ri}) is error-free. On the other hand, if SY is a non-zero vector, there is change in bits, which means there are some bit errors.

Tables 4 and 5 show the EP and SY for LCPC (9, 4) code for single bit error and the cases of double bit errors in the received codeword. Tables 6, 7, 8, and 9 show the EP and SY for LCPC (8, 3) and LCPC (7, 3) for single bit error and the cases of double bit errors in the received codeword, respectively.

Algorithm 1: LCPC Encoding and Decoding

1: // **LCPC Encoding**, $\mathbf{SD_i}$ information symbol, $\mathbf{CD_{Ti}}$ transmitted codeword, and **G** generator matrix
2: $CD_{Ti} = SD_i \times G$
3: // **LCPC Decoding**
4: M = 0 // flag to indicate the correction is done completely
5: $CD_{Ri} = CD_{Ti} + EP$ // $\mathbf{CD_{Ri}}$ received codeword, **EP** error pattern
6: $SY = H \cdot CD_{Ri}^{T}$ // **SY** syndromes vector, **H** parity check matrix
7: **if** SY = 0 // No Errors in the received codeword
8: $CD_{Ti} = CD_{Ri}$
9: $SD_i = AND(CD_{Ti}, 111100000)$
10: **end if**
11: **Goto** 33
12: **if** SY ≠ 0 // Errors in the received codeword
13: The type of bit error (single or double) and the **EP** value that will used to correct is specified depending on the **SY** value.
14: Obtain **SY** value
15: **end if**
16: **if** SY value indicates there is a single bit error then
17: // Obtain the **EP** from Lookup Table1* that stored in memory
18: $\overline{CD_{Ti}} = CD_{Ri} + EP$ // correction process
19: **else if** SY value indicates there is a two bit error then
20: // obtain the **EP** from Lookup Table2* that stored in memory
21: $\overline{CD_{Ti}} = CD_{Ri} + EP$ // correction process
22: **end if**
23: // after correction process, check the **SY** to make sure the correction is done completely
24: $SY = H \cdot \overline{CD_{Ti}}$
25: **if** SY = 0 // correction is done completely
26: $SD_i = AND(\overline{CD_{Ti}}, 111100000)$ // obtain the original source symbol data
27: M=0
28: **return**
29: **else**
30: // SY ≠ 0 , the correction is NOT done completely, there are more than double error bits
31: M=1 // flag to indicate there are more than double bit error
32: **end if**
33: **return**

Lookup Table_1* , include EP and SY for single bit error.
Lookup Table_2* , include EP and SY for double bit error.

FIGURE 2. Pseudo code for error detection and correction function of LCPC codes.

The number of error pattern can be computed using Eq. (12), where, n is the codeword length (in the proposed code $n = 9$), and $e \in (1, 9)$ is the number of bit errors that may occur in the codewords. In the proposed code, we assume the lookup tables that include the EP of each SY for single and double bit errors are stored in the memory.

$$NoEP = \frac{n!}{e!(n-e)!} \quad (12)$$

Depending on the SY value, the EP can be determined from the lookup tables that is stored in memory. Fig. 2 illustrates the Pseudo code for error detection and correction functions of the proposed LCPC codes.

2) ERROR CORRECTION

In the error correction process, the EP is chosen (depending on the SY value) and is fetched from the lookup tables in

memory. The correction process is achieved as shown in Eq. (13).

$$\overline{CD_{Ti}} = CD_{Ri} \oplus EP \tag{13}$$

Where, $\overline{CD_{Ti}}$ is defined as the corrected transmitted codeword after being received.

If SY indicates that if there is single or double bit errors in CD_{Ri}, LCPC code can correct the received codeword (CD_{Ri}). This correction is achieved by adding the specific EP to error received codeword CD_{Ri}. Next, the decoding of the corrected received codeword is carried out. The decoder is implemented by masking the last four bits on the left side of the codeword. After correction process is completed, SD_i can be obtained by implementing the decoding process on the $\overline{CD_{Ti}}$. The third process is the decoder, which is used to decode the $\overline{CD_{Ti}}$ and obtain the original source symbol data sent. The masking process for the last left four bits of the correct received codeword must be done by AND operation for the $\overline{CD_{Ti}}$ with (111100000) as shown in Eq. (14).

$$SD_i = AND(\overline{CD_{Ti}}, 111100000) \tag{14}$$

III. COMPLEXITY ANALYSIS

This section presents the methodology of computational complexity analysis measurement of the LCPC codes. Counts of additions and multiplications, in addition to the size of required memory to save EP and SY tables for the LCPC code are used to measure the approach complexity. Table 1 shows the memory size requirement of the proposed types of LCPC codes, where n is the codeword length, k is the symbol data length, r (i.e., $n - k$) is the parity check bits length. NoEP is the number of error pattern for single and double bit error. NoSY is the number of syndrome vector for single and double bit error. The NoEP is computed as in Eq. (12). The computation of NoSY is based on the simulation program and using the Eq. (7).

