COPYRIGHT STATEMENT

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior consent.
Improving Group Integrity of Tags in RFID Systems

by

Reem Ibrahim AlKanhel

A thesis submitted to the University of Plymouth

in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Engineering, Computing and Mathematics

December 2019
Acknowledgements

I would like to express my deep and sincere gratitude to my Director of Studies, Dr. Marcel Adrian Ambroze. His professionalism, experience, logical way of thinking, and guidance has been invaluable during the course of my PhD.

I would also like to express my sincere gratitude to my other supervisors, Prof. Ingo Stengel, and Prof. Nathan Clarke, for their excellent advices, proof reading, and constructive comments that have been very useful to this study.

My thanks also go to Dr Jalal Almuhtadi for his advice during the course of my PhD.

Many thanks go to Ubaid Fayaz for his efforts with Matlab implementations.

Special thanks to my husband for all his advice, guidance and sacrifices that he has made. Without him, I will not be able to reach this far.

Special thanks also go to my best friend Eatedal for her support.

A great deal of appreciation goes to my beloved family members-my mother, daughters, son, sisters, and brothers for their love and support over the years. I am grateful that these people had faith in me and my abilities and have always been by my side throughout my studies.
Author’s Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award without prior agreement of the Doctoral College Quality Sub-Committee.

Work submitted for this research degree at the University of Plymouth has not formed part of any other degree either at the University of Plymouth or at another establishment.

This study was financed with the aid of a scholarship from Princess Nourah bint Abdulrahman University at the Kingdom of Saudi Arabia.

Relevant scientific seminars and conferences were attended at which work was often presented and several papers were published and prepared for publication.

Word count of main body of thesis: 48009 words

Signed: ...............Reem.........................

Date: ...............1/12/2019.........................
Abstract

Improving Group Integrity of Tags in RFID Systems

Reem Ibrahim AlKanhel

Checking the integrity of groups containing radio frequency identification (RFID) tagged objects or recovering the tag identifiers of missing objects is important in many activities. Several autonomous checking methods have been proposed for increasing the capability of recovering missing tag identifiers without external systems. This has been achieved by treating a group of tag identifiers (IDs) as packet symbols encoded and decoded in a way similar to that in binary erasure channels (BECs). Redundant data are required to be written into the limited memory space of RFID tags in order to enable the decoding process. In this thesis, the group integrity of passive tags in RFID systems is specifically targeted, with novel mechanisms being proposed to improve upon the current state of the art.

Due to the sparseness property of low density parity check (LDPC) codes and the mitigation of the progressive edge-growth (PEG) method for short cycles, the research is begun with the use of the PEG method in RFID systems to construct the parity check matrix of LDPC codes in order to increase the recovery capabilities with reduced memory consumption. It is shown that the PEG-based method achieves significant recovery enhancements compared to other methods with the same or less memory overheads. The decoding complexity of the PEG-based LDPC codes is optimised using an improved hybrid iterative/Gaussian decoding algorithm which includes an early stopping criterion. The relative complexities of the improved algorithm are extensively analysed and evaluated, both in terms of decoding time and the number of operations required. It is demonstrated that the improved algorithm considerably reduces the operational complexity and thus the time of the full Gaussian decoding algorithm for small to medium amounts of missing tags.

The joint use of the two decoding components is also adapted in order to avoid the iterative decoding when the missing amount is larger than a threshold. The optimum value of the threshold value is investigated through empirical analysis. It is shown that the adaptive algorithm is very efficient in decreasing the average decoding time of the improved algorithm for large amounts of missing tags where the iterative decoding fails to recover any missing tag. The recovery performances of various short-length irregular PEG-based LDPC codes constructed with different variable degree sequences are analysed and evaluated. It is demonstrated that the irregular codes exhibit significant recovery enhancements compared to
the regular ones in the region where the iterative decoding is successful. However, their performances are degraded in the region where the iterative decoding can recover some missing tags.

Finally, a novel protocol called the Redundant Information Collection (RIC) protocol is designed to filter and collect redundant tag information. It is based on a Bloom filter (BF) that efficiently filters the redundant tag information at the tag’s side, thereby considerably decreasing the communication cost and consequently, the collection time. It is shown that the novel protocol outperforms existing possible solutions by saving from 37% to 84% of the collection time, which is nearly four times the lower bound. This characteristic makes the RIC protocol a promising candidate for collecting redundant tag information in the group integrity of tags in RFID systems and other similar ones.
# Table of Contents

List of Figures .......................................................................................................................... xi  
List of Tables .......................................................................................................................... xiv  
List of Algorithms ................................................................................................................... xv  
List of Acronyms ..................................................................................................................... xvi  

1 Introduction and Overview ................................................................................................. 1  
   1.1 Introduction ................................................................................................................... 1  
   1.2 Aims and Objectives .................................................................................................... 7  
   1.3 Contributions to Knowledge ....................................................................................... 8  
   1.4 Methodology ................................................................................................................ 10  

2 An Overview of Group Integrity Methods in RFID Systems .............................................. 12  
   2.1 Literature Review ......................................................................................................... 12  
   2.2 Conclusions .................................................................................................................. 20  

3 RFID Systems ..................................................................................................................... 21  
   3.1 RFID System Components ......................................................................................... 21  
   3.1.1 RFID Tags ............................................................................................................... 22  
   3.1.1.1 Types of RFID Tags .......................................................................................... 22  
   3.1.1.2 Types of Memory in RFID Tags ....................................................................... 23  
   3.1.1.3 Classes of RFID Tags ....................................................................................... 25  
   3.1.2 RFID Readers .......................................................................................................... 26  
   3.2 RFID Standards .......................................................................................................... 27  
   3.3 Operating Frequencies ............................................................................................... 28  
   3.4 Data Coding ................................................................................................................. 29  
   3.5 Multi-access Procedures (Anti-collision) .................................................................... 30  
   3.5.1 Space Division Multiple Access Technique .......................................................... 30  
   3.5.2 Frequency Division Multiple Access Technique ..................................................... 31
3.5.3 Code Division Multiple Access Technique ......................................................... 31
3.5.4 Time Division Multiple Access Technique ......................................................... 31
  3.5.4.1 Aloha-based Protocols ..................................................................................... 32
  3.5.4.2 Tree-based Protocols ..................................................................................... 38
3.6 Conclusions .............................................................................................................. 41

4 LDPC Codes .................................................................................................................. 43
  4.1 Preliminaries and Notations ...................................................................................... 44
  4.2 Encoding Process ...................................................................................................... 47
  4.3 Decoding Algorithms ............................................................................................... 50
    4.3.1 Iterative Decoding Algorithms ........................................................................... 52
    4.3.2 Maximum Likelihood Decoding Algorithms ....................................................... 54
    4.3.3 Reduced Complexity Decoding Algorithms ....................................................... 57
  4.4 Construction Methods for LDPC Codes .................................................................... 59
    4.4.1 Random Constructions ...................................................................................... 59
      4.4.1.1 Gallager Codes ............................................................................................... 59
      4.4.1.2 MacKay Codes ............................................................................................. 61
      4.4.1.3 Progressive Edge Growth (PEG) Methods .................................................... 61
      4.4.1.4 Approximate Cycle Extrinsic Message Degree (ACE) Methods ....................... 63
    4.4.2 Algebraic Constructions ...................................................................................... 64
      4.4.2.1 Quasi-Cyclic Codes ....................................................................................... 64
      4.4.2.2 Euclidean Geometry Codes ......................................................................... 65
      4.4.2.3 Chinese Remainder Theorem (CRT) Methods ............................................... 66
    4.4.3 Main Comparisons of Construction Methods ..................................................... 67
  4.5 Conclusions .............................................................................................................. 68

5 Extended Grouping of RFID Tags Based on PEG Methods ......................................... 70
  5.1 System Model .......................................................................................................... 70
  5.2 Encoding Process ..................................................................................................... 70
  5.3 Decoding Process ..................................................................................................... 72
  5.4 Performance Evaluation .......................................................................................... 74
    5.4.1 Using the GE and IT Decoding Algorithms ......................................................... 75
    5.4.2 PEG-based vs CRT-based LDPC Codes ............................................................. 83
5.4.3 PEG-based vs ACE-based LDPC Codes .................................................. 85
5.5 Conclusions .............................................................................................. 86

6 Improved Hybrid IT/GE Decoding Algorithm ........................................ 87
   6.1 Early Termination of the IT Decoding Algorithm ............................. 87
   6.2 Missing Recovery Capability Analysis .......................................... 92
   6.3 Decoding Complexity Analysis ......................................................... 93
     6.3.1 Decoding Time Analysis .............................................................. 93
     6.3.2 Operational Complexity Analysis ............................................ 97
       6.3.2.1 IT Decoding Algorithm Analysis ...................................... 97
       6.3.2.2 GE Decoding Algorithm Analysis ...................................... 97
       6.3.2.3 Improved Hybrid Decoding Algorithm Analysis ............... 99
   6.4 Conclusions ......................................................................................... 101

7 Adaptive Hybrid Decoding Algorithm ................................................. 103
   7.1 Threshold Analysis ........................................................................... 103
   7.2 Irregular PEG-based LDPC Codes ................................................... 106
   7.3 Conclusions ......................................................................................... 112

8 Bloom Filters .................................................................................... 114
   8.1 Preliminaries of Bloom Filters .............................................................. 114
   8.2 Bloom Filter Features ........................................................................ 118
   8.3 Bloom Filter Limitations ..................................................................... 119
   8.4 Bloom Filter Variants .......................................................................... 120
     8.4.1 Counting Bloom Filters ............................................................... 120
     8.4.2 $d$-Left Counting Bloom Filters ............................................... 121
     8.4.3 Space Code Bloom Filters ........................................................... 122
     8.4.4 Scalable Bloom Filters ................................................................. 122
     8.4.5 Weighted Bloom Filters ............................................................... 123
   8.5 Applications of Bloom Filters ............................................................ 123
     8.5.1 Filtering Duplicate Data from RFID Data Streams .................... 125
   8.6 Conclusions ......................................................................................... 129
9 Efficient Redundant Information Collection Protocol .................................. 131

9.1 Tag Information Collection Protocols ................................................. 132

9.2 System Model and Problem Statement ............................................. 136

9.3 Performance Lower Bound ............................................................... 137

9.4 Possible Solutions ............................................................................. 137

9.4.1 Baseline Collection Protocol ......................................................... 138

9.4.2 ID Collection Protocol ................................................................. 138

9.4.3 Multi-Hash Information Collection Protocol ................................. 139

9.5 Redundant Information Collection Protocol ....................................... 140

9.5.1 Protocol Description ....................................................................... 141

9.5.2 Performance Analysis ................................................................. 144

9.5.3 Cardinality Estimation ................................................................. 146

9.5.4 Hash Functions ............................................................................ 147

9.6 Simulation Results ........................................................................... 147

9.6.1 Simulation Settings ....................................................................... 148

9.6.2 Collection Time Comparison Under Different Numbers of Tags and Information Lengths ................................................................. 148

9.6.3 Collection Time Comparison Under Different Settings of the False Positive Rate 151

9.7 Conclusions ..................................................................................... 152

10 Conclusions and Future Work .............................................................. 153

10.1 Contributions and Achievements of the Research ............................ 153

10.2 Limitations of the Research ............................................................. 156

10.3 Further work .................................................................................. 157

References .............................................................................................. 159

Appendices ............................................................................................. 175

Appendix A-Some Source Codes of the Simulations ................................. 175

Appendix B-Some Constructed LDPC-Based Codes .................................. 185
List of Figures

Figure 1.1: The main components of an RFID system (Jones and Chung, 2007) ...................... 1
Figure 1.2: Example scenario of a group integrity checking process (Sato et al., 2012) ............ 2
Figure 1.3: The three main phases of the complete system .................................................. 7
Figure 2.1: A hierarchical structure of three tags (Ban’atre et al., 2010) ............................... 12
Figure 2.2: Memory space organisation of the leaves (Sinha et al., 2014) .............................. 13
Figure 2.3: 4-regular graph representation of a group of eight tags (Glouche and Couderc, 2012) .............................................................................................................................. 14
Figure 2.4: Chain of tag IDs modelling (Sayers et al., 2010) ................................................. 15
Figure 2.5: Group of pallets modelling (Sayers et al., 2010) ................................................... 16
Figure 2.6: Generation and distribution of a codeword for a group of tags (Chaves et al., 2013) ........................................................................................................................................ 16
Figure 2.7: Checking group integrity using ECCs (Chaves et al., 2013) ................................... 17
Figure 3.1: RFID tags in various shapes and sizes (Coresonant, 2014) ................................. 22
Figure 3.2: RFID system standards and frequency bands (Knospe and Pohl, 2004) ............... 27
Figure 3.3: Data coding in RFID systems (Finkenzeller, 2010) ............................................. 29
Figure 3.4: Collision behaviour of NRZ coding (Finkenzeller, 2010) ....................................... 41
Figure 3.5: Collision behaviour of Manchester coding (Finkenzeller, 2010) .......................... 41
Figure 4.1: The Tanner graph illustration of $H$ in Equation (4.1) (Ryan, 2003) .................. 46
Figure 4.2: The parity check matrix in the approximate lower-triangular form (Richardson and Urbanke, 2001) ........................................................................................................ 48
Figure 4.3: An example of a stopping set (Di et al., 2002) .................................................... 54
Figure 4.4: Reduced complexity ML decoder on $H_k$ (Paolini et al., 2012) ......................... 56
Figure 5.1: An example of the encoding process and the information stored in RFID tags belonging to a group of 6 RFID tags .................................................................................. 72
Figure 5.2: An example of the decoding process of a group of 6 RFID tags .......................... 73
Figure 5.3: The IT decoding algorithm in the group integrity of RFID tags ......................... 74
Figure 5.4: Flowchart of the simulation process ..................................................................... 75
Figure 5.5: The average error rate of the identification of missing tag IDs from different group sizes: (a) $n = 25$; (b) $n = 49$; (c) $n = 121$; (d) $n = 169$, using the PEG-based LDPC codes with the IT and GE decoding algorithms ........................................................................................................... 77
Figure 5.6: The average error rate of the identification of missing tag IDs from a group of size $n = 1000$ using the PEG-based codes with the IT decoding algorithm..........................80
Figure 5.7: The average error rate of the identification of missing tag IDs from a group of size $n = 1000$ using the PEG-based codes with the GE decoding algorithm.........................80
Figure 5.8: The histogram of unrecovered missing tag IDs of the (200, 3, 6) PEG-based LDPC codes with the GE decoding algorithm.................................................................81
Figure 5.9: The average error rate of the identification of missing tag IDs from a group of size $n=169$, where $j=4$ and the related decoding time for both the IT and GE decoding algorithms ..................................................................................................................82
Figure 5.10: The average error rate of the identification of missing tag IDs from a group of size $n = 512$ using the PEG-based and CRT-based codes with the IT decoding algorithm.....84
Figure 5.11: The average error rate of the identification of missing tag IDs from a group of size $n = 512$ using the PEG-based and CRT-based codes with the GE decoding algorithm...84
Figure 5.12: The average error rate of the identification of missing tag IDs from a group of size $n = 502$ using the PEG-based and ACE-based codes with the IT and GE decoding algorithm ..................................................................................................................85
Figure 6.1: The iteration numbers required for decoding two group sizes: (a) $n = 49$; (b) $n = 169$, where $j = 3$, using the PEG-based LDPC codes .................................................................89
Figure 6.2: The enhanced iteration numbers required for decoding two group sizes: (a) $n = 49$; (b) $n = 169$, where $j = 3$, using the PEG-based LDPC codes .......................................................91
Figure 6.3: The average error rate of the identification of missing tag IDs from different group sizes: (a) $n = 25$; (b) $n = 49$ using the PEG-based LDPC codes with the GE and improved HB decoding algorithm ..................................................................................................92
Figure 6.4: The average error rate of the identification of missing tag IDs from different group sizes: (a) $n = 121$; (b) $n = 169$ using the PEG-based LDPC codes with the GE and improved HB decoding algorithm ........................................................................................................93
Figure 6.5: Histogram of the GE decoding time with respect to the simplified system size for a group of size $n = 169$, where $j = 3$, using the PEG-based LDPC codes.............................95
Figure 6.6: The average decoding time for recovering missing tag IDs from two group sizes: (a) $n = 49$ and (b) $n = 169$, where $j = 3$, using the PEG-based LDPC codes under the IT with early termination, the GE, and improved HB decoding algorithms .............................96
Figure 6.7: The average decoding time for recovering missing tag IDs from two group sizes: (a) $n = 49$ and (b) $n = 169$, where $j = 4$, using the PEG-based LDPC codes under the IT with early termination, the GE, and improved HB decoding algorithms .............................96
Figure 6.8: Operational complexity comparison between the IT, GE, and improved HB decoding algorithms for a group of \( n = 49 \), where (a) \( j = 3 \) and (b) \( j = 4 \). ........................................... 101
Figure 6.9: Operational complexity comparison between the IT, GE, and improved HB decoding algorithms for a group of \( n = 169 \), where (a) \( j = 3 \) and (b) \( j = 4 \). ........................................... 101
Figure 7.1: Reduction in the number of missing tags with iterations for \( n = 169 \) tags, where \( j = 3 \) ................................................................................................. 104
Figure 7.2: The average decoding time for recovering missing tag IDs from two group sizes: (a) \( n = 49 \) and (d) \( n = 169 \), where \( j = 3 \), using the PEG-based method under the IT with early termination, the GE, and improved HB decoding algorithms .................................................. 105
Figure 7.3: The average error rate of the identification of missing tag IDs for irregular and regular PEG-based codes with \( n = 256 \) under the IT decoding algorithm ........................................... 107
Figure 7.4: The average error rate of the identification of missing tag IDs for irregular and regular PEG-based codes with \( n = 512 \) under the IT decoding algorithm ........................................... 108
Figure 7.5: Reduction in the number of missing tags with iterations for: (a) regular and (b) irregular PEG-based LDPC code with \( n = 256 \) ................................................................................................. 108
Figure 7.6: Influence of changing the fractions of low degree variable nodes on the performance of half rate short-length irregular PEG-based LDPC codes with \( n = 256 \) .................. 109
Figure 7.7: Influence of changing high degree variable nodes on the performance of half rate short-length irregular PEG-based LDPC codes with \( n = 256 \) ................................................................................................. 110
Figure 7.8: Influence of changing the fractions of high degree variable nodes on the performance of half rate short-length irregular PEG-based LDPC codes with \( n = 256 \) .................. 111
Figure 8.1: An example of a Bloom filter ................................................................................................. 115
Figure 8.2: An example of a CBF ............................................................................................................. 121
Figure 8.3: Basic duplicate readings filtering (Yaacob and Mahdin, 2016) ........................................... 125
Figure 8.4: Duplicate filtering method of an RFID data stream (Kamaludin et al., 2016) ....... 126
Figure 9.1: An example of the tag identification process with a BF in Yue et al. (2014) ....... 134
Figure 9.2: An illustrative example of the MIC protocol (Chen et al., 2011) ........................................ 140
List of Tables

Table 3.1: RFID system features at available frequencies.................................................................28
Table 3.2: The number of tags versus the optimal frame size and the number of groups in the EDFSA (Lee et al., 2005) ......................................................................................................................................................38
Table 4.1: the average number of recovered erasures for some analysed codes in Tomlinson et al. (2007)..................................................................................................................................................................51
Table 5.1: Comparisons of the PEG-based LDPC codes with other LDPC-based codes........78
Table 5.2: Comparisons of the PEG-based and CRT-based LDPC codes.................................83
Table 5.3: Comparisons of the PEG-based and ACE-based LDPC codes ...............................85
Table 6.1: Number of operations for the improved HB decoding algorithm.......................99
Table 7.1: The values of θ_{th} for different parameters of the PEG method.........................105
Table 7.2: Variable degree sequences for irregular codes in Figure 7.6...............................109
Table 7.3: Variable degree sequences for the irregular codes in Figure 7.7..........................110
Table 7.4: Variable degree sequences for the irregular codes in Figure 7.8..........................112
Table 9.1: Estimation time comparison (in second) under different group sizes ............146
Table 9.2: Collection time comparison (in seconds) for different parity check values m ....149
Table 9.3: Collection time comparison (in seconds) for the same parity check values m ....150
Table 9.4: Collection time comparison (in second) under different values of P_{FP}, where n = 2,000 ........................................................................................................................................................................152
List of Algorithms

Algorithm 4.1: Construction of the parity check matrix using the PEG method .................. 63
Algorithm 6.1: Improved IT Decoding ............................................................................. 90
Algorithm 9.1: Constructing the BF ................................................................................. 142
Algorithm 9.2: Membership testing .................................................................................. 143
Algorithm 9.3: The RIC protocol ....................................................................................... 143
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>Approximate Cycle Extrinsic Message Degree</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>BC</td>
<td>Baseline Collection</td>
</tr>
<tr>
<td>BEC</td>
<td>Binary Erasure Channel</td>
</tr>
<tr>
<td>BER</td>
<td>Bit Error Rate</td>
</tr>
<tr>
<td>BF</td>
<td>Bloom Filter</td>
</tr>
<tr>
<td>BIC</td>
<td>Bloom-filter-based Information Collection</td>
</tr>
<tr>
<td>CBF</td>
<td>Counting Bloom Filter</td>
</tr>
<tr>
<td>CDMA</td>
<td>Code Division Multiple Access</td>
</tr>
<tr>
<td>CRT</td>
<td>Chinese Remainder Theorem</td>
</tr>
<tr>
<td>ECC</td>
<td>Error Correction Code</td>
</tr>
<tr>
<td>EDFSA</td>
<td>Enhanced Dynamic Framed Slotted Aloha</td>
</tr>
<tr>
<td>EG</td>
<td>Euclidean Geometry</td>
</tr>
<tr>
<td>EMD</td>
<td>Extrinsic Message Degree</td>
</tr>
<tr>
<td>EPC</td>
<td>Electronic Product Code</td>
</tr>
<tr>
<td>ETOP</td>
<td>Enhanced Tag Ordering Polling</td>
</tr>
<tr>
<td>FDMA</td>
<td>Frequency Domain Multiple Access</td>
</tr>
<tr>
<td>FER</td>
<td>Frame Error Rate</td>
</tr>
<tr>
<td>FSA</td>
<td>Framed Slotted Aloha</td>
</tr>
<tr>
<td>GE</td>
<td>Gaussian Elimination</td>
</tr>
<tr>
<td>GF</td>
<td>Galois Field</td>
</tr>
<tr>
<td>HB</td>
<td>Hybrid</td>
</tr>
<tr>
<td>HF</td>
<td>High Frequency</td>
</tr>
<tr>
<td>ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standard Organization</td>
</tr>
<tr>
<td>IT</td>
<td>Iterative</td>
</tr>
<tr>
<td>LDPC</td>
<td>Low Density Parity Check</td>
</tr>
<tr>
<td>LF</td>
<td>Low Frequency</td>
</tr>
<tr>
<td>LLR</td>
<td>Log-Likelihood Ratio</td>
</tr>
<tr>
<td>MF</td>
<td>Microwave Frequency</td>
</tr>
<tr>
<td>MIC</td>
<td>Multi-hash Information Collection</td>
</tr>
<tr>
<td>ML</td>
<td>Maximum Likelihood</td>
</tr>
<tr>
<td>ML-CCBF</td>
<td>Multi-Layer Compressed Counting Bloom Filter</td>
</tr>
<tr>
<td>NACK</td>
<td>Negative Acknowledgement</td>
</tr>
<tr>
<td>NRZ</td>
<td>Non-Return to Zero</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PC</td>
<td>Physical Characteristic</td>
</tr>
<tr>
<td>PEG</td>
<td>Progressive Edge-Growth</td>
</tr>
<tr>
<td>PIC</td>
<td>Perfect-hashing-based Information Collection</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>RHS</td>
<td>Right Hand Side</td>
</tr>
<tr>
<td>RIC</td>
<td>Redundant Information Collection</td>
</tr>
<tr>
<td>RS</td>
<td>Reed-Solomon</td>
</tr>
<tr>
<td>RTD</td>
<td>Resolvable Transversal Design</td>
</tr>
<tr>
<td>RTF</td>
<td>Reader-Talk-First</td>
</tr>
<tr>
<td>RZ</td>
<td>Return to Zero</td>
</tr>
<tr>
<td>SA</td>
<td>Slotted Aloha</td>
</tr>
<tr>
<td>SCBF</td>
<td>Space Code Bloom Filter</td>
</tr>
<tr>
<td>SDMA</td>
<td>Space Division Multiple Access</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>SIC</td>
<td>Single-hash Information Collection</td>
</tr>
<tr>
<td>SSF</td>
<td>Strongly Selective Family</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time Domain Multiple Access</td>
</tr>
<tr>
<td>TOP</td>
<td>Tag Ordering Polling</td>
</tr>
<tr>
<td>TTF</td>
<td>Tag-Talk-First</td>
</tr>
<tr>
<td>UHF</td>
<td>Ultra-High Frequency</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
</tbody>
</table>
Chapter 1 - Introduction and Overview

1 Introduction and Overview

1.1 Introduction

Radio frequency identification (RFID) is a technology that uses radio waves to identify and track objects automatically through radio waves. Typically, an RFID system is composed of tags, readers, and database servers. A tag is a small device that is attached to a physical object and stores information about the object in its user memory. In addition to the user memory, each tag has a special memory that stores the 96-bit tag identifier (ID), which uniquely identifies the physical object. The reader gathers information from multiple tags at a time through multiple access technique, then relaying them to the back-end database servers for further analysis and processing (Nambiar, 2009). A block diagram of such a system is shown in Figure 1.1. Tags come in three types, including passive, active, and semi-passive tags. Passive tags are cheaper, smaller in size, and have longer life than the other types of tags and hence, they are widely used in different applications. On the other hand, they have smaller memory size, only storing up to 1 kilobytes of information in commonly used models (Roberti, 2011).

![Figure 1.1: The main components of an RFID system (Jones and Chung, 2007)](image)

Figure 1.1: The main components of an RFID system (Jones and Chung, 2007)
RFID technology has been used in many application domains, including security, inventory, logistics, and public transportation. Many activities need the integrity verification of a set of objects in addition to individual object checking. The basic idea of a group integrity verification, is to check that a group of tagged objects, parts, animals, or people remains consistent during some process; whilst also determining the missing tag IDs, if they occur. In logistics, it is a common practice that items form a group and are handled together. It is also common to check personal belongings at airports to avoid loss or to prevent similar possessions being wrongly exchanged with those of someone else. A further example is when a group of children requires to be regularly checked during a school excursion to ensure that no child is missing. Figure 1.2 demonstrates an example scenario of such a process.

Verifying the integrity of a received group of shipments, for example, is usually achieved by checking the shipment list, where the reader requires access to on-line databases. There are other methods for accomplishing verification of a group of tags. Juels (2004) studied the integrity of two RFID tags from the security perspective and presented yoking-proof, whereby a message authentication code is exchanged between the two tags and a verifier. This was enhanced and generalised for a group of tags in studies by Saito and Sakurai (2005),
Liu et al. (2013), Sundaresan et al. (2014), and Shen et al. (2015). Whilst all the proposed grouping-proofs provide integrity evidence for complete groups of tags, they do not address incomplete ones. Potdar et al. (2007) presented an integrity checking method which compares the total weight of a group of tags recorded in a database and their reading results.

Autonomous verification of a group of RFID tags without external systems is an essential feature in RFID systems, because it ensures standalone operations as well as scalability, privacy, and security. Banâtre et al. (2009) introduced the concept of “coupled objects” to describe groups of objects in autonomous group integrity checking, being a group of physical objects that represents a logical group. An essential characteristic of a coupled object is that the group information is self-contained within the objects, such that the group verification process is possible without using extra systems and hence, additional data are written into the user memory of RFID tags. A simple method to represent the coupled object or a group of objects would be to let each tag of a group store the set of IDs of all members. However, given that the tag ID is of size 96 bits and the passive tags have limited memory space, this approach is not scalable to large groups. Another method is to use a digest value to represent a group of tags (Banâtre et al., 2009). For a group of $n$ tags, group creation includes the following steps:

1) The reader collects the tag IDs denoted by $TID_1$, $TID_2$, $TID_3$, ..., $TID_n$ from the group and orders them sequentially;

2) A hash function is applied to the collected tag IDs to obtain the digest value as hash $(TID_1, TID_2, TID_3, ..., TID_n)$ and the resulting digest value is treated as a group ID;

3) The reader records the group ID in the user memory of the tags in the group.

Once a group is formed, group integrity checking includes the following steps:

1) The reader collects the tag IDs and the stored group IDs from a group of tags having the same group ID;
2) The collected tag IDs are ordered sequentially;

3) The same hash function of the group creation is applied over the collected tag IDs;

4) The computed hash value is compared with the stored group ID. If there is a mismatch, the group integrity is invalid, meaning that one or more tags are missing; otherwise, the group preserves its integrity.

Even though Ban’atre et al.’s method enables fully autonomous group integrity checking, it is not able to recover the missing tag IDs. A number of methods have been proposed to solve this issue by writing additional data into the user memory of the RFID tags, examples being, each tag in the group stores the tag IDs of some members that are arranged as chains (Sayers et al., 2010), regular connected graphs (Glouche and Couderc, 2012), or tree structures (Sinha et al., 2014). The robustness of these methods is strengthened by increasing the amount of redundant tag IDs stored among tags at the expense of memory costs. As a result, Sato, Igarashi, et al. (2012) presented another method in which a group of tag IDs is treated as packet symbols, which are encoded and decoded in a way similar to that of binary erasure channels (BECs). Due to the sparseness property of low density parity check (LDPC) codes as well as their good erasure recovery capabilities, the parity check matrix of LDPC codes is used as a group generator matrix to divide the main group into multiple overlapping subgroups virtually, henceforth referred to as the LDPC-based method. Each tag records all the parity check values in which they are involved. Therefore, tags of the same row of the matrix maintain the same set of information. The authors used the Gallager method (Gallager, 1962) to construct the parity check matrix and since the matrix is constructed randomly, two or more missing tags may have identical memberships. In other words, they belong to stopping sets, which degrades the performance of the method in recovering missing tag IDs with iterative decoding. To overcome this shortcoming, three nonrandom methods were utilised in the construction of the parity check matrices of the LDPC codes, aimed at increasing the recovery performance (Su et al., 2013; Su and Tonguz, 2013; Su, 2014).
performance of the existing LDPC-based methods can be increased by increasing the number of subgroups which each RFID tag belongs to, but at the expense of the tag’s user memory. Su (2014) introduced a method to reduce the amount of information stored in a tag’s user memory; however, it causes higher decoding complexity.

For the group integrity checking and decoding processes, the reader needs to collect the redundant information from all the tags in a group efficiently, because RFID systems have subsequent processes to perform. When the time is long, the task of collecting tag information may delay these processes of the system, which is undesirable, especially in large-RFID environments. Many existing RFID studies mainly focus on proposing ID collection protocols that collect tag IDs in the interrogation zone of the reader. These protocols are mainly classified into ALOHA-based (Cha and Kim, 2006; Vogt, 2002a) and tree-based (Law et al., 2000; Hush and Wood, 1998) protocols. In ALOHA-based protocols, the reader sends a request message to the tags. Then, each tag transmits its ID to the reader at randomly chosen time slots. Consequently, more than one tag may possibly transmit their IDs at the same time slot; in this case, the time slot becomes a collision slot. On the other hand, if only one tag transmits its ID at a certain time, the tag is successfully identified. In tree-based protocols, collisions are reduced by building tree traversal approaches. When a collision occurs, the reader divides the tags into two subgroups and requests the subgroups with a smaller number of tags. The reader continues to divide the tags until all the tags are identified. Whilst the ID collection protocol can be used to collect tag information, the transmission overhead and time delay could be excessively high for redundant tag information.

In sensor-augmented RFID systems, tags are combined with miniaturised sensors to provide information about the surrounding environment. Collecting such information from tags with low communication costs and thus, short time spans, has inspired many studies in recent
Chapter 1 - Introduction and Overview

years. Chen et al. (2011) proposed the Single-hash Information Collection (SIC) protocol and the improved version, the Multi-hash Information Collection (MIC) protocol, to periodically collect information from sensor-based RFID systems with a minimum collection time. By utilising several hash functions, the MIC protocol can resolve most hash collisions in a frame and thus, significantly enhance the protocol performance. Qiao et al. (2011; 2016) studied the same problem, but with respect to energy efficiency for active tags. The tag ordering polling protocol and the improved protocol were presented to read information from tags with the smallest amount of energy consumption. A follow-up work by Yue et al. (2012) complemented previous work with an additional protocol that is primarily designed to read sensor-produced information from RFID tags with multiple readers. The extended study in Yue et al. (2014) targeted an unknown subset of tags with one reader. The proposed protocols utilised Bloom filters that are distributively built and transferred from the tags to the reader to identify the target tags efficiently. The reader can then build a hash vector from a frame of time slots to link each identified tag to each slot in order to send its information. In sensor-based RFID systems, the tags have no knowledge about the information the reader has and should send all their information.

Many methods in the literature filter out the duplicate information in RFID systems (Mahdin and Abawajy, 2011; Jiang et al., 2012; Mezzi and Akaichi, 2013). Duplicate data are produced because an RFID reader reads the same tag multiple times, or the same tag is read by multiple readers. All these methods concentrated on filtering duplicate information at the reader’s side before the information is transmitted to back-end database servers for processing, which causes communication overheads between the reader and the tags. The majority of the proposed methods are based on Bloom filters. Generally, the received data are maintained in Bloom filters to eliminate duplicates and thus, reduce memory and communication consumptions.
Motivated by the need to increase the missing recovery capabilities of the checking methods with low memory and decoding overheads; and to read redundant tag information from a group of tags with low communication overheads, several methods are designed in this thesis aimed at increasing the functionality of the complete system of group integrity application in resource constrained environments of RFID systems. Figure 1.3 shows the three main phases of the complete system.

![Figure 1.3: The three main phases of the complete system](image)

### 1.2 Aims and Objectives

This research is devoted to the development of group integrity application in RFID systems. The aim is to propose novel mechanisms to improve the performance in recovering missing tag IDs with low memory consumptions and decoding complexity, whilst also collecting redundant tag information from a group of tags with low communication overheads. In order to achieve these, the following research objectives are established:

1) To review existing autonomous group integrity checking methods and define their limitations;
2) To investigate RFID systems with a focus on important features directly related to the thesis;
3) To investigate LDPC codes in BECs with special attention to their construction methods and decoding algorithms;
4) To design and evaluate a new method, with the aim of increasing the missing recovery capabilities with low memory overheads;
To design, analyse, and evaluate an optimised decoding algorithm, with the aim of achieving high recovery performance with low decoding complexity;

To review existing tag information filtering and collection protocols, with a focus on their designs and limitations;

To investigate a Bloom filter with a focus on its features, limitations, and the ways in which the Bloom filter has been utilised in some systems, especially in RFID ones;

To design and evaluate a novel tag information collection protocol that is used to collect redundant tag information from a group of tags in a group integrity application with low communication overheads.

1.3 Contributions to Knowledge

The following list summarise the main contributions made by the thesis.

1) Using the progressive edge-growth (PEG) method to increase the recovery capability of missing tags in group integrity applications of RFID systems. It has been found that the recovery performance of the method outperforms existing methods, with either the same or less memory overheads.

2) Optimising the decoding complexity using an improved hybrid decoding algorithm which includes an early stopping criterion to avoid unnecessary iterations of iterative decoding for undecodable blocks, thus saving decoding time. This improved algorithm has achieved the optimal missing recovery capabilities of the full Gaussian elimination decoding algorithm.

3) Conducting extensive analysis and evaluations of the improved hybrid decoding complexity, both in terms of the number of operations required and the decoding time. It has been found that the improved hybrid decoding algorithm considerably reduces the operational complexity and also the time of the full Gaussian decoding algorithm for
small to medium amounts of missing tags up to a certain threshold value, which depends on the missing recovery capabilities of the iterative decoding algorithm.

4) Designing an adaptive algorithm to avoid the iterative decoding when the missing amount is larger than a threshold. It has been found that the adaptive decoding algorithm is very efficient in decreasing the average decoding time of the improved hybrid decoding, thus saving more decoding time of up to 10% for large amounts of missing tags, where the iterative decoding fails to recover any missing tags.

5) Conducting numerous simulations to analyse and evaluate the missing recovery performance of various half rate short-length irregular PEG-based LDPC codes constructed with different variable degree sequences. It has been found that the irregular codes achieve better missing recovery than regular codes only in the region where the iterative decoding is successful and the density evolution method is good for the irregular PEG construction.

6) Designing a novel tag information collection protocol based on a Bloom filter to filter redundant tag information at the tag's side efficiently and thus, saving collection time. The designed protocol includes a method for estimating the number of the target parity check values. The performance of the designed protocol outperforms the existing solutions by saving from 37% to 84% of the collection time.

Several papers have been published and submitted for publication, including:


1.4 Methodology

A schematic diagram of the research methodology is presented in Figure 1.4 to show its phases for achieving the thesis objectives and the relationships between these objectives and the chapters. The thesis is organised into ten chapters. References are located at the end of this thesis, together with two appendices containing some source codes of the simulations and some constructed LDPC-based codes.
Aim: to improve the recovery performance of missing tag IDs with low memory consumption and decoding complexity; and to collect redundant tag information from a group of tags with low communication overheads.

Objectives:
1) To review existing autonomous group integrity checking methods and define their limitations.
2) To investigate RFID systems.
3) To investigate LDPC codes in BECs.
4) To design and evaluate a new method, with the aim of increasing the missing recovery capabilities with low memory overheads.
5) To design, analyse, and evaluate an optimised decoding algorithm, with the aim of achieving high recovery performance with low decoding complexity.
6) To review existing tag information filtering and collection protocols.
7) To investigate Bloom filters.
8) To design and evaluate a novel tag information collection protocol to collect redundant tag information with low communication overheads.

Reviewing the existing autonomous group integrity checking methods of RFID tags to provide an insight into the designs, developments, and limitations.

Investigating RFID systems, including major components, the standards, mode of communications between the reader and tags, anti-collision protocols, and important features directly related to the thesis.

Investigating LDPC codes in BECs with a focus on the construction methods, encoding process, and the most prominent decoding algorithms.

• Designing a new method that is used to construct the parity check matrix of LDPC codes to increase the missing recovery capabilities with low memory overheads.
• Testing and evaluating the designed method with Matlab R2015a & R2018b simulations: the codes of the method are constructed using MainPEG.C (Hu, 2008). Appendix A contains some source codes of the simulations.

• Improving the decoding algorithm with a hybrid decoding which includes a mechanism to speed up the decoding time with high recovery performance.
• Analysing, testing, and evaluating (with Matlab) the complexity of the improved decoding algorithm, both in terms of the decoding time and the number of operations required.

• Designing an adaptive algorithm to speed up further the decoding time.
• Testing and evaluating (with Matlab) the decoding time of the adaptive algorithm.

• Reviewing the existing tag information filtering protocols, with a focus on their designs and limitations.
• Investigating a Bloom filter with a focus on its construction, features, limitations, variations, and the ways in which the Bloom filter has been utilised in some systems, especially in RFID ones.

• Reviewing the existing tag information collection protocols with a focus on their designs and limitations.
• Designing a novel tag information collection protocol to collect redundant tag information from a group of tags in a group integrity application with low communication overheads.
• Testing and evaluating the designed protocol (with Matlab R2018b: the simulation settings follow the specifications of the EPCglobal Class-1 Generation-2 standard).

Contributions and achievements of the research.
Limitations and future work.

Figure 1.4: A schematic diagram of the research methodology
Chapter 2 - An Overview of Group Integrity Methods in RFID Systems

2 An Overview of Group Integrity Methods in RFID Systems

A number of researchers have proposed autonomous methods for group integrity checking without external systems, by writing additional data into the limited memory space of RFID tags. These data reference other tags or represent their membership in the group. This chapter provides a literature review of different autonomous group integrity checking methods of RFID tags, with a focus on their design and limitations.

2.1 Literature Review

Banâtre et al. (2009) used a digest value to represent a group of tags logically, as presented in Chapter 1. Later, in 2010 and 2011, the authors extended the basic principle to have objects belonging to several groups in a hierarchical structure. Instead of having a flat addressing scheme where all tags are present at the same level, hierarchical data structures or coupled object trees allow for the independent checking of each tree or subtree. Each tagged object stores the hash value that corresponds to the group or the level to which the object belongs. Figure 2.1 represents two groups O₁ and O₂, where O₁ consists of two tags f₁ and f₂, whereas O₂ consists of the group O₁ = {f₁, f₂} and the tag f₃.

![Figure 2.1: A hierarchical structure of three tags (Banat et al., 2010)](image-url)
Whilst Ban’atre et al.’s method enables autonomous group integrity checking, it cannot recover the missing tag IDs and for this reason, some methods have been proposed to overcome this. Sinha et al. (2014) utilised a tree structure to check the integrity of composite objects, with hierarchical representation of the tree being distributed among its leaf nodes. In other words, the data about internal nodes are stored on an RFID tag’s memory attached to the leaf. Figure 2.2 shows an example of Sinha et al.’s model, where each leaf node records three types of information, including its sibling IDs, its parent ID, its ancestor node ID, and associated child IDs. The distributed information forms a logical link between leaves, which helps to assemble the composite object, check the integrity of the structure, and identify missing tags. When missing all leaves belonging to any subtree, Sinha et al’s model is not able to identify missing tags, because each leaf node stores two sets of data only. One set is associated with its siblings and is used to identify missing siblings, whilst the other is related to its ancestor subtree and is used to construct the tree. The robustness of the method can be increased by increasing the redundancy of the information stored in the tags at the expense of memory costs.

![Figure 2.2: Memory space organisation of the leaves (Sinha et al., 2014)](image)
Glouche and Couderc (2012) proposed a different method in which a set of tags is arranged as a regular connected graph. Each vertex of the graph is represented by a tag, which records the graph structure, that is, the IDs of its neighbours as shown in Figure 2.3. The presence of each tag is evenly recorded on the other tags so the same amount of memory is consumed in each. When a tag is missing, it can be recovered using the information stored in its neighbours. In a $k$-regular graph, each tag stores the IDs of its $k$ neighbours. When, at most, $k$ tags are missing, the structure guarantees that the IDs of all the missing tags will be recovered. However, when more than $k$ tags are missing, only $k$ of them can be recovered.

The amount of memory required to record the IDs of the neighbours is a multiple of $k$ and is proportionally increased when $k$ increases. The amount of missing tags that can be recovered is increased by increasing the degree of the graph, which affects the number of IDs to be recorded on tags. The authors, interestingly, make the following comment in their paper:

“Unfortunately, as RFID tags provide very limited memory capacity, it is difficult to implement redundant storage and robust data structures distributed over a set of RFID tags”

Figure 2.3: 4-regular graph representation of a group of eight tags (Glouche and Couderc, 2012)
Sayers et al., (2010) proposed a chain of IDs to check the integrity of a group of tags on a pallet. For a group of tags, each stores the IDs of the next two tags in the sequence. This ensures the redundancy of the IDs, which helps to recover missing tags. For example, the ID of tag C in Figure 2.4 is redundantly stored in the previous two tags, that is, tag A and tag B. In order to enable the system to differentiate between real missing items and those that are present, but cannot be read while the items are on the pallet, the physical characteristics (PC), such as the weight or total number of items are also stored on the tags and pallets. For example, the weight of the pallet is compared with either PC_p stored on tag D or stored on tag P. If there is a mismatch, this guarantees that one or more tags are missing. In addition, the physical characteristics can be used for an individual tag to verify its authenticity. A disadvantage of this method is that when more than two consecutive tags are missing, it cannot detect and recover some missing tag IDs.

![Figure 2.4: Chain of tag IDs modelling (Sayers et al., 2010)](image)

To increase the robustness of the system, Sayers et al. (2010) also stored the IDs of a given pallet across multiple pallets, as illustrated in Figure 2.5, where ID_{PA}, for example, indicates the IDs for all the tags on pallet A. The robustness of the method depends on the amount of the redundant data as this consumes more memory spaces especially with a large group of tags.
Chaves et al., (2013) proposed another method using Error Correction Codes (ECCs) applied to a group of tag IDs. In a set of tags, all the IDs are assembled to a message that is encoded using an ECC and then, the codeword is split up into a number of parts. Each part is stored on each tag in the group. The codeword is stored with an address to ensure that is assembled in a specific order, as shown in Figure 2.6. The EPC in the figure is an abbreviation for “The Electronic Product Code”, which is equivalent to the tag ID used in this thesis.

In a checking process, all available portions of the codeword are assembled based on their addresses. The recovery of the missing portions depends on the remaining parts and the type of ECC used, with Figure 2.7 depicting an example of the checking process. Whilst Chaves et al.’s method presents a compact representation of the redundancy, missing a chain of consecutive tags may become an obstacle to the recovery process.
Another direction of the research in this field was proposed in Sato et al. (2011; 2012), in which a group of tag IDs is treated as packet symbols and encoded and decoded in a way similar to that of binary erasure channels (BECs). Due to the sparseness property of low density parity check (LDPC) codes as well as their good erasure recovery capabilities, the parity check matrix of LDPC codes is constructed with the Gallager method and then used as a group generator matrix to divide the main group into multiple overlapping subgroups virtually. For a group of tags, the group ID is calculated as the bitwise exclusive or XOR of the hashed tag IDs. In addition to the group ID, each tag stores all subgroup IDs, that is, the parity check values to which it belongs. Subgroup IDs are calculated in the same way as the group ID. The number of missing IDs is estimated as the rank of a matrix containing missing tag IDs. Then, Sato, Igarashi, et al. (2012) extended the previous method to recover the IDs of missing tags. In the extended method, also called extended grouping, extra information is added to the tag’s memory in addition to subgroup IDs. The subgroup IDs are computed as the XOR of tag IDs, and therefore, the bit length of the subgroup IDs is equal to the bit length of the tag IDs, i.e. 96 bits. In addition, a short ID that is a portion of the tag ID is assigned to each tag. The bit length of each short ID is set to the minimum number of bits necessary to distinguish a group of tags. For example, a short ID of 8 bits is enough for a group less than
256 tags. Each tag stores all short IDs of other tags belonging to the same subgroup (parity check constraint). The short IDs are used to re-establish the Tanner graph during the decoding process and thus, recover the IDs of the missing tags. During the decoding process, missing tag IDs are computed as the XOR of the subgroup IDs and the known IDs of all tags belonging to the same subgroup. It has been assumed that a situation where all the tags belonging to one subgroup are missing does not occur. The procedure of the group decoding is similar to the iterative decoding algorithm in the BEC. The properties of the matrices used for the encoding, including the way of division, the number of subgroups which each tag belongs to, and the number of tags in each subgroup, are important, because they have a direct relationship with the recovery performance and the amount of information stored in the tag’s memory. Because the Gallager’s parity check matrix is constructed randomly, the method fails to resolve the exact amount of missing tags and recover some missing tag IDs when they belong to stopping sets. This degrades the performance of the method in resolving the accurate amount of missing tags or recovering missing tag IDs with the iterative decoding.

To overcome the above shortcoming, three nonrandom methods were utilised in the construction of the parity check matrices of LDPC codes, aiming to maximise the performance in resolving the accurate number of missing IDs or recovering more. Su et al. (2013) constructed the matrices based on the properties of strongly selective families (SSFs) whereas Su and Tonguz (2013) utilised the redundancy property of the remainder presentation in the Chinese remainder theorem (CRT). Su (2014) also used resolvable transversal designs (RTDs) to construct the matrices. The author introduced a method called group splitting, which splits rows of the parity check matrix into smaller ones to improve the recovery performance of the proposed method and to reduce the amount of information stored in the tag’s memory. However, group splitting causes higher decoding complexity. The performance of the aforementioned LDPC-based methods can be enhanced by increasing
Chapter 2 - An Overview of Group Integrity Methods in RFID Systems

the number of subgroups that each RFID tag belongs to (i.e. the column weight of the parity check matrix), but at the expense of the tag’s memory, as concluded by their authors. For example, the SSF-based method can recover up to five missing tag IDs by writing 528-bit additional information into the user memory of each tag in a group of 121. Increasing the memory overhead to 704-bit, the largest number of missing tag IDs that can be recovered by the method is increased to 13.

All the LDPC-based methods aim to provide equal missing protection for all tags in a group, as each tag is included in the same number of subgroups. However, there are some cases where not all tags need the same protection level. For instance, when tracking laundry with RFID tags, expensive clothes are considered to be more important than others. This application requires unequal missing protections amongst the tags. Accordingly, Su and Wang (2015) and Su et al. (2015) introduced the concept of unequal missing protection for the design of grouping of RFID tags. The design is based on joining some group generator matrices with equal missing protection. The amount of protection levels that the new matrix offers depends on the joint sub-matrices and the number of overlapping columns. Only the tags associated with the overlapping columns have the highest protection levels with additional stored information and thus, more memory overheads.

From the security perspective, Juels (2004) presented a yoking-proof protocol of two RFID tags, which verifies the simultaneous presence of two tags. A message authentication is exchanged between the two tags and a verifier. Saito and Sakurai (2005) developed an improved version to overcome the vulnerability of replay attack. They also generalised the problem of the yoking-proof to a grouping-proof, which verifies the presence of a group of tags in the range of a reader. Later, several protocols were presented aiming to improve the security of grouping-proof protocols (Liu et al., 2013; Sundaresan et al., 2014; Shen et al., 2015), which use an off-line or on-line verifier to generate a proof. However, tags and
Chapter 2 - An Overview of Group Integrity Methods in RFID Systems

Verifiers must previously share a secret key. Despite all the proposed grouping-proofs providing integrity evidence for complete groups of tags, they do not address incomplete ones. In particular, they do not give any information about missing tags. Following the same principles of Sato, Igarashi, et al. (2012), Burmester and Munilla (2016) utilised Reed-Solomon (RS) erasure codes to identify the missing tags with a grouping-proof protocol. More precisely, they used $RS(n, k)$ code to encode the IDs of a group of tags. The concatenation of tag IDs is rearranged into blocks and then encoded to get an RS codeword, which is then split into blocks and stored in the memory of each tag in the group. RS decoding can only be performed, if the scanned blocks are ordered correctly with gaps for the missing blocks. For this purpose, control information is needed to ensure the order of tag IDs at the time of encoding. The authors highlighted how the optimal code is costly when the group of tags is large. Therefore, the number of tags and missing tags is assumed to be small.

2.2 Conclusions

It can be seen that many autonomous methods have been proposed for checking the integrity of a group of tags in RFID systems without external systems. The compact representation of group members in LDPC-based methods is of significant importance to accommodate the limited memory space of passive RFID tags. The performance of the methods in recovering missing tag IDs can be enhanced by increasing the amount of redundant information at the expense of memory costs. Even though there is an effort to reduce memory consumption, the decoding complexity is increased accordingly. Having the same memory consumptions, the performance of the existing LDPC-based methods is mainly affected by the structure of the parity check matrix used for the encoding. More precisely, the presence of stopping sets limits the recovery capability of the iterative decoding.
3 RFID Systems

RFID is an automatic technology that enables the identification and tracking of objects through radio waves (Bolic et al., 2010). It has been successfully used in different applications, including supply chain management, objects tracking management, production process control, agriculture management, transportation, healthcare and medicine, and services to name a few. The main advantage of RFID systems is the automated identification and data capture, which has brought significant improvements to resource optimisation, end-user enhancements, and process efficiency across a wide range of business activities (Jia et al., 2012). This chapter presents an overview of RFID systems, including major the components, its standards, mode of communications between the readers and the tags, anti-collision protocols, and important features directly related to the thesis.

3.1 RFID System Components

A typical RFID system consists of three main components: RFID tags, readers, back-end database servers. The tag is a small device that is attached to an object as the identifier of the object. It stores and transmits information to readers through radio waves. The reader generates magnetic fields that enable the RFID system to locate tagged objects that are within its field. The reader gathers information from multiple tags and relays them to the back-end database servers for further analysis and processing (Nambiar, 2009). The back-end database server is used to store and process the information centrally. The back-end database server uses the gathered information in the field by the readers to perform different functions, such as verifying identities and granting authorisation, keeping inventories, alerting suppliers when new inventories are required, tracking the movement of items, and redirecting them (Hunt et al., 2007). Unlike typical RFID systems, the autonomous group integrity verification of this
thesis eliminates the use of the back-end database servers to ensure standalone operations as well as scalability, privacy, and security.

3.1.1 RFID Tags

An RFID tag, also known as a transponder, is composed of an antenna, a microchip that gives the tag some limited functionality, and a substrate or printed circuit board. Information is received and transmitted by the antenna, while the chip stores and processes the information. The printed circuit board holds the tag parts together. Generally, the chip is used to store data about an object the tag is attached to. The object is provided with a unique identification number, which, along with other related information, such as product type and manufacturing date are stored in the chip (Brown, 2007). The size of the tags is determined by the size of their antenna since the microchip is usually very small in size. Tags are available in many different shapes, sizes, and protective housings depending on the application and the environment in which they will be used. Figure 3.1 demonstrates the different shapes and sizes of RFID tags.

![Figure 3.1: RFID tags in various shapes and sizes (Coresonant, 2014)](image)

3.1.1.1 Types of RFID Tags

Three types of RFID tags are in use, including active, passive and semi-passive ones, which are described below (Ahson and Ilyas, 2008).
Active tag: It contains its own power source in the form of a battery or a solar cell, which is used to power up its microchip and transmits the signal to the RFID reader. The active tag has a larger memory up to 128 kilobytes (Roberti, 2011), longer read range up to 100 metres, and greater processing power than other types of tags. On the other hand, the larger size and higher cost limit their popularity. These tags can be ideal for asset management in large systems.

Passive tag: It does not need any power source. It derives the power from the signal transmitted by the RFID reader and thus, there is a constraint on the read range of passive tags to receive a strong signal from the reader. The read range of passive tags is limited to 7 metres and typically, they can store from 64 bits to 1 kilobyte of data (Roberti, 2011). However, new passive tags can store up to 64 kilobytes of data in new Fujitsu models (Fujitsu, 2014). Passive tags are less expensive, smaller in size, and have a longer life than active ones and therefore, they are widely used in different applications. The focus of this research is devoted to this type of tag, where the limited memory space is of high consideration.

Semi-passive tag: It can also be called a semi-active tag and has a power source similar to active ones. This power source is used to power up the circuit on its microchip only. Semi-passive tags transmit the signal using the same way as that of passive tags, but at a higher rate. The overall features of semi-passive tags are located in between those of passive and active ones.

3.1.1.2 Types of Memory in RFID Tags

Tags come in different types based on their memory capabilities as follows (Finkenzeller, 2010).


**Read-only tag:** The information is permanently stored on the tags during manufacture or at the user’s site. The communication between the tag and the reader is unidirectional from the former. If a read-only tag is placed in the reader region, the tag starts to broadcast its ID continuously. It is important to ensure that there is only one tag in the reader region, otherwise two or more tags will send their IDs simultaneously, which results in a collision and thus, the tags are unidentified by the reader. Despite this limitation, this type of tag is generally inexpensive and consumes low power. It also suits some applications where one tag ID is transmitted such as the identification of pallets, containers or animals. Read-only tags can operate at all frequencies of RFID systems.

**Read-write tag:** It is occupied by a writable memory, such as an electrically erasable programmable read-only memory (EEPROM) for passive tags and static random-access memory (SRAM) for active ones. The user can change the tag ID and add additional information to the tag’s memory by the reader. This gives the user the flexibility to store and process information and therefore, eliminates the requirement for the back-end database server which is applicable for the autonomous operation of group integrity application. Read-write tags can process reader commands as well as support anti-collision and cryptological procedures. Similar to read-only tags, they can operate at all frequencies of RFID systems. However, they are more expensive and consume more power than read-only tags.

The RFID tags have four memory banks, as described in the EPCglobal Class-1 Generation-2 standard (EPCglobal, 2015):

**Reserved memory:** It stores the access and kill password and cannot store data besides these two passwords. The role of the 32-bit access password is to enable and disable the tag’s write capabilities, whereas the 32-bit kill password permanently disables the tag. This memory bank is read and write protected, only being writable if a certain password needs to be specified.
**EPC memory:** It has a minimum of 96 bits of writable memory and it stores the 96-bit tag ID called the EPC, which uniquely identifies the physical object. The EPC memory also stores a 16-bit cyclic redundancy check code computed over the contents of the EPC code, and some control bits.

**Tag manufacture information memory:** It stores the information that describes the tag itself, including the manufacturer ID and a code that specifies the tag model. It may also contain information that describes tag capabilities.

**User memory:** It comes blank from the manufacturer and enables users to store additional data of different sizes based on the type of the tags (see subsection 3.1.1.1). This area of memory is used to store the information needed by the autonomous group integrity verification of this thesis.

### 3.1.1.3 Classes of RFID Tags

RFID Tags are constructed to comply with a specification called a class. The EPCglobal organisation describes six tag classes ranging from 0 to 5 that increasingly have greater capabilities and more features. These classes are grouped into generations. Generation-1 includes classes 0 and 1 and defines the physical and logical requirements for read-only passive tags. Generation-2 defines the physical and logical requirements for RFID system operating in the 860–960 MHz frequency range and includes classes 1–5. A general description of these classes is as follows (EPCglobal, 2015):

**Class-0 and class-1 Generation-1:** Provide the basic identification capability for read-only passive tags. Class 0 is for factory programmed tags, whereas class 1 is for user-programmable tags after manufacture;
Chapter 3 - RFID Systems

Class-1 Generation-2: Defines read-write passive tags. The simulation settings in Chapter 9 follow the specifications of this class;

Class-2: Additional functionalities are added to read-write passive tags, including encryption and access control. It is not yet completed;

Class-3: Provides longer range and broadband communications for semi-passive tags that are powered by on-board batteries. It is not yet fully defined;

Class-4: Provides peer-to-peer communications and additional sensing for active tags. It is still in the early definition stage;

Class-5: Defines active tags that contain enough power to activate other tags and communicates with devices other than readers. It is also still in the early definition stage.

3.1.2 RFID Readers

An RFID reader is also called an interrogator since it queries the tags when they enter its range. It is an electronic device that can be used as a standalone or be integrated with other devices. The reader consists of a high frequency (HF) interface that has a transmitter and receiver, control unit, and one or more antennas. The antenna can be integrated into the reader, or it can be a separate device. Larger systems usually separate the antennas from the reader, whereas handheld units combine the two. The HF interface generates power to activate the tag, and sends and receives information from it. The control unit codes and decodes the signal received from the tag. It also uses a microprocessor to control the communication with the tag (Finkenzeller, 2010). In more complex RFID systems, the control unit usually performs some complex processes, including the execution of an anti-collision scheme, encryption and decryption of the transmitted data as well as authentication between tags and readers (Jia et al., 2012). In addition, the control unit of this research needs to perform the decoding and filtering processes. The reader may be self-contained and store
the data internally or it may be a part of other systems such as a Wide Area Network (WAN). There is also a middleware, which controls the reader as well as the information from the tags. It forwards the gathered data to the database systems. It performs basic processes including integration, filtering, and controlling of the reader (Hunt et al., 2007).

3.2 RFID Standards

RFID systems have a significant number of associated standards, with Figure 3.2 demonstrating the most relevant ones. The majority are defined by the International Standard Organisations (ISO) and the EPCglobal organisation. The standards cover three key areas of RFID applications and uses, including:

1) Air interface;
2) Conformance and interoperability between applications and RFID systems;
3) Data encoding and contents.

The purpose of RFID standards is to create a level of product consistency in the RFID industry and thus, enhance the efficiency of RFID systems, make the system more cost effective, and lead to industry development (Hunt et al., 2007). The simulation settings in Chapter 9 follow the specifications of the EPCglobal standard.

![Figure 3.2: RFID system standards and frequency bands (Knospe and Pohl, 2004)](image)
### 3.3 Operating Frequencies

A range of frequencies exists for the use of RFID systems and each of them has different capabilities. The use of tags, the read range needed, the material of the tagged item, and the type of surrounding environment determine the frequency required. Table 3.1 summarises the operating frequencies and their characteristics (Ward and Kranenburg, 2006; Hunt et al., 2007; Datta, 2016). Whilst the protocols for the communications at UHF are defined in the EPCglobal standard, as used for the current research, other frequencies are also applicable for autonomous group integrity verification.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Low Frequency (LF)</th>
<th>High Frequency (HF)</th>
<th>Ultra-High Frequency (UHF)</th>
<th>Microwave Frequency (MF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency range</td>
<td>125–134 KHz</td>
<td>13.56 MHz</td>
<td>860-960 MHz</td>
<td>2.45 or 5.8 GHz</td>
</tr>
<tr>
<td>Type of tag</td>
<td>Passive</td>
<td>Passive</td>
<td>Active and passive</td>
<td>Active and passive</td>
</tr>
<tr>
<td>Approximate read range</td>
<td>1–50cm</td>
<td>1.5m</td>
<td>4–7m for passive tags</td>
<td>1m for passive tags</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100m for active tags</td>
<td>100m for active tags</td>
</tr>
<tr>
<td>Data transfer rate</td>
<td>&lt; 10kbits/s</td>
<td>&lt; 100kbits/s</td>
<td>&lt; 100kbits/s</td>
<td>&lt; 200kbits/s</td>
</tr>
<tr>
<td>Tag cost</td>
<td>High</td>
<td>Lower than LF tags</td>
<td>Lowest</td>
<td>Highest</td>
</tr>
<tr>
<td>Typical RFID applications</td>
<td>Access control, animal and vehicle identification</td>
<td>Smart cards, baggage handling, item tracking, and libraries</td>
<td>Toll collection, pallet tracking, baggage handling, warehouse management, and supply chain management</td>
<td>Toll collection and real time item tracking</td>
</tr>
<tr>
<td>Performance near metal or wet surface</td>
<td>Best</td>
<td>Worst</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of tags read per second</td>
<td>Lowest</td>
<td>Highest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power consumption</td>
<td>Lowest</td>
<td>Highest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Passive tag size</td>
<td>Largest</td>
<td>Smallest</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.1: RFID system features at available frequencies**
3.4 Data Coding

Data encoding is the process of interpreting binary data that are transferred between the tags and readers as line codes or a stream of symbols. Different coding schemes of data are used in RFID systems, as shown in Figure 3.3. The coding schemes can be classified into two main types as follows (Chabanne et al., 2011):

**Level coding**: The logic value relates to the voltage level of the signal. The non-return to zero (NRZ) coding is an example of this type. In NRZ coding, a logic value 1 is coded by a high signal and a logic value 0 is coded by a low one;

**Transition coding**: The logic value relates to the difference between the two voltage levels of the signal. Manchester coding, the most commonly used coding scheme, generates a transition at the centre of the bit. It uses a positive transition to represent a logic value 0 and a negative transition to represent a logic value 1.

RFID systems usually use a different coding scheme for tag to reader communications than for reader to tag ones. The coding scheme should be selected with various considerations, including power sources of tags, bandwidth available, and collision detection ability. In the EPCglobal standard, UHF passive tags encode the data using FM0 or Miller coding whereas the reader uses Manchester coding.

![Diagram of various coding schemes](image)

**Figure 3.3**: Data coding in RFID systems (Finkenzeller, 2010)
3.5 Multi-access Procedures (Anti-collision)

The operation of RFID systems usually involves reading multiple tags simultaneously, such as identifying the entire contents of a pallet at once for a group integrity checking process. A reader issues signals to interrogate the tag IDs or the information stored in them. This mode of communication is called multi-access and it is one of the key features of RFID systems. The communication channel between the reader and the tags has a certain capacity and it is specified by the data rate. The shared capacity must be divided between the tags in a way that their signals are transmitted to a single reader without collisions. Collisions occur when two or more tags transmit at the same time. Anti-collision protocols are of great importance to the performance of RFID systems in this research, because without them the tags have to retransmit their data when collisions occur, which consumes time and power. The existing anti-collision protocols are generally categorised into four techniques, including Space Division Multiple Access (SDMA), Frequency Domain Multiple Access (FDMA), Time Domain Multiple Access (TDMA) and Code Division Multiple Access (CDMA) techniques. The objective of these techniques is to receive a tag ID with low complexity, transmission power, and time delay (Shih et al., 2006).

3.5.1 Space Division Multiple Access Technique

SDMA divides the channel into spatially separated domains. It can be implemented using multiple directive antennas and readers to reduce their reading range and to form an array of ranges in the space. Another way of implementing this technique is to use an electronically controlled directional antenna. The directional beam can be directed to a tag and thus, different tags can be distinguished by their angular location in the range of the reader. This multi-access technique is expensive and hence, is limited to a few particular applications (Finkenzeller, 2010).
3.5.2 Frequency Division Multiple Access Technique

FDMA allows different communication channels to work together simultaneously using different operating frequencies. This technique is implemented using tags with an adjustable communication frequency. A tag can receive any signal at the optimally suited reader frequency and then, respond to one of its available operation frequencies. For example, the communication channel from the reader to the tag operates on 135 KHz, whereas that from the tag to the reader operates in the range 860-960 MHz. A disadvantage of this technique is the cost since one reader must be available for each communication channel and therefore, it is limited to a few particular applications (Finkenzeller, 2010).

3.5.3 Code Division Multiple Access Technique

There are different variants of the CDMA technique based on how the spreading is done, but generally, each tag is assigned a different pseudo random binary sequence to spread its data over the entire spectrum. It provides multiple simultaneous communications using the same frequency operation and the reader is able to differentiate each signal from the others by their signatures. CDMA brings significant benefit in anti-collision for security purposes. However, it needs a very complicated system, which has restrained its adaption for RFID systems (Karmakar, 2010).

3.5.4 Time Division Multiple Access Technique

Most of the RFID communication systems use the TDMA technique, which allocates the available channel capacity to readers and tags with respect to time. Communications between the reader and the tags can be Tag-Talk-First (TTF) or Reader-Talks-First (RTF). TTF modes are very slow and inflexible and thus most applications use RTF modes. Generally, the signal from the reader synchronises the clocks of the tags. A reader starts each cycle of communication by transmitting commands and parameters in a time slot to select an
individual from a large group of tags. Then, the communication is established between the selected tag and the reader, with the selected tag subsequently responding. After reception of the tag’s message, it can be muted or its transmission rate can be minimised. The RFID tag anti-collision protocols using TDMA techniques are mainly classified into Aloha-based and tree-based protocols (Sarangan et al., 2008). In the next subsections, some of the more frequently used anti-collision protocols are presented. The protocols are briefed to a certain extent where the operational principle of the protocol can be understood without complications.

### 3.5.4.1 Aloha-based Protocols

The basic idea of ALOHA-based protocols is that the reader broadcasts a query request to the tags and then, each transmits its ID to the reader at the randomly generated time slots. Consequently, there is a certain possibility that more than one tag may transmit their IDs during the same time slot and in this case, the time slot turns out to be a collision one. On the other hand, if only one tag transmits its ID in a certain time, then the tag is successfully identified and will not participate in the subsequent process. ALOHA-based protocols are widely used in RFID systems. They are simple, have moderate performance, consume low power, and promise dynamic adaptability to different loads of RFID tags. However, they experience the problem of tag starvation, whereby a tag may never be identified because its signals collide every time (Shih et al., 2006). In the following, some common ALOHA-based protocols are presented.

**Pure Aloha Protocol:** This protocol is the simplest of all the TDMA protocols. It is used exclusively with read-only tags, for which they send their tag IDs only. These data are transmitted to the reader in a cyclical communication. After receiving the reader’s signal, the tag replies with its ID at a random time. Then, the tag waits for the reader’s reply, which could be a positive acknowledgment (ACK), meaning the transmission was successful, or a negative acknowledgment (NACK), indicating the existence of a collision, in which case the
tag needs to retransmit the same information. Since the tags transmit their IDs randomly when signaled by the reader, without any control, collisions are inevitable. The efficiency of this protocol is degraded when reading a large group of tags (Liu and Lai, 2006).

**Slotted Aloha (SA) Protocol:** This protocol is an enhancement to the pure ALOHA protocol. In this protocol, the time is divided into a number of synchronous time slots and the reader controls this number. The transmission time of the tag’s message is equal to the size of the slot. Each tag sends its ID in a predefined time slot. If collisions occur, tags retransmit their messages in other time slots after a random waiting time. The synchronisation among the tags is controlled by the reader. The synchronisation between the tags and the reader as well as the response to the tag at a predefined time decrease the probability of the collisions and therefore increase the performance of the protocol. However, it still has insufficient performance when the amount of tags is large (Liu and Lai, 2006).

**Framed Slotted Aloha (FSA) Protocol:** It is an improvement of the SA protocol where a tag is allowed to send only once in a frame. It contains frames and read cycles as well as the slots. A frame contains a number of time slots and a read cycle involves a frame to read the tag IDs or identify the tags. The frame size is fixed during the identification process. The reader starts each cycle of the reading with a request message containing the frame size and a random number. Then, each tag uses the random number to select a slot in the subsequent frame and replies in that slot. If no tag responds in the slot, the slot is said to be empty. If one tag responds, the slot is treated as a singleton slot, whilst if two or more tags respond, the slot turns out to be a collision one. Singleton or collision slots can also be called nonempty slots. The unidentified tags must respond in the next read cycle and this process is repeated until all the tags have transmitted their message successfully (Zhen et al., 2005). Because of the fixed frame size of the FSA protocol, its performance degrades when there is unexpected variation
in the amount of tags. Moreover, additional delay will result when a large frame size is used with fewer tags (Liu and Lai, 2006).

**Dynamic Frame Slotted Aloha (DFSA) Protocol:** In the DFSA protocol, the reader can adjust the frame size in each read cycle of the communication between the reader and the tags. The frame size is determined by collecting the information about the previous read cycle, including the number of singleton \( C_1 \), empty \( C_0 \) and collision \( C_k \) slots. This information is used to estimate the number of tags for the current round and thus, adapt the frame size accordingly (Huang and Le, 2007). Generally, when the number of tags is large, the probability of collisions is minimised by increasing the frame size. On the other hand, when the number of tags is small, the frame size is decreased. The performance of this protocol is better than that of the FSA protocol since setting the frame size to the number of tags will achieve the lowest identification delay. However, if the number of tags increases its performance degrades, because the frame size cannot be increased indefinitely beyond 256 for the accuracy of the estimation techniques. Hence, this protocol performs well when the number of tags is quite small (Vogt, 2002b; Kaewsirisin et al., 2008). The DFSA algorithm has several versions, depending on the estimation technique that change the frame size. The main difference amongst these techniques is how and what they estimate. Accuracy is the primary requirement of a tag estimation technique, for inaccurate ones cause empty or collided slots, which consumes time and power. The popular and most accurate tag estimation functions, as evaluated by (Klair et al., 2007) are briefly explained below.