The memory size requirement to store the error pattern and syndrome vector tables for the proposed LCPC codes is very low as shown in Table 1. For example the memory size requirement to store the error pattern and syndrome vector tables for LCPC (9, 4) is 420 bits, 364 bits for LCPC (8, 3) and 253 bits for LCPC (7, 3). The total memory size requirement for the three LCPC codes proposed is 1037 bits (130 Byte). Therefore, the time and the space complexities are still low for LCPC codes. Even with LCPC (12, 6), the memory size requirement to store the error pattern and syndrome vector tables is 1404 bits (176 Byte), which is still a very low memory size. For LCPC (14, 7) the memory size requirement to store the error pattern and syndrome vector tables is 2205 bits (276 Byte). For LCPC (24, 12) the memory size requirement to store the error pattern and syndrome vector tables is 10224 bits (1278 Byte). Accordingly, the LCPC code can be implemented on a memory chip with a very limited capacity (less than tens of Kbytes). Table 1 shows the memory size requirement to store the error pattern and syndrome vector tables for the different types of LCPC codes.

TABLE 1. Memory size required for different types of LCPC codes.

Type of code LCPC (n, k)	Size of SY table (n-k) × NoSY	Size of EP table n × NoEP	Total size (bits)
LCPC (7, 3)	4 × 23 = 92	7 × 23 = 161	253 (32 Byte)
LCPC (8, 3)	5 × 28 = 140	8 × 28 = 224	364 (46 Byte)
LCPC (9, 4)	5 × 30 = 150	9 × 30 = 270	420 (53 Byte)
LCPC (12, 6)	6 × 78 = 468	12 × 78 = 936	1404 (176 Byte)
LCPC (14, 7)	7 × 105 = 735	14 × 105 = 1470	2205 (276 Byte)
LCPC (24, 12)	12 × 300 = 3600	24 × 300 = 7200	10224 (1278 Byte)

In comparison to the literature, our approach is more efficient where the memory requirement for the existing decoders of LDPC codes are as follows; 1321 KB [26], 658 KB [27], 1331 KB [28], and 581 KB [28].

Table 2 shows the number of addition and multiplication operations for encoding and decoding processes of LCPC (9, 4) code. It also shows the Big O complexity. Table 3 shows the comparison complexity between the proposed LCPC (9, 4) code with different types of decoding algorithm for LDPC codes. Table 3 shows that the proposed LCPC (9, 4) code has low complexity $O(n)$ compared with the LDPC codes.

Three main differences between the LDPC code and the proposed LCPC code are used for comparisons in this section. The first discrimination is that, the complexity of the proposed LCPC code is lower than LDPC code. The second difference is the number of iteration, in the proposed LCPC code no need to iteration in the decoding process. The third difference is the long latency of LDPC code compared with the proposed LCPC code in the decoding process that results from the longer codeword length and number of iteration. These differences make the proposed LCPC code is very useful in real time communications for the futuristic wireless networks applications, due to the lower complexity and latency.

The capability of the proposed LCPC code in error detection and correction is also studied. The minimum Hamming distance defined as $d_{min} = n - k$, where $d_{min} \geq 3$ is a positive integer. The number of errors that a block code can detect and correct is determined by its minimum Hamming distance d_{min}. This is defined as the minimum number of places where any two codewords differ. In general, the number of errors (u) that can be detected for a block code is $u = d - 1$. For example, at $m = 3$, the codeword length $n = 7$, message length $k = 4$ and $d_{min} = 3$, where t is the number of errors that a block code can correct $t = \lfloor (n - k)/2 \rfloor$. Since the Hamming code has a minimum Hamming distance $d_{min} = 3$, it can only correct 1 bit error for each 7 bits transmitted. Therefore, the error correction is $1/7 = 14.285\ \%$.

Likewise, in the case of RS codes, the number and type of errors that can be corrected depend on the characteristics of the RS code. The RS code is specified as RS (n, k) with s-bit symbols. This means that the encoder takes k data symbols of s bits and adds parity symbols to make an $n = 2^s - 1$ symbol codeword. There are $n - k$ parity symbols of each s bits. The RS decoder can correct up to t symbols that contain errors in an error codeword, where $2t = n - k$. If $s = 3$ bits,

TABLE 2. Number of addition and multiplication operations for encoding and decoding stages of LCPC (9, 4) code.

Operation Type	Encoding	Detection	Correction	Total No. of Operation	Error States
No. of MUL (n)	4	-	-	4	No Error
No. of MUL (m)	-	9	-	9	
No. of ADD (n)	3	-	-	3	
No. of ADD (m)	-	8	-	8	
Complexity	$O(4n + 3n)$	$O(9m + 8m)$	-	$O(4n + 3n + 9m + 8m)$	$O(7n + 17m) = O(n)$
No. of MUL (n)	4	-	-	4	Single bit error or
No. of MUL (m)	-	9	-	9	Double bit errors with
No. of ADD (n)	3	-	1	4	one EP
No. of ADD (m)	-	8	-	8	
Complexity	$O(4n + 3n)$	$O(9m + 8m)$	$O(n)$	$O(4n + 4n + 9m + 8m)$	$O(8n + 17m) = O(n)$

TABLE 3. Comparison complexity of LDPC and LCPC codes.

Reference	Decoder Complexity	Note
(Davey and MacKay, 1998) [29]	$O(q^2)$	GF(q), $q = 64, 256$
(Barnault and Declercq, 2003)[30], (Chun-Hao et al., 2008) [31]	$O(q \log q)$	GF(q), $q = 64, 256$
(Declercq and Fossorier, 2007) [32]	$O(n_m q)$	$n_m \ll q$, $q = 64$, $n_m = 16$ or 32
(Voicila et al., 2010) [21]	$O(n_m \log n_m)$	$n_m \ll q$, $q = 64$, $n_m = 16$ or 32
(Xinmiao and Fang, 2011) [33]	$O(q^2 d_c)$	d_c is the check node degree for NB codes over GF(q)
(Proposed LCPC (9, 4) code)	$O(n)$	n is the codeword length

$n = 7$, and when the number of parity is 3, $k = 4$. The number of symbols containing errors that RS code can correct is t, where $t = \lfloor (n - k)/2 \rfloor$.

So, based on t value, the RS (7, 4) code can only correct one symbol error from the 7 codeword symbols sent. If the symbol size is 3 bits, the worst case happens only when a one-bit error occurs in separate symbols. In this case, the error correction is 1/21 = 4.7619 %, which is small compared with the percentage of error correction in the Hamming (7, 4) code, and this explains the reason why the Hamming (7, 4) code has a better BER performance when compared with the RS (7, 4) code, as shown in Fig. 4. The best case for RS (7, 4) code occurs when all bits in an one symbol are errors. This means that, the percentage of error correction is the number of errors in one symbol over the total number of symbol bits transmitted, i.e. (3/21 = 14.285 %).

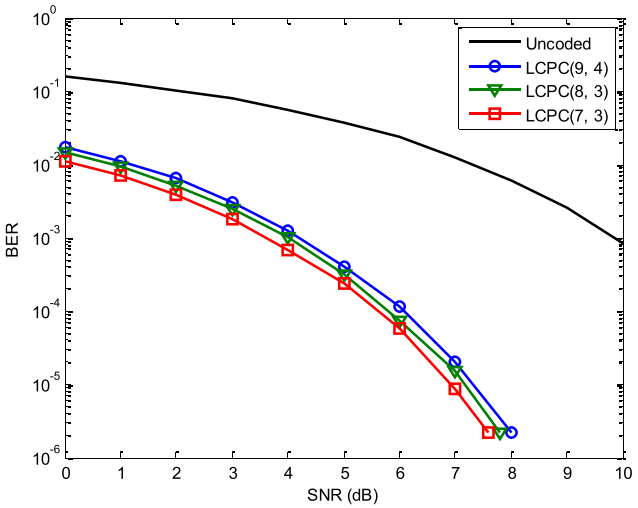

FIGURE 3. BER versus SNR for LCPC codes under AWGN and BPSK.

IV. SIMULATION RESULTS

In this section, simulation results of LCPC codes are presented. The simulations are carried out to validate the performance of the proposed LCPC codes using Binary Phase Shift Keying (BPSK) modulation over an Additive White Gaussian Noise (AWGN) and Rayleigh fading channels. The BER performance of different code rates is investigated. Simulation result as shown in Fig. 3 shows that the LCPC (7, 3) code has good BER performance compared with other codes over AWGN channel using BPSK modulation. Fig. 3 shows that the LCPC (9, 4) code provide BER = 10^{-5} at SNR 7.3 dB, 7.1 dB for LCPC (8, 3) and 6.9 dB for LCPC (7, 3). The opportunity of bit error decreases for the short codeword length, this explains the difference between the proposed LCPC codes.

The performance of the LCPC codes is compared with other codes, such as Hamming, BCH, RS and LDPC codes [34] using various values of codeword length. The Hamming, RS, BCH Soft, BCH Hard codes, and some decoding algorithms of the LDPC (8, 4) code such as bit flip, log domain and log domain simple are implemented using MATLAB.

Fig. 4 presents the comparison between LCPC, Hamming, RS, and BCH codes at codeword lengths (7, 4), over AWGN channels using BPSK modulation. Fig. 4 shows that the LCPC code improves the BER performance when compared with the Hamming, RS, and BCH codes.