Vogt (2002a; 2002b) proposed two estimation techniques. One is based on a simple assumption that at least two tags can cause a collision. The number of estimated tag \( n_{\text{est}} \) is calculated in each read cycle, as:

\[
  n_{\text{est}} = C_1 + 2C_k
\]

\[ (3.1) \]
The second technique is based on the Chebyshev inequality principle, which aims to reduce the distance between the actual read result vector \(<C_0, C_1, C_k>\) and the theoretical calculated result vector \(<E_0^{f,n}, E_1^{f,n}, E_k^{f,n}>\), as:

\[
n_{est} = \min \left| \begin{pmatrix} E_0^{f,n} \\ E_1^{f,n} \\ E_k^{f,n} \end{pmatrix} - \begin{pmatrix} c_0 \\ c_1 \\ c_k \end{pmatrix} \right| \tag{3.2}
\]

where, \(f\) is the current frame size, \(n\) is the number of tags, \(E_0^{f,n}\) represents the expected number of empty slots, \(E_1^{f,n}\) is the expected number of singleton slots, and \(E_k^{f,n}\) is the expected number of collision slots. The probability that \(r\) tags reply in one slot is a binomial distribution, as:

\[
p(y = r) = \binom{n}{r} p^r (1 - p)^{n-r} \tag{3.3}
\]

where, \(p\) is success probability and \(y\) is a random variable. Since each slot in the frame has equal probability, Equation (3.3) can be written, as:

\[
p(r) = \binom{n}{r} \left( \frac{1}{f} \right)^r \left( 1 - \frac{1}{f} \right)^{n-r} \tag{3.4}
\]

Therefore, the expected number of slots filled with \(r\) responding tags out of \(n\) is given by:

\[
E_r^{f,n} = f \cdot \binom{n}{r} \left( \frac{1}{f} \right)^r \left( 1 - \frac{1}{f} \right)^{n-r} \tag{3.5}
\]

From Equation (3.5), the values of \(E_0^{f,n}, E_1^{f,n}\), and \(E_k^{f,n}\) can be calculated, as:

\[
E_0^{f,n} = f \cdot \left( 1 - \frac{1}{f} \right)^n \tag{3.6}
\]

\[
E_1^{f,n} = n \cdot \left( 1 - \frac{1}{f} \right)^{n-1} \tag{3.7}
\]

\[
E_k^{f,n} = f - E_0^{f,n} - E_1^{f,n} \tag{3.8}
\]

Cha and Kim (2006) also proposed two tag estimation techniques. The first simply estimates the number of tags, as:

\[
n_{est} = C_1 + 2.39 C_k \tag{3.9}
\]

The second estimation technique is based on the calculation of the ratio of the number of collided slots to the frame size, as:
C_{ratio} = \frac{C \cdot k}{f} \quad (3.10)

By substituting the value of $C_{ratio}$ into the following equation:

$$C_{ratio} = 1 - (1 - \frac{1}{f})^{n_{est}} (1 + \frac{1}{f}) \quad (3.11)$$

the value of $n_{est}$ can be calculated after each read cycle.

The aforementioned simple techniques are easier to implement and have low computational overheads. They are more accurate for low amounts of tags. However, their accuracy decreases when the number of tags is increased due to the usage of the constant multiplier, which does not take into account the collisions of a large number of tags in a slot. On the other hand, the second techniques are more robust for a large number of tags, because they use the read results after each cycle and the theoretical calculated values at the expense of computation overheads (Klair et al., 2007).

EPCglobal has also developed a tag estimation algorithm called $Q$-algorithm which is specified in the standard. The algorithm needs the reader to increase or decrease the frame size with a constant. First, the reader broadcasts a request message containing a slot counter $Q$ and a frame of size $2Q$, where $Q$ is an integer in the range $0 \leq Q \leq 8$. Each tag selects a slot randomly in the range $[0 - (2Q-1)]$. In the case of a collision slot, the reader increases $Q$ by a constant $c$ where $0.1 \leq c \leq 0.5$. For an idle slot, it decrements the value of $Q$ by $c$. A slot that has a single response does not change the value of $Q$. The frame size of the next round is then set based on the resulting value of $Q$. This algorithm can enhance the read performance when the number of tags is much larger or smaller than the frame size (Chen and Kao, 2011).

**Enhanced Dynamic Frame Slotted Aloha (EDFSA) Protocol**: To solve the problem of identifying large numbers of tags, Lee et al. (2005) proposed the EDFSA protocol by restricting the number of responding tags to nearly equal to the frame size to achieve optimal system efficiency. When the number of responding tags exceeds the maximum frame size
(i.e. 256), the tags are split into a number of groups in order to achieve optimal system efficiency. In each read cycle, the number of unread tags is estimated using Equation (3.1). If the estimated number of unread tags is much larger than the maximum frame size that gives the optimal system efficiency, those unread ones are split into a number of groups. Only one group of unread tags is allowed to reply to the reader. The number of groups $M$ is computed, as:

$$M = \left\lceil \frac{\text{The number of estimated tags}}{\text{the maximum frame size}} \right\rceil$$  \hspace{1cm} (3.12)

and the system efficiency is calculated, as:

$$\text{System Efficiency} = \frac{\text{The number of singleton slots}}{\text{the current frame size}}$$  \hspace{1cm} (3.13)

Then, the reader sends a request including the number of groups and a random number to the tags. The tags receiving the request message use the random number and their tag IDs to generate a new number and then, divide it by the number of groups. Only the tags that have a zero remainder reply to the reader. When the number of estimated tags is much smaller than the maximum frame size, the reader reduces the frame size without grouping and in this case it sends a request message with the number of groups equal to 1. The reader estimates the number of unread tags and adapts the frame size after each read cycle. This process is repeated until all the tags have been read. Table 3.2 illustrates the values of $M$ for different tag ranges. The authors also proposed the optimal frame sizes for various tag ranges to achieve optimal system efficiency. When the number of unread tags is less than 355, the value of $M$ is one which allows every unread tag to respond. On the other hand, when the number of unread tags increases, the group of tags is divided into $M$ groups and thus, the reader read tags on a group basis. This type of protocol is one of the existing solutions that is implemented and compared with the designed protocol in Chapter 9.
Chapter 3 - RFID Systems

<table>
<thead>
<tr>
<th>The number of unread tags</th>
<th>Frame size</th>
<th>The number of groups $M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 11</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>12 – 19</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>20 – 40</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>41 – 81</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>82 – 176</td>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>177 – 354</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>355 – 707</td>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>708 – 1416</td>
<td>256</td>
<td>4</td>
</tr>
<tr>
<td>1417 – 2831</td>
<td>256</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 3.2: The number of tags versus the optimal frame size and the number of groups in the EDFSA (Lee et al., 2005)

3.5.4.2 Tree-based Protocols

In tree-based protocols, collisions are reduced by building tree traversing approaches. When a collision occurs, the reader divides the tags into two subgroups and queries that with fewer tags. The reader continues to divide the tags until all the tags have been successfully identified (Capetanakis, 1979). A weak point of these protocols is that they must restart the reading process when new tags enter the reader’s range while other tags are being read. In addition, they are complex and require more time to identify tags than that for the ALOHA-based protocols. However, they are free of the tag starvation problem (Shih et al., 2006). The tags get divided into subgroups based on different algorithms. In the following, some common algorithms are introduced.

**Tree Splitting Algorithm**: This algorithm operates by splitting collided tags into $b$ subgroups using a $b$ random number generator and these subgroups become smaller until they contain just one tag. Each tag has a counter to maintain its position in the tree and when its value is zero, the tag enters the transmit mode. Otherwise, the tag enters the sleep or wait mode. In the case of a collision, the tag in the transmit mode generates a random number and adds it to the value of its counter, whilst tags in the wait mode increment their counter by one. When the
reader receives a single reply, the tags in the wait mode decrease their counter by one. All tags enter the sleep mode after being identified (Hush and Wood, 1998).

**Query Tree Algorithm:** In this algorithm, the reader broadcasts a prefix to all tags in its region. Then, a tag the ID of which matches the prefix replies to the reader. When a collision occurs, the reader appends the prefix with a binary 0 or 1 until none exists. Once a tag is identified, the reading cycle is terminated and a new one is issued with a new prefix (Law et al., 2000). Some extensions to the query tree have been proposed in Law et al. (2000) to enhance the performance of the original algorithm, as follows:

1) **Shortcutting:** The reader appends the prefix with the binary 0 and 1, subsequently pushing the two new prefixes onto the stack. The reader then transmits one of the prefixes. If there is no reply, this means that two or more tags have the second new prefix. In this case, if the reader sends the second new prefix, a collision will exist. Thus, it removes the second new prefix from the stack and appends it with the binary 0 or 1 and then, pushes them onto the stack instead.

2) **Aggressive advancement:** The reader appends the prefix with multiple bits instead of just one.

3) **Categorisation:** If a group of tag IDs has the same prefix, the identification process can be enhanced by identifying each group independently.

4) **Incremental matching:** This algorithm minimises the length of the transmitted prefix by requiring the tags to store the last prefix. Therefore, instead of sending the same prefix appended with the binary 0 or 1, the reader sends only bit 0 or 1 in the next read cycle.

**Binary Search Algorithm:** In this algorithm, the reader recognises the position of the collided bit by using Manchester coding. When a collision occurs, the reader divides tags into subgroups based on the collided bits. To show the benefit of using this coding in tracing out the collided bit, its collision behaviour is compared with NRZ coding. If at a time at least one
tag sends a signal, the reader interprets the received signal using the NRZ coding as a high signal and decodes it as logic value 1, as shown in Figure 3.4. In this case, the reader cannot trace back the decoded signal and decide whether it is from one tag or many tags. The transmission error in the data block can be detected by using the block checksum. On the other hand, when two or more tags send different data simultaneously, the positive transitions cancel the negative transitions using the Manchester code. If there is no transition, it is interpreted as a collision having occurred between at least two tags, as depicted in Figure 3.5. In both coding schemes, when two or more tags send identical data simultaneously, they successfully received and interpreted by the reader. This feature is utilised to read redundant tag information in the designed protocol in Chapter 9.

The binary search algorithm divides the collided tags into two groups. Those that have first collided bit 0 are allowed to send in the subsequent request, whilst the tags with first collided bit 1 are not allowed to do so. Assume that there are four tags and their tag IDs are of 8-bits. Initially, the reader sends a request message asking all tags that have IDs less than or equal to 11111111 to reply. All tags will reply with their tag IDs to the reader, thus resulting in collisions in some bits. Assume collisions occurred in the received tag ID at bit 0, bit 3 and bit 5, then clearly bit 5 is the highest collided bit. This indicates that there is at least one tag the ID of which is less than 11011111. The reader sends a new request message asking tags that have IDs less than or equal to 11011111 to reply in the next read cycle. Now assume that the received tag IDs have collisions at bit 4. Again, the reader sends a request message to the tags that have the tag IDs less than or equal 11001111 to reply in the next read cycle. Tag 2 only replies in this round and is thus, identified by the reader. By repeating these procedures, all tags are identified (Finkenzeller, 2010).
Figure 3.4: Collision behaviour of NRZ coding (Finkenzeller, 2010)

Figure 3.5: Collision behaviour of Manchester coding (Finkenzeller, 2010)

3.6 Conclusions

An overview of RFID systems has been presented in this chapter, providing all the necessary and important bases for the thesis. Specifically, there has been explanation of the main system components, the standards, and the operating frequencies. The different types of tags have been reviewed, concentrating on their important features directly in relation to the
thesis, including the types of memory. There has also been coverage of the differences between ALOHA-based and tree-based anti-collision protocols, including a detailed explanation to help understand how the protocols work. Their advantages and limitations have been highlighted as well.
4 LDPC Codes

In modern communication systems, transmissions can be affected by data loss caused by
router congestion, intermittent connectivity, or poor channel situations. In such cases, erasure
codes represent the main solution. They are a type of forward error correction codes for the
BEC that has been used for modelling the transmission of data over the Internet (Ryan and
Lin, 2009). This channel was introduced by Elias in 1955 (Elias, 1955). A transmitted bit is
either completely erased with probability $p$ or correctly received with probability $1 - p$. Elias
proved that there exist codes of rate $R$ for any $R < 1 - p$ that can be used to transmit over
channels of capacity $1 - p$. Erasure codes rely on augmenting the transmitted message bits
with some redundant information called parity check bits and the resulting message is called
a codeword. The parity check bits are inserted in such a way that each codeword is
adequately distinct from one another and thus, allowing receivers to recover the original
message (Huffman and Pless, 2003). Among the different classes of erasure codes, the
attention in this thesis is limited to LDPC codes, because of their sparseness, which suits the
limited memory space of passive tags. In addition, they exhibit good erasure recovery
capabilities and comparably low decoding complexity (Liva et al., 2009). This chapter
provides a comprehensive overview of LDPC codes in the BEC with a focus on the encoding
process, the most prominent decoding algorithms, and various construction techniques. This
chapter provides sufficient relevant preliminary information for comprehending the
subsequent ones.
4.1 Preliminaries and Notations

LDPC codes were first introduced by Gallager in 1962 (Gallager, 1962). They are codes the parity check matrices of which, as the name implies, are sparse. In other words, the parity check matrix $H$ of an LDPC consists of low density of “1” entries and this characteristic suits the limited memory space of passive tags. Each column of $H$ relates to a bit in the codeword and each row to a parity check equation. The parity check matrix $H$ is of size $m \times n$, where $m$ is the number of parity check bits and $n$ is the length of the codeword. A parity check matrix is said to be regular when every column of $H$ has the same weight $j$ (the number of “1” entries in each column) and every row has the same weight $l$ (the number of “1” entries in each row), where $l = j \times (n/m)$ and $j < l$. If the number of “1” entries in each column or row is not constant, then the code is an irregular LDPC one (Shedsale and Sarwade, 2012). LDPC codes are usually represented as $(n, j, l)$ LDPC code or as $(n, k)$ LDPC code, where $k$ is the number of message bits.

The theory of LDPC codes is connected to a field of mathematics called graph theory and some basic notations and definitions used in it are briefly presented in what follows (Tomlinson et al., 2017).

**Definition 1:** (Node, Edge, Neighbouring) A graph, represented by $P(V, E)$, contains an ordered set of nodes and edges.

- A **node** (or a **vertex**) is usually represented as a dot. The set $V(P)$ contains nodes of $P(V, E)$. If $v$ is a node of $P(V, E)$, it is represented as $v \in V(P)$. The notation $|V(P)|$ denotes the cardinality of the nodes.
- An **edge** $(u, v)$ joins two nodes $u \in V(P)$ and $v \in V(P)$ and it is represented as a line joining the two. The set $E(P)$ consists of two nodes of $V(P)$. 
Chapter 4 – LDPC Codes

• If \((u, v) \in E(P)\), then \(u \in V(P)\) and \(v \in V(P)\) are **neighbouring** nodes of \(P(V, E)\). In the same way, the two nodes are connected to the edge \((u, v)\).

**Definition 2**: The **degree** of a node \(v \in V(P)\) is the number of edges that are connected to it.

**Definition 3**: The **Bipartite** or **Tanner graph** \(P(V, E)\) contains two disjoint sets of nodes, \(V_v(P)\) and \(V_c(P)\), such that \(V(P) = V_v(P) \cup V_c(P)\), and every edge \((v_i, c_j) \in E(P)\), such that \(v_i \in V_v(P)\) and \(c_j \in V_c(P)\) for some integers \(i\) and \(j\).

The matrix defined in (4.1) is an example of a \(5 \times 10\) parity check matrix for the regular \((10, 2, 4)\) LDPC code.

\[
H = \begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1
\end{bmatrix}
\tag{4.1}
\]

LDPC codes are usually exemplified in graphical ways using a Tanner graph (Tanner, 1981). It gives a complete illustration of the code and it helps in the explanation of the decoding process. The Tanner graph contains two groups of nodes, which are \(n\) variable nodes for the codeword bits, and \(m\) parity check nodes for the parity check equations. An edge connects a variable node to a parity check node, if that variable node is contained in the related parity check equation. Therefore, the number of edges in the Tanner graph is equivalent to the number of “1” entries in \(H\) (Richardson and Urbanke, 2001).

**Definition 4**: A **cycle** in a Tanner graph is a sequence of joined nodes that starts and ends at the same node, which contains other nodes no more than once. The length of a cycle is the number of edges it includes.

**Definition 5**: The **girth** of a Tanner graph is the minimum cycle length of the graph. The **local girth** of a node \(v \in V(P)\) is the length of the shortest cycle that passes through that node.
Since the Tanner graph is bipartite, the minimum length of a cycle is four. Short cycles, especially of length 4 degrade the iterative decoding performance of LDPC codes as will be explained in subsection 4.3.1 (Ryan, 2003). Figure 4.1 shows the Tanner graph of $H$ defined in Equation (4.1). The variable nodes are represented by circles while the parity check nodes are squares and a cycle of length 6 is exemplified by the bold edges.

![Figure 4.1: The Tanner graph illustration of $H$ in Equation (4.1) (Ryan, 2003)](image)

For an irregular LDPC code, the variable and parity check node degree sequences are used to represent the code. The polynomials:

$$\Lambda_\lambda(x) = \sum_{i \geq 1} \lambda_i x^i$$ \hspace{1cm} (4.2)

and

$$\Lambda_\rho(x) = \sum_{i \geq 1} \rho_i x^i$$ \hspace{1cm} (4.3)

specify the variable and parity check degree sequence, respectively, where $\lambda_i$ and $\rho_i$ are the fractions of nodes of degree $i$ (Tjhai, 2007). Degree sequences should satisfy:

$$\sum_{i \geq 1} \lambda_i = 1$$ \hspace{1cm} (4.4)

and
\[ \sum_{i \geq 1} \rho_i = 1 \] (4.5)

For example,

\[ \Lambda_d(x) = 0.4x^2 + 0.27x^3 + 0.24x^6 + 0.09x^9 \] (4.6)

means that 40\%, 27\%, 24\%, and 9\% of the columns have weights of 2, 3, 6, and 9, respectively, whereas,

\[ \Lambda_{\rho}(x) = 0.70x^{13} + 0.30x^{14} \] (4.7)

means that 70\% and 30\% of the rows have weights of 13 and 14, respectively. Another representation considers the fraction of edges that are linked to the nodes of degree \( i \). The former representation is used throughout this thesis.

### 4.2 Encoding Process

A message of \( k \) bits can be encoded using a generator matrix \( G \). The encoding process introduces redundancy of size \( m \) bits, where \( m = n - k \). If the \( k \) message bits are represented by the vector \( u = [u_1, u_2, \cdots, u_k] \), the codeword \( x \) of a binary message \( u \) can be generated using:

\[ x = uG \] (4.8)

where the operation is carried out under modulo-2 arithmetic. The ratio \( k/n \) is the rate of the code which represents the amount of redundancy introduced by the code, where higher rate means smaller \( m \) and vice versa. For a regular code, the rate can also be expressed as \( 1 - j/l \), since \( m \times l = j \times n \). Generally, a code can have any values of \( m \), however, only \( n - k \) of them will be linearly independent and thus, the rank of \( H \) is equal to \( n - k \). The generator matrix is usually constructed by performing the Gauss-Jordan elimination on \( H \) to transform it into:

\[ H = [-P, I_m] \] (4.9)
where $P$ is an $m \times k$ matrix and $I_m$ is the identity matrix. The identity matrix, $I_m$, is an $m \times m$ matrix with “1” entries on the diagonal from the top left corner to the bottom right corner and “0” entries elsewhere. The generator is then formed, as:

$$G = [I_k, P^T]$$

(4.10)

where, $(\cdot)^T$ denotes the transpose of its argument. A code is called systematic, if the first $k$ codeword bits represent the message bits. In this case, the first $k$ columns of the generator matrix represent the $k \times k$ identity matrix, otherwise, it is called non-systematic code (Gravano, 2001). Transforming $H$ into the desired form as in Equation (4.10) needs $O(n^3)$ operations. The limitation of encoding through $G$ is that the submatrix $P^T$ is not sparse and thus, the encoding complexity is $O(n^2)$ (Ryan and Lin 2009). Therefore, Richardson and Urbanke (2001) proposed an effective encoding method that takes advantage of the parity check matrix sparseness. The encoding overhead can be minimised to, at most, $O(n + g^2)$, where $g$ is a small constant, and in the worst circumstances as a small fraction of $n$. The columns and rows of $H$ are, firstly, permuted into an approximate lower-triangular form, as shown in Figure 4.2.

![Figure 4.2: The parity check matrix in the approximate lower-triangular form (Richardson and Urbanke, 2001)](image)

The five matrices A, B, C, D, and E are sparse, whereas the matrix T is in a lower triangular form which has “1” entries on the diagonal. The result can be written, as:
$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}$ (4.11)

Then, GE is applied to $E$ to zero out the matrix which is similar to multiplying Equation (4.11) by:

$$\begin{pmatrix} I & 0 \\ ET^{-1} & I \end{pmatrix}$$ (4.12)

To get:

$$\begin{pmatrix} A \\ -ET^{-1}A + C \\ -ET^{-1}B + D \\ 0 \end{pmatrix}$$ (4.13)

The source vector $u$ of length $k$ can be encoded as $c = [u, p_1, p_2]$, where $p_1$ and $p_2$ denote the parity part of length $g$ and $m-g$, respectively. More precisely, $p_1$ and $p_2$ are calculated so that the codeword $c$ must satisfy Equations (4.14) and (4.15), as:

$$ Au^T + Bp_1^T +Tp_2^T = 0 $$ (4.14)

$$ (-ET^{-1}A + C)u^T + (-ET^{-1}B + D)p_1^T = 0 $$ (4.15)

Following the same interest, Mackay (2005) proposed the staircase codes, where the parity check matrix is divided into two submatrices. The left submatrix $H_1$ is an $m\times k$ submatrix, the columns of which, relate to message bits. The right submatrix $H_2$ is an $m \times m$ submatrix with columns relating to parity check bits of weight 2 or less arranged in a staircase structure, as:

$$ H = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} $$ (4.16)

The staircase structure converts the codes into systematic forms and thus, it can be easily encoded in a linear time. If the data $u$ are loaded into the first $k$ bits of the codeword, then the $m$ parity bits $p_1...p_m$ can be calculated one at a time from left to right in a linear time, using only the message bits and the previously computed parity check bit, as:
\( P_1 = \sum_{n=1}^{k} H_{1n}u_n \)
\( P_2 = P_1 + \sum_{n=1}^{k} H_{2n}u_n \)
\( P_3 = P_2 + \sum_{n=1}^{k} H_{3n}u_n \)
\[ \vdots \]
\( P_m = P_{m-1} + \sum_{n=1}^{k} H_{mn}u_n \)

Chapter 4 – LDPC Codes

The encoding overhead is \( O(n) \), if the sparsity of the matrix \( H \) is exploited when calculating the sums in Equation (4.17). Unlike encoding in the BEC, the autonomous group integrity verification utilises an \( m \times n \) parity check matrix of an LDPC code as a group generator matrix to encode a group of tag IDs of size \( n \). Further details are given in Chapter 5.

### 4.3 Decoding Algorithms

A received codeword \( y \), is said to be valid, if it satisfies:
\[ Hy^T = 0 \]  
\[ (4.18) \]

Denoting the number of erasures by \( n_e \in \{1, 2, \ldots, n\} \), the received codeword with erasures can be represented, as:
\[ H_k y_k^T = H_k y_k^T \]  
\[ (4.19) \]

where, \( y_k \) means the set of loss bits of length \( n_e \) of the codeword, and \( y_k \) refers to the set of known bits of length \( n_e = n - n_e \). Similarly, \( H_{k'} \) is the matrix containing the columns of \( H \) related to \( y_{k'} \), whereas \( H_k \) is the matrix containing the columns of \( H \) related to \( y_k \). The capability of a code to recover erasures is measured by the minimum Hamming distance \( d_{min} \) of the code. The Hamming distance between two codewords is the number of bit positions in which they differ, whereas the \( d_{min} \) represents the smallest Hamming distance between any pair of the codeword. A code has a good distance, if \( d_{min}/n \) tends to a constant greater than zero. Gallager (1962) found that a typical \( (n, j, k) \) LDPC code with a fixed \( j \geq 3 \) has a \( d_{min} \) that grows linearly with \( n \). Column weight \( j \) has also been proven to affect the \( d_{min} \). Increasing only the column weight \( j \) or increasing it together with row weight \( l \) will increase the value of \( m \) and thus, decrease the rate. However, the \( d_{min} \) is increased accordingly. A code of minimum
distance $d_{\text{min}}$ can recover, at most, $d_{\text{min}} - 1$ erasures. If the code can recover from any set of $k$ coordinates of the codeword, that is, if $d_{\text{min}} - 1 = n - k$, the code is optimal with respect to recovering erasures (Luby et al., 2001a; Ryan and Lin, 2009). On the other hand, analysis conducted by Tomlinson et al. (2007) of the erasure recovering performance of linear binary codes has proven that many $(n, k)$ codes, such as the extended Bose, Chaudhuri, and Hocquenghem (BCH) codes and extended quadratic residue codes, can recover, on average, nearly $n - k$ erasures. It is also been proven that codes designed with iterative decoding, including the turbo and LDPC codes, exhibit weak performance compared to algebraic codes designed for maximum $d_{\text{min}}$. Table 4.1 illustrates the average number of recovered erasures for some analysed codes.

The performance of LDPC codes under any decoding algorithm over the BEC is usually described by plotting the Frame-Error-Rate (FER) or the Bit-Error-Rate (BER) curve as a function of the erasure probability. The behaviour of the curve typically consists of three regions. The first, known as the error floor region, occurs at smaller values of erasure probability and it shows the decoding failure to recover some erasures. As the erasure probability is increased, the error rate starts to increase rapidly forming a waterfall region. The dramatic change in the error rate with the increase of erasure probability depends on the type of the decoding algorithm. The third region, known as the erroneous region, occurs at higher values of erasure probability and it shows the inability of the decoder to recover nearly all of the erasures. In the next subsections, the most prominent decoding algorithms over the BEC that employed in this research are presented.

<table>
<thead>
<tr>
<th>$(n, k, d_{\text{min}})$ Code</th>
<th>Average number of recovered erasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>(128, 99, 10) Extended BCH Code</td>
<td>27.44</td>
</tr>
<tr>
<td>(256, 207, 14) Extended BCH Code</td>
<td>47.4</td>
</tr>
<tr>
<td>(512, 457, 14) Extended BCH Code</td>
<td>53.4</td>
</tr>
<tr>
<td>(200, 100, 32) Extended Quadratic Residue</td>
<td>98.4</td>
</tr>
<tr>
<td>(200, 100, 10) PEG-LDPC Code</td>
<td>93.19</td>
</tr>
<tr>
<td>(240, 120, 16) Turbo Code</td>
<td>116.5</td>
</tr>
</tbody>
</table>

Table 4.1: the average number of recovered erasures for some analysed codes in Tomlinson et al. (2007)
4.3.1 Iterative Decoding Algorithms

Luby et al. (2001a) were the first who successfully applied iterative (IT) decoding algorithms for LDPC codes over the BEC. The IT decoding is also known as a message-passing algorithm and is equivalent to Gallager’s soft decoding algorithm (Gallager, 1962). Whilst the IT decoding is suboptimal in recovering erasures, it has the advantage of demonstrating a linear decoding complexity of $O(n)$ (Richardson and Urbanke, 2008). The IT decoding algorithm recovers erasures by recursively exchanging messages along the edges of a Tanner graph between variable nodes and parity check nodes. Firstly, all parity check equations having only one erased bit are found and the erased bits are recovered. Then, new equations that have only one erased bit may be formed and used to recover further erased bits. The decoding process can be simplified as follows (Johnson, 2009):

1) Each variable node $v_i$ sends the same message $M_{v_i} \in \{0, 1, e\}$, where $e$ denotes the erased bit, to each of its connected parity check nodes;

2) If a parity check node receives only one $e$, it replaces the erased bit with the XOR of the known bits in its check equation;

3) Each parity check node $c_j$ sends different messages $E_{c_j,v_i} \in \{0, 1, e\}$ to each of its connected variable nodes $v_i$. These messages represent the results of the previous step;

4) If the variable node of an erased bit receives $E_{c_j,v_i} \in \{0,1\}$, the variable node updates its value to the value of $E_{c_j,v_i}$.

The IT decoding algorithm iterates the above process until all the erasure bits have been recovered successfully or a predetermined maximum number of iterations $I_{max}$ has passed without successful decoding due to the existence of stopping sets (Di et al., 2002). A stopping set is a set of variable nodes, such that all parity check nodes (neighbours) of the set are connected to the set at least twice. It is also a set of linearly dependent columns of $H$. In
general, a stopping set consists of multiple cycles and the only stopping set that contains a single cycle is one that contains all degree-2 variable nodes (Tian et al., 2003). Figure 4.3 shows an example of a stopping set containing variable nodes \( v_7, v_8, v_9, \) and \( v_{10} \) and marked by bold lines. The size of a stopping set is the number of variable nodes it contains. They play a vital role in the IT decoding algorithm of LDPC codes over the BEC. If all stopping set symbols are erased, none of them can be recovered. The minimum stopping set size of a code identifies the minimum number of erasures causing a decoding failure (Di et al., 2002), which is termed the stopping distance, \( d_{\text{stop}} \). Generally, the performance of the IT decoding algorithm is limited by the number and size of the minimum stopping sets. Hence, a good LDPC code must have no or very few small stopping sets. Orlistky et al. (2002; 2005) proved that the sizes of minimum stopping sets of regular \((n, j, l)\) LDPC codes with girths 4, 6, and 8, are 2, \( j + 1 \), and \( 2xj \), respectively. Tian et al. (2003) proved that a code that has a \( d_{\text{min}} \) includes at least one small stopping set of size \( d_{\text{min}} \). Therefore, avoiding all small stopping sets of size \( d_{\text{stop}} \leq t \) guarantees that \( d_{\text{min}} > t \). Since a stopping set contains cycles, removing short cycles decreases the probability of small stopping sets. Di et al. (2002) showed that in the BECs the remaining set of unrecovered erasures after the IT decoding is equal to the maximum size stopping set. The distribution of the stopping set in LDPC codes defines the erasure patterns for which the IT decoding will not succeed on the BEC. Finding the stopping set distribution of any parity check matrix is a difficult process. However, the average stopping set distribution can be found instead. This process is known as a finite-length analysis. It involves computing the exact average bit and error probabilities for any regular LDPC codes.
4.3.2 Maximum Likelihood Decoding Algorithms

Maximum likelihood (ML) decoding algorithms on the BEC turn into solving the system of $m$ linear equations and finding $n_e$ unknowns (erasures) in Equation (4.19). According to Di et al., (2002), optimal decoding is possible, if Equation (4.19) has a unique solution. The system of linear equations has a unique solution, if and only if, the columns of $H_k$ are linearly independent or the matrix $H_k$ has full column rank, that is, $\text{rank}(H_k) = n_e$. Since $H_k$ has $m$ rows, its columns can be linearly independent only, if $|y_k| \leq m$. In addition, since any $d_{\text{min}} - 1$ columns of $H$ and thus, columns of $H_k$ are linearly independent, the system of linear equations is solvable whenever $|y_k| < d_{\text{min}}$ (Ryan and Lin, 2009). For solving a system of linear equations, three classes of algorithms were proposed. The first, contains the conjugate gradient and Lanczos methods. These two algorithms are the variant of the standard conjugate gradient and Lanczos algorithms for solving a system of linear equations over real entities (Odlyzko, 1985; Coppersmith et al., 1986). A drawback of this class of algorithm is that it is not guaranteed to derive a solution when the system of linear equations is solvable (Odlyzko, 1985; Teitelbaum, 1998). Hence, Teitelbaum (1998) improved Lanczos algorithms.
in order to avoid this shortcoming. Both algorithms need about $O(n^2)$ operations. The second class is the Wiedemann method and its derivatives (Wiedemann, 1986). This algorithm employs the Berlekamp-Massey algorithm and guarantees deriving a solution in about $O(n^2)$ operations, but it is complicated to program (LaMacchia and Odlyzko, 1990). The third class is structured Gaussian elimination (GE) algorithms, also called intelligent GE algorithms, which were introduced by Odlyzko (1985). The classical GE algorithm needs a row permutation process and then, an equation solving one. Employing the classical GE algorithm or GE for short, on the matrix $H_k$ in Equation (4.19) has a complexity scaling as $O(n^3)$ and therefore, ML decoding becomes impractical for long block lengths. On the other hand, the structured GE algorithm converts a system of sparse linear equations into a non-sparse system with smaller dimensions of the unknown. Such unknowns are called pivots or reference variables (Liva et al., 2009). The purpose of the structured GE algorithm is to limit the average number of pivots from which the remaining unknowns can be recovered using back substitution operations, while the pivots are solved using the GE algorithm. LaMacchia and Odlyzko (1990) analysed the performance of different algorithms using problems of discrete logarithm calculations and integer factorisations. The authors concluded that a very large sparse system can be solved efficiently by using structured GE algorithms. Burshtein and Miller (2004) proposed a practical decoding algorithm similar to the structured GE algorithm to decode LDPC codes over the BEC efficiently. They exploited the sparseness of the matrix $H_k$ to reduce the decoding complexity of the GE algorithm and thus to enable the ML decoding for long block lengths. The description of the algorithm is as follows.

1) The columns of $H$ are first permuted so that the left part contains all the columns corresponding to known variables in $H_k$, while the right part contains all those corresponding to the erased variables in $H_k$.

2) $H_k$ is transformed into an approximate triangular matrix, as proposed by Richardson and Urbanke (2001). This means searching for degree-1 rows in $H_k$, then performing row and
column permutations to bring the found “1” entries into a diagonal form. The results are the submatrices \( C, T \) which is in a lower triangular form, whilst \( R_U \) and \( R_L \) have columns that cannot be put into lower triangular form. The \( \alpha \) variables related to the columns of \( R_U \) and \( R_L \) form the pivots. The matrices \( C, R_U, \) and \( R_L \) are sparse, as illustrated in Figure 4.4(a).

3) Perform row additions on \( C \) repeatedly to zero out the matrix. The matrix \( T \) is also transformed into an identity matrix by row additions. Both \( R_U \) and \( R_L \) become dense due to the row additions, as illustrated in Figure 4.4(b).

4) Recover the \( \alpha \) pivots by applying the GE algorithm only on the rows of the equations corresponding to the columns of \( R_L \). If all pivots are recovered, then the remaining \( e - \alpha \) unknowns can be obtained using back substitution operations.