Fig. 4 also shows that the LCPC (9, 4) code provide BER = 10^{-5} at SNR 7.3 dB, whereas the RS (7, 4) code provide the same BER (10^{-5}) in 9.4 dB SNR. The Hamming (7, 4) code provides the same BER (10^{-5}) in 9 dB SNR. Whereas, to obtain the same BER (10^{-5}), we need SNR equals to 8.1 dB for BCH Soft (7, 4) code, and 9.2 dB for BCH Hard (7, 4) code. The code gain that obtained in this case is 2.1 dB over the RS (7, 4), 1.7 dB over the Hamming (7, 4), 0.8 dB over BCH soft (7, 4) and 1.9 dB over BCH Hard (7, 4) codes. The LCPC codes have the capability to correct single bit and many

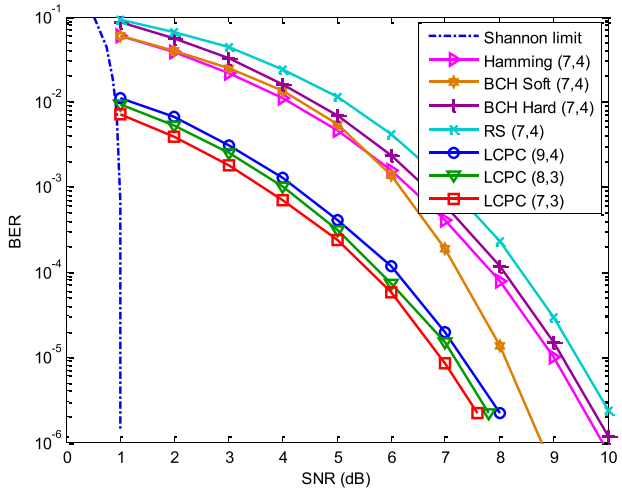

FIGURE 4. Comparison between LCPC (9, 4) code and other codes with Shannon limit over AWGN channel.

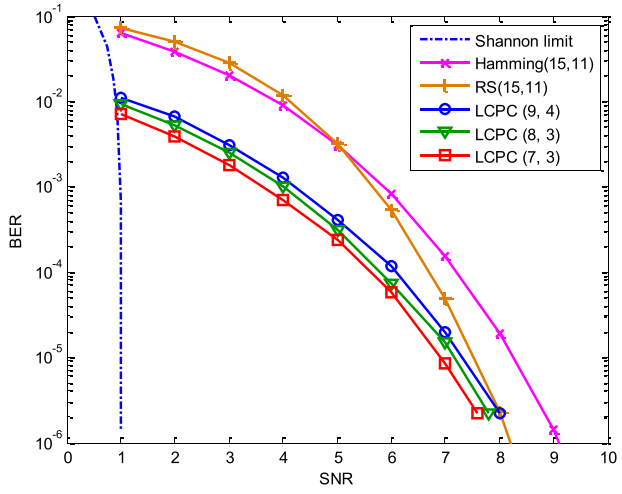

FIGURE 5. LCPC codes versus the Hamming (15, 11), and RS (15, 11) codes over AWGN channel.

cases of double bits error without the need for any iteration in the decoding process, whereas the Hamming (7, 4) code has the ability to correct one bit error and RS (7, 4) code has the ability to correct one bit error in each symbol.

The saving power is one of the benefits of LCPC codes. Therefore, the proposed codes can be effectively used in WSN because of its huge reduction in the power consumption, and saves the battery life to a large extent which is a very important consideration in WSN.

To investigate the BER performance of LCPC codes with the other codes that have codeword length greater than the codeword length of the LCPC codes, Fig. 5 shows the comparison of BER performance of the proposed LCPC (9, 4) and (8, 3) codes with Hamming (15, 11) and RS (15, 11) codes. Fig. 5 shows that the performance of LCPC codes is still better than the Hamming and RS codes, although the block length increases.

Similarly, Fig. 6 presents the BER performance comparison between LCPC (9, 4) code and binary LDPC (8, 4)

TABLE 4. Error Pattern and Syndrome Vector for single bit error of LCPC (9, 4) code.

Error Pattern EP	Syndrome Vector SY
000000000	0 0000
000000001	0 0001
000000010	0 0010
000000100	0 0100
000001000	0 1000
000010000	1 0000
000100000	1 0111
001000000	1 1011
010000000	1 1101
100000000	1 1110

TABLE 5. Error Pattern and Syndrome Vector for double bit errors of LCPC (9, 4) code.

Error Pattern EP	Syndrome Vector SY
000000011	0 0011
000000101	0 0101
000001001	0 1001
000010001	1 0001
000100001	1 0110
001000001	1 1010
010000001	1 1100
100000001	1 1111
000000110	0 0110
000001010	0 1010
000010010	1 0010
000100010	1 0101
001000010	1 1001
000001100	0 1100
000010100	1 0100
000100100	1 0011
000011000	1 1000
000110000	0 0111
001010000	0 1011
010010000	0 1101
100010000	0 1110

TABLE 6. Error Pattern and Syndrome Vector for single bit error of LCPC (8, 3) code.