![Figure 4.4: Reduced complexity ML decoder on \( H_F \) (Paolini et al., 2012)](image)

This decoding algorithm succeeds, if the rank \( (R_L) \geq \alpha \) after the zero matrix step. During the triangulation and zero matrix steps, all operations are performed also in parallel on \( H_k y_k^T \).

The main advantage of this algorithm is that the GE algorithm is employed only on \( R_L \) and not on the whole set of unknowns. The GE step is the critical step of Burshtein and Miller’s algorithm and has a complexity of \( O(\alpha^3) \). However, this can be considerably decreased, if the number of pivots is very small and hence, it is important to keep the size of \( R_L \) as small as
possible. This can be achieved by using one of the sophisticated methods for choosing the pivots presented by Burshtein and Miller (2004) and improved by Liva et al. (2009).

Eliminating the permutation process can also reduce the computational complexity of the GE algorithm, as proposed by Tomlinson et al. (2004). The proposed “In-Place” algorithm consists of two processes, including: a polynomial update process and back filling process. At the beginning of the algorithm, the erasure bits are substituted in the matrix $H$. Then, the first process marks the first equation containing the first erased bit. This equation is subtracted from all other equations having the same erased bit and not yet marked to generate a new group of equations. The process repeats until either no marked equations exist having the same erased bit or no erased bits exist that are not in marked equations. Having a set of erasure bits in the last marked equations, the second process begins with the last row of equations and solves the erasures backwards. Compared with the GE algorithm, the complexity of Tomlinson et al.’s algorithm remains $O(n^3)$, however, the multiplicative constant is significantly decreased.

### 4.3.3 Reduced Complexity Decoding Algorithms

Reduced complexity decoding algorithms for LDPC codes over the BEC can be implemented as Hybrid (HB) decoding algorithms. These algorithms combine different decoding algorithms with the aim of improving erasures recovery performance or decoding complexity over the BECs. A number of HB decoding algorithms have been proposed in the literature for different LDPC codes in the BECs. In four independent studies by Cunche and Roca (2008), Paolini et al. (2008), Yang et al. (2008), and Huang et al. (2011), HB IT/ML decoding algorithms were proposed, with the basic idea of employing the IT decoding first. If this decoding fails to recover all the erased bits due to the existence of stopping sets, the ML decoding is activated to complete the decoding process and resolve a simplified system. The ML decoding algorithm can fully recover the erasures in the simplified system, if the
columns of the decoding matrix containing erasures are linearly independent. It has been shown that implementing HB decoding on LDPC-triangle and LDPC-staircase codes achieves a high performance in terms of erasure recovery capabilities similar to that of the ML decoding as well as decoding speeds close to that of an ideal code, even with small block sizes (Cunche and Roca, 2008). The same conclusion has been drawn for generalised irregular repeat accumulate codes (Paolini et al., 2008), for turbo Gallager codes (Yang et al., 2008), and for fountain codes in the BEC (Huang et al., 2011). Mattoussi and Roca (2012) proposed a variant HB decoder that combines four types of decoding algorithms, including IT, Reed Solomon, binary and non-binary GE decoding algorithms. The proposed HB decoding was used to decode the generalised LDPC-staircase code for the BEC which is constructed by extending LDPC-staircase codes through Reed-Solomon codes using “quasi” Hankel matrices. Even though the HB decoding utilises a non-binary GE decoder that is more complex, the system on which it is applied is simplified by the previous three decoders. This allows for reaching the maximum recovery capabilities of the code. Decoding succeeds, if one or some of these decoding algorithms succeed. Recently, Bocharova et al. (2018) proposed a reduced complexity near-ML decoding algorithm for very long quasi-cyclic LDPC codes over the BEC. The proposed algorithm is a HB decoding that combines the IT decoding of the quasi-cyclic LDPC code and a sliding-window of the ML decoding. The second component is applied “quasi-cyclically” to a quite short sliding window of the ML decoding of a zero-tail terminated LDPC convolutional code. The complexity of the proposed algorithm is polynomial in the window length. On the other hand, it is linear in the length of the code.

Besides the HB decoding algorithms, the reduced complexity suboptimal decoding algorithms for LDPC codes over the BEC are generally based on appending redundant rows to the parity check matrix and employing post-processing of variable guessing when the IT decoding is not successful. Enhancing the performance of the IT decoding of linear codes
over the BECs by employing redundant parity check matrices has been studied, such as in Sankaranarayanan and Vasić (2005), Schwartz and Vardy (2006) and recently considered by Bocharova et al. (2017). A predetermined number of dual codewords is appended to the parity check matrix of an LDPC code, which can later be used by the IT decoder instead of the original matrix. The dual codewords are appended in ascending order of their weights, such that \( d_{\text{stop}} = d_{\text{min}} \). The proposed method can be applied to short LDPC codes since finding low weight dual codewords needs complex operations. Employing post-processing of variable guessing was first studied in Pishro-Nik and Fekri (2004) for short-length LDPC codes, where the complexity is quite small. The proposed method needs a minimum number of trials of the IT decoding equal to a polynomial function of the number of bit guesses. The complexity of the decoder increases exponentially with the number of guessed variables. An improved version in Vellambi and Fekri (2007) is based on the observation that a significant fraction of unsatisfied neighbours of a stopping set are of weight 2. This observation is exploited to simplify the decoding process by replacing the related pairs of variable nodes by one variable node. In Olmos et al. (2010), the method is improved for finite length codes. It eliminates parity check and variable nodes and thus, its complexity is identical to the IT decoding algorithm.

### 4.4 Construction Methods for LDPC Codes

This section presents some of the methods for constructing LDPC codes with special attention to the PEG algorithm on which the current research is based.

#### 4.4.1 Random Constructions

##### 4.4.1.1 Gallager Codes

In the original paper of Gallager (1962), the rows of \( H \) are divided into \( j \) submatrices with \( m / j \) rows in each. The first submatrix of independent rows includes \( l \) consecutive “1” entries ordered from left to right through the columns. That is, for \( i \leq m / j \), the \( i \)-th row contains ones
in columns \((i - 1) \times l + 1\) through \(i \times l\) and zeros elsewhere. The other submatrices of rows are simply random column permutations of the first submatrix. Hence, each column of \(H\) has a “1” entry once in every one of the \(j\) submatrices. Due to the random nature of Gallager’s LDPC code, the absence of short cycles especially of length 4, is not guaranteed (Ryan and Lin, 2009). The following is the parity check matrix of the \((20, 3, 4)\) Gallager-LDPC code.

\[
H = \begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\tag{4.20}
\]

Many studies have been focused on enhancing the performance of the LDPC codes. The irregular LDPC code introduced by Luby et al. (2001b) is one example that exhibits a significant enhancement compared to regular codes. Variable nodes as well as parity check nodes have different degrees and thus, the level of missing protection differs over a codeword. The columns of the matrix \(H\) that have greater numbers of “1” entries, provide higher missing protection levels than those with fewer of these entries. During the IT decoding process, erasures whose columns of \(H\) have a greater number of “1” entries require a few iterations to be recovered. Therefore, in the successive iterations, the recovered erasures will then be used to recover the remaining ones. To design good irregular degree distribution pairs, Richardson et al. (2001) proposed a density evolution method. This method is used to determine the maximum level of channel noise, called the threshold, where the IT decoding can recover erasures and to optimise degree distributions. Whilst Chung et al., (2001) used a Gaussian approximation to simplify the analysis of the decoding algorithm and to estimate the threshold. These two approaches are based on a cycle-free assumption, which
is only valid for infinite length LDPC codes. However, there are different methods for designing good finite length LDPC codes, as presented below.

### 4.4.1.2 MacKay Codes

MacKay (1999) provided algorithms to generate semi-randomly sparse parity check matrices and consequently decrease short cycles in the Tanner graphs of Gallager codes. Some of them that are free of length 4 cycles are listed below:

1) $H$ is constructed by randomly generating columns of weight $j$ while row weight is kept to being as uniform as possible. The number of overlapping “1” entries between any two rows is less than 2;

2) $H$ is constructed as above, except there are no two columns having overlap larger than one;

3) $H$ is constructed as above. A small number of columns are also deleted to avoid short cycles.

One limitation of the MacKay codes is that their structure is inefficient to enable low encoding complexity (Ryan, 2003).

### 4.4.1.3 Progressive Edge Growth (PEG) Methods

Hu et al. (2001; 2005) proposed the PEG method to avoid short cycles and thus, minimum stopping sets in the Tanner graph. The PEG method has good performance even for large block lengths (Shedsale and Sarwade, 2012). The method constructs Tanner graphs in an edge-by-edge or progressive manner with large local girths at variable nodes. When the cycles are long, the IT decoding algorithm iterates several times and thus, the recovery performance increases. The PEG method is initialised by three parameters, including the number of variable nodes ($n$), the number of parity check nodes ($m$), and the variable node degree sequence ($D_v$), which represents the list of degrees for each of the variable nodes. The
method starts with the lowest degree variable nodes and proceeds to those of higher degrees, with the aim of maximising the local girth of every variable node. The method does not progress to the next variable node until all of the edges of the present variable node have been connected.

For a given variable node, the first edge links it to a lowest degree parity check node under the current setting of the graph. Following edges of the variable node are linked in a way that either no cycle is generated or, when this is inevitable, that generated has the largest local girth possible under the current graph setting. Hence, if the current state of the graph shows that one or more parity check nodes are unreachable from the current variable node by traversing the linked edges, the new edge links an unreachable parity check node and thus, no cycle is generated. If all parity check nodes are reachable from the current variable node, the new edge should be linked to the parity check node with the lowest degree that generates the maximum local girth. Multiple choices may exist to choose the parity check node from the set of those available. That is, all choices result in the same local girth under the current graph setting. This issue can be solved by either choosing the first parity check node or randomly selecting any parity check node from the set. Both choices result in the same performance.

Denote the set of parity check nodes by $C = \{c_1, c_2, ..., c_m\}$, the set of variable nodes by $V = \{v_1, v_2, ..., v_n\}$, and the set of edges by $E$, where $E \subseteq C \times V$. An edge $(c_i, v_j) \in E$, if and only if, $h_{i,j} = 1$, where $h_{i,j} \in H$, $1 \leq i \leq m$, $1 \leq j \leq n$. Also, let $D_v = \{d_{v_1}, d_{v_2}, ..., d_{v_n}\}$ represent the variable node degree sequence in ascending order, where $d_{v_j}$ is the degree of variable node $v_j$. Similarly, let $D_c = \{d_{c_1}, d_{c_2}, ..., d_{c_m}\}$ represent the parity check node degree sequence in ascending order, where $d_{c_j}$ is the degree of parity check node $c_j$. Let $N_{v_j}^t$ represent the set of parity check nodes that are connected to $v_j$ within depth $t$, whereas $\bar{N}_{v_j}^t$ denotes the complement of the set. Finally, let $E_{v_j}^z$ represent the $z^{th}$ edge linked to $v_j$, where $1 \leq z \leq d_{v_j}$. 
Algorithm 4.1 shows the details of the PEG method (Hu et al., 2001), which is employed to improve the autonomous group integrity verification in this research.

**Algorithm 4.1:** Construction of the parity check matrix using the PEG method

**Input:** $(n, m, D_i)$

**Output:** $H$

1: for $j = 1$ to $n$

2: for $y = 1$ to $d_{v_j}$ do

3: if $y = 1$ then

   $E^1_{v_j} \leftarrow \text{edge}(c_i, v_j)$, where $E^1_{v_j}$ is the first edge connected to $v_j$, and $c_i$ has the minimum degree in the current graph state $E_{v_1} \cup E_{v_2} \cup \ldots \cup E_{v_j}$.

4: else

   Extend a tree from $v_j$ up to depth $z$, such that $^\prime N^z_{v_j} \neq \emptyset$ but $^\prime N^{z+1}_{v_j} = \emptyset$, or the number of elements in $N^z_{v_j}$ is not increasing, but is less than $m$. Then, $E^y_{v_j} \leftarrow \text{edge}(c_i, v_j)$, where $c_i$ is selected from $^\prime N^z_{v_j}$ and has the minimum degree.

5: end if

6: end for

7: end for

### 4.4.1.4 Approximate Cycle Extrinsic Message Degree (ACE) Methods

The approximate cycle extrinsic message degree (ACE) method proposed by Tian et al. (2003; 2004) is another method that accounts for cycles, especially the overlapping of multiple cycles in the Tanner graph of an LDPC code. The method prevents stopping sets of small sizes by maximising the connections between the variable nodes within a short cycle and parity check nodes outside the cycle. Consequently, useful information can flow from outside the cycle into the cycle. An extrinsic check node of variable nodes in a cycle is one
that is connected to the cycle only once. The extrinsic message degree (EMD) for a cycle is the number of these extrinsic check nodes connected to the cycle. The approximate cycle EMD (ACE) for a cycle of length $2\alpha$ is the maximum possible EMD for the cycle. Since a variable node with degree $j$ in a cycle can be linked to, at most, $j-2$ extrinsic parity check nodes, the ACE for a cycle of length $2\alpha$ is $\sum_{i=1}^{\alpha}j_i - 2$. The proposed method constructs a Tanner graph or parity check matrix in such a way that short cycles have large EMD values. This has the effect of increasing the size of the smallest stopping set as the EMD of a stopping set is zero. The method produces an $(\alpha, c_{ACE})$ LDPC code, with cycles of length $2\alpha$ or less having ACE values of at least $c_{ACE}$. The method starts with the lowest weight column of the parity check matrix and proceeds to columns of higher weight. If the ACE value of a cycle is at least $c_{ACE}$, the candidate column for the parity check matrix is added. There is a tradeoff between $\alpha$ and $c_{ACE}$, such that increasing one parameter unavoidably causes the other parameter more difficult to implement. For $\alpha = 6, 7, 9, 10, 13$, the maximum $c_{ACE}$ values that can be achieved are 7, 5, 4, 3, 2, respectively. The constructed code with the ACE method is always irregular (Tian et al., 2004), which may prove to be an obstacle for the group integrity application of RFID systems. Xiao and Banihashemi (2004) combined the ACE and PEG algorithms to obtain a hybrid ACE-PEG one, which results in codes with good iterative decoding performance. In Chapter 5, the performances of the PEG-based LDPC codes in recovering missing tag IDs in the autonomous group integrity verification are compared with those of the ACE-based LDPC codes.

### 4.4.2 Algebraic Constructions

#### 4.4.2.1 Quasi-Cyclic Codes

Lan et al. (2007) proposed quasi-cyclic codes. A code is called as such, if for any $\sigma$ cyclic shifts of a codeword, this results in another codeword. A cyclic code is also a quasi-cyclic with $\sigma = 1$. The simplest form of quasi-cyclic codes is row circulant codes, as:
where, \( A_1, A_2, \ldots, A_q \) are binary \( j \times j \) circulant matrices. Having one invertible circulant matrices, for example \( A_q \), \( G \) can be built in systematic form, as:

\[
G = \begin{bmatrix}
I_{j(q-1)} & (A_q^{-1}A_1)^T \\
(A_q^{-1}A_2)^T \\
\vdots \\
(A_q^{-1}A_{q-1})^T
\end{bmatrix}
\]  

(4.22)

The resulted quasi-cyclic code is of length \( j \times q \) and dimension \( j \times (q - 1) \). The algebra of \( j \times j \) binary circulant matrices is similar to that of polynomials modulo \( x^j - 1 \) over Galois Field (2) or GF(2) for short. In this case, the codeword is encoded as \( c(x) = [i(x), p(x)] \), where \( i(x) \) describes \( k \) message bits and \( p(x) \) is given by:

\[
p(x) = \sum_{l=1}^{q-1} i_l (x) \times (a_q^{-1}(x) \times a_l(x))^T
\]  

(4.23)

Quasi-cyclic codes possess structures that are free of length 4 cycles as well as allowing low decoding complexity at the cost of poor performances for large \( n \). Cunche et al. (2010) showed that a class of quasi-cyclic LDPC codes can be used to decrease the complexity of ML decoding significantly in the implementation of HB decoding. This is accomplished by a number of row and column permutations, scaled as \( k\sqrt{k} \), which transform a quasi-cyclic parity check matrix into a pseudo-band form. This method can be effectively used to decode quasi-cyclic LDPC codes of medium and short lengths or of high rates as well.

### 4.4.2.2 Euclidean Geometry Codes

Kou et al. (2001) constructed Euclidean geometry (EG) codes using points and lines of Euclidean geometries over finite fields. The following simply explains the fundamental construction of EG codes. Let EG with \( n \) points and \( m \) lines have four structural properties, as follows:

1) Every line contains \( p \) points;

2) Any two points are linked by one and only one line;
3) Every point is intersected by \( l \) lines;

4) Any two lines are either parallel or they intersect at one and only one point.

An \( m \times n \) parity check matrix is formed where rows and columns relate to the lines and points of the EG, respectively. \( h_{i,j} = 1 \), if and only if the \( i \)-th line of EG contains the \( j \)-th point of EG. A row in the parity check matrix represents the points on a specific line of EG and has weight \( p \), whereas a column represents the lines that intersect at a certain point in EG and has weight \( l \). The parity check matrix can be simply seen as the incidence matrix of the lines over the points in EG. The Tanner graphs of these types of codes have girth 6. However, the values of \( j \) and \( l \) are relatively large, which is undesirable in some applications, including the autonomous group integrity application. In addition, there is no flexibility in the choice of the length and rate of the code (Ryan, 2003).

### 4.4.2.3 Chinese Remainder Theorem (CRT) Methods

To overcome the random nature of the Gallager LDPC codes, Su and Tonguz (2013) utilised the redundancy property of the remainder presentation in the Chinese remainder theorem (CRT) to construct the parity check matrix of LDPC codes. Let \( a_1, a_2, \ldots, a_w \) be any \( w \) integers and \( b_1, b_2, \ldots, b_w \) be positive integers that are pairwise relatively prime, where \( \gcd(b_p,b_q) = 1, \forall \ p \neq q, \ 1 \leq p, \text{ and } q \leq w \). According to the theorem, the system of congruences:

\[
    x \equiv a_i \pmod{b_i} \quad \text{for } i=1, 2, \ldots, w
\]

has a unique solution module \( \prod_{i=1}^{w} b_i \) (Ding et al., 1996). A positive integer \( x \) is uniquely identified by the residues \( a_1, a_2, \ldots, a_w \) of \( x \) module \( b_1, b_2, \ldots, b_w \). Thus, having \( j \geq w \) integers that are relatively prime in pairs \( b_1 < b_2 < \ldots < b_w \), where \( x < \prod_{i=1}^{w} b_i \), the residues of \( x \) module \( b_1, b_2, \ldots, b_w \) are a redundant representation of \( x \). The feature that allows for the building of the parity check matrix is that by choosing \( j \geq w \) integers that are relatively prime in pairs \( b_1 < b_2 < \ldots < b_w \), any two positive integers \( x_1, x_2 < \prod_{i=1}^{w} b_i \) disagree on at least \( j - x \)
+1 residues module \(b_1, b_2, \ldots, b_w\) (Goldreich et al., 2000). The CRT algorithm proposed by Su and Tonguz (2013) builds the parity check matrix \(H\) of LDPC codes by, firstly, identifying three parameters, including the length of the codeword \(n\), the column weight \(j\), and a positive integer \(w \leq j\) subsequently, performing the following procedure.

1) Choose \(j\) positive integers \(b_1 < b_2 < \ldots < b_j\) that are relatively prime in pairs, where,

\[
b_j \leq \frac{n}{2}
\]

\[
\prod_{i=1}^{w} b_i \geq n
\]

2) For each \(b_i\), where \(1 \leq i \leq j\), build a parity check matrix \(H(i)\) of size \(b_i \times n\), where the \((p,q)\)th entry is set to 1, if,

\[
q - 1 \equiv p - 1 \pmod{b_i}
\]

is true, otherwise, it is set to 0.

3) The submatrices \(H(i)\) are concatenated to build the main parity check matrix \(H(n, j, w)\) of size \(m \times n\), where \(m = \sum_{i=1}^{j} d_i\), as:

\[
H = \begin{bmatrix}
    H(1) \\
    H(2) \\
    \vdots \\
    H(j)
\end{bmatrix}
\]

The constructed CRT-based LDPC code is always irregular and there is no flexibility in the choice of its rate. Whilst, Su and Tonguz (2013) employed this method to improve autonomous group integrity verification, these drawbacks eliminate deriving a realistic baseline performance comparison with other codes. The performances of the CRT-based LDPC codes in recovering missing tag IDs in autonomous group integrity verification are compared with those of the PEG-based LDPC codes in Chapter 5.

\subsection*{4.4.3 Main Comparisons of Construction Methods}

In general, long random codes perform better in the waterfall region than algebraic codes of the same parameters. The encoding process of the random codes involves implementing the GE process first and then storing the result matrix in a memory. On the other hand, algebraic
codes usually have a lower error floor. Codes with large minimum Hamming distances can be constructed more easily than random ones. Furthermore, since algebraic codes, such as quasi-cyclic codes can be defined using polynomials, the encoding process may be simplified using a linear-feedback shift-register circuit where the memory overhead is least. They also have advantages in integrated circuits decoder implementation, because of their cyclic symmetry, which results in simple modular structure and regular wiring (Lan et al., 2007).

4.5 Conclusions

In the family of erasure codes over the BECs, a significant role is played by LDPC codes, because of their good performance and their comparably low decoding complexity. Their parity check matrices are characterised by a low density of “1” entries, which suits the limited memory space of passive tags. A message is encoded using a generator matrix that is formed from the parity check matrix. The performance of LDPC codes, both in terms of erasure recovery capabilities and decoding speeds, is largely dependent on the decoding algorithm used. The two most prominent decoding algorithms to recover erasures over the BECs are the IT and ML decoding algorithms. The IT decoding algorithm features a linear decoding complexity, but it remains suboptimal in recovering erasures, as it is affected by the existence of stopping sets. On the other hand, the ML decoding algorithm for the BEC, which can be implemented as the GE algorithm, provides the optimum decoding in terms of erasure recovery capabilities, but at the price of high decoding complexity. A good decoding strategy may consist of combining the IT and ML decoding algorithms. This hybrid IT/ML decoding algorithm allows for a compromise between performance and complexity. Different methods exist in the literature for constructing LDPC codes. Mitigating short cycles and thus, stopping sets are of primary concern when constructing LDPC codes for the autonomous group integrity verification of tags to enable linear time decoding with high recovery capabilities. Hence, the PEG method is used to construct the parity check matrix of LDPC codes in this
research. The GE decoding algorithm is deployed to represent the ML decoding throughout this thesis.
5 Extended Grouping of RFID Tags Based on PEG Methods

In this chapter, extended grouping of RFID tags is improved in terms of missing recovery capabilities. Due to the sparseness property of LDPC codes and the mitigation of the PEG method for short cycles, this method is used to construct the parity check matrix $H$ of LDPC codes in order to increase the performance of the code in recovering missing tag IDs with low memory consumptions. The recovery performance of PEG-based LDPC codes is analysed and assessed under the IT and GE decoding algorithms. The properties of the constructed PEG Tanner graphs are also analysed and compared with the Tanner graphs of other codes.

5.1 System Model

Consider an RFID system with one reader and many passive tags. On the basis of the EPCglobal Class-1 Generation-2 standard, each tag stores the 96-bit ID that uniquely identifies the tagged object. Tags can perform some calculations, such as hashing. They have special memories, as presented in subsection 3.1.1.2 of Chapter 3, in addition to the user memory which enables users to store additional information. The communication between the reader and the tags uses a framed, slotted Aloha protocol, which follows the RTF mode. The signal from the reader synchronises the clocks of the tags. In each cycle of communication, a reader broadcasts a request in a time slot and then, multiple tags reply in the following slotted frame. As stated in the standard, the transmission rate between the reader and the tags is asymmetric and subject to the environment.

5.2 Encoding Process

Algorithm 4.1 of the PEG method, which is described in subsection 4.4.1.3 of Chapter 4, is used to construct the parity check matrix $H$ of LDPC codes. The algorithm is initialised with
three parameters, including the number $n$ of tag IDs, the number $m$ of subgroups, and the variable node degree sequence. Then, a group of $n$ RFID tag IDs denoted by $TID_1$, $TID_2$, $TID_3$, ..., $TID_n$ is encoded using the constructed $H$ of size $m \times n$, as:

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = H \cdot \begin{bmatrix} TID_1 \\ TID_2 \\ \vdots \\ TID_n \end{bmatrix}$$  \hspace{1cm} (5.1)$$

to generate $m$ subgroup IDs, where $c_i, i \in \{1..m\}$ is the subgroup ID of the $i$-th subgroup. As represented in coding theory, the resultant codeword block length denoted by $x$ is of size $n + m$. $c_i$ is computed as the XOR (denoted by $\oplus$) of the IDs of the tags belonging to the same subgroup. The set of information needed to be recorded in the memory of each RFID tag in a group is calculated based on the number of subgroups in which the tag is involved (the column weight $j$ of $H$) and the number of short IDs of other tags belonging to the same subgroup (the row weight $l$ of $H$). The bit length of each short ID $s_i$ is set to the minimum number of bits that are necessary to distinguish a group of tags. For example, a short ID of 8 bits is enough for a group of less than 256 tags. Note that the short ID of each tag is not needed to be recorded in its memory. Hence, the set of information needed to be recorded for each subgroup is a 96-bit subgroup ID $c_i$ in addition to $s_i \times (l - 1)$. Therefore, the total length of information recorded by each tag in its memory is equal to:

$$j \times (c_i + s_i \times (l - 1)) \hspace{1cm} (5.2)$$

Figure 5.1 shows an example of the encoding process of 6 tag IDs denoted by $TID_1$, $TID_2$, ..., $TID_6$ and all the information recorded in these tags. Since, $Tag_1$ is involved in the parity check value $c_1$ with $Tag_4$ and $Tag_5$, it records $c_1$ as well as $s_4$ and $s_5$. 
5.3 Decoding Process

During the decoding process, the interrogator obtains tag IDs, parity check values, and short IDs from the available tags, then checks the integrity of each subgroup. Based on the results of the checking process, Equation (5.1) can be rearranged into the following form:

$$H_k y_k^T = C^T \oplus H_k y_k^T$$

(5.3)

where, $y_k$ and $y_k$ are the arrays of lost and known tag IDs, respectively. Further, $C$ represents the set of subgroup IDs. The interrogator can obtain $H_k$ from the short IDs of missing tags. Figure 5.2 shows the decoding process for the example presented in Figure 5.1, where $TID_3$ and $TID_6$ are missing. Equation (5.3) represents a set of linear equations with only $y_k^T$ as the unknown and it also describes a Tanner graph containing two groups of nodes. One group
represents the tag IDs and the other, the subgroup IDs. An edge links a tag node to a subgroup node, if the tag is part of the subgroup. The missing tag IDs in Equation (5.3) can be recovered using the IT decoding or GE decoding algorithms, each with different performance and complexity. Figure 5.3 illustrates the IT decoding algorithm for the example presented in Figure 5.2. The Tanner graph is initialised as presented in part (a), where the dark squares denote the missing tag IDs. Bold and thin lines represent identified and unidentified edges, respectively. Known variables (tag IDs) are passed to subgroup IDs as in part (b). Then, missing tag IDs (TID₃ and TID₆) are recovered in part (c). The second iteration of variable passing illustrated in part (d) confirms that the recovered tag IDs satisfy the parity check constraint c₃. In the remainder of the thesis, for simplicity, the term “parity check values” is used to represent “subgroup IDs” and the term “parity check constraints” to represent “subgroups”.

\[
\begin{bmatrix}
0 & 0 \\
1 & 0 \\
1 & 1 \\
0 & 1 \\
\end{bmatrix}
\begin{cases}
TID₃ \\
TID₆
\end{cases}
= \begin{cases}
c₁ \\
c₂ \\
c₃ \\
c₄
\end{cases}
\oplus
\begin{bmatrix}
1 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}
\begin{cases}
TID₁ \\
TID₂ \\
TID₄ \\
TID₅
\end{cases}
\]

\[TID₃ = c₂ \oplus TID₁ \oplus TID₄\]
\[TID₃ \oplus TID₆ = c₃ \oplus TID₂\]
\[TID₆ = c₄ \oplus TID₂ \oplus TID₅\]

**Figure 5.2: An example of the decoding process of a group of 6 RFID tags**
5.4 Performance Evaluation

This section evaluates the missing recovery capability of the PEG-based LDPC codes with Matlab R2015a and R2018b simulations. Different parity check matrices are constructed using MainPEG.C (Hu, 2008), which outputs different matrices for different parameters, including $n$, $m$, and $j$. Generally, the procedures of the simulation can be applied to test the performance of any LDPC-based code in the research. In the following, the process of the simulation is summarised and the related flowchart is shown in Figure 5.4.

1) The simulator generates $n$ random 96-bit TIDs;

2) The simulator performs the encoding algorithm on the group of $n$ TIDs in order to obtain $m$ parity check values $c_i$;
3) The simulator removes \( r \) TIDs from the group of \( n \) (not including the parity check values), where \( r \) varies from 1 to \( n - 1 \). Missing tag IDs are chosen randomly and uniformly from the main group;

4) The simulator applies the decoding algorithm to known tag IDs and parity check values in order to recover the missing tag IDs. If the result of the decoding algorithm is different from the missing tag IDs, it is treated as an error;

5) The simulator repeats the above two processes \( 10^5 \) times for each value of \( r \) and then, computes the average error rate as the mean value of errors existing in the \( 10^5 \) rounds in order to obtain an accurate value of the statistics.

![Flowchart of the simulation process](image)

**Figure 5.4: Flowchart of the simulation process**

### 5.4.1 Using the GE and IT Decoding Algorithms

The performances of the PEG-based LDPC codes are examined under the GE and IT decoding algorithms. For comparison with other LDPC-based methods presented in Chapter 2, which are based on encoding using the parity check matrices of LDPC codes, the same parameters are used in the construction of the parity check matrices. The size of the tested group of RFID tags is assumed to be equal to \( n = 25, 49, 121, \) and 169. Since the aim of the
autonomous group integrity verification in RFID systems is to minimise the memory consumption, only a small value of $j$ is considered. The column weight, $j$, varies between 3 and 4 whereas the row weight, $l$, is fixed for each group. It is important to highlight that some constructed matrices using the PEG method are nearly regular with minor rows have different weights. Appendix B presents some of the constructed matrices. Figure 5.5 depicts the performance of the PEG-based LDPC codes in recovering missing tag IDs with the IT and GE decoding algorithms for different values of $n$. Two performance criteria are considered, including error rates and memory costs. The error rate is the percentage that the decoder cannot recover missing RFID tags. The memory cost represents the information needed to be recorded in the memory of each tag, which can be calculated using Equation (5.2). For example, using $(25, 3, 5)$ code, each tag needs to store an amount of information equal to $3 \times (96 + 8 \times (5 – 1)) = 384$ bits, where the length of the short ID $s_i$ is 8 bits. The results in the figure show that the recovery capabilities for both decoding algorithms increase with increasing $j$. The main reason for this behaviour is that increasing $j$ increases the value of $m$ and so too, the $d_{mn}$ of the code, which results in increasing the value of $d_{stop}$ and the rank of the matrix $H_k$. However, the amount of memory consumption increases as a consequence. Table 5.1 illustrates the comparison of the PEG-based method with other extended grouping LDPC-based methods. The performance of the other methods is obtained from Su and Tonguz (2013). It is clearly shown that the aforementioned finding is true for all methods. For example, if 10 tags are missing from a group of $n = 49$, all of their IDs can be recovered with at least $j = 3$, using the PEG-based methods under the IT decoding algorithm. In this case, each tag records 432-bit, additional information. However, if 13 IDs are missing, each tag is required to store 576-bit additional information in order to achieve 100% reliable recovery (zero average error rate). If the reliability can be reduced to 95%, the consumed memory spaces can be decreased to 432-bit where the IT and GE decoding algorithms cannot recover some missing tag IDs. As the number of missing tags increases, the performance of the two
decoding algorithms decreases. For example, when the number of missing tags is 20, the average error rate for the IT decoding algorithm is 0.8, meaning that a few number of missing IDs is recovered.

![Graphs showing average error rate vs number of missing tags for different group sizes](image)

**Figure 5.5:** The average error rate of the identification of missing tag IDs from different group sizes: (a) \( n = 25 \); (b) \( n = 49 \); (c) \( n = 121 \); (d) \( n = 169 \), using the PEG-based LDPC codes with the IT and GE decoding algorithms.

It is also clear that the selection of tags for subgroups needs to be carried out in a systematic way. Table 5.1 shows that, with the same memory consumption, the PEG-based LDPC code achieves greater improvement in recovering missing tag IDs compared to the other codes. The simulation reveals that the PEG-based LDPC code under the IT decoding algorithm can recover up to 13 missing tag IDs with a 0% error rate by writing 576-bit additional information to each tag in a group of 49 tags. Lowering the memory overhead to 432-bit, the largest number of missing IDs that can be recovered by the code without any error is 10.
which is still higher than that provided by other codes. This reduction in memory overhead can save 25% of each tag’s memory while outperforming the performance of other methods.

<table>
<thead>
<tr>
<th>Parameters $(n, j, l)$</th>
<th>LDPC Codes</th>
<th>Maximum number of missing tags that can be identified within a specified tolerance of error rate</th>
<th>$m$</th>
<th>Tag memory consumption in bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0%</td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>(25, 3, 5)</td>
<td>Gallager-based</td>
<td>3</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>SSF-based</td>
<td>5</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRT-based</td>
<td>5</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTD-based</td>
<td>5</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based under IT</td>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based under GE</td>
<td>7</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>(25, 4, 5)</td>
<td>Gallager-based</td>
<td>5</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>SSF-based</td>
<td>7</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRT-based</td>
<td>9</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with IT</td>
<td>9</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with GE</td>
<td>12</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>(25, 3, 7)</td>
<td>Gallager-based</td>
<td>1</td>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>SSF-based</td>
<td>5</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRT-based</td>
<td>5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with IT</td>
<td>10</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with GE</td>
<td>13</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>(49, 3, 7)</td>
<td>Gallager-based</td>
<td>3</td>
<td>16</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>SSF-based</td>
<td>7</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRT-based</td>
<td>8</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with IT</td>
<td>13</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with GE</td>
<td>16</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>(121, 3, 11)</td>
<td>Gallager-based</td>
<td>1</td>
<td>13</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>SSF-based</td>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRT-based</td>
<td>5</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTD-based</td>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with IT</td>
<td>16</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with GE</td>
<td>20</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>(121, 4, 11)</td>
<td>Gallager-based</td>
<td>6</td>
<td>27</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>SSF-based</td>
<td>13</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRT-based</td>
<td>15</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with IT</td>
<td>23</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with GE</td>
<td>30</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>(169, 3, 13)</td>
<td>Gallager-based</td>
<td>1</td>
<td>13</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>SSF-based</td>
<td>5</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRT-based</td>
<td>5</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with IT</td>
<td>17</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with GE</td>
<td>19</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>(169, 4, 13)</td>
<td>Gallager-based</td>
<td>1</td>
<td>31</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>SSF-based</td>
<td>9</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRT-based</td>
<td>16</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with IT</td>
<td>28</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PEG-based with GE</td>
<td>31</td>
<td>45</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: Comparisons of the PEG-based LDPC codes with other LDPC-based codes
The results of Table 5.1 also depict that the performance of the PEG-based LDPC code degrades with the IT decoding algorithm compared with its performance with the GE decoding algorithm. This is caused by the presence of small size of stopping sets, which are phenomena of the IT decoding only. The simulation reveals that when applying the PEG method to a group of 169 tags, the largest number of missing tag IDs that can be recovered without any error is 28 with the IT decoding algorithm. On the other hand, applying the GE decoding algorithm causes a noticeable improvement of up to 31 using the same parameters, where \( j = 4 \).