Error Pattern EP	Syndrome Vector SY
00000000	0 0000
00000001	0 0001
00000010	0 0010
00000100	0 0100
00001000	0 1000
00010000	1 0000
00100000	0 1101
01000000	1 1011
10000000	1 0101

code with various decoding algorithms using BPSK modulation over AWGN channel. The binary LDPC (8, 4) code is implemented for three different types of decoding algorithms (i.e., Bit Flip, Log Domain, and Log Domain Simple). The performance of LCPC (9, 4) code is better than the binary LDPC code when the Bit Flip decoding method (BF) is used, and the coding gain is around 3 dB at BER $= 10^{-5}$. Whereas, the coding gain is 1 dB at BER $= 10^{-5}$ in case Log Domain decoding algorithm is used. The error correction capability of

TABLE 7. Error Pattern and Syndrome Vector for double bit errors of LCPC (8, 3) code.

Error Pattern EP	Syndrome Vector SY
00000011	0 0011
00000101	0 0101
00001001	0 1001
00010001	1 0001
00100001	0 1100
01000001	1 1010
10000001	1 0100
00000110	0 0110
00001010	0 1010
00010010	1 0010
00100010	0 1111
01000010	1 1001
10000010	1 0111
01000100	1 1111
00011000	1 1000
01001000	1 0011
10001000	1 1101
01010000	0 1011
01100000	1 0110
11000000	0 1110

TABLE 8. Error Pattern and Syndrome Vector for single bit error of LCPC (7, 3) code.

Error Pattern EP	Syndrome Vector SY
0000001	0001
0000010	0010
0000100	0100
0001000	1000
0010000	0011
0100000	0110
1000000	1100

TABLE 9. Error Pattern and Syndrome Vector for double bit errors of LCPC (7, 3) code.

Error Pattern EP	Syndrome Vector SY
0000101	0101
0001001	1001
0100001	0111
1000001	1101
0000110	0110
0001010	1010
0010010	0001
0100010	0100
1000010	1110
0001100	1100
1000100	1000
0011000	1011
0101000	1110
1001000	0100
1010000	1111
1100000	1010

binary LDPC code depends on the codeword length and the characteristic of the parity check matrix. The decoder gives a better performance with a larger codeword (i.e., big size of **G** and **H** matrices). The matrix multiplication for that larger codeword length demands large memory size, computational requirements and more complex decoding [5], [7].

In addition to the better performance of the BER, the main advantage of the proposed LCPC code is the low complexity

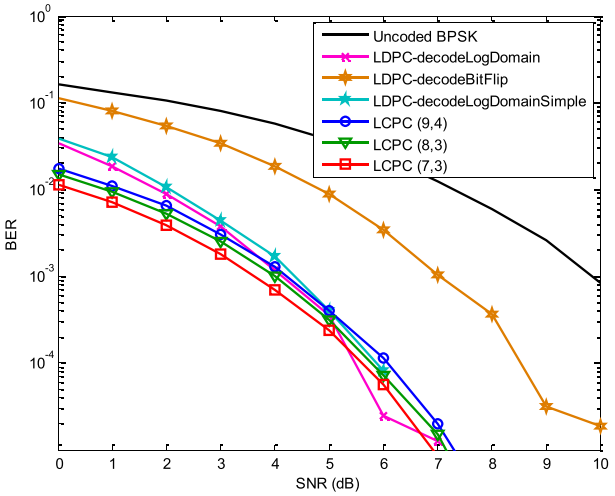

FIGURE 6. Comparison between LCPC (9, 4) code and binary LDPC (8, 4) with different types of decoding using BPSK over AWGN channel.

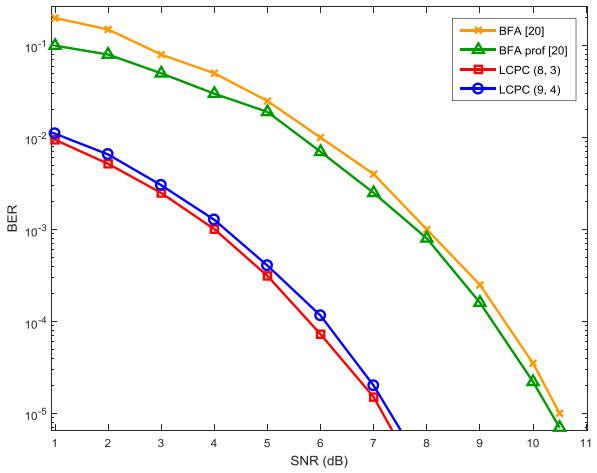

FIGURE 7. BER performance comparison between the proposed LCPC code and non-binary LDPC (7; 3; 4) code.

of encoding and decoding process when compared with the LDPC and RS codes. Furthermore, the proposed code does not require reiteration during the decoding of error correction, which is a salient feature of this code and is an important improvement over previous codes.