The simulation also reveals the advantages of using the PEG method with larger groups of tags. Applying the PEG method to a group of \( n = 121 \) and 169 tags results in dramatic recoveries with a 0% error rate, which is not the case with other LDPC-based codes. With the same memory consumption, the PEG-based code can recover up to 17 missing tag IDs from a group of 169 tags, where \( j=3 \), using the IT decoding algorithm, which is three times the performance of the CRT-based code. This results from the fact that the PEG method accounts for small stopping sets in Tanner graphs by maximising the girth, which is more achievable when the group size is large. The Tanner graph of the \((169, 3, 13)\) PEG-based LDPC code has a girth of 6 and by maximising the length of the code, the girth is also increased. For example, the \((512, 3, 6)\) PEG-based LDPC code has a girth of 8 and further increasing the length of the code can cause an increase in the girth to 10, such as for the \((1000, 3, 5)\) PEG-based LDPC code.

The effects of changing \( l \) on the performance of the IT and GE decoding algorithms with long length codes, such as \( n = 1000 \), are shown in Figure 5.6 and Figure 5.7, respectively. The results of the IT decoding clearly reveal that although the \((1000, 4, 5)\) code where \( m=800 \), has a girth of 8, smaller than the girth of the \((1000, 3, 5)\) code with \( m=600 \), which is 10, it achieves higher performance, as with shorter length codes that are examined above, due to the larger
value of \( m \) and also, its bigger \( d_{\text{min}} \) it has. On the other hand, decreasing the \( l \) of the code from 8 to 6 with the same \( m=500 \) and girth=8 enhances the IT performance. The same conclusion can also be drawn for the performance of the GE decoding, but with an improvement of the (1000, 4, 8) code over the (1000, 3, 6) code when recovering less than 500 missing tag IDs.

Figure 5.6: The average error rate of the identification of missing tag IDs from a group of size \( n = 1000 \) using the PEG-based codes with the IT decoding algorithm

Figure 5.7: The average error rate of the identification of missing tag IDs from a group of size \( n = 1000 \) using the PEG-based codes with the GE decoding algorithm
To confirm the performance of the PEG-based LDPC codes with that of Tomlinson et al. (2007), the (200, 3, 6) PEG-based LDPC code is constructed with $n=200$, $m=100$, and $j=3$ and then tested with the GE decoding algorithm. The constructed code has a girth equal to 10 and a group of 200 tag IDs is encoded through the constructed matrix to generate a codeword $x$ of length 300. Figure 5.8 shows the histogram of the unrecovered missing RFID tag IDs, with the average number of recovered missing tag IDs being 96.59. When a group of 100 tag IDs is encoded to generate a codeword $x$ of length 200, as normally done in the BEC, the average number of recovered erasures is 96.64. This discloses an interesting finding that for (200, 3, 6) PEG-based LDPC codes, the average number of recovered erasures is nearly equal to $m$, which is higher than the finding of Tomlinson et al. (2007) of 93.19. Appendix B presents the constructed (200, 3, 6) PEG-based LDPC matrix.

![Figure 5.8: The histogram of unrecovered missing tag IDs of the (200, 3, 6) PEG-based LDPC codes with the GE decoding algorithm](image)

To provide a preliminary comparison between the complexity and the time of the GE and IT decoding algorithms, the running time of decoding the PEG-based LDPC codes is measured.
on a 2.6 GHz Intel i7-6600U processor. For (49, 3, 7) PEG-based LDPC codes, the GE and IT decoding algorithms take about 0.63 and 0.021 milliseconds (ms), on average, respectively, to recover 15 missing tags. Increasing $j$ causes an increase in the average running time of the GE decoding to 0.96 ms and in the IT decoding to 0.052 ms. On the other hand, increasing the group size to 169, where $j = 3$, causes a large increase in the average time, especially for the GE decoding, to 4.6 ms, in order to recover 25 missing tag IDs, whilst the IT decoding only requires 0.084 ms. Figure 5.9 shows the average error rate of the identification of missing tag IDs from a group of $n=169$, where $j=4$ and the related decoding time for both the IT and GE decoding algorithms. It is clearly shown that the latter can recover more missing tag IDs than the former. However, its decoding time is higher due to the fact that the IT decoding algorithm has a linear decoding complexity of $O(x)$, whereas the GE has a complexity of $O(x^3)$, where $x$ is the codeword block length. Generally, the GE decoding algorithm provides the optimum decoding in terms of missing recovery capabilities. However, the complexity and thus, the time needed for the decoding procedure could make it impractical in real RFID systems, especially for a large group of tags. Thus, a good decoding strategy may combine the GE and IT decoding algorithms in order to make a compromise between the performance and complexity.

![Figure 5.9: The average error rate of the identification of missing tag IDs from a group of $n=169$, where $j=4$ and the related decoding time for both the IT and GE decoding algorithms](image)
5.4.2 PEG-based vs CRT-based LDPC Codes

This subsection evaluates the performance of the \((n, j, l)\) PEG-based and \((n, j, w)\) CRT-based LDPC codes with large group sizes with the IT and GE decoding algorithms. In order to derive a realistic baseline performance comparison, the same group size \(n\) and column weight \(j\) are used to build the matrices of the two codes. Table 5.2 compares the girth as well as the memory consumption of the PEG-based and CRT-based codes. Clearly, according to the results, the PEG-based codes outperform their counterpart by having larger girths and low memory consumption. Even though the \((512, 3, 19)\) and \((1024, 3, 29)\) PEG-based codes have girths similar to the CRT-based codes, each tag needs to record only 828-bit and 1245-bit, respectively, of additional information which is smaller than those of the CRT-based codes. The large number of parity check values \(m\) in the PEG-based codes may affect the decoding complexity. However, this can be simplified by using the improved hybrid decoding algorithm designed in the next chapter. Increasing the group size also increases the girth of the PEG-based codes to 10, while that of the CRT-based one remains stable. For the CRT-based codes, increasing \(w\) from 2 to 3 degrades the girth to 4 and increases the memory consumption. Figure 5.10 and Figure 5.11 show the performance of these codes in recovering missing tag IDs with the IT and GE decoding algorithms, respectively. It is clear that the PEG-based codes have advantages over the CRT-based codes in preventing small stopping sets with low memory consumption. Appendix B contains the constructed \((512, 3, 2)\) and \((512, 3, 3)\) CRT-based LDPC matrices as well as the \((512, 3, 6)\) and \((512, 3, 19)\) PEG-based LDPC matrices.

<table>
<thead>
<tr>
<th>Codes</th>
<th>Girth</th>
<th>(m)</th>
<th>Tag memory consumption in bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>((512, 3, 2)) CRT-based</td>
<td>6</td>
<td>82</td>
<td>948</td>
</tr>
<tr>
<td>((512, 3, 3)) CRT-based</td>
<td>4</td>
<td>30</td>
<td>2478</td>
</tr>
<tr>
<td>((512, 3, 6)) PEG-based</td>
<td>8</td>
<td>256</td>
<td>438</td>
</tr>
<tr>
<td>((512, 3, 19)) PEG-based</td>
<td>6</td>
<td>82</td>
<td>828</td>
</tr>
<tr>
<td>((1024, 3, 2)) CRT-based</td>
<td>6</td>
<td>106</td>
<td>1377</td>
</tr>
<tr>
<td>((1024, 3, 3)) CRT-based</td>
<td>4</td>
<td>38</td>
<td>3357</td>
</tr>
<tr>
<td>((1024, 3, 6)) PEG-based</td>
<td>10</td>
<td>512</td>
<td>486</td>
</tr>
<tr>
<td>((1024, 3, 29)) PEG-based</td>
<td>6</td>
<td>106</td>
<td>1245</td>
</tr>
</tbody>
</table>

Table 5.2: Comparisons of the PEG-based and CRT-based LDPC codes
Figure 5.10: The average error rate of the identification of missing tag IDs from a group of size $n = 512$ using the PEG-based and CRT-based codes with the IT decoding algorithm

Figure 5.11: The average error rate of the identification of missing tag IDs from a group of size $n = 512$ using the PEG-based and CRT-based codes with the GE decoding algorithm
5.4.3 PEG-based vs ACE-based LDPC Codes

The performance of the PEG-based and ACE-based LDPC codes with different group sizes are evaluated under the IT and GE decoding algorithms. The two codes are constructed using the same parameters, including \( n, m, \) and \( j \). Table 5.3 compares the girths as well as the memory consumptions of the PEG-based and ACE-based codes. The PEG-based codes outperform their counterpart by consuming lower memory overheads. Figure 5.12 shows the performance of the codes with the IT and GE decoding algorithms. It can be concluded that the PEG-based codes have advantages over the ACE-based ones in preventing small stopping sets with low memory consumption for low to medium amounts of missing tags. However, the ACE codes outperform their counterparts for high amounts of missing tags. Appendix B presents the constructed \((502, 3)\) PEG-based and ACE-based LDPC matrices.

<table>
<thead>
<tr>
<th>((n, j))</th>
<th>Codes</th>
<th>Girth</th>
<th>( m )</th>
<th>Tag memory consumption in bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(256, 3)</td>
<td>ACE-based</td>
<td>4</td>
<td>128</td>
<td>558</td>
</tr>
<tr>
<td></td>
<td>PEG-based</td>
<td>8</td>
<td></td>
<td>423</td>
</tr>
<tr>
<td>(502, 3)</td>
<td>ACE-based</td>
<td>4</td>
<td>251</td>
<td>612</td>
</tr>
<tr>
<td></td>
<td>PEG-based</td>
<td>8</td>
<td></td>
<td>423</td>
</tr>
</tbody>
</table>

Table 5.3: Comparisons of the PEG-based and ACE-based LDPC codes

![Figure 5.12: The average error rate of the identification of missing tag IDs from a group of size \( n = 502 \) using the PEG-based and ACE-based codes with the IT and GE decoding algorithm](image)

\[ \]
5.5 Conclusions

This chapter has presented the PEG-based LDPC codes for improving the recovery performance of missing RFID tag IDs in the extended grouping of group integrity application. A group of RFID tag IDs has been treated as packet symbols and encoded and decoded in a way similar to that of the BECs. Due to the mitigation of the PEG method for small stopping sets, it has been used to construct the parity check matrix $H$ of LDPC codes of size $m \times n$. Then, the matrix $H$ was utilised as a generator matrix to encode a group of $n$ RFID tag IDs and generate $m$ parity check values. The missing recovery performance of the constructed codes has been tested and evaluated with the IT decoding and GE decoding algorithms. The properties of the constructed PEG Tanner graphs have also been analysed and compared with the Tanner graphs of other codes. The simulation results show that the PEG-based LDPC codes can provide significant missing recovery enhancement with the same or less memory overheads compared to other existing LDPC-based codes.
6 Improved Hybrid IT/GE Decoding Algorithm

This chapter focuses on optimising the decoding complexity of the PEG-based LDPC code for the extended grouping of RFID tags using a hybrid (HB) decoding algorithm. It combines the IT with the GE decoding algorithms, with the basic idea of employing the IT decoding first. If it fails to recover all the erased symbols, the GE decoding is activated to complete the decoding process and resolve the simplified system. This HB decoding is improved by including an early stopping criterion to avoid unnecessary iterations of the IT decoding algorithm for undecodable blocks, thus saving decoding time. Extensive simulations have been carried out to analyse and assess the performance achieved with decoding the PEG-based LDPC codes under the improved HB decoding algorithm, both in terms of missing recovery capabilities and decoding complexities.

6.1 Early Termination of the IT Decoding Algorithm

Early termination of the IT decoding algorithm has been widely researched within correcting errors introduced by channels. In each iteration, the algorithm detects decodable blocks by checking the parity check constraints. If all parity check constraints are satisfied, the decoding terminates, otherwise, the decoding always completes $I_{\text{max}}$ iterations for undecodable blocks. The decoding time increases linearly with the number of decoding iterations. To avoid unnecessary decoding iterations when processing undecodable blocks, an effective stopping criterion is required to terminate the decoding process early and hence, save decoding time and tag power consumption (Mohsenin et al., 2011). Different early stopping criteria have been proposed for the IT decoding algorithm, aiming to detect undecodable blocks and to stop the decoding process in its early stage. These criteria can be divided into two types as follows.
Chapter 6 - Improved Hybrid IT/GE Decoding Algorithm

- One type utilizes log-likelihood ratios (LLRs), for example the criterion presented by Kienle and Wehn (2005). It identifies undecodable blocks using variable node reliability, which is the addition of the absolute values of all variable node LLRs. The decoding process is stopped, if the variable node reliability remains unchanged or is decreased within two consecutive iterations. This is a consequence of the observation that there is a consistent increase of the variable node reliability from a decodable block. Similar criteria have been proposed by Li et al. (2006), Cai et al. (2008), and Chen (2012), which monitor the convergence of the mean magnitude of the LLRs at the end of each iteration to identify undecodable blocks.

- The other types of criteria are those that observe the numbers of satisfied or unsatisfied parity check constraints. For example, the criterion in Shin et al. (2007) and the improved version in Alleyne and Sodha (2008) compare the numbers of satisfied parity check constraints in two consecutive iterations. If they are equal, the decoding process is halted. On the other hand, the criteria in Han and Liu (2010) and Mohsenin et al. (2011) use the number of unsatisfied parity check constraints to detect undecodable blocks.

During the HB decoding process of the PEG-based codes, the decoding time of the IT decoding for undecodable blocks becomes longer as the decoding iterates for $I_{\text{max}}$. Figure 6.1 shows the number of iterations as a function of the missing tags for two variants of group sizes, including 49 and 169 tags, where $j=3$. $I_{\text{max}}$ is set to 50 and 150, respectively. It is explicit that as the number of missing tags increases, iteration for $I_{\text{max}}$ starts to occur due to the existence of undecodable blocks. This can clearly be seen in the region where the IT decoding has failed, for example, when recovering more than 10 and 17 missing tags from a group of 49 and 169, respectively. Therefore, taking into account an early stopping criterion, which adaptively adjusts the number of decoding iterations, arises as an attractive solution to avoid unnecessary iterations and thus, save decoding time. Since the main advantage of the HB decoding is to reduce the GE decoding complexity, the frequency of its usage should be

88
as small as possible even for a large number of missing tags. In addition, when GE decoding is utilised, the size of the simplified system should be as small as possible. This suggests a stopping criterion for the PEG-based LDPC code that permits partial decoding of undecodable blocks up to a point where the IT decoding algorithm cannot recover any further missing tag IDs. The proposed criterion can distinguish between decodable and undecodable blocks. It does not cause any computation overheads, as it utilises the information available during the decoding process. Algorithm 6.1 outlines the improved IT decoding algorithm with the early termination criterion. Let \( v_j \) denote the set of variables in the \( j^{th} \) check equation of the code, and \( E_{j,v_i} \) is the outgoing message from the check node \( j \) to the variable node \( i \).

The improved algorithm is based on the observation of the amount of erasures \( n^e_I \) at the end of each iteration \( I \). It halts the decoding process, if the amount remains unchanged or does not decrease within two consecutive decoding iterations, as in Step 24. This is as a result of the observation that the decodable blocks exhibit a consistent decrease in the amount of erasures.

![Successful region of IT decoding](image1.png)

![Successful region of IT decoding](image2.png)

Figure 6.1: The iteration numbers required for decoding two group sizes: (a) \( n = 49 \); (b) \( n = 169 \), where \( j = 3 \), using the PEG-based LDPC codes
Algorithm 6.1: Improved IT Decoding

**Input:** the received codeword \( y \)

**Output:** \( M = [M_1, \ldots, M_n] \) where \( M_i \in \{0, 1, e\} \)

1: \( I = 1 \) \quad \triangleright \text{Initialisation}

2: \( n_q^{I-1} = n_e \)

3: for \( i = 1 \) to \( n \) do

4: \( M_i = y_i \)

5: end for

6: repeat

7: for \( j = 1 \) to \( m \) do

8: for all \( i \in v_j \) do

9: if a check node \( j \) receives only one \( M_i = 'e' \) then

10: \( E_{c_j, v_i} = \sum_{i' \in v_j, i' \neq i} M_{i'} \oplus c_j \) \quad \triangleright \text{Calculating check nodes messages}

11: else

12: \( E_{c_j, v_i} = 'e' \)

13: end if

14: end for

15: end for

16: for \( i = 1 \) to \( n \) do

17: if \( M_i = 'e' \) then

18: if there is at least one \( E_{c_j, v_i} = 0 \) or \( 1 \) then

19: \( M_i = E_{c_j, v_i} \) \quad \triangleright \text{Recovering missing tag ID using incoming messages}

20: end if

21: end if

22: end for
Chapter 6 - Improved Hybrid IT/GE Decoding Algorithm

23: \( n_e^I = \text{count 'e' in } M_i \ \forall \ i = 1 \ldots n \)

24: \[ \text{if } (M_i \neq 'e' \ \forall \ i = 1 \ldots n) \ \text{or } (I = I_{\text{max}}) \ \text{or } (n_e^{I-1} = n_e^I) \ \text{then} \]

25: \quad \text{Terminate the IT decoding and start the GE decoding}

26: \quad \text{else}

27: \quad \text{Increment } I

28: \quad n_e^{I-1} = n_e^I

29: \quad \text{end if}

30: \quad \text{until terminated}

To illustrate the impact of the early stopping criterion on decoding time, Figure 6.2 shows iteration numbers taken for various missing amounts to be compared with those of the conventional IT decoding, as depicted in Figure 6.1. One can observe that the stopping criterion greatly reduces the average number of iterations in the region where the IT decoding is not successful. Figure 6.2 shows that undecodable blocks need, at most, 16 and 22 iterations, respectively, when recovering more than 10 and 17 missing tags.

![Figure 6.2: The enhanced iteration numbers required for decoding two group sizes: (a) \( n = 49 \); (b) \( n = 169 \), where \( j = 3 \), using the PEG-based LDPC codes](image)
6.2 Missing Recovery Capability Analysis

To illustrate the performance of the improved HB decoding algorithm in recovering missing tag IDs, Figure 6.3 and Figure 6.4 show the average error rate as a function of the number of missing tags under the improved HB and GE decoding algorithms for the same parameters in Table 5.1. It is clear that the performance of the two decoding algorithms is identical, which means that the improved HB algorithm achieves the optimal missing recovery capabilities of the full GE decoding algorithm, since the remaining systems of the IT decoding are solved with this algorithm. Similar to the performance of the IT and GE decoding algorithms, in subsection 5.4.1 of Chapter 5, increasing $j$ increases the value of $m$ and thus, the $d_{\text{min}}$ of the code, which results in increasing the value of $d_{\text{stop}}$ and the rank of the matrix $H_k$. However, the amount of memory consumption increases as a consequence. For example, if 13 tags are missing from a group of $n = 49$, all their IDs can be recovered with at least $j = 3$ under the improved HB decoding. In this case, each tag records 432-bit additional information. However, if 16 IDs are missing, the column weight needs to be adjusted to 4 in order to achieve 100% reliable recovery (zero average error rate) and thus, each tag is required to store 576-bit additional information.

![Figure 6.3](image_url)

**Figure 6.3:** The average error rate of the identification of missing tag IDs from different group sizes: (a) $n = 25$; (b) $n = 49$ using the PEG-based LDPC codes with the GE and improved HB decoding algorithm.
Figure 6.4: The average error rate of the identification of missing tag IDs from different group sizes: (a) $n = 121$; (b) $n = 169$ using the PEG-based LDPC codes with the GE and improved HB decoding algorithm

6.3 Decoding Complexity Analysis

The relative complexities of the improved HB decoding algorithm are analysed in terms of decoding time and the number of operations required. The entire decoding complexity is equal to the complexity of the IT decoding plus that of the GE decoding, if stopping sets exist. This is mainly determined by the efficiency of the first stage of the IT decoding; the better its performance, the less the complexity and thus, the time of the improved HB decoding.

6.3.1 Decoding Time Analysis

Two experiments are conducted:

1) Decoding time measurements when the IT decoding fails and thus, the GE decoding is required. This is the worst case from a decoding point of view;
2) Decoding time measurements as a function of the number of missing tags to compare two cases, one where the IT decoding is successful and the other where the IT decoding fails and the GE decoding is needed. This experiment is more illustrative of real situations.

Decoding Time when the GE Decoding is Required (Worst Case): Figure 6.5 illustrates the decoding time during the GE decoding process of the improved HB decoding for a group of 169 tags, where \( j = 3 \). By changing the number of missing tags, the resulting average simplified system size changes, which affects the decoding time. The figure illustrates this time as a function of the simplified system size and the related result. The systems size is equal to the number of unrecovered missing tag IDs after the IT decoding phase. The region where the IT decoding recovers some missing tag IDs is only considered in this analysis. In the second region, where the IT decoding cannot recover any missing tag IDs, the GE decoding time is equal to that of the full systems. The improved HB decoding algorithm efficiently decreases the average decoding time of the GE decoding, because the complex GE decoding is only utilised when required over a simplified system resulting from the IT decoding. The histogram demonstrates that all simplified systems have an average decoding time close to their minimum, which indicates that most systems have a decoding time below their average. It can clearly be seen that the maximum average decoding time remains nearly below 0.004 s related to a maximum decoding time of 0.004 s. In contrast, the GE decoding on the full systems is slower, especially for a large group of tags. When the system size increases, the GE decoding time also increases, following an \( O(x^3) \) law. This is confirmed in Figure 6.6 part (b) presented in the next section. For a group of size 169, where \( j = 3 \), it is evident that the decoding time lasts, on average, 0.006 s.
Figure 6.5: Histogram of the GE decoding time with respect to the simplified system size for a group of size $n = 169$, where $j = 3$, using the PEG-based LDPC codes

**Decoding Time of the Improved Hybrid Decoding:** Figure 6.6 and Figure 6.7 show the average decoding time as a function of the number of missing tags under the IT with early termination, the GE, and improved HB decoding algorithms. It is evident that the decoding time depends on the missing amounts experienced. With small numbers of missing tags, the decoding time of the improved HB decoding is fast and is similar to that of the IT decoding. This is clear in the first region, where the IT decoding is successful. After this, the decoding time of the improved HB decoding progressively increases since the IT decoding turns out to be ineffective in recovering some missing tags and thus, the size of the simplified system, which is solved by the GE decoding, starts to increases. On the other hand, due to the stopping criterion, the IT decoding still continues, trying to decode missing tags as much as possible, as shown on the figure. Then, above a certain threshold which depends on the missing recovery capabilities of the IT decoding, the decoding time of the improved HB decoding gradually approaches the full GE decoding as shown in the third region. It is clear that the $n$ and $j$ parameters also have major impacts on the decoding time. As $n$ or $j$ increases,
the improved HB decoding time increases too. This is due to the increasing in the number of operations required by the IT and GE decoding algorithms, as seen in the next subsection. If the HB decoding time is compared with the GE decoding time for the full system, it can be concluded that this decoding is faster for small to medium numbers of missing tags, even when the GE decoding is required.

Figure 6.6: The average decoding time for recovering missing tag IDs from two group sizes: (a) $n = 49$ and (b) $n = 169$, where $j = 3$, using the PEG-based LDPC codes under the IT with early termination, the GE, and improved HB decoding algorithms

Figure 6.7: The average decoding time for recovering missing tag IDs from two group sizes: (a) $n = 49$ and (b) $n = 169$, where $j = 4$, using the PEG-based LDPC codes under the IT with early termination, the GE, and improved HB decoding algorithms
6.3.2 Operational Complexity Analysis

This subsection focuses on analysing the operational complexity of decoding the PEG-based LDPC codes with the improved HB decoding algorithm.

6.3.2.1 IT Decoding Algorithm Analysis

During the IT decoding process, if at row $i$ of $H$ there exists more than one erasure, then there is insufficient information to decode each erasure. Thus, these rows are not calculated during the operational complexity analysis, except for counting the number of bit operations. A bit operation is a test operation on a single bit variable. The search for erasures on the $i$-th row needs $n$ bit operations and for all $m$ rows, $m \times n$ bit operations are needed. In each iteration of the IT decoding algorithm, the number of erasures $n_e$ is reduced. The IT decoding begins with the parity check equation with a single erasure. For each equation having only a single erasure, the single recovery process needs $n-1$ multiplications and $n-1$ XOR operations. Therefore, the required number of operations for the IT decoding algorithm ($N_{IT}^o$) in terms of $n_e$ erasures with $I_{max}$ iterations is:

$$N_{IT}^o = \sum_{l=2}^{I_{max}} ( (n_e^{l-1} - n_e^l) (2n−2) + m \times n )$$  \hspace{1cm} (6.1)

6.3.2.2 GE Decoding Algorithm Analysis

The GE decoding algorithm solves Equation (5.3), by reducing $H_k$ to the row echelon form. The process of the algorithm is composed of two phases: The forward elimination phase to find $n_e$ independent equations and an equation-solving phase to solve the set of $n_e$ linearly independent equations. The forward elimination phase first reduces the system into an upper triangular form, which can be done using elementary row operations, as follows: Starting from $i = 0$, search for a nonzero entry in column $i$, the pivot, with row $j \geq i$. Rearrange rows $i$ and $j$ and then, add the row $i$ to all the rows having nonzero elements of column $i$. At the
same time, the same process is performed on the simplified right-hand side (RHS) of the system, that is, the right entry of the i-th row is added to right entries of related rows. The search for the first 1 on the i-th column needs a maximum of \( m \) bit operations and for all \( n_e \) columns, \( m \times n_e \) bit operations are needed. Once a “1” entry is found in row \( j \), test whether \( H_{ki_b} = 1 \) for \( b = 1, 2, \ldots, j-1, j+1, \ldots, m \), and then, add row \( j \) to row \( b \). This phase needs a single bit operation, which checks whether \( H_{ki_b} = 1 \) and \( n_e +1 \) operations for the addition of row \( j \) to row \( b \). Since these operations have to be performed for \( b = 1, 2, \ldots, j-1, j+1, \ldots, m \) and \( i = 1, 2, \ldots, n_e \), a total of \( n_e(m - 1)(2 + n_e) \) operations is needed. Hence, the total number of operations needed by the GE decoding algorithm to triangulate an \( m \times n_e \) matrix is:

\[
(m \times n_e) + n_e (m - 1)(2 + n_e) \tag{6.2}
\]

\[
mn_e^2 + 3mn_e - n_e^2 - 2n_e \tag{6.3}
\]

The algorithm continues with a backward substitution phase, recursively resolving erased symbols until the solution is found. The last symbol of an upper triangular matrix gives the value of the symbol on the RHS. Then, this value is replaced in all the equations in which it is included. With \( n_e \) equations to be solved in a linear system, the calculated value of each decoded bit requires \( n_e \) multiplications and \( n_e - 1 \) XOR operations. Thus, the required number of operations for the second phase is:

\[
n_e (2n_e - 1) \tag{6.4}
\]

\[
2n_e^2 - n_e \tag{6.5}
\]

For the RHS of the system, it needs \( n_e \) multiplications and \( n_e \) XOR operations, where \( n_e \) is the number of known tag IDs. Therefore, the required number of operations to simplify the RHS of the system is:

\[
2n_e \times m \tag{6.6}
\]
For a complete GE decoding algorithm, the required number of operations \( N_o^{GE} \) is obtained by adding Equations (6.3), (6.5), and (6.6) to get:

\[
N_o^{GE} = 3n_e (m - 1) + n_e^2 (m + 1) + 2mn_e. 
\]  

(6.7)

### 6.3.2.3 Improved Hybrid Decoding Algorithm Analysis

The improved HB decoding algorithm first employs IT decoding with an early stopping criterion. If this fails to recover all the erased symbols owing to the existence of stopping sets, GE decoding is activated to complete the decoding process and to resolve the simplified system. Since the improved HB decoding algorithm combines the IT decoding and GE decoding algorithms, the entire decoding complexity is equal to the complexity of IT decoding plus that of GE decoding to solve the simplified system. Three cases are considered here relating to the behaviour of the IT decoding algorithm.

- The first case is when the IT decoding algorithm successfully recovers all missing tag IDs and the complexity in this case includes only that of this algorithm.
- The next case is when the IT decoding algorithm fails to recover some missing tag IDs and the complexity here includes the IT decoding computations and GE computations for the unrecovered missing IDs after the IT decoding algorithm.
- The final case is when the IT decoding algorithm fails to recover any missing tag IDs, and the GE computations on the full system are only considered here. The complexity of the improved HB decoding algorithm is illustrated in Table 6.1, where \( n_e^* \) is the number of unrecovered missing tag IDs after the IT decoding phase.

<table>
<thead>
<tr>
<th>Case</th>
<th>Number of Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Case</td>
<td>( N_o^{IT} )</td>
</tr>
<tr>
<td>Second Case</td>
<td>( N_o^{IT} + 3n_e^* (m - 1) + n_e^2^* (m + 1) + 2mn_e^* )</td>
</tr>
<tr>
<td>Third Case</td>
<td>( N_o^{GE} )</td>
</tr>
</tbody>
</table>

Table 6.1: Number of operations for the improved HB decoding algorithm
Figure 6.8 and Figure 6.9 depict a comparison of the operational complexity between the IT, GE, and improved HB decoding algorithms for two group sizes, including 49 and 169, where \( j \) varies between 3 and 4. It is clear that the decoding complexity depends on the missing amounts \( n_e \). For a small amount of \( n_e \) where IT decoding is successful, the complexity of the improved HB decoding is similar to that of IT decoding, such as when \( n_e \) is below 10 for a group of 49 tags and \( j = 3 \). For a medium amount of \( n_e \) where IT decoding fails to recover some missing tag IDs, the complexity of IT decoding starts to decrease due to the stopping criterion, whereas the complexity of the improved HB decoding gradually approaches that of the full GE decoding, because the latter is required more often. Then, above a threshold where IT fails to recover any missing tag IDs, the complexity of the improved HB decoding is equal to that of the full GE decoding algorithm. It is clear that the PEG method parameters also affect the decoding complexity. As a group size \( n \) or column weight \( j \) increases, the improved HB decoding complexity also increases as shown in the black ovals on the y-axis. This can be attributed to particular phenomena. The complexity of the IT decoding also depends on the number of XOR and multiplication operations, which is determined by the number of variables in each parity check constraint, whereas the complexity of GE decoding mainly depends on the number of linear equations and the number of variables. Therefore, by increasing \( n \) or \( j \), the number of operations required by the IT and GE decoding algorithms is increased accordingly.

It can be concluded that the improved HB decoding considerably reduces the decoding complexity and thus the time of the full GE decoding for small and medium amounts of missing tags, which are proportional to the total number of tags in a group. For example, missing a medium amount of tags from a group of 49 tags would be considered as a small amount for a group of 169 tags. The improved HB decoding could enhance the group integrity applications, such as grouping or categorising animals, based on certain features, to produce different food in a factory.
Figure 6.8: Operational complexity comparison between the IT, GE, and improved HB decoding algorithms for a group of \( n = 49 \), where (a) \( j = 3 \) and (b) \( j = 4 \)

Figure 6.9: Operational complexity comparison between the IT, GE, and improved HB decoding algorithms for a group of \( n = 169 \), where (a) \( j = 3 \) and (b) \( j = 4 \)

6.4 Conclusions

The decoding complexity of the PEG-based LDPC codes has been optimised using an improved HB IT/GE decoding algorithm. This improved HB decoding includes an early
Chapter 6 - Improved Hybrid IT/GE Decoding Algorithm

Stopping criterion to avoid unnecessary iterations of IT decoding for undecodable blocks and thus, saving decoding time. The stopping criterion greatly reduces the average number of decoding iterations for undecodable blocks and therefore, reduces decoding time when compared to the conventional IT decoding algorithm. It has been shown that the improved HB decoding algorithm achieves the optimal missing recovery capabilities of the full GE decoding, since the remaining systems of IT decoding are solved with the GE decoding algorithm. The relative complexities of the improved HB decoding algorithm have been analysed both in terms of decoding time and the number of operations required. Simulation results have been presented demonstrating that the improved HB decoding algorithm considerably reduces the operational complexity and thus, the time of the full GE decoding for small to medium amounts of missing RFID tags, up to a certain threshold value, which depends on the missing recovery capabilities of IT decoding. This makes the improved HB decoding algorithm a promising candidate for decoding the PEG-based LDPC codes in the extended grouping of group integrity application in RFID systems.
7 Adaptive Hybrid Decoding Algorithm

In this chapter, the joint use of the two decoding components of the improved HB decoding algorithm is adapted based on the missing amounts of RFID tags in order to avoid the IT decoding when the missing amount is larger than a threshold and thus, save more decoding time. The optimum value of the threshold value has been investigated for different missing amounts of tags and different parameters of the PEG method through empirical analysis. Various simulations have been conducted in order to analyse and evaluate the impact of the threshold value on the improved HB decoding time. This chapter also analyses and assesses the missing recovery performance of various short-length irregular PEG-based LDPC codes constructed with different variable degree sequences.

7.1 Threshold Analysis

The improved HB decoding algorithm can be easily tailored to avoid the IT decoding phase and thus, save more decoding time when the number of missing tags exceeds a threshold ($\theta_{th}$). This threshold represents the capability of the IT decoding algorithm to recover missing tag IDs, which depends on the missing amounts $n_e$ and the parameters of the PEG method, including $n$ and $j$. The optimum value for $\theta_{th}$ is obtained by empirical analysis. The simulation is implemented with different parameters of the PEG method over a range of the numbers of missing tags in the region where the IT decoding recovers some missing tag IDs. Figure 7.1 demonstrates the reduction in the number of missing tags across iterations without the stopping criterion for a group of 169 tags, where $j = 3$. It can clearly be seen from part (a) of the figure, that for values of $n_e \leq 40$, the number of missing tags decreases, but for values of $n_e \geq 70$, the number of missing tags does not, even as $I \rightarrow I_{max}$. The transition value of $n_e$ between these two results is called the $\theta_{th}$ value. To close in on $\theta_{th}$ further, the simulation is applied to
a range of numbers of missing tags between these two values. Part (b) shows that, for values of \( n_e \geq 58 \), the number of missing tags does not decrease by one, even as \( I \) gets very large, but for values of \( n_e \leq 47 \), this decreases by at least two missing tags. To close in on \( \theta_{th} \) even further, the simulation is applied between these two values as in Part (c) and therefore, it can be concluded that the value of \( \theta_{th} \) is in the range 50–53. Even though the value of \( \theta_{th} \) is larger than the amount of the parity check values, the GE decoding phase of the HB decoding can still recover some missing tag IDs, as depicted in Figure 5.5 of Chapter 5. Table 7.1 illustrates the values of \( \theta_{th} \) for different parameters of the PEG method. The results reveal that this value increases by increasing \( n \) or \( j \). This is because larger LDPC codes built with the PEG methods have elegant properties in regard to their girth.