Fig.7 shows the comparison of BER performance of proposed LCPC (9, 4) and (8, 3) codes with non-binary LDPC (7; 3; 4) code in case of Bit-Flipping A algorithm (BFA) and Bit-Flipping A proposed probabilistic algorithm (BFA prob) is used [35], where block length (n =7), column weight ($c_w = 3$) and row weight($r_w = 4$). Fig. 7 shows that the LCPC (9, 4) code provides BER $= 10^{-5}$ at SNR 7.3 dB and 7.1 dB in case LCPC (8, 3), whereas the LDPC code provides the same BER (10^{-5}) in more than 10.5 dB SNR, in case the simulation BFA and BFAprob decoding algorithms is used. The code gain obtained is 3.2 dB and 3.4 dB for LCPC (9, 4) and LCPC (8, 3) codes, respectively.

Fig. 8 shows the BER performance comparison of LCPC (9, 4) and (8, 3) codes with non-binary LDPC codes

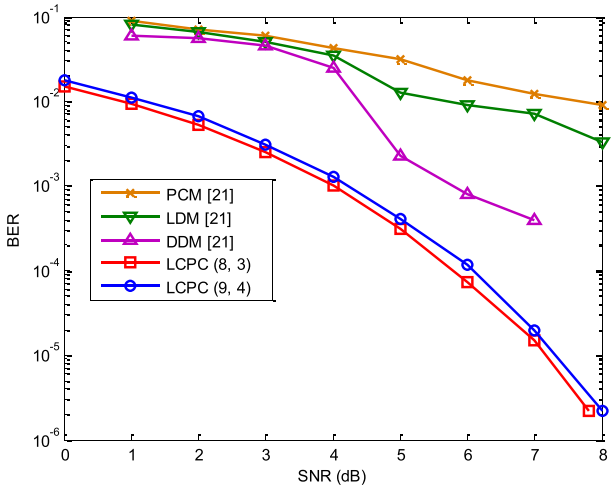

FIGURE 8. BER performance comparison between proposed LCPC codes and PCM, LDM and DDM for non-binary LDPC code GF(4), based on FFT-SPA decoding.

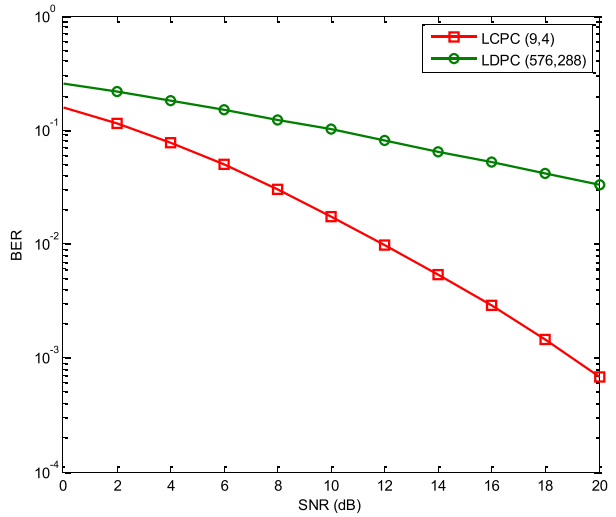

FIGURE 9. Comparison between LCPC (9, 4) code and LDPC (576, 288) at BPSK over Rayleigh fading channel at 41472 bit.

with Parity Check Matrix (PCM), Lower Diagonal based PCM (LDM) and Doubly Diagonal based PCM (DDM) for GF(4) [36]. Fig. 8 demonstrates that the BER performance of proposed LCPC codes is better than the codes presented in [36]. Fig. 8 shows that the LCPC (9, 4) code provides BER $= 10^{-5}$ at SNR 7.3 dB, and 7.1 dB for LCPC (8, 3) code, whereas the non-binary LDPC code with GF(4) with DDM provide the BER (0.5×10^{-3}) in 7 dB SNR. The non-binary LDPC code with LDM and PCM provides the BER (0.2×10^{-2}) and (10^{-2}) in 7 dB SNR, respectively [36].

The BER performance of the proposed LCPC codes over the Rayleigh fading channel using BPSK modulation is investigated. Fig. 9 shows the BER performance comparison between the LCPC (9, 4) code and LDPC (576, 288) with 41472 bits. Fig. 9 shows that the LCPC (9, 4) code outperforms the LDPC (576, 288) code over Rayleigh fading channel.

V. CONCLUSIONS

In this paper, a short codeword length approach has been proposed. The performance of the proposed Low Complexity Parity Check (LCPC) codes with BPSK modulation over AWGN and Rayleigh fading channels is investigated. The BER performance comparisons are made between the LCPC codes with Hamming, RS, BCH, binary and non-binary LDPC codes. The simulation results show significant enhancement in the BER performance of LCPC code as compared with the renowned LDPC, RS, BCH and Hamming codes. LCPC code has characteristics that distinguish it from LDPC codes such as; low complexity in the encoding and decoding processes, low memory size requirement (only 420 bits for LCPC (9, 4)), and no iterations in decoder process when compared with LDPC codes that need more than 20 times of iterations to correct the error codeword.