![Figure 7.1: Reduction in the number of missing tags with iterations for n=169 tags, where j=3](image)

(a)  
(b)  
(c)
Parameters \((n, j, l)\) & The value of \(\theta_{th}\)  
(25, 3, 5) & 9-11  
(25, 4, 5) & 11-13  
(49, 3, 7) & 22-24  
(49, 4, 7) & 23-25  
(121, 3, 11) & 40-43  
(121, 4, 11) & 42-45  
(169, 3, 13) & 50-53  
(169, 4, 13) & 54-57  

Table 7.1: The values of \(\theta_{th}\) for different parameters of the PEG method

To illustrate the impact of \(\theta_{th}\) on the improved HB decoding time, Figure 7.2 shows a plot of the average decoding time of the adaptive algorithm for a group of 49 and 169, where \(j = 3\). The process of the simulation is repeated for each value of \(\theta_{th}\) in the range depicted in Table 7.1 in order to achieve an optimum time. Compared with the results obtained in Figure 6.6, the results shown in Figure 7.2 indicate that the adaptive decoding algorithm is very efficient in decreasing the average decoding time of the improved HB decoding and thus, save more decoding time of up to 10% for large amounts of missing RFID tags where the IT decoding fails to recover any missing tags.

![Figure 7.2: The average decoding time for recovering missing tag IDs from two group sizes: (a) \(n = 49\) and (d) \(n = 169\), where \(j = 3\), using the PEG-based method under the IT with early termination, the GE, and improved HB decoding algorithms](image-url)
7.2 Irregular PEG-based LDPC Codes

Grouping RFID tags with regular codes provides an equal missing protection level for all the members since each tag is involved in the same number of parity check constraints (or subgroups). On the other hand, there are applications where not all the members need the same missing protection level. For example, in the application of tracking personal luggage at the airport, some bags are regarded as more important than others, because they contain valuable items. Therefore, the important tagged objects should have higher missing protection levels, such that they are more likely to be recovered. Since variable nodes of higher degrees in irregular codes converge faster (Luby et al., 2001b), they are assigned to valuable tagged objects.

Hu et al. (2002; 2005) built irregular LDPC codes using the PEG method, where IT decoding exhibits a significant performance compared with randomly constructed codes. This performance is not surprising in the light of a study rigorously accentuating the power of irregular graphs in designing erasure correcting codes conducted by Luby et al. (2001b). Some studies have been dedicated to examining the convergence thresholds for regular and irregular LDPC codes, such as those by Luby et al. (1997) and Richardson et al. (2001). The results show that the thresholds of irregular LDPC codes generally outperform regular codes.

This section investigates the performance of half rate short-length irregular PEG-based LDPC codes constructed from different variable node degree sequences, where \( n = 256 \). In the PEG construction, the variable node distribution is only considered and the parity check node distribution is kept to be as uniform as possible (Hu et al., 2001). The half rate irregular PEG-based codes are constructed based on the optimised degree distribution given in Table I and II of Richardson et al. (2001), with maximum variable node degrees 5, 8, 11, 15, and 20. On the other hand, five regular codes are constructed with the same parameters (with \( n=256, m=128, \) and \( j=3 \)), but with different seeds in order to get different distributions. It has been remarked
that the simulated performance for the five (256, 3, 6) regular codes exhibits identical 
behaviour and for this reason only one code is presented in the plot. Figure 7.3 shows the 
performance of the constructed codes, where it is clear that the performance of the irregular 
codes outperforms the regular one in the first regions where IT decoding is successful. This 
discloses that for short block lengths, the density evolution is an effective method for the 
designing of good irregular PEG-based LDPC codes. However, their performances are lower 
than the regular code in the second region, where the IT decoding recovers some missing tag 
IDs, as clearly illustrated inside the orange oval. Among the constructed irregular codes, the 
one with the maximum variable degrees of 8 achieves the best performance in the first region. 
The performance of the regular code versus the irregular codes is also clear when increasing 
n to 512, as illustrated in Figure 7.4. 

![Image](image.png)

**Figure 7.3:** The average error rate of the identification of missing tag IDs for irregular and 
regular PEG-based codes with \( n=256 \) under the IT decoding algorithm
Figure 7.4: The average error rate of the identification of missing tag IDs for irregular and regular PEG-based codes with \( n=512 \) under the IT decoding algorithm

To gain a clear insight into the second region where the IT decoding fails to recover some missing tag IDs, especially inside the orange oval, the previous empirical threshold analysis conducted in section 7.1 is extended to compare the performance of the irregular code with maximum variable degrees of 8 and the regular one. From Figure 7.5, one can observe that the capability of the regular code in recovering missing tag IDs in the second region exceeds that of the irregular one and thus, its \( \theta_{th} \) value is larger as marked by the green oval.

Figure 7.5: Reduction in the number of missing tags with iterations for: (a) regular and (b) irregular PEG-based LDPC code with \( n=256 \)
Figure 7.6 shows the influence of changing the fractions of low degree variable nodes, particularly $\lambda_2$ and $\lambda_3$, on the performance of half rate short-length irregular codes, while the fractions of high degree variable nodes are kept the same. Table 7.2 shows the variable degree sequences of the simulated irregular codes based on Table 2.4 of Tjhai (2007). The results show that as the fraction of low degree variable nodes increases, the performance of the IT decoding in recovering missing tag IDs also increases in the two regions, as demonstrated by Code 3, Code 4, Code 5, and Code 6. The main reason for this behaviour is that these irregular codes with a high fraction of low degree variable nodes have a higher minimum Hamming distance $d_{\text{min}}$ and a lower amount of short cycles. Among all of these codes, Code 6 achieves the best performance.

![Graph showing the influence of changing the fractions of low degree variable nodes on the performance of half rate short-length irregular PEG-based LDPC codes with $n=256$](image)

**Figure 7.6: Influence of changing the fractions of low degree variable nodes on the performance of half rate short-length irregular PEG-based LDPC codes with $n=256$**

<table>
<thead>
<tr>
<th>Code</th>
<th>$\lambda_2$</th>
<th>$\lambda_3$</th>
<th>$\lambda_4$</th>
<th>$\lambda_5$</th>
<th>$\lambda_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code 1</td>
<td>0.200</td>
<td>0.100</td>
<td>0.550</td>
<td>0.050</td>
<td>0.100</td>
</tr>
<tr>
<td>Code 2</td>
<td>0.250</td>
<td>0.200</td>
<td>0.400</td>
<td>0.050</td>
<td>0.100</td>
</tr>
<tr>
<td>Code 3</td>
<td>0.350</td>
<td>0.400</td>
<td>0.100</td>
<td>0.050</td>
<td>0.100</td>
</tr>
<tr>
<td>Code 4</td>
<td>0.450</td>
<td>0.200</td>
<td>0.200</td>
<td>0.050</td>
<td>0.100</td>
</tr>
<tr>
<td>Code 5</td>
<td>0.500</td>
<td>0.175</td>
<td>0.175</td>
<td>0.050</td>
<td>0.100</td>
</tr>
<tr>
<td>Code 6</td>
<td>0.550</td>
<td>0.150</td>
<td>0.150</td>
<td>0.050</td>
<td>0.100</td>
</tr>
</tbody>
</table>

**Table 7.2: Variable degree sequences for irregular codes in Figure 7.6**
Figure 7.7 illustrates the influence of changing the high degree variable nodes of half rate short-length codes on the performance of recovering missing tag IDs. Table 7.3 shows the variable degree sequences of the simulated irregular PEG-based LDPC codes. These irregular codes have the same degree sequences except for their maximum variable node degree. It is evident that the recovery performance of the irregular code that has a maximum variable node degree of 7 outperforms other simulated irregular codes in both regions. On the other hand, the recovery performance degrades as the irregular code has a higher maximum variable node degree, as demonstrated by Code 11. This is due to a specific phenomenon. Increasing the maximum variable node degree causes an increase in the amount of short cycles.

![Figure 7.7: Influence of changing high degree variable nodes on the performance of half rate short-length irregular PEG-based LDPC codes with n=256](image)

<table>
<thead>
<tr>
<th>Code</th>
<th>$\lambda_2$</th>
<th>$\lambda_3$</th>
<th>$\lambda_4$</th>
<th>$\lambda_5$</th>
<th>$\lambda_7$</th>
<th>$\lambda_9$</th>
<th>$\lambda_{11}$</th>
<th>$\lambda_{13}$</th>
<th>$\lambda_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code 7</td>
<td>0.550</td>
<td>0.150</td>
<td>0.150</td>
<td>0.050</td>
<td>0.100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code 8</td>
<td>0.550</td>
<td>0.150</td>
<td>0.150</td>
<td>0.050</td>
<td>0.100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code 9</td>
<td>0.550</td>
<td>0.150</td>
<td>0.150</td>
<td>0.050</td>
<td>0.100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code 10</td>
<td>0.550</td>
<td>0.150</td>
<td>0.150</td>
<td>0.050</td>
<td></td>
<td></td>
<td></td>
<td>0.100</td>
<td></td>
</tr>
<tr>
<td>Code 11</td>
<td>0.550</td>
<td>0.150</td>
<td>0.150</td>
<td>0.050</td>
<td></td>
<td></td>
<td></td>
<td>0.100</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3: Variable degree sequences for the irregular codes in Figure 7.7

Opposite to Figure 7.6, Figure 7.8 illustrates the influence of changing the fractions of high degree variable nodes on the recovery performance of half rate short-length irregular PEG-
based LDPC codes, while the fractions of low degree variable nodes are kept the same. Table 7.4 shows the variable degree sequences of the simulated irregular codes. In contrast to the findings of Tjhai (2007) on additive white Gaussian noise (AWGN) channels, the results show that increasing the fraction of the high degree variable nodes degrades the recovery performance in the two regions, as demonstrated by the performance of Code 14, Code 18, and Code 22. One the other hand, Code 13, Code 17, and Code 21, which have the smallest fraction of high degree variable nodes achieve the best performance among other simulated irregular codes in their group.

Figure 7.8: Influence of changing the fractions of high degree variable nodes on the performance of half rate short-length irregular PEG-based LDPC codes with \( n=256 \)
Chapter 7 - Adaptive Hybrid Decoding Algorithm

<table>
<thead>
<tr>
<th>Code</th>
<th>$\lambda_2$</th>
<th>$\lambda_3$</th>
<th>$\lambda_4$</th>
<th>$\lambda_5$</th>
<th>$\lambda_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code 12</td>
<td>0.250</td>
<td>0.200</td>
<td>0.400</td>
<td>0.050</td>
<td>0.100</td>
</tr>
<tr>
<td>Code 13</td>
<td>0.250</td>
<td>0.215</td>
<td>0.400</td>
<td>0.050</td>
<td>0.085</td>
</tr>
<tr>
<td>Code 14</td>
<td>0.250</td>
<td>0.171</td>
<td>0.400</td>
<td>0.050</td>
<td>0.129</td>
</tr>
<tr>
<td>Code 15</td>
<td>0.250</td>
<td>0.186</td>
<td>0.400</td>
<td>0.050</td>
<td>0.114</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code 16</td>
<td>0.250</td>
<td>0.200</td>
<td>0.400</td>
<td>0.050</td>
<td>0.100</td>
</tr>
<tr>
<td>Code 17</td>
<td>0.250</td>
<td>0.200</td>
<td>0.415</td>
<td>0.050</td>
<td>0.085</td>
</tr>
<tr>
<td>Code 18</td>
<td>0.250</td>
<td>0.200</td>
<td>0.371</td>
<td>0.050</td>
<td>0.129</td>
</tr>
<tr>
<td>Code 19</td>
<td>0.250</td>
<td>0.200</td>
<td>0.386</td>
<td>0.050</td>
<td>0.114</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code 20</td>
<td>0.250</td>
<td>0.200</td>
<td>0.400</td>
<td>0.050</td>
<td>0.100</td>
</tr>
<tr>
<td>Code 21</td>
<td>0.250</td>
<td>0.200</td>
<td>0.400</td>
<td>0.065</td>
<td>0.085</td>
</tr>
<tr>
<td>Code 22</td>
<td>0.250</td>
<td>0.200</td>
<td>0.400</td>
<td>0.021</td>
<td>0.129</td>
</tr>
<tr>
<td>Code 23</td>
<td>0.250</td>
<td>0.200</td>
<td>0.400</td>
<td>0.036</td>
<td>0.114</td>
</tr>
</tbody>
</table>

Table 7.4: Variable degree sequences for the irregular codes in Figure 7.8

### 7.3 Conclusions

The joint use of the two decoding components of the improved HB decoding algorithm is adapted based on the missing amounts of RFID tags in order to avoid IT decoding when the missing amount is larger than a threshold and thus, save more decoding time. The optimum value of the threshold value has been investigated for different missing amounts of tags and different parameters of the PEG method through empirical analysis. It has been shown that the adaptive decoding algorithm is very efficient in decreasing the average decoding time of the improved HB decoding for large amounts of missing RFID tags where IT decoding fails to recover any missing tags. More simulations have been conducted to analyse and evaluate the missing recovery performance of various short-length irregular PEG-based LDPC codes constructed with different variable degree sequences. It has been demonstrated that the irregular PEG-based LDPC codes exhibit noticeable missing recovery enhancements compared to the regular PEG-based LDPC codes in the region where IT decoding is successful. However, their performances are degraded in the second region where IT decoding can recover some missing tag IDs. It has been empirically found that for half rate
short block lengths, the density evolution method is good for irregular PEG construction. Simulation results have also shown that as the fraction of low degree variable nodes increases, the performance of IT decoding in recovering missing tag IDs also increases in the two regions. On the other hand, increasing the fraction of the high degree variable nodes degrades the recovery performance in the two regions. In addition, the recovery performance of the code that has a maximum variable node degree of 7 outperforms other simulated codes with maximum variable node degrees of 9, 11, 13, and 15 in both regions.
8 Bloom Filters

A Bloom filter (BF) is a simple data structure that can store the items of a set in a space efficient way and support membership queries that ask: “Is element a in set B?” Burton Bloom presented BFs in 1970 (Bloom, 1970). Since that time, BFs have received increased attention and been extensively employed in various systems due to their space efficiency. This has been reflected in the latest studies and numerous new proposed algorithms that are directly or indirectly based on BFs, as shown in the existing surveys targeted different systems other than RFID systems and accomplished by Broder and Mitzenmacher (2004), Tarkoma et al. (2012); Geravand and Ahmadi (2013), Saravanan and Senthilkumar (2015), and Luo et al. (2018). Despite BFs enabling false positives, the space optimisation always outweighs this limitation in most practical applications as long as the probability of the false positive is sufficiently small and controlled. This chapter provides an overview of BFs with a focus on their construction, features, limitations, variations, and the ways in which they have been utilised in some systems, especially in RFID ones. This review gives preliminary information relevant to the designing of the novel redundant tag information collection protocol presented in the next chapter.

8.1 Preliminaries of Bloom Filters

A BF of a set \( W = \{w_1, w_2, \ldots, w_y\} \) containing \( y \) elements is described by a hashed Boolean vector of \( b \) bits, all initialised to 0. A BF uses \( k \) independent hash functions \( h_1, \ldots, h_k \), which map each element \( w \in W \) to the bits \( h_z(w) \in \{1, 2, \ldots, b\} \) for \( 1 \leq z \leq k \) that are set to 1. For ideal performance, each of the \( k \) hash functions should be a part of the class of universal hash functions. In other words, it should hash each item to a uniform random number. In order to check whether an item \( q \in W \), a \( k \) hash function is applied to \( q \) and whether all \( h_z(q) \) have
been set to 1 is checked, otherwise, \( q \notin W \). Testing a membership using a BF may yield a false positive due to hash collisions. That is, it suggests that an item \( q \in W \), but it is not when all indexed bits were previously set to 1 by other items. Figure 8.1 illustrates an example of a BF, where the filter is initialised with zeros. Each item \( w_i \) in the set is hashed \( k \) times and each hash result represents a bit location which is set to 1. To test whether an element \( q_i \) is a member in the set, it is hashed \( k \) times and the related bits in the filter are checked. The element \( q_1 \) is definitely not a member because one of the related bits is 0. On the other hand, the element \( q_2 \) is either a member or the related bits have by chance been set to 1 during the insertion of other elements.

![Figure 8.1: An example of a Bloom filter](image)

As abovementioned, even though the BF allows false positives, for many applications in practical situations the space savings always outweigh this limitation when the false positive rate is controlled to be adequately small (Broder and Mitzenmacher, 2004). According to Bloom (1970), the probability of a false positive for an element not in the set or the false positive rate \( P_{FP} \) of the BF constructed from \( m \) elements is derived as follows. By assuming that a hash function chooses each position in the BF with equal probability, the probability that a certain bit is not set to 1 by a hash function when adding an element to the filter is:

\[
\text{Probability of a bit not being set to 1 by a hash function} = \left( 1 - \frac{1}{2^k} \right)^m
\]
With \( k \) hash functions, the probability that none of them are set a certain bit to 1 is:

\[
(1 - \frac{1}{b})^k
\]  

(8.2)

The probability that a given bit is remain 0 after adding \( m \) elements to the BF is:

\[
(1 - \frac{1}{b})^{mk}
\]  

(8.3)

Therefore, the probability that the bit is 1 is:

\[
1 - (1 - \frac{1}{b})^{mk}
\]  

(8.4)

During membership testing, an element is said to be a member of a set, if all of the \( k \) positions in the BF are set to 1. Consequently, the probability of this occurring when the element is not a member can be represented as:

\[
P_{FP} = (1 - (1 - \frac{1}{b})^{mk})^k
\]  

(8.5)

From the Taylor series expansion, the value of \( 1 - 1/b \) is almost the same as \( e^{-1/b} \) and thus:

\[
P_{FP} \approx \left( 1 - e^{\frac{-mk}{b}} \right)^k
\]  

(8.6)

\( P_{FP} \) can be reduced by reducing the value of the right-hand side of Equation (8.5). The minimum value of \( P_{FP} \) can be found by taking the derivative of Equation (8.5) with respect to \( k \) as:

\[
\frac{dP_{FP}}{dk} = 0
\]  

(8.7)

which gives the optimal value of \( k \) as:

\[
k = \frac{b}{m} \times \ln 2
\]  

(8.8)

and thus, the minimum value of \( P_{FP} \) is \( 0.61285^{b/m} \). At this point, the BF achieves a balanced configuration in which nearly half of the bits are set to 0 and half are set to 1. The more elements that are added to the set, the higher the probability of false positives. Therefore,
with the given $k$, to have a fixed false positive probability, the value of $b$ must increase linearly with the number of elements added to the BF. As stated in Gremillion (1982) and Mullin (1983), the probability of the false positive in practice is larger than the value given by Equation (8.5). Theoretically, Bose et al. (2008) showed that Equation (8.5) offers a lower bound of the false positive rate. Given a desired $P_{FP}$ and the number of elements $m$, the optimal length of the BF can be computed as:

$$b = -\frac{m \times \ln P_{FP}}{(\ln 2)^2} \quad (8.9)$$

For example, if the desired $P_{FP}$ is 0.0001, signifying accuracy of 99.99 %, the optimal length of the BF is around $19.2 \times m$ bits. In practice, the probability of a false positive $P_{FP}$ of a BF depends on three parameters, including the number of elements added to the set $m$, which is application dependable and cannot be controlled, the number of bits $b$ or the length of the BF, and the number of hash functions $k$. To meet the accuracy requirement, the value of $b$ should be optimised. There is a trade-off between the probability of the false positive and the length of the BF. A lower $P_{FP}$ requires a larger value of $b$, which may increase the communication overheads of transmitting the BF between the reader and tags or the memory spaces to store the BF. Hence, for applications where the time or the space is critical, additional strategies may be required to remedy misjudged elements. However, the simulation results in subsection 9.6.3 of Chapter 9 show that the designed protocol is faster than other collection protocols even with the lowest tested value of $P_{FP}$. As $k$ is proportional to $b$, increasing the latter causes an increase in the former. Implementing $k$ hash functions on tags may cause processing overheads, because each tag needs to perform the calculations independently by using its limited resources. On the other hand, the implementation of hash functions can be simplified to decrease the complexity of a tag’s circuit, as presented in subsection 9.5.4 of Chapter 9.
8.2 Bloom Filter Features

BFs exhibit the following features, which attract the designing of the novel redundant tag information collection protocol in this research.

**Space efficient:** BFs can considerably compress elements at the cost of a small false positive rate. The set of elements is compressed into a $b$ bit vector, regardless of the size of the elements in bits. The resulting space cost only relates to the number of encoded elements $m$ and not the size of the elements in bits (Bloom, 1970). For example, a BF with $P_{FP}$ is equal to 0.0001 and an accuracy of 99.99 %, needs only around 19.2 bits per element to compress a set of parity check values each of size 96 bits. Having a set of 500 parity check values, the BF can save 80% of the bits needed. As a result, the BF is space efficient for compressing aggregated parity check values between the reader and the tags, which is the main reason why it has been included in the design of the novel redundant tag information collection protocol in this research.

**Constant time of insertion and querying:** The time complexity of inserting as well as querying an element in the BF is $O(k)$, which is less than that of trees, that is, $O(\log m)$. In addition, it is less than the time needed by lists or tables, which is equal to $O(m)$ (Guo et al., 2010).

**One inevitable error:** Despite a BF suffering from inevitable false positives, it is free of false negatives. When a BF states that an element is not a member in a set, there is no doubt about the decision. In contrast, when a BF states that an element is a member of a set, there is a probability that the decision is incorrect (Broder and Mitzenmacher, 2004).
8.3 Bloom Filter Limitations

Even though BFs provide a compact structure for representing a set and supporting membership queries, they do experience some limitations and they are considered in the design of the redundant information collection protocols in this research. These limitations are in the following (Luo et al., 2018):

**False positives**: The false positive rate can be decreased by extending the length of the BF. However, the misleading membership may cause some influences on the applications, which should be considered.

**Adaptability**: The three parameters of the BF are defined before deployment. Any bit is not allowed to be altered once it has been set to the binary 1. In addition, a BF is only representing a static set. That is, the encoded elements cannot be changed or deleted. Once implemented, the length of the BF and the employed hash functions cannot be altered.

**Functionality**: BFs support only membership queries, that is, additional operations, such as deletion and multiplicity queries, are not supported. When an element is deleted from the BF, the indexed bits cannot be set to 0, because they may affect another element and thus, label it as not a member of the BF, which leads to a false negative. This limitation will not affect the current state of the designed protocol.

**Implementation**: Whilst BFs are easy to implement, some requirements such as filter access, space optimisation, and cost calculation may cause some difficulties for the applications. In addition, implementing high performance hash functions requires complex operations, which may not suit lightweight hardware. The aforementioned requirements get even worse when there is a large amount of elements to be encoded. However, the implementation of hash functions can be simplified to decrease the complexity of a tag’s circuit. For example, a pre-
stored logical ring of random numbers may be used to produce hash values, as described in subsection 9.5.4 of Chapter 9.

Despite these limitations, in particular, the false positives, the compact structure for representing a set of parity check values with a BF and thus, the reduced time to send these values between the reader and tags outweighs them, as long as the probability of the false positives is sufficiently small and controlled in the designed protocol in the next chapter.

### 8.4 Bloom Filter Variants

Several modifications and developments have been made over the standard BF from different aspects to overcome its limitations. In fact, the BF variants introduced below are the results of improving various systems. In this section, some important and commonly used variants are presented.

#### 8.4.1 Counting Bloom Filters

The standard BF only allows for the insertion of elements into a BF. To enable the deletion of elements, a counting Bloom filter (CBF) was proposed by Fan et al. (2000), which replaces a vector of bits with one of counters. The CBF has $d$ counters along with the $d$ bits. When an element is inserted into or removed from the CBF, the related counters are increased or decreased respectively. Figure 8.2 demonstrates an example, where the counter values relate to the number of elements linked to the counter. The length of the counter should be sufficiently large enough to prevent counter overflow. In the analysis accomplished by the authors, it is concluded that a counter of 3 or 4 bits should suffice for most applications. CBFs have the same properties as BF$s$, at the expense of 3 or 4 times memory space cost. Ficara et al. (2008) proposed an enhanced structure called a Multi-Layer Compressed Counting Bloom Filter (ML-CCBF). The structure uses a hierarchy of hash-based filters in addition to CBF$s$ in order to provide space to counters and thus to avoid overflows. Huffman
coding is also used to compress counter values even further. However, the costs of addition and deletion are increased accordingly. Rothenberg et al. (2010) improved upon the deletion process in terms of memory space, with an enhanced method that splits the filter into $r$ regions. The collision information is represented by a compact representation composed of a bitmap of size $r$ that sets to 0 when the region is free of collisions and thus, the deletions are permitted, otherwise, it is set to 1.

![Figure 8.2: An example of a CBF](image)

### 8.4.2 $d$-Left Counting Bloom Filters

Bonomi et al. (2006) proposed a data structure that utilises $d$-left hashing and fingerprints. It works in a way similar to that of a CBF, but with a less memory space. The $d$-left hashing method splits a hash table into $d$ sub-tables with equal sizes. Each sub-table has $r/d$ segments, where $r$ is the total number of segments. Each segment contains a number of cells of fixed size and each cell maintains a counter and a fingerprint of the element. If an element is added to the table, $d$ segments are selected by calculating $d$ independent hash values of the element. Parallel search of the $d$ sub-tables is used by element lookups to get the related fingerprint and counter. If an element is deleted, the counter is decreased by one. The counters of this type of filter are much smaller than those in the CBF owing to the utilisation of the $d$-left hashing and fingerprints that generate a smaller amount of collisions.
8.4.3 Space Code Bloom Filters

Kumar et al. (2006) proposed space code Bloom filters (SCBFs) to encode a multiset and to manipulate the number of existence of an element in the multiset. Typically, a group of standard BFs is used and a maximum likelihood estimation method to measure the existence of an element in the multiset. To add an element in the SCBF, the element is hashed using one set of randomly chosen hash functions and then, the related bits in the filter are set to 1. To perform a membership query for an element, the number of sets that the element matches is counted and this number is utilised to approximate multiplicity of the element in the multiset.

8.4.4 Scalable Bloom Filters

One limitation of a BF is to set the filter length $d$ in advance based on the number of elements $m$ and the desired value of $P_{FP}$. When the number of elements is unavailable, the filter length may be over-estimated and consequently, will consume memory spaces. Hence, Almeida et al. (2007) proposed a scalable BF to adapt the filter length to the number of elements added dynamically. The authors constructed a sequence of heterogeneous BFs in an incremental way. When a BF reaches its limit with respect to the false positive probability, a new one is inserted. Each BF uses different lengths and hash functions, which results in large overheads when performing membership queries. Later, in 2010, Guo et al. enhanced dynamic set representations with homogenous BFs structured in a dynamic matrix, the rows of which are related to the BFs. The matrix initially contains one active BF and the elements of the set are added to it. When the BF reaches the limit of its false positive probability, it is stored and set to be inactive and a new one is built and set to be active. The membership query process is similar to that of a standard BF, whereas deletion of an element is implemented in a way similar to that of CBFs. Despite the proposed structure enabling the representation of a dynamic set, the false positive probability may become too large and thus, it is necessary to
occasionally rebuild the whole structure. In addition, the upper bound on the number of elements $m$ must be set in advance (Guo et al., 2010).

### 8.4.5 Weighted Bloom Filters

The weighted BF, as proposed by Bruck et al. (2006), uses the number of queries to assign weights for the elements in BFs. The elements that have high frequencies of queries are allocated a high amount of hash functions in order to minimise the occurrence of false positives. The false positive probability differs for each input and the average false positive probability is calculated based on the frequencies of all the elements in the filter. This type of BF accomplishes a high level of optimisation for real time applications.

### 8.5 Applications of Bloom Filters

Originally, a BF was employed in dictionaries to search words and to support spell check operations in text editors, such as UNIX spell-checkers (Mullin and Margoliash, 1990). Instead of storing the whole dictionary, a compact representation of a BF is stored. If a false positive occurs, the text editor will spell the word incorrectly and in this case, it can be ignored. In distributed databases, rather than exchanging lists of information between hosts holding part of the database, BFs of these lists are sent to decrease communication overheads between hosts (Mullin, 1990). In Peer-to-Peer (P2P) networks, Guo and Li (2013) and Skachek and Rabbat (2014) employed BFs to find the set difference in order to solve the problems of approximate set reconciliation with minimum amount of information transmitted between two peers. Approximate set reconciliation is a common and fundamental task in P2P systems, where each member of a node pair wants to send its set of unique items to another member. For example, suppose there are two sets of items $Set_x$ and $Set_y$, relating to two peers, $x$ and $y$, respectively. An application requires to recognise those items that exist in set $x$, but not in set $y$, that is, items in $Set_x – Set_y$. A typical approach is to have $y$ send a BF of its items
to $x$. Then, $x$ will check all its items against the BF and transmit items that are not members of it. Due to the false positive probabilities, not all items in $\text{Set}_x \setminus \text{Set}_y$ will be transmitted, but most will.

Luo et al. (2017) extended the set reconciliation to multisets reconciliation in which each set of items allows an item to appear multiple times, that is, the multiplicity of each item can be greater than one. Hence, each cell of the constructed BF includes two fields that store the IDs and the number of items linked to it in order to identify different items and differences in the multiplicities of these between the two multisets. Chen et al. (2012) used BFs in a similar fashion as employed in approximate set reconciliation problems, except that their goal was to find the set intersection and union instead of the set difference in multi-keyword search to decrease the transmission overheads. To find the set intersection, that is, $\text{Set}_x \cap \text{Set}_y$, $y$ sends a BF of its items to $x$. Then, $x$ will check all its items against the BF and transmits items that are members of it. False positives may be eliminated by recalculating the intersection operation of $y$ and the result set. For the union operation, that is, $\text{Set}_x \cup \text{Set}_y$, the process is similar to the intersection operation except that the results set is obtained by choosing the items that are not in $y$ by testing those in $x$ using the BF. Finding the set difference between two pairs using the BF has inspired the designing of the novel redundant tag information collection protocol in this research.

In RFID systems, the literature contains a wealth of material based on the BF and its variants to solve various issues, including filtering duplicate data, as presented below, searching for a group of tags (Chen et al., 2016), unknown tag scanning (Qian et al., 2016), missing tag detection (Yu et al., 2017), grouping (Liu et al., 2016), and estimating the number of tags (Li et al., 2015). Generally, BFs have been employed to compact the transmitted tag IDs between the reader and tags in order to reduce transmission overheads and the parameters of the BFs are set based on the predefined false positive rate requirement.
8.5.1 Filtering Duplicate Data from RFID Data Streams

RFID systems naturally produce a large amount of duplicate readings. Duplicate readings are produced because an RFID reader reads the same tag multiple times or the same tag is read by multiple readers due to their overlapped regions. Any reading is treated as a duplicate when it is recurrent and does not provide new information to the system. Duplicate readings need to be removed, because they waste memory and processing resources as well as causing communication overheads in RFID systems (Bashir et al., 2013). Figure 8.3 shows the basic filtering concept of duplicate readings.

![Diagram of duplicate readings filtering](image)

**Figure 8.3: Basic duplicate readings filtering (Yaacob and Mahdin, 2016)**

Various BF-based approaches have been proposed in the literature to detect and filter duplicate readings from an RFID data stream. Typically, these approaches consider filtering tag readings that take place in readers (local filtering) or middleware (global filtering) or both, after the data are received from the tags and before being forwarded to the backend database servers for subsequent processing, as shown in Figure 8.4. The region of an RFID reader is generally divided into major and minor regions, where the latter represents the overlapping region. Wang et al. (2008) proposed eager and lazy methods that employ the
standard a BF to filter duplicate data. When new data arrive at a local reader, it is added to the BF, then, the BF is transmitted to the middleware for the update. The middleware manages readings from readers under its network before forwarding them to the backend database server. In the eager method, a copy of the BF is transmitted to every reader in order to eliminate reading duplicate data again. However, it is costly to inform all the readers about a new reading every time it arrives. In the lazy method, only a reader that transmits new data has a copy of the BF from the middleware. This method needs to update the middleware each time a new reading is coming in, which may result in wasting lots of bandwidth.

![Diagram of duplicate filtering method of an RFID data stream](image)

**Figure 8.4: Duplicate filtering method of an RFID data stream (Kamaludin et al., 2016)**

To support deletion of old readings, Mahdin and Abawajy (2011) introduce the decaying BF. Even though the CBF supports deletion of elements, it does have any way of knowing which counters represent the elements in its filter. The decaying BF can delete old readings by including sliding windows in its filter. The counters in the filter are initialised to the sliding window size and are decreased by one when a new reading is inserted. If the reading already exists, all the relevant counters are set to the sliding window size again. When there is no reading coming in from a tag, which means that the tagged object has left the reader region,
the reading will be removed from the filter as a result of the decreasing process. A false positive happens, if all the counters are not set to 0 for a non-member element. The probability of one is increased when the BF is full, when all or nearly all the counters have been utilised to represent the elements. For this reason, the length of the BF should be long enough to accommodate the number of incremented elements and the old elements should be deleted to decrease the false positive probability. In addition, in the same context, Lee and Chung (2011) proposed time BF, where the filtering process takes place at the middleware’s side. The time BF uses the time information to test whether the reading is duplicated or not. The filter uses an integer array in place of a bit one, where each entry is set to the identified time of a tag. If the entries are already set to the time of the previous reading, they will be overwritten with the time of the new reading. To test whether an element is a duplicate or not, all the related $k$ entries are checked. If there is at least one entry where the result of the subtraction of its time from that of the tested element is larger than a threshold, this means that the tested element is not a duplicate. Due to the transmission of duplicates to the middleware, a lot of bandwidth is wasted and consequently, a bottleneck exists on the middleware’s side. Mezzi and Akaichi (2013) discussed the same idea, but with some differences. They proposed a method based on the difference of the time of the current and the previous flow. Jiang et al. (2012) stressed the importance of eliminating duplicates as early as possible in order to decrease network bandwidth. For this reason, they proposed two layers of filtering methods. The first is at the reader’s side, where a BF is used to filter local duplicates. The other filtering method is employed at the middleware’s side, where a distributed time BF is utilised to filter global duplicates. The two layers filtering process generates a small false positive rate and causes latency at the reader’s side as well, because it requires filtering the duplicate data before forwarding them to the middleware. Following the same concept, Kamaludin et al. (2016) modified the BF and used only a single hash function to reduce the complexity and thus speed up the filtering process. The authors employed two
filters in addition to the landmark window to allow for the deletion of old elements. The proposed method tests the time to delete the reading and if the condition is met, all the counters are reset to 0. Every incoming reading is hashed using a single hash function, with the result of the hashing being treated as the counter index in the two arrays. The algorithm first tests whether the reading is duplicated or not and if so, the reading will be rejected. Then, if the first array index at the counter is empty, then that index will record the tag ID, or else, if the second array index at that counter is empty, then that index will record the tag ID. If not, the reading is a false positive. Yongsheng and Zhijun (2013) combined CBFs with dynamic chain lists to detect and filter duplicate readings produced from reading the same tag multiple times. The dynamic chain is implemented as a linked list and the CBF is constructed from an array of times to enable the deletion of old data. The counter is increased and decreased when an element is inserted into and removed from the filter, respectively. The cell is set when the counter is changed from 0 to 1. This method has some limitations such as the reader producing a large volume of data. In addition, when a duplicate is found, the reader has to search for a match in the double linked list, which causes latency. Dutta et al. (2013) constructed a BF that is able to store elements from multiple sets. Each element of the stream is hashed to a $p$-bit fingerprint using Rabin’s method (Rabin, 1981) and the results of the hashing are considered as the input to the filter. Therefore, the addition of an element involves the addition of its hash value and conversely removing one involves the removing of its hash value. The authors also proposed a dynamic filter to handle the variations in the stream behaviour, which is constructed from an array of pointers. When an element is to be added to the filter, a bucket is dynamically allocated and is linked to the pointer in the related index of the vector. On the other hand, removal of an element involves the deletion of the bucket in which the hash of the element was recorded and then, the free cells are added to the main pool of free spaces. Membership testing requires checking the buckets linked to the pointer of the array index in which the element is hashed.
A study by Guoqiong et al. (2016) supported filtering of RFID data streams in mobile environments. The proposed filtering method considers both the time and location information of readings. The single vector of a BF is extended to a vector of two dimensions with one dimension recording reader IDs, whilst the other records the time of the detection event. The filter is initialised with all bits being set to 0. When a new reading x is received, it is encoded using the k hash functions and the indexes are set to the related reader ID and time. If one of the times of the resulting indexes is equal to 0 this means that the reading is for non-redundant data. If all the time values of the k indexes are not equal to 0, but there is one entry having the same reader ID, this indicates that the location of the tag has been changed and also, a non-redundant data. The authors also adopted a random decay method for the removal of old data by randomly selecting a number of cells in the filter to decrease their time by 1. This results from the fact that, if the time of a cell has not been changed for a long period, it reaches 0 after a number of decreasing processes, which is identical to the process of removing the item of the cell. In 2017, Guoqiong et al. studied filtering redundant data for uncertain RFID data streams gathered from areas far away from the reader. The proposed BF consists of four tuples \( p, et, f, r \) to store the existential probability of the tag, the latest time, the sum of the decay weights, and the location. Each hash function relates to a different counter. The minimum existential probability is obtained by comparing the existential probabilities of all the k counters and then, comparing these with the minimum probability threshold. If it is larger than the threshold, the location entry is different from that of the reading, and the time difference is less than a threshold, then this indicates that the reading is a non-duplicate.