APPENDIX

G matrix for LCPC (8, 3)

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{15}$$

H matrix for LCPC (8, 3)

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

G matrix for LCPC (7, 3)

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{17}$$

H matrix for LCPC (7, 3)

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{18}$$

REFERENCES

[1] F. M. Al-Turjman, "Towards smart eHealth in the ultra large-scale Internet of Things era," in *Proc. Int. Iranian Conf. Biomed. Eng.*, Tehran, Iran, Nov. 2016, pp. 1–6.
[2] F. Al-Turjman, "Cognitive caching for the future sensors in fog networking," *Pervasive Mobile Comput.*, vol. 42, pp. 317–334, Dec. 2017, doi: 10.1016/j.pmcj.2017.06.004.
[3] F. Al-Turjman, "Impact of user's habits on smartphones' sensors: An overview," in *Proc. Int. IEEE Symp. HONET-ICT*, Kyrenia, Cyprus, Oct. 2016, pp. 70–74.
[4] F. Al-Turjman, "Hybrid approach for mobile couriers election in smart-cities," in *Proc. IEEE Local Comput. Netw. (LCN)*, Dubai, United Arab Emirates, Nov. 2016, pp. 507–510.
[5] E. Tsimbalo, X. Fafoutis, and R. J. Piechocki, "CRC error correction in IoT applications," *IEEE Trans. Ind. Informat.*, vol. 13, no. 1, pp. 361–369, Feb. 2017.
[6] J. Huang, S. Zhou, and P. Willett, "Near-Shannon-limit linear-time-encodable nonbinary irregular LDPC codes," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov. 2009, pp. 1–6.

[7] H. Zhong and T. Zhang, "Block-LDPC: A practical LDPC coding system design approach," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 4, pp. 766–775, Apr. 2005.
[8] C. A. Cole, S. G. Wilson, E. K. Hall, and T. R. Giallorenzi, "Regular 4, 8 LDPC codes and their lowerror floors," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, Oct. 2006, pp. 1–7.
[9] R. A. Carrasco and M. Johnston, *Non-Binary Error Control Coding for Wireless Communication and Data Storage*. Hoboken, NJ, USA: Wiley, 2009.
[10] Z. Su, Q. Qiu, and H. Zhou, "Analysis and elimination of short cycles in LDPC convolutional codes," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Chengdu, China, Oct. 2016, pp. 1128–1132.
[11] F. Wang, Y. Kou, M. Jiang, and Y. Xu, "Design of short quasi-cyclic LDPC codes for next generation broadcast wireless systems," in *Proc. 83rd IEEE Veh. Technol. Conf. (VTC Spring)*, Nanjing, China, May 2016, pp. 1–4.
[12] K. Vakilinia, T.-Y. Chen, S. V. S. Ranganathan, A. R. Williamson, D. Divsalar, and R. D. Wesel, "Short-blocklength non-binary LDPC codes with feedback-dependent incremental transmissions," in *Proc. IEEE Int. Symp. Inf. Theory*, Honolulu, HI, USA, Jun. 2014, pp. 426–430.
[13] C.-L. Wang, X. Chen, Z. Li, and S. Yang, "A simplified min-sum decoding algorithm for non-binary LDPC codes," *IEEE Trans. Commun.*, vol. 61, no. 1, pp. 24–32, Jan. 2013.
[14] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.
[15] A. Salbiyono and T. Adiono, "LDPC decoder performance under different number of iterations in mobile WiMax," in *Proc. Int. Symp. Intell. Signal Process. Commun. Syst. (ISPACS)*, Dec. 2010, pp. 1–4.
[16] M. Baldi, F. Chiaraluce, N. Maturo, G. Liva, and E. Paolini, "A hybrid decoding scheme for short non-binary LDPC codes," *IEEE Commun. Lett.*, vol. 18, no. 12, pp. 2093–2096, Dec. 2014.
[17] S. V. S. Ranganathan, D. Divsalar, and R. D. Wesel, "Design of improved quasi-cyclic protograph-based Raptor-like LDPC codes for short block-lengths," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017, pp. 1207–1211.
[18] T. V. Nguyen and A. Nosratinia, "Rate-compatible short-length protograph LDPC codes," *IEEE Commun. Lett.*, vol. 17, no. 5, pp. 948–951, May 2013.
[19] H. Xu, B. Bai, M. Zhu, B. Zhang, and Y. Zhang, "Construction of short-block nonbinary LDPC codes based on cyclic codes," *China Commun.*, vol. 14, no. 8, pp. 1–9, 2017.
[20] D. Deng, H. Xu, B. Bai, and J. Zhang, "A two-stage decoding algorithm for short nonbinary LDPC codes with near-ML performance," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017, pp. 1202–1206.
[21] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Low-complexity decoding for non-binary LDPC codes in high order fields," *IEEE Trans. Commun.*, vol. 58, no. 5, pp. 1365–1375, May 2010.
[22] L. Trifina, J. Ryu, and D. Tarniceriu, "Up to five degree permutation polynomial interleavers for short length LTE turbo codes with optimum minimum distance," in *Proc. Int. Symp. Signals, Circuits Syst. (ISSCS)*, Iasi, Romania, Jul. 2017, pp. 1–6.
[23] E. Cojocariu, D. Tarniceriu, L. Trifina, and A. G. Lazar, "Performance of asymmetric turbo codes on Rayleigh fading channels for small interleaver length," in *Proc. 3rd Int. Symp. Elect. Electron. Eng. (ISEEE)*, Galati, Romania, Sep. 2010, pp. 54–57.
[24] G. Liva, E. Paolini, B. Matuz, S. Scalise, and M. Chiani, "Short turbo codes over high order fields," *IEEE Trans. Commun.*, vol. 61, no. 6, pp. 2201–2211, Jun. 2013.
[25] M. Andrei, L. Trifina, and D. Tarniceriu, "Influence of trellis termination methods on turbo code performances," in *Proc. 4th Int. Symp. Elect. Electron. Eng. (ISEEE)*, Galati, Romania, Oct. 2013, pp. 1–6.
[26] Z. Yang, N. Jiang, K. Peng, and J. Wang, "High-throughput LDPC decoding architecture," in *Proc. Int. Conf. Commun., Circuits Syst.*, May 2008, pp. 1240–1244.
[27] N. Jiang, K. Peng, J. Song, C. Pan, and Z. Yang, "High-throughput QC-LDPC decoders," *IEEE Trans. Broadcast.*, vol. 55, no. 2, pp. 251–259, Jun. 2009.
[28] L. Zhao, R. Liu, Y. Hou, and X. Zhang, "High hardware utilization and low memory block requirement decoding of QC-LDPC codes," *Chin. J. Aeronautics*, vol. 25, no. 5, pp. 747–756, 2012.
[29] M. C. Davey and D. MacKay, "Low-density parity check codes over GF (q)," *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165–167, Jun. 1998.
[30] L. Barnault and D. Declercq, "Fast decoding algorithm for LDPC over GF (2^q)," in *Proc. Inf. Theory Workshop*, Mar./Apr. 2003, pp. 70–73.
[31] C.-H. Liao, C.-Y. Wang, C.-H. Liu, and T.-D. Chiueh, "An O(qlogq) log-domain decoder for non-binary LDPC over Gf(q)," in *Proc. IEEE Asia Pacific Conf. Circuits Syst. (APCCAS)*, Nov./Dec. 2008, pp. 1644–1647.
[32] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over GF(q)," *IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633–643, Apr. 2007.
[33] X. Zhang and F. Cai, "Reduced-complexity decoder architecture for non-binary LDPC codes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 7, pp. 1229–1238, Jul. 2011.
[34] C.-Y. Chen, Q. Huang, C.-C. Chao, and S. Lin, "Two low-complexity reliability-based message-passing algorithms for decoding non-binary LDPC codes," *IEEE Trans. Commun.*, vol. 58, no. 11, pp. 3140–3147, Nov. 2010.
[35] N. Miladinovic and M. P. C. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 4, pp. 1594–1606, Apr. 2005.
[36] S. Aruna and M. Anbuselvi, "FFT-SPA based non-binary LDPC decoder for IEEE 802.11 n standard," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2013, pp. 566–569.

SALAH ABDULGHANI ALABADY received the Ph.D. degree in computer engineering/wireless networks from the School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Pulau Pinang, Malaysia, in 2014. From 1999 to 2010, he was a Lecturer with the Computer Engineering Department, University of Mosul, Iraq, where he is currently a Senior Lecturer. From 2011 to 2014, he was a Research Assistant in wireless networks with the School of Electrical and Electronic Engineering, Universiti Sains Malaysia. His research interests include wireless channel coding, joint channel–network coding, error correction codes, and cross layer of wireless sensor networks.

FADI AL-TURJMAN received the Ph.D. degree in computing science from Queen's University, Canada, in 2011. He is a leading authority in the areas of smart/cognitive, wireless and mobile networks' architectures, protocols, deployments, and performance evaluation. He is also a Professor with the Computer Engineering Department, Antalya Bilim University, Turkey. His record spans over 170 publications in journals, conferences, patents, books, and book chapters, in addition to numerous keynotes and plenary talks at flagship venues. He is the sole author for three recently published books about cognition and wireless sensor networks' deployments in smart environments with Taylor and Francis, CRC New York (a top tier publisher in the area). He is the Publication Chair at the IEEE International Conference on Local Computer Networks (LCN'18). He is serving as the Lead Guest Editor in several journals, including the *IET Wireless Sensor Systems*, *Sensors* (MDPI), and Wiley.