**8.6 Conclusions**

A BF is a simple data structure that can store the items of a set in a space efficient way and support membership queries. Due to its space efficiency, it has been widely employed in various systems. Regarding RFID systems, the literature contains a wealth of proposed
methods based on the BF to solve various issues including filtering duplicate data from RFID data streams. However, all the methods concentrate on filtering duplicate data at the reader or the middleware’s side after they are received from the tags and before being forwarded to the backend database servers for processing, which results in communication overheads between the reader and the tags. The main limitation of using the BF is that it enables false positives. However, the space optimisation and thus, the reduced time to send the BF between the reader and tags outweighs this limitation, as long as the probability of the false positives is sufficiently small and controlled, as in the designed protocol. This chapter has provided an overview of the BF with a focus on its construction, features, limitations, variations, and the ways in which it has been utilised in some systems, especially RFID ones. This review has provided preliminary information relevant to the designing of the novel redundant tag information collection protocol presented in the next chapter.
9 Efficient Redundant Information Collection Protocol

The extant autonomous group integrity methods in RFID systems write redundant data into the memory of RFID tags. These data reference other tags or represent their membership in the group. In LDPC-based methods, parity check values and short IDs are written into the memory of RFID tags, as shown in Figure 5.1 of Chapter 5. For the group integrity checking and decoding processes, the reader needs to collect the parity check values and their short IDs from the available tags. Designing an efficient collection protocol is needed for reading such redundant information from the tags in RFID systems. The key performance measure for the protocol is time efficiency, because RFID systems have subsequent processes to perform, including integrity checking and decoding processes. When the time is long, the task of collecting tag information may delay these processes of the system, which is undesirable, especially in large-RFID environments. The study in this chapter is aimed at efficiently collecting such redundant information from a group of tags, thus enhancing the complete system of group integrity applications. Whilst there are many methods in the literature that filter out duplicate information in RFID systems, as presented in subsection 8.5.1 of Chapter 8, all these concentrate on filtering duplicate information at the reader or the middleware’s side after the information is received from the tags and before being transmitted to the backend database servers for processing. This causes communication overheads between the reader and the tags. Existing possible solutions are also not efficient in solving the problem since they do not distinguish redundancy among tags, which result in long collection times. To achieve time efficiency, a novel protocol called the Redundant Information Collection (RIC) protocol is designed. It is based on a BF that efficiently filters redundant tag information at the tag’s side. Extensive simulations have been carried out to analyse and assess the performance achieved with the designed protocol.
Chapter 9 - Efficient Redundant Information Collection Protocol

9.1 Tag Information Collection Protocols

In sensor-augmented RFID systems, tags are combined with miniaturised sensors to provide information about the surrounding environment for example humidity and temperature. Collecting such information from tags with low communication costs and thus, short time spans, has inspired many studies in recent years. Chen et al. (2011) proposed the Single-hash Information Collection (SIC) protocol and the improved version, the Multi-hash Information Collection (MIC) protocol, to periodically collect information from sensor-augmented RFID systems with a minimum execution time. By utilising several hash functions, the MIC protocol can resolve most hash collisions in a time slotted frame and thus, significantly enhance the protocol performance compared to the SIC protocol (more details are given below in section 9.4.3). Qiao et al. (2011; 2016) studied the same problem from the aspect of energy efficiency for active tags. The Tag Ordering Polling (TOP) protocol and the enhanced version (ETOP) were proposed to read information from a group of tags in large RFID systems with minimum energy consumption. The information here contains that of the sensor information as well as the link between the information and the tag so that the reader can correctly linked sensor information to the related object. In an earlier version presented in 2011, the reader sends a vector instead of tag IDs. Each tag ID in the group is mapped to a bit in the vector using a hash function and the corresponding bit in the vector is set to 1. The vector is divided into segments of size 96 bits to send each segment in one time slot of the current frame. Each tag only requires checking a specific bit in the vector at a location specified by the hash of its ID. Only a tag that finds its value in the vector is 1 will stay active and report its information to the reader in the subsequent time slotted frame. The vector also identifies the order in which the polled tags will report their information in order to eliminate collisions. If a tag finds that there are \( i \) 1s in the vector preceding its bit, it determines that its order is \((i + 1)\) when sending its information to the reader.
In the enhanced version presented in 2016, the vector is implemented as a partitioned BF to reduce the occurrence of false positives. Each segment is evenly split into \( k \) partitions. For each tag in the group, the reader hashes its ID to get some hash bits. It uses these bits to link each tag to its related segment, and then deploys \( k \) hash bits to link further each tag to one related bit in each partition of the segment. The partitioned BF allocates \( k \) bits for each hashed tag which is similar to a standard BF; however, it spreads the \( k \) related bits in \( k \) partitions. The reader broadcasts each segment with the total number of tags \( x \) encoded in all the preceding segments. Upon receiving its segment, a tag performs the \( k \) hash bits to check its membership. Then, it calculates its location in the reporting order by adding the value of \( x \) to the number of 1s preceding its bit in the partition that has the largest number of 1s.

A follow-up work by Yue et al. (2012) complements previous work with a new protocol primarily designed to read sensor-produced information from RFID tags with multiple readers. The extended study in Yue et al. (2014) targets an unknown subset of tag IDs with one reader. They proposed the Bloom-filter-based Information Collection (BIC) protocol, in which tags build their BF and transfer them simultaneously to the reader in order to identify the target tags efficiently. Different from the existing protocols, in that of Yue et al., the reader has no knowledge about the target tag IDs from which it collects information. Therefore, the reader, first, is required to identify the tags in its region in order to gather information from them. Figure 9.1 illustrates a simple example of the protocol in identifying the interrogated tags. The reader sends a request message, including the length of the BF \( b \) and the number of hash functions \( k \), as in part (a). Upon receiving the reader’s message, each tag constructs its BF using the reader’s parameters. Then, it encodes its ID into the filter and sends this to the reader. All the interrogated tags send their BF simultaneously as in part (b).

In the physical layer, an idle carrier represents a logic value 0 and a busy one, a logic value 1. For each received signal at the reader’s side, if the channel is idle, the received signal is set to 0. If the channel is busy, which indicates that at least one tag is sending, the received signal is
set to 1. After the transmissions of all the BF from the tags, the reader can create a new BF that represents all tags, as in part (c). The reader uses the new BF to filter the interrogated tag set from a set of all tag IDs stored in the back-end database server. Then, can start collecting information from the interrogated tag set by sending an ordering vector to schedule the tags’ transmissions, as in the TOP protocol. The authors set the parameters of the BF, including its length $b$ and its hash functions $k$, according to the size of the set to be encoded and the desired false positive rate (i.e. $1 \times 10^{-4}$), as discussed in section 8.1 of Chapter 8.

Figure 9.1: An example of the tag identification process with a BF in Yue et al. (2014)

To increase the frame utilisation, Xie et al. (2017) proposed a protocol called the minimal Perfect hashing-based Information Collection (PIC) protocol that establishes a one-to-one mapping between target tags and slots. To eliminate target tag collisions, the protocol starts with the assignment process, which allocates target tags to successive indices. The assignment process contains a number of rounds and in each round, the reader sends an order vector to allocate a portion of target tags to their indices. Upon receiving the order vector, each tag will check its related bit. If it is 1, the tag will be allocated to a slot at index $(n + y)$, where $n$ represents the number of target tags that have been allocated in the preceding rounds, and $y$ is the number of 1s preceding this bit. In order to filter out the non-target tags and thus, preventing them from interfering with the target tags, an improved BF is employed. It
Chapter 9 - Efficient Redundant Information Collection Protocol

consists of $b$ items and each of them is a $k$ bit fingerprint of the related target tag. Each fingerprint is calculated based on the target tag ID. The reader transmits each fingerprint in a slot of the next time frame. Each tag aiming to respond in the $i$-th slot and finding that the $i$-th fingerprint is not identical to its fingerprint will keep silent. The transmission order of the fingerprints is based on the slot indices allocated to target tags in the assignment process. Since the reader has the knowledge of the tag IDs and the hashing parameters used in the filtering phase, the error of the false positives can be solved by broadcasting the non-target tag IDs that pass the membership test in order to deactivate them.

Another direction of the research was proposed recently in Liu et al. (2017), who studied category information collection problems in multi-category RFID systems. In such systems, the tag ID consists of a category ID, which specifies the category that a tag is a member in; and a member ID, which indicates a certain tag in the category. All tags belonging to the same category store the same category information and hence, there is no need to collect category information from all tags in the same category; one tag’s response is enough to satisfy the requirement. The proposed protocol consists of two phases. The first phase aims to separate a category from others, while the second attempts to poll one tag in a category according to the geometry distribution of tag IDs by using a 4-bit index. The reader first broadcasts a request message, including the frame size $f$ and a random seed $r$. Upon receiving the message, each tag randomly selects a slot in the frame at the index $h(r, CID)$, where $h(.)$ is a hash function shared by all tags with range $[0, ..., f-1]$ and $CID$ is a category ID. As each tag takes its $CID$ as the hash seed, the tags from the same category choose the same slot. The reader can know which slots in the frame are selected by tags belonging to the same category from the receiving category IDs and in this case, these are called useful slots, otherwise, they are called useless ones. The reader then broadcasts an ordering vector, as in the TOP protocol of Qiao et al. (2011; 2016). Each bit in the vector relates to a slot in the frame and is set to 0 to represent a useless slot or 1 to represent a useful one. If the vector is too long to fit into one
time slot, it is split into segments of size 96 bits. In order to single out a tag in an efficient way, each selects an index $R(h(TID, r) \mod 2^X)$, where $X$ is a constant and $R(\cdot)$ is the location of the right-most bit of 1 in the input. Since the reader has the knowledge of tag IDs, it knows the hashing result of each tag. The reader then sends a frame of slots, where each slot contains a $\log_2 X$-bit index to pull a single tag from a category, while other tags in the category will keep silent. In all the work presented above, it has been assumed that the reader has the knowledge of the tag IDs presented in the system by being connected to a back-end database server that stores the IDs of all the tags.

9.2 System Model and Problem Statement

The underlying RFID system here is the same as that in section 5.1 of Chapter 5. Consider a group of RFID tag IDs denoted as $ID = \{ID_1, ID_2, ID_3, \ldots, ID_n\}$ corresponding to a group of passive tags represented as $TAG = \{tag_1, tag_2, tag_3, \ldots, tag_n\}$, where $n$ is the number of tags in the group. These tag IDs are encoded using a parity check matrix of size $m \times n$, which is constructed using the PEG method to generate a set of parity check values of size $m$ denoted by $C = \{c_1, c_2, c_3, \ldots, c_m\}$. The bit length of each $c_i$ is equal to that of the tag ID, that is, 96 bits. To enable autonomous integrity checking without using external systems, each tag stores its parity check values ($c_i$), as well as a set of short IDs ($s_i$) referring to other tags belonging to the same row of the parity check matrix. For the group integrity checking and decoding processes, the reader needs to collect the parity check values and their short IDs from available tags. Designing a fast collection protocol to read such redundant information from the tags is needed in order not to delay these processes. The goal of the designed protocol is to collect all $c_i$ efficiently and their $s_i$ in the interrogation zone of the reader within a desired false positive rate. More precisely, the design should consider the redundancy among tags and collect each $c_i$ and its $s_i$ only once. Hence, the complete system, which includes the three main phases, as shown in Figure 1.3 of Chapter 1 is enhanced. In this study, for the purpose
of collecting redundant tag information, a group of tags belonging to the same row of the
parity check matrix maintains the same order of short IDs and consequently, the short ID of
the tag needs to be recorded in its memory.

9.3 Performance Lower Bound

Let $t_{\text{bit}}$ denote the time needed to send one bit. In group integrity application, the information
related to one parity check value and its short IDs is denoted by $c_i + s_i \times l$ and can be expressed
as $x$, where $l$ represents the row weights of the parity check matrix used for the encoding. All
the tags contain the same amounts of parity check information, which are denoted by
$j \times (c_i + s_i \times l)$ and can be expressed as $j \times x$, where $j$ is the column weights of the parity check
matrix. Therefore, each tag needs a time, the length of which is equal to $x \times t_{\text{bit}}$ in order to send
one piece of parity check information. For collecting all tag information, the lower bound on
the collection time for a group of $n$ tags is $n \times j \times x \times t_{\text{bit}}$. For collecting redundant information,
the lower bound is much less than $n \times j \times x \times t_{\text{bit}}$. The reader needs to collect each $c_i$ and its $s_i$
only once. Thus, the lower bound on the collection time is $m \times x \times t_{\text{bit}}$. It is unlikely that this
lower bound is achievable, because the reader requires extra time to transmit its request and
coordinate the protocol process. Despite this, it provides a benchmark for assessing the
performance of the designed protocol.

9.4 Possible Solutions

In this section, three existing solutions to general and different problems that could be applied
to collect redundant tag information are examined, and these are simply called possible
solutions. None of them is efficient in solving the redundant information collection problem
since they do not distinguish redundancy among tags, which results in long collection times.
This gives clear knowledge about the solution of the redundant tag information collection
problem, which motivates the designing of the novel protocol presented in section 9.5.
9.4.1 Baseline Collection Protocol

A baseline collection (BC) protocol for collecting tag information is to let the RFID reader broadcast the ID of one tag. Then, it receives the information of the related tag and the reader repeats this process until it gets all the information needed. As regard to collecting redundant information, the tag does not have knowledge of the information that the reader has, so it must send it. Thus, the total time to collect one piece of parity check information from a tag is $(ID_i \times t_{bit} + x \times t_{bit})$, and the optimal total time for collecting all parity check information from a group is $m \times (ID_i \times t_{bit} + x \times t_{bit})$. The advantage of this protocol is that the request and reply is a one-to-one mapping. In other words, collisions cannot occur in the communication between the reader and the tags. On the other hand, the reader has to transmit the tag ID in order to collect related tag information, which increases communication costs because the tag ID is 96 bits. Consequently, the time is significantly above the lower bound value of $m \times x \times t_{bit}$.

9.4.2 ID Collection Protocol

As presented in subsection 3.5.4 of Chapter 3, the RFID tag anti-collision protocols using TDMA techniques are mainly classified into Aloha-based and tree-based protocols. Using these protocols, the reader can collect tag IDs in its interrogation zone. If these ID collection protocols are borrowed to collect redundant tag information, the collection time is high, because each tag has to send all its information. In addition, the protocol has to consider the entire tag group $n$ each time when collecting tag information. In the enhanced version of ALOHA-based protocols, as presented in Lee et al. (2005), the optimal time for collecting $n$ tag information is $e \times n \times x \times t_{bit}$, where $e$ is the natural constant and is approximately equal to 2.72. This means that each tag resends its information nearly 2.72 times, on average, which is the theoretically ideal number. Experiments and simulation results demonstrated that the best performance of tree-based protocols is equivalent to that of the ALOHA-based ones (Chen et al., 2011).
9.4.3 Multi-Hash Information Collection Protocol

Chen et al. (2011) presented the SIC and MIC protocols for collecting tag information in sensor-based RFID systems. Only the MIC protocol is considered here, because it is an improved one, which is performed stage by stage. In each stage, the reader links tags to time slots using $k$ hash functions. Then, it uses a hash selection vector to notify the tags about the links. Each item in the vector is related to a time slot at the same index location. The reader sets the item to 0, if no tag is linked to a slot. If a tag is linked to a slot with the $z$-th hash function, where $1 \leq z \leq k$, then the reader sets the related item to $z$, which is of size $\lceil \log_2(k+1) \rceil$. After receiving the vector, each tag only requires checking the items in the vector that it is mapped to by using the $k$ hash functions. Let $T_z$, $1 \leq z \leq k$, be the item that the tag is linked to by the $z$-th hash function. Each tag checks the items in the order from $T_1$ to $T_k$. If an item $T_z$ has the value of $z$, the tag concludes that it must be allocated to a slot using the $z$-th hash function. The above process iterates stage after stage until all tags have sent their information. The first stage considers $n$ tags for slot allocation and the size of its frame size is equal to $n$. Each successive stage considers a fewer number of tags and thus, its frame size is smaller.

Figure 9.2 gives an illustrative example of the MIC protocol where $k = 2$. The arrows denote the links from the tags to the slots based on the results of the hash functions. The allocations of the tags to the slots are represented by thick arrows. In the first round of the protocol when $H[1]$ is applied, tags with ID$_1$ and ID$_5$ are allocated to slots. When $H[2]$ is applied in the second round, tag 3 and tag 4 are allocated. In part (c), a hash selection vector is built by the reader based on the slot allocation. Upon receiving the vector, each tag checks the items in the vector that it is linked to. For example, the third tag finds that the value of the item linked by $H[2]$ is equal to 2 and thus, it concludes that it must be allocated to the related slot.
The expected total collection time of the MIC protocol is:

\[
\frac{n}{32P_k} \times ID_i \times t_{bit} + \frac{n}{P_k} \times j \times x \times t_{bit} \quad (9.1)
\]

where, \( P_k \) is the probability that a tag is linked to one time slot in every stage when the \( k \) hash function is used. It has been proven that when \( k = 7 \), the collection time reaches the optimal value of \( 0.036 \times n \times ID_i \times t_{bit} + 1.16 \times n \times j \times x \times t_{bit} \), where \( P_7 = 86.1\% \). Similar to the BC protocols, when the MIC is used for collecting redundant information, each tag does not have knowledge about the information that other tags have sent, so it must send its information. Thus, the optimal total time for collection \( m \) parity check information from a group is:

\[
0.036 \times m \times ID_i \times t_{bit} + 1.16 \times m \times x \times t_{bit} \quad (9.2)
\]

### 9.5 Redundant Information Collection Protocol

This section presents the novel RIC protocol that utilises a BF to filter redundant tag information from a group of tags efficiently. The designed protocol aims to avoid the redundant transmissions of \( c_i \), as well as the corresponding \( s_i \). The process of the RIC protocol consists of several rounds that are repeated until all \( c_i \) have been read. In each round,
the reader sends a request, and then the tags reply with their information in the successive slotted time frame. Each tag must check whether its information has been sent previously by other tags before it sends any information. The reader maintains a list of all collected $c_i$ and its $s_i$. Reporting the list to the tags in order to prevent transmitting redundant information in the subsequent rounds is too costly, especially when the set of tags in the group is large. To accomplish a high level of filtering efficiency, the collected parity check values are compressed into a BF, which is then sent instead of the actual parity check values. In other words, the BF is used as a tool to carry the aggregated $c_i$ and filter the tag’s information at the latter’s side.

### 9.5.1 Protocol Description

The RIC protocol consists of several rounds. The communication between the tags and the reader uses a framed slotted Aloha, which follows the RTF mode based on the EPCglobal Class-1 Generation-2 standard. In each round of the communication, the reader sends a request message containing two parameters, $r$ and $f$, where the former is a random number and the latter is the frame size, that is, the number of slots in the frame. Upon receiving a request, each interrogated tag is randomly mapped to a slot in the frame at the index $h(r, c_i) \mod f$, where $h(.)$ is a hash function shared by all tags with a range of $[0, ..., f-1]$. As each tag takes its $c_i$ as the hash seed, those having the same parity check values $c_i$ choose the same slot. Then, they send their identical parity check information $x$, and thus, the signals can be decoded by the reader. In this case, the slot can be called a successful slot. On the other hand, tags with different parity check values $c_i$ may respond at the same slot, and the slot here is treated as a collision slot. Hence, the reader cannot decode the signals encoded from a set of unidentical parity check information. The reader stores the collected parity check values $c_i$ in a list denoted as $D = \{d_1, d_2, d_3, \ldots, d_a\}$. Then, it constructs a BF vector by mapping the collected parity check values to $b$ bits array using $k$ hash functions, as described in
Algorithm 9.1. Given a desired false positive rate $P_{FP}$ and the number of elements in $D$, the optimal values of the hash functions and the optimal length of the BF constructed in the RIC protocol can be computed using Equations (8.8) and (8.9), respectively.

The reader again sends another request to collect a new set of parity check values $c_i$. It also sends the constructed BF and corresponding parameters $b$ and $k$. Before sending any parity check information, each tag must check whether its $c_i$ has been sent previously by other tags, and here is where the BF plays an important role by meeting this requirement. Each tag performs membership tests for all $c_i$ it contains by using Algorithm 9.2. For each one, it performs $k$ hash functions, which are identical to those used to construct the BF at the reader’s side. It then tests whether the bit $h_z(c_i)$ is set to 1 for $1 \leq z \leq k$. If so, the tag will treat the tested $c_i$ as redundant information and will not send it to the reader, otherwise, the tested $c_i$ and all its $s_i$ are sent to the reader. After receiving the tag information, the reader updates its list and reconstructs the BF. The process of filtering and collecting information will be repeated until all the tags have reported all their information. The processes involved in all rounds are identical, except that at the beginning of the first round the list of the reader is empty, and thus, $b$ is equal to zero, which means that the BF is not sent. Algorithm 9.3 outlines the designed protocol.

Algorithm 9.1: Constructing the BF

**Input:** $D$, $P_{FP}$

**Output:** $b$, $k$, BF

1: Calculate $b$ and $k$

2: $BF = \text{allocate } b \text{ bits initialised to 0}$

3: for each $d_i$ in $D$:

4: for $z = 1$ to $k$ do
Algorithm 9.2: Membership testing

**Input:** \( elm, BF, k \)

**Output:** Yes/No

1: for \( j = 1 \) to \( k \) do
2:   if \( BF[h_j(elm)] \neq 1 \) then
3:     return No
4:   end if
5: end for
6: return Yes

Algorithm 9.3: The RIC protocol

**Input:** \( P_{FP} \)

**Output:** \( D \) containing all \( c_i \) and its \( s_i \) in the interrogated region

1: \( D = \phi \)
2: Repeat
3:  Constructing BF \((D, P_{FP})\) return \((b, k, BF)\)
4:  The reader sends \( \{r, f, b, k, BF\} \)
5:  for each \( c_i \) in \( tag_i \) :
6:    membership testing \((c_i, BF, k)\) return Yes/ No
7:    if No then
8:      \( tag_i \) sends its \( c_i \) and the corresponding \( s_i \) at index \( h(r, c_i) \)
9 : \[ D = D \cup c_i \text{ sent by } tag_i \]

10 : go to step (13) \[ \triangleright \text{tag is allowed to send only once in a frame} \]

11 : end if

12 : end for each

13 : Until all tags send their information

### 9.5.2 Performance Analysis

In this section, the expected collection time of the RIC protocol and the optimal value of the frame size \( f \) are derived. The expected collection time \( T_i \) of the \( i \)-th round is composed of that needed by the reader to send the BF vector and that needed by the tags to send the information. The reader broadcasts a request message, including \( r, f \), the BF vector and its parameters. The transmissions from the reader, except for the BF vector, are very small and thus, can be ignored. Note that the BF vector may be too long, so it can be split into multiple segments of 96 bits, each being sequentially transmitted, as in Chen et al. (2011) and Qiao et al. (2016). Thus, the expected time to send \( b \) bits vector of the BF is calculated, as:

\[
\frac{b}{96} \times t_{\text{bit}}
\]

(9.3)

From Equation (8.9), \( b \) can be expressed, as:

\[
- \frac{|D| \times \ln P_{FP}}{\ln 2^2}
\]

(9.4)

The time for the tags to test the set of parity check values from the BF is also ignored. Recall that one parity check information is denoted by \( x \), then the expected time for the tags to send their \( x \) information can be calculated, as:

\[
E_{\text{succ}} \times x \times t_{\text{bit}}
\]

(9.5)

where, \( E_{\text{succ}} \) is the expected number of successful slots in the frame. Therefore, the expected collection time \( T_i \) of the \( i \)-th round is:
\[ T_i = \frac{b_i}{96} \times t_{bit} + E_{\text{succ},i} \times x \times t_{bit} \]  \hfill (9.6)

In the ID collection protocols of such as Vogt (2002a), Lee et al. (2005), and Cha and Kim (2006), when the reader uses a frame of size \( f \) and the number of replying tags is \( n \) and from Equations (3.3) and (3.4), the expected number of read tags during one read cycle can be expressed, as:

\[ \lambda = f \left( \frac{n}{f} \right) \left( 1 - \frac{1}{f} \right)^{n-1} \]  \hfill (9.7)

In the RIC protocol, each tag uses its \( c_i \) to choose one slot and so the tags having the same \( c_i \) must share the same slot, which is called a successful one. By treating this group of tags as a single tag, the successful slot can be seen as a singleton. Hence, the expected number of successful slots can be calculated, as:

\[ E_{\text{succ},i} = f_i \left( \frac{m_i}{1} \right) \left( 1 - \frac{1}{f_i} \right)^{m_i-1} \]

\[ \approx m_i \left( 1 - \frac{1}{f_i} \right)^{m_i-1} \]

\[ \approx m_i \cdot e^{-\frac{m_i-1}{f_i}} \approx m_i \cdot e^{-\frac{m_i}{f_i}} \quad \text{as} \quad m_i, f_i \to \infty \]  \hfill (9.8)

where, \( m_i \) is the number of \( x \) to be collected before the \( i \)-th round. By letting the first derivative of \( T_i \) be zero, the minimum value of \( T_i \) can be found, as:

\[ \frac{d(T_i)}{d f_i} = \frac{d(E_{\text{succ},i})}{d f_i} \times x \times t_{bit} = 0 \]  \hfill (9.9)

where, \( \frac{d(E_{\text{succ},i})}{d f_i} \) can be obtained from Equation (9.8), as:

\[ \frac{d(E_{\text{succ},i})}{d f_i} \approx m_i e^{-\frac{m_i}{f_i^2}} \]  \hfill (9.10)

when \( f_i \) is large. Using Taylor series, Equation (9.10) can be written, as:
By solving Equation (9.9) numerically, the optimal value of $f_i$ that minimises $T_i$ is approximately equal to $m_i$.

### 9.5.3 Cardinality Estimation

The reader can estimate the number of the target parity check values $m$ before running the RIC protocol. To accomplish the estimation, the reader transmits a random number $r$ and a frame size $f$. Each interrogated tag sends a short-response at a time slot $h(r, c_i) \mod f$. Some tags having the same $c_i$ will respond at the same slot, whilst others, having different parity check values $c_i$, may respond at the same slot. According to the EPCglobal Class-1 Generation-2 standard, only one bit is required to distinguish an empty slot from a nonempty one. For each received signal at the reader’s side, if the channel is idle, the received signal is set to 0, whilst if it is busy, which indicates that at least one tag has sent, the received signal is set to 1. The tag replies a short response of one bit at the corresponding time slot. After receiving the responses, the RFID reader measures the state of the slots and consequently, by counting the number of empty slots, $m$ can be estimated. The additional time overhead for estimating $m$ of the interrogated tags is measured under various group sizes $n$. Table 9.1 demonstrates the estimated time of the RIC protocol, which is very small. In subsection 9.6.2 below, it is shown that the RIC protocol outperforms other information collection protocols. Therefore, taking into account the estimation overhead will not affect the performance of the RIC protocol.

<table>
<thead>
<tr>
<th>$n$</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation time</td>
<td>0.32</td>
<td>0.64</td>
<td>0.96</td>
<td>1.6</td>
</tr>
</tbody>
</table>

**Table 9.1: Estimation time comparison (in second) under different group sizes**
Chapter 9 - Efficient Redundant Information Collection Protocol

9.5.4 Hash Functions

The problem on how to simplify the implementation of hash functions on RFID tags in order to decrease the complexity of a tag’s circuit has been discussed in Li et al. (2010) and Chen et al. (2011). In general, the hash value can be obtained from a ring of pre-stored random bits. A random number generator is implemented offline with the tag ID as a seed to produce a string of random bits. This string is then pre-stored as a logical ring in the tag before deployment and when \( k \) hash values are required, \( k \) portions of the string are stored instead.

For example, to obtain the hash value of \( h_z(TID, r) \), where \( 1 \leq z \leq k \), a tag obtains a portion of bits from the first \( z \) rings after the \( r \)-th bit. Another way is to start from the \( r \)-th bit and select one bit after each \( r \) bits. The final hash value is then computed as the number represented by these bits modulo the length of the BF or the frame size depending on the application. Li et al. (2015) also simplified the implementation of three hash functions by pre-storing a 32-bits random number on each tag. The reader produces three random seeds in the binary form and sends them to the tags in its region. Each tag then calculates the three hash values using:

\[
h(TID) = \text{getlbit} (RN \oplus RS(i), 13:1) \tag{9.12}
\]

where, \( \text{getlbit} \) is a function used to obtain the lowest 13 bits from the XOR results, \( RN \) is a random number in the binary form, and \( RS(i) \) where \( i \in \{1, 2, 3\} \), is the random seeds in the binary form. With the improvement in RFID tags, the hash functions are expected to be feasible at even more lightweight passive tags (Hutter et al., 2012; Yang, 2017).

9.6 Simulation Results

In this section, the performance of the RIC protocol is evaluated with Matlab R2018b. As no existing method is aimed at collecting redundant tag information, three protocols, including the BC, MIC with seven hash functions, and the enhanced version of the ALOHA-based
Chapter 9 - Efficient Redundant Information Collection Protocol

(EDFSA) protocols, as presented in Lee et al. (2005), are implemented. Their collection times are measured and compared with those of the RIC protocol.

9.6.1 Simulation Settings

The simulation settings in Matlab follow the specifications of the EPCglobal Class-1 Generation-2 standard. A waiting time of 302 μs is used to separate any two successive communications between the reader and the tags, with the transmission rate of a tag being 53 Kb/s. A tag needs 18.87 μs to send one bit. The length of time slot a tag needs to send its information is obtained as the addition of the time for sending the information and a waiting time. For instance, if the tag information is 600 bits, the length of a time slot is around 11,623 μs. The transmission rate of the reader is 26.5 Kb/s. The reader needs 37.74 μs to send one bit. Thus, it takes nearly 3925 μs (including the waiting time) to send a message of length 96 bits. For all the implemented protocols, the total number of parity check values \( m \) is assumed to be known a priori in order to derive a realistic baseline performance comparison. In the RIC protocol, the frame size will change based on the number of parity check values to be collected. The first round considers collecting \( m \) parity check values, and its frame has \( m \) slots. Each consequent round considers a smaller value of \( m \) and consequently, has a smaller frame size. All the presented results are obtained by averaging over 100 simulation runs in order to obtain an accurate value of the statistics.

9.6.2 Collection Time Comparison Under Different Numbers of Tags and Information Lengths

The performance of the RIC protocol is evaluated under different numbers of group sizes \( n \). The group sizes vary from 1,000 to 5,000. Each tag in a group stores the same number of parity check values \( j \) that is set to 3 and 4. The value of \( P_{FP} \) is set to \( 1 \times 10^{-4} \) to achieve the accuracy of 99.99%. Table 9.2 presents the results of the simulated protocols for different
values of $m$ within the same group, where the parity check information size $x$ is the same. The collection times of all the protocols increase as the number of tags $n$ or the number of parity check values $m$ increases. This is intuitive, because the tags need to transmit more information. The RIC protocol obviously outperforms the other protocols by having a minimum time, followed by MIC protocol, BC and, finally, EDFSA. For example, when the number of tags in a group is 3,000 and the number of parity check values is 1,800, the collection time of the EDFSA protocol is 138.31 s, which is around 24 times the lower bound. The BC protocol follows with 47.32 s, and the MIC protocol further minimises the time to 33.63 s. The RIC protocol considerably reduces the time of the MIC protocol by around 30% to 23.56 s, with its collection time being about four times the lower bound.

Examining the second columns of $n = 3,000$, where the number of parity check values is increased to 2,400, shows that the EDFSA protocol needs 183 s. The BC protocol reduces the collection time by nearly 62% to 68.83 s, whilst the MIC protocol requires 45.32 s, which is around six times the lower bound. The RIC protocol also considerably reduces the collection time to a minimum value of 32.06 s. Again, the collection time of the RIC protocol is about four times the lower bound, and this conclusion can also be drawn from all columns of Table 9.2.

<table>
<thead>
<tr>
<th>$n$</th>
<th>1,000</th>
<th>2,000</th>
<th>3,000</th>
<th>5,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>600</td>
<td>800</td>
<td>1,200</td>
<td>1,600</td>
</tr>
<tr>
<td>$j$</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$x$</td>
<td>146</td>
<td>151</td>
<td>156</td>
<td>161</td>
</tr>
<tr>
<td>Total tag information</td>
<td>438</td>
<td>584</td>
<td>453</td>
<td>604</td>
</tr>
<tr>
<td>EDFSA</td>
<td>30.31</td>
<td>40.1</td>
<td>76.86</td>
<td>101.6</td>
</tr>
<tr>
<td>BC</td>
<td>15.36</td>
<td>22.34</td>
<td>31.13</td>
<td>45.28</td>
</tr>
<tr>
<td>MIC</td>
<td>10.58</td>
<td>14.3</td>
<td>21.78</td>
<td>29.4</td>
</tr>
<tr>
<td>RIC</td>
<td>6.71</td>
<td>8.93</td>
<td>14.68</td>
<td>20.1</td>
</tr>
<tr>
<td>Lower bound</td>
<td>1.83</td>
<td>2.45</td>
<td>3.78</td>
<td>5.04</td>
</tr>
</tbody>
</table>

Table 9.2: Collection time comparison (in seconds) for different parity check values $m$
Table 9.3 presents the collection times when the amount of parity check values $m$ is fixed for each group and the parity check information size $x$ is changed. Again, similar conclusions can be drawn. For example, when the number of tags in a group is 3,000 and $j=3$, the collection time of the RIC protocol outperforms the other simulated ones. Its collection time is 18.29 s which is 51% of the time required by the MIC protocol, 38% of the time required by the BC protocol, and just 12.3% of the time required by the EDFSA protocol. Increasing $j$ to 4, causes an increase in the collection times of the MIC, BC, and EDFSA protocols. On the other hand, the collection time of the RIC protocol is slightly better than that with $j = 3$. When further scrutinising their average number of rounds, the RIC protocol with $j = 4$ needs an average of 9.37 round while the protocol with $j=3$ needs 10.9 round on average. This behaviour is due to the increase in the row weight $l$ of the parity check matrix, which causes an increase in the amount of short IDs stored on tags. Maximising the distribution of the parity check values among the tags has the effect of enhancing the process of reporting tag information in each round. In addition, eliminating the transmission of the redundant tag information with the BF reduces the communication cost and therefore, speeds up the RIC protocol. On the other hand, the value of $l$ should be carefully chosen as it can affect the performances of the decoding algorithms, as depicted in Figure 5.6 and Figure 5.7 in Chapter 5.

<table>
<thead>
<tr>
<th>$n$</th>
<th>1,000</th>
<th>2,000</th>
<th>3,000</th>
<th>5,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$j$</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$x$</td>
<td>156</td>
<td>176</td>
<td>162</td>
<td>184</td>
</tr>
<tr>
<td>Total tag information</td>
<td>468</td>
<td>704</td>
<td>486</td>
<td>736</td>
</tr>
<tr>
<td>EDFSA</td>
<td>32.31</td>
<td>48.07</td>
<td>82.27</td>
<td>123.24</td>
</tr>
<tr>
<td>BC</td>
<td>15.54</td>
<td>23.58</td>
<td>31.57</td>
<td>48.1</td>
</tr>
<tr>
<td>MIC</td>
<td>11.21</td>
<td>16.82</td>
<td>23.16</td>
<td>34.98</td>
</tr>
<tr>
<td>RIC</td>
<td>5.28</td>
<td>4.53</td>
<td>11.58</td>
<td>10.35</td>
</tr>
<tr>
<td>Lower bound</td>
<td>1.62</td>
<td>1.81</td>
<td>3.36</td>
<td>3.77</td>
</tr>
</tbody>
</table>

Table 9.3: Collection time comparison (in seconds) for the same parity check values $m$
9.6.3 Collection Time Comparison Under Different Settings of the False Positive Rate

The relationship between the collection time of the RIC protocol and $P_{FP}$ is also investigated. Table 9.4 compares those required by the RIC under different values of $P_{FP}$ ranging from $1\times10^{-2}$ to $1\times10^{-6}$, where the group size is set to $n = 2,000$ and the length of one piece of parity check information $x$ is 151 bits. Generally, the collection time of the RIC protocol keeps increasing with a decrease in $P_{FP}$. For example, when $P_{FP}$ is set to $1\times10^{-2}$ in order to achieve accuracy of 99 %, the collection time of the RIC reaches 10.19 s when $j=3$. When $P_{FP}$ is reduced to $1\times10^{-3}$ in order to achieve accuracy of 99.9%, the time of the RIC protocol increases to 12.48 s. This behaviour is due to the fact that the relation between $P_{FP}$ and $b$ is inversely proportional, as presented in Equation (8.9). That is, a low amount of $P_{FP}$ causes a larger $b$. Consequently, a larger $b$ in the RIC protocol increases the communication overhead and thus, the time needed for transmitting the BF vector between the reader and the tags. The collection time of the RIC protocol reaches a higher value equal to 19.45 s when $P_{FP}$ is set to the lowest tested value, which is $1\times10^{-6}$, in order to achieve accuracy of 99.9999 %. However, the RIC protocol is still faster than the collection times of the other simulated protocols as shown in the column of $n=2,000$ with $j=3$ of Table 9.2 and the same conclusion can also be drawn when $j$ is set to 4. In all cases of $P_{FP}$, the small amount of unreported parity check information would be mitigated with the spread of tag IDs to the parity check constraints. In other words, increasing the column weight would compensate the loss in parity check information; however, the value of $P_{FP}$ could be tuned based on the missing amount. For example, when the missing amount is high, the value of $P_{FP}$ should be decreased in order to minimise its effect on the recovery performance of the decoding process. Thus, the RIC protocol is applicable for collecting redundant tag information for the group integrity checking and decoding processes as long as the probability of false positives is sufficiently small and controlled.
### Table 9.4: Collection time comparison (in second) under different values of $P_{FP}$, where $n = 2,000$

<table>
<thead>
<tr>
<th>$P_{FP}$</th>
<th>$1 \times 10^{-2}$</th>
<th>$1 \times 10^{-3}$</th>
<th>$1 \times 10^{-4}$</th>
<th>$1 \times 10^{-5}$</th>
<th>$1 \times 10^{-6}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection time when $j = 3$</td>
<td>10.19</td>
<td>12.48</td>
<td>14.68</td>
<td>17</td>
<td>19.45</td>
</tr>
<tr>
<td>Collection time when $j = 4$</td>
<td>14</td>
<td>17.01</td>
<td>20.1</td>
<td>22.83</td>
<td>26.26</td>
</tr>
</tbody>
</table>

#### 9.7 Conclusions

The study presented in this chapter has been aimed at efficiently collecting redundant information from a group of tags for the group integrity checking and decoding processes, thus enhancing the complete system of group integrity applications. Extant solutions to general and different problems are not efficient since they do not distinguish the redundancy among tags, which results in long collection times. To achieve time efficiency, a novel protocol called the RIC protocol has been designed for this research. In this protocol, a BF is utilised as a tool to carry aggregated information in order to filter redundant tag information at the tag’s side efficiently and thus, save collection time. The performance lower bound for collecting redundant information has been provided to assess the performance of the designed protocol and the expected collection time has also been derived. Extensive simulations have been conducted to evaluate the performance of the designed protocol as well as the existing possible solutions. The results demonstrate that the designed protocol outperforms other existing possible solutions by saving between 37% and 84% of the collection time, which is also around four times the lower bound. This makes the RIC protocol a promising candidate for collecting redundant tag information for the group integrity of tags as long as the probability of false positives is sufficiently small and controlled.
10 Conclusions and Future Work

This chapter concludes the thesis by first highlighting the contributions and achievements that have been made through the research. This is followed by outlining the limitations and obstacles encountered during it, and finally, there is identification of potential areas for further research work.

10.1 Contributions and Achievements of the Research

This research has been devoted to the development of group integrity application in RFID systems. The first part of the thesis focused on improving the performance in recovering missing tag IDs with low memory consumption and decoding complexity. In the second part, the research was focused mainly on designing a novel protocol to read redundant tag information from a group of tags with low communication overheads. During the research programme, there several achievements that have contributed to the field, which are as follows.

1) Motivated by the properties of Tanner graphs, or equivalently, parity check matrices constructed using PEG methods, extended grouping via the PEG method was designed. The PEG method was used to construct the parity check matrix of LDPC codes in order to increase the missing recovery capabilities of the group integrity application. Different matrices were constructed with different parameters. A group of RFID tag IDs is treated as packet symbols and encoded through the constructed parity check matrix. The constructed PEG-based LDPC codes are decoded under the IT and GE decoding algorithms. The recovery performances of the two decoding algorithms were analysed and evaluated, then being compared with the performance of other existing LDPC-based methods. The properties of the constructed PEG Tanner graphs were also analysed and
compared with the Tanner graphs of other codes, which were shown to have higher girths. It was elicited that the PEG-based LDPC codes achieve significant missing recovery enhancements compared to other existing LDPC-based codes, with either the same or less memory overheads. (Chapter 5)

2) In order to strike a balance between the decoding complexities and performances of the IT and GE decoding algorithms, improved HB decoding was designed. This HB decoding algorithm includes an early stopping criterion to avoid unnecessary iterations of the IT decoding for undecodable blocks and thus, speeds up the decoding time. The practical performance of the early stopping criterion was evaluated and compared with the conventional IT decoding algorithm. The designed stopping criterion greatly reduces the average number of decoding iterations for undecodable blocks and therefore, reduces the decoding time when compared to the conventional IT decoding algorithm. (Chapter 6)

3) The recovery performance of the improved HB decoding algorithm was analysed and evaluated with different parameters. It achieved the optimal missing recovery capabilities of the full GE decoding algorithm. The decoding complexity of the improved HB decoding algorithm was extensively analysed and evaluated, both in terms of the number of operations required and the decoding time. It was demonstrated that the improved HB decoding algorithm considerably reduces the operational complexity and also, the time of the full GE decoding algorithm for small to medium amounts of missing RFID tags up to a certain threshold value, which depends on the missing recovery capabilities of the IT decoding algorithm. (Chapter 6)

4) The joint use of the two decoding components of the improved HB decoding algorithm was also adapted based on the missing amounts of RFID tags in order to avoid IT decoding when the missing amount is larger than a threshold and thus, save more decoding time. The optimum value of the threshold value was investigated for different missing amounts of tags and different parameters of the PEG method through empirical
analysis. Various simulations were conducted in order to analyse and evaluate the impact of the threshold value on the improved HB decoding time. It was elicited that the adaptive decoding algorithm is very efficient in decreasing the average decoding time of the improved HB decoding, thus saving more decoding time of up to 10% for large amounts of missing RFID tags, where the IT decoding fails to recover any missing tags. (Chapter 7)

5) Many simulations were carried out to analyse and evaluate the missing recovery performance of various short-length irregular PEG-based LDPC codes constructed with different variable degree sequences. The influence of low and high degree variable nodes on the performance of recovering missing tag IDs was also examined. It was demonstrated that the irregular PEG-based LDPC codes exhibit significant missing recovery enhancements compared to the regular PEG-based LDPC ones in the region where IT decoding is successful. However, their performances are degraded in the region where IT decoding can recover some missing tag IDs. It was empirically found that for half rate short block lengths, the density evolution method is good for irregular PEG construction. Simulation results also show that as the fraction of low degree variable nodes increases, the performance of the IT decoding in recovering missing tag IDs also increases in the two regions. On the other hand, increasing the fraction of the high degree variable nodes degrades the recovery performance in these regions. In addition, the recovery performance of the code that has a maximum variable node degree of 7 outperforms other simulated codes, with a maximum variable node degree of 9, 11, 13, and 15 in both regions. (Chapter 7)

6) A novel protocol called the Redundant Information Collection (RIC) Protocol was designed to filter and collect redundant information from a group of tags in group integrity applications with low communication overheads. To the best of this researcher’s knowledge, there has been no prior work on how to collect and filter redundant tag
information at the tag’s side efficiently. In the RIC protocol, A Bloom filter was constructed for the reader to filter the redundant tag information efficiently, which significantly decreases the communication overheads. The performance lower bound for collecting redundant information was provided to assess the performance of the designed protocol. The expected collection time and the cardinality estimation method were also derived. (Chapter 9)

7) Three existing possible solutions to general and different problems that could be applied to collect redundant tag information were implemented, analysed and examined. It was shown how all of them are not efficient at solving the redundant tag information collection problem since they do not distinguish redundancy among tags, which results in long collection times. (Chapter 9)

8) The performance of the RIC protocol was implemented, analysed and evaluated with different parameters and then, compared with the existing possible solutions. It was shown that the RIC protocol outperforms the existing solutions by saving between 37% and 84% of the collection time, which is nearly four times the lower bound. This characteristic makes the RIC protocol a promising candidate for collecting redundant tag information in the group integrity of tags as long as the probability of false positives is sufficiently small and controlled. (Chapter 9)

10.2 Limitations of the Research

Whilst the objectives of this research have been met, a number of issues have arisen, which may have imposed limitations upon the work progress and findings in one way or another.

The key limitations of the research are as follows.

1) The focus of the research was only on the widely used passive RFID tags. With this restriction, the efforts of improving the application of group integrity of RFID tags were
limited to the characteristics of this type of tags. On the other hand, the features of the other types of RFID tags may open a wider area of improvements.

2) The proposed mechanisms and protocol were not evaluated on real RFID systems. It would be desirable to conduct a series of experimental evaluations involving different numbers of RFID tags to confirm the validity of the numerical simulations and to appraise the practical usefulness of the proposed approaches. However, it is contended that having already established the empirical analysis have been useful and provided a fair insight about the performance.

Despite these aforementioned limitations, it is maintained that the research has made valid contributions to knowledge and provided sufficient proof of concept for the ideas proposed.

10.3 Further work

A number of suggestions for future work outside the scope of this study have been identified and they are as follows:

1) *The encoding process complexity*: A new method needs to be developed in order to reduce the relative complexities of encoding a group of tag IDs through the parity check matrix both in terms of the number of operations required and the encoding time;

2) *Average error rate bounds*: lower and upper bounds for the IT and GE decoding average error rates for the PEG-based LDPC codes in group integrity applications need to be derived theoretically;

3) *Dynamic Bloom filter constructions*: The construction of the BF in the RIC protocol could be implemented dynamically. To enable this feature, further development needs to be accomplished where the length of the BF is adapted in each round of the protocol instead of reconstruction the filter;
4) **Security features:** Further research and development is required in providing some security aspects such as protecting the information in a tag’s memory and the transmitted information between the reader and tags.

5) **Channel error:** This is an important implementation issue, which may corrupt the information transmitted between the reader and the tags and thus, interrupting the process of group integrity checking and the RIC protocol as well. Hence, more work needs to be carried out to overcome the effect of channel errors. The group integrity method and the RIC protocol may need to be modified accordingly;

6) **Multi reader RFID systems:** Some large scale RFID systems include more than one reader. In contrast to the one reader deployed in small RFID system, in multi reader ones, each reader has to know which tags exist in its area in order to collect information from them. The readers must communicate with each other to read and recover the missing tag IDs. Consequently, research and software developments are required to enable such a system;

7) **Hashing of tags:** Due to the development of cheap and lightweight RFID tags that have limited resources, the addition of hash-based technique forms very interesting research area. That is, additional investigations and developments are needed in the area of lightweight hash functions in order to reduce the complexity of the tag’s circuit even on more constrained passive tags.
References


References


161


References


164

References


References


RFID Tags’, *IEEE Transaction on Communications*, 61 (11), pp. 4741–4753.


Number of Erasures Correctable by a Binary Linear Code and the Link to Low-Weight Codewords’, *IET Communications*, 1 (3), pp. 539–548.


Appendices

Appendix A - Some Source Codes of the Simulations

MainPEG.C

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <math.h>
#include "BigGirth.h"
#include "Random.h"
#include "CyclesOfGraph.h"

#define EPS  1e-6
using namespace std;

int main(int argc, char * argv[]){
  int i, j, m, N, M;
  int sglConcent=1;  // default to non-strictly concentrated parity-check distribution
  int targetGirth=100000; // default to greedy PEG version
  char codeName[100], degFileName[100];
  int *degSeq, *deg;
  double *degFrac;
  BigGirth *bigGirth;
  CyclesOfGraph *cog;

  int numArgs=(argc-1)/2;
  if (argc<9) {
    USE:
    cout<<"********************************************************************
    ***********************"<<endl;
    cout<<" Usage Reminder: MainPEG -numM M -numN N -codeName CodeName -
    degFileName DegFileName " <<endl;
    cout<<" option: -sglConcent SglConcent " <<endl;
    cout<<" sglConcent==0 ----- strictly concentrated parity-check " <<endl;
    cout<<" degree distribution (including regular graphs)" <<endl;
    cout<<" sglConcent==1 ----- Best-effort concentrated (DEFAULT) " <<endl;
  }
  USE:
cout"" option: -tgtGirth TgtGirth "" <<endl;
cout"" TgtGirth=4, 6 ...; if very large, then greedy PEG (DEFAULT) "" <<endl;
cout"" IF sglConcent==0, TgtGirth is recommended to be set relatively small"" <<endl;
cout"" Remarks: File CodeName stores the generated PEG Tanner graph. The first line contains"" <<endl;
cout"" the block length, N. The second line defines the number of parity-checks, M."" <<endl;
cout"" The third line defines the number of columns of the compressed parity-check matrix. "" <<endl;
cout"" Each of the M rows contains the indices (1 ... N) of 1's in the compressed row of parity-check matrix. If not all column entries are used, the column is filled up with 0's. "" <<endl;
cout"" File DegFileName is the input file to specify the degree distribution (node perspective). The first line contains the number of various degrees. The second defines the row vector of degree sequence in the increasing order. The vector"" <<endl;
cout"" of fractions of the corresponding degree is defined in the last line. "" <<endl;
cout"" A log file called 'leftHandGirth.dat' will also be generated and stored in the current directory, which gives the girth of the left-hand subgraph of j, where 1<=j<=N. The left-hand subgraph of j is defined as all the edges emanating from bit nodes {1 ... j} and their associated nodes. "" <<endl;
cout"" The last point is, when strictly concentrated parity-check degree distribution"" <<endl;
cout"" is invoked, i.e. sglConcent==0, the girth might be weaken to some extent as compared to the generic PEG algorithm. "" <<endl;
cout""********************************************************************
**************************"" <<endl;
exit(-1);
} else {
    for(i=0;i<numArgs;i++){
        if (strcmp(argv[2*i+1], "-numM")==0) {
            M=atoi(argv[2*i+2]);
        } else if(strcmp(argv[2*i+1], "-numN")==0) {
            N=atoi(argv[2*i+2]);
        } else if(strcmp(argv[2*i+1], "-codeName")==0) {
            strcpy(codeName, argv[2*i+2]);
        } else if(strcmp(argv[2*i+1], "-degFileName")==0) {
            strcpy(degFileName, argv[2*i+2]);
        } else if(strcmp(argv[2*i+1], "-sglConcent")==0) {
            sglConcent=atoi(argv[2*i+2]);
        } else if(strcmp(argv[2*i+1], "-tgtGirth")==0) {
            targetGirth=atoi(argv[2*i+2]);
        } else{
            goto USE;
        }
    }
    if(M>N) {
        cout<<"Warning: M must be smaller than N"
        exit(-1);
    }
}

degSeq=new int[N];
ifstream infn(degFileName);
if (!infn) {cout << "Cannot open file " << degFileName
    << endl; exit(-1); }
infn >>m;
deg=new int[m];
degFrac=new double[m];
for(i=0;i<m;i++) infn>>deg[i];
for(i=0;i<m;i++) infn>>degFrac[i];
infn.close();
double dtmp=0.0;
for(i=0;i<m;i++) dtmp+=degFrac[i];
cout.setf(ios::fixed, ios::floatfield);
if(fabs(dtmp-1.0)>EPS) {
    cout.setf(ios::fixed, ios::floatfield);
    cout <<"Invalid degree distribution (node perspective): sum != 1.0 but"
    "setprecision(10)"<<dtmp<<endl; exit(-1);
}
for(i=1;i<m;i++) degFrac[i]+=degFrac[i-1];
for(i=0;i<N;i++) {
    dtmp=(double)i/N;
}
for(j=m-1;j>=0;j--) {
    if(dtmp>degFrac[j]) break;
} 
if(dtmp<degFrac[0]) degSeq[i]=deg[0];
else degSeq[i]=deg[j+1];
}
bigGirth=new BigGirth(M, N, degSeq, codeName, sglConcent, targetGirth);
(*bigGirth).writeToFile_Hcompressed();

//computing local girth distribution
if(N<10000) {
    cout<<" Now computing the local girth on the global Tanner graph setting. "<<endl;
    cout<<" might take a bit long time. Please wait ... "<<endl;
    (*bigGirth).loadH();
    cog=new CyclesOfGraph(M, N, (*bigGirth).H);
    (*cog).getCyclesTable();
    (*cog).printCyclesTable();
    delete cog;
    cog=NULL;
}

delete [] degSeq; degSeq=NULL;
delete [] deg; deg=NULL;
delete [] degFrac; degFrac=NULL;
delete bigGirth;
}

**Iterative Decoding**

function [erasure_vec, avg_error] = get_iterative(fname)
addpath './lib/
% the max word size of word in Matlab
max_word_size = 64;
alist_params = readalist_peg(fname);
H = alist2parity(alist_params);
[~, nc] = size(H);
[newH] = H;
G_s = full((newH));
[m, n] = size(G_s);
data_word = [];
while length(unique(data_word))~=n
    data_word = round((2^max_word_size-2)*rand(1, n)) + 1;
end
D_de_rep = repmat(data_word, m, 1).*G_s;
parity_words = zeros(m, 1);
for i=1:n
    parity_words = bitxor(D_de_rep(:, i), parity_words);
end

disp('Input Data Values');
disp(data_word);
disp('Generator Matrix');
disp(G_s);
disp('Parity Values');
disp(parity_words);
maxFrames = 100000;
erasure_vec = (1:nc);
avg_error = zeros(1, length(erasure_vec));
[nCheck, nBit] = size(newH);
Nedge = nnz(newH);
indexLinear = find(newH);
[indexCheck, indexBit] = find(newH); % For sparse matrix, find() is fast.
% The following initialization of the check nodes takes ONE TnewHIRD of the running
time!
checkNode = struct('indexBitConnected', 0);
check = repmat(checkNode, nCheck, 1);
for iCheck = 1:nCheck
    check(iCheck).indexBitConnected = indexBit(indexCheck == iCheck); % Connections to
    check nodes
end % for iCheck
nIterationMax = 80;
for erasure_index=1:length(erasure_vec)
    cFrame = 1;
    while cFrame < maxFrames,
        y = randperm(n, erasure_vec(erasure_index));
        missing_index = false(1, n);
        missing_index(y) = true;
        % iterative decoding
        data_word_rx = data_word.*(~missing_index);
        [messageBit, nIteration] = bpdecfast(data_word_rx, parity_words, nIterationMax,
        indexLinear, indexCheck, indexBit, check, nCheck, nBit, nEdge, 0);
        avg_error(erasure_index) = avg_error(erasure_index) +
        sum(messageBit~=data_word)/erasure_index;
        cFrame = cFrame + 1;
    end
    avg_error(erasure_index) = avg_error(erasure_index)/cFrame;
end

**Gaussian Decoding**

function [erasure_vec, avg_error] = getgaussian(fname)
addpath './lib/'

% the max word size of word in Matlab
max_word_size = 64;
alist_params = readalist_peg(fname);
H = alist2parity(alist_params);
[nr, nc] = size(H);
[newH] = H;
G_s = full(fliplr(newH));
[m, n] = size(G_s);
data_word = round((2^max_word_size-1)*rand(1, n)) + 1;
D_de_rep = repmat(data_word, m, 1).*G_s;
parity_words = zeros(m, 1);
for i=1:n
    parity_words = bitxor(D_de_rep(:, i), parity_words);
end
maxFrames = 100000;
erasure_vec = (1:nc);
avg_error = zeros(1, length(erasure_vec));
for erasure_index=1:length(erasure_vec)
cFrame = 1;
    while cFrame < maxFrames,
        y = randperm(n, erasure_vec(erasure_index));
        missing_index = false(1, n);
        missing_index(y) = true;
        % Making system of equation for unknown words.
        bprep = [parity_words D_de_rep.*repmat(~missing_index, m, 1)];
        % Computation of b in Ax = b form of the equation.
        b = zeros(m, 1);
        for k=1:n+1
            b = bitxor(bprep(:, k), b);
        end
        % Computation of A in Ax = b form of the equation.
        A = G_s(:, missing_index);
        % Using Gauss elimination to solve the system of equations.
        x = bin_gauss_solve(A, b);
        avg_error(eraser_index) = avg_error(eraser_index) ... 
            + (eraser_index-sum(x==data_word(missing_index)))/eraser_index;
        cFrame = cFrame + 1;
    end
avg_error(eraser_index) = avg_error(eraser_index)/cFrame;
end

**Hybrid Decoding**

function [eraser_vec, avg_error_hyb] = get_hybrid_early(fname)
addpath './lib/'
% the max word size of word in Matlab
max_word_size = 64;
alist_params = readalist_peg(fname);
H = alist2parity(alist_params);
[nr, nc] = size(H);
[newH] = H;
G_s = full((newH));
[m, n] = size(G_s);
N = n;
M = m;
data_word = [];
while length(unique(data_word))~=n
    data_word = round((2^max_word_size-2)*rand(1, n)) + 1;
end
D_de_rep = repmat(data_word, m, 1).*G_s;
parity_words = zeros(m, 1);
for i=1:n
    parity_words = bitxor(D_de_rep(:, i), parity_words);
end
maxFrames = 100000;
erasure_vec = (1:nc);
avg_error_it = zeros(1, length(erasure_vec));
avg_error_hyb = avg_error_it;
[nCheck, nBit] = size(newH);
NEdge = nnz(newH);
indexLinear = find(newH);
[indexCheck, indexBit] = find(newH); % For sparse matrix, find() is fast.
% The following initialization of the check nodes takes ONE TnewHIRD of the running
% time!
checkNode = struct('indexBitConnected', 0);
check = repmat(checkNode, nCheck, 1);
for iCheck = 1:nCheck
    check(iCheck).indexBitConnected = indexBit(indexCheck == iCheck); % Connections to
    check nodes
end % for iCheck
nIterationMax = 80;
Hi = -1*ones(m, max(sum(H, 2)));
Hj = -1*ones(max(sum(H, 1)), n);
for i=1:m
    Hi(i, 1:sum(H(i, :))) = find(H(i, :));
end
for i=1:n
    Hj(1:sum(H(:, i)), i) = find(H(:, i));
end
Hi = int32(Hi-1);
Hj = int32(Hj-1);
for erasure_index=1:length(erasure_vec)
    cFrame = 1;
    while cFrame < maxFrames,
        y = randperm(n, erasure_vec(erasure_index));
        missing_index = false(1, n);
        missing_index(y) = true;
        % iterative decoding
        data_word_rx = data_word.*(~missing_index);
        time1 = tic;
        [messageBit, nIteration] = bpd_early(Hi(:), Hj(:), int32(data_word_rx(:)),
            int32(parity_words(:)), m, n, nIterationMax, max(sum(H, 2)), max(sum(H, 1)));
        errt_index = sum(messageBit==0);
        if errt_index>0
            % if unable to decode using iterative, use Gaussian.
            mi = false(1, N);
            mi(messageBit==0) = true;
            missing_index = mi;
            D_de_rep = repmat(messageBit, M, 1).*G_s;
            % Making system of equation for unknown words.
            bprep = D_de_rep.*repmat(~missing_index, M, 1);
            % Computation of b in Ax = b form of the equation.
            b = zeros(M, 1);
            for k=1:N
                b = bitxor(bprep(:, k), b);
            end
            b = bitxor(b, parity_words);
            % Computation of A in Ax = b form of the equation.
            A = G_s(:, missing_index);
            % Using Gauss elimination to solve the system of equations.
            x = bin_gauss_solve(A, b);
            messageBit(missing_index) = x;
        end
        avg_error_hyb(erasure_index) = avg_error_hyb(erasure_index) +
            sum(messageBit~=data_word)/erasure_index;
        cFrame = cFrame + 1;
    end
    avg_error_it(erasure_index) = avg_error_it(erasure_index)/cFrame;
    avg_error_hyb(erasure_index) = avg_error_hyb(erasure_index)/cFrame;
end

**RIC Protocol**

close all; clear all; clc;
addpath('libs');
case_no = 1;
m = 1000; % number of values
maxframes = 100; % maximum number of frames.
%n = 2*m; % number of tags
n=2000;
rate_t2r = 53e3;
rate_r2t = 26.5e3;
twait = 302e-6;
falseprob = 10^-4;
if case_no==1
    nbits = 162; % number of bits
    H = get_tags_vals(m, n, [0 1]);
else
    nbits = 184; % number of bits
    H = get_tags_vals(m, n, [0 0 1]);
end
Hmat = H;
tottime = 0;
round=0
for k=1:maxframes,
    f = m;
    H = Hmat;
    T = [];
    while f~=0
        rval = randi(1000, 1);
        % Bloom filter parameters
        b = ceil((m-f)*log(falseprob)/log(2)^2);
        k = ceil(log(2)*b/(m-f));
        tottime = tottime + ceil(b/96)*(twait+96/rate_r2t);
        tottime = tottime + f*(twait + nbits/rate_t2r);
        tagsloc = zeros(n, 1);
        tagsval = zeros(n, 1);
        for q=1:n,
            % for each tag, find its first value
            v_val = find(H(:, q), 1);
            % if the tag is not empty, find its slot.
            if ~isempty(v_val)
                tagsval(q) = v_val;
                tagsloc(q) = hashfunc_tags(v_val, rval, f);
            end
        end
        M = bsxfun(@eq, tagsloc, (1:f));
        Q = sum(M);
        T = find(sum(M(:, Q==1), 2));
        c1 = length(T);
for q=2:max(Q),
P = find(Q==q);
for t=1:length(P),
    Z = find(M(:, P(t))); 
    if sum(tagsval(Z) == tagsval(Z(1)))==length(Z)
        T = [T; Z];
        c1 = c1 + 1;
    end
end
vvalvec = [];
for q=1:length(T),
    vval = find(H(:, T(q)), 1, 'first');
    vvalvec =[vvalvec; vval];
    H(vval, :) = 0;
end
round=round+1;
f = f-length(unique(vvalvec));
end
round=round/maxframes;
disp('average round');
disp(round);
fprintf('Average Time %6.4f seconds...
', tottime/maxframes);
### Appendix B-Some Constructed LDPC-Based Codes

<table>
<thead>
<tr>
<th>(25, 3, 5) PEG-based</th>
<th>(49, 3, 7) PEG-based</th>
<th>(121, 3, 11) PEG-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 6 11 17 22</td>
<td>1 9 16 24 31 39 45</td>
<td>1 12 23 35 46 57 67 79</td>
</tr>
<tr>
<td>2 7 12 18 23</td>
<td>2 10 17 21 31 37 46</td>
<td>9 10 30 41 52 69 80 92</td>
</tr>
<tr>
<td>2 8 13 19 22</td>
<td>2 11 18 25 32 40 47</td>
<td>2 13 24 36 47 58 68 80</td>
</tr>
<tr>
<td>3 9 11 19 20</td>
<td>3 10 19 26 33 40 48</td>
<td>2 14 25 33 48 59 67 81</td>
</tr>
<tr>
<td>1 9 13 16 21</td>
<td>4 11 19 23 30 41 46</td>
<td>2 14 25 33 48 59 67 81</td>
</tr>
<tr>
<td>4 10 12 16 22</td>
<td>5 12 20 27 32 42 46</td>
<td>3 14 26 35 49 60 69 82</td>
</tr>
<tr>
<td>5 6 13 20 23</td>
<td>1 12 19 25 34 37 49</td>
<td>4 15 23 37 50 61 70 81</td>
</tr>
<tr>
<td>7 17 14 23</td>
<td>6 8 21 24 35 42 49</td>
<td>5 16 27 38 49 62 71 83</td>
</tr>
<tr>
<td>5 9 12 17 24</td>
<td>7 11 22 27 31 38 0</td>
<td>1 10 30 37 47 62 79 93</td>
</tr>
<tr>
<td>4 8 15 20 21</td>
<td>3 8 20 25 36 43 0</td>
<td>2 10 31 42 52 57 74 86</td>
</tr>
<tr>
<td>6 14 18 25</td>
<td>8 13 14 27 33 44 0</td>
<td>2 10 17 24 43 64 75 87</td>
</tr>
<tr>
<td>3 8 16 18 24</td>
<td>6 9 22 28 33 41 47</td>
<td>3 17 21 37 13 73 82 92</td>
</tr>
<tr>
<td>1 7 15 19 25</td>
<td>7 12 21 28 30 43 45</td>
<td>4 14 17 28 32 39 0</td>
</tr>
<tr>
<td>2 10 11 15 24</td>
<td>5 14 18 24 34 43 48</td>
<td>4 9 20 26 35 38 0</td>
</tr>
<tr>
<td>5 10 14 21 25</td>
<td>3 15 22 29 37 42 45</td>
<td>1 15 17 30 36 44 48</td>
</tr>
<tr>
<td>(25, 3, 5) PEG-based</td>
<td>2 13 16 26 34 41 0</td>
<td>1 21 17 29 45 61 83</td>
</tr>
<tr>
<td>(25, 3, 5) PEG-based</td>
<td>5 10 16 29 35 44 47</td>
<td>7 13 23 29 36 40 49</td>
</tr>
<tr>
<td>(25, 3, 5) PEG-based</td>
<td>6 15 18 23 38 39 0</td>
<td>(169, 3, 13) PEG-based</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(169, 3, 13) PEG-based</th>
<th>(200, 3, 6) PEG-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 14 27 42 54 64 79 92</td>
<td>1 35 69 103 137 170</td>
</tr>
<tr>
<td>2 18 28 43 55 68 80 93</td>
<td>2 36 70 104 138 171</td>
</tr>
<tr>
<td>3 15 29 43 56 69 81 94</td>
<td>3 37 71 105 139 172</td>
</tr>
<tr>
<td>4 37 51 64 76 82 95 104</td>
<td>4 37 72 106 140 173</td>
</tr>
<tr>
<td>5 28 32 46 59 70 83 96</td>
<td>5 38 73 107 141 174</td>
</tr>
<tr>
<td>6 19 30 46 58 72 84 96</td>
<td>6 39 74 103 142 174</td>
</tr>
<tr>
<td>7 20 32 47 59 73 85 98</td>
<td>7 40 75 108 143 175</td>
</tr>
<tr>
<td>8 21 33 46 60 68 86 98</td>
<td>8 34 72 109 144 176</td>
</tr>
<tr>
<td>9 22 34 47 60 72 87 99</td>
<td>9 40 64 106 145 177</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>10 39 76 110 127 178</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>3 41 77 111 146 179</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>11 42 78 111 147 173</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>12 43 76 112 139 180</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>13 44 79 107 134 175</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>5 45 80 113 138 181</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>14 46 81 105 148 171</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>1 37 82 114 149 182</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>15 47 71 115 150 183</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>12 48 83 116 151 184</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>10 44 84 117 152 167</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>4 39 77 111 146 179</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>11 42 78 111 147 173</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>12 43 76 112 139 180</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>13 44 79 107 134 175</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>5 45 80 113 138 181</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>14 46 81 105 148 171</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>1 37 82 114 149 182</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>15 47 71 115 150 183</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>12 48 83 116 151 184</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>10 44 84 117 152 167</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>9 35 85 118 153 185</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>7 49 86 109 136 178</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>16 50 83 110 146 186</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>5 51 87 119 148 187</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>17 52 88 97 154 175</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>18 53 77 117 155 171</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>8 54 80 120 156 168</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>19 55 89 121 145 181</td>
</tr>
<tr>
<td>(169, 3, 13) PEG-based</td>
<td>12 56 70 122 142 188</td>
</tr>
<tr>
<td>192</td>
<td>77 149 244 264 402 507 0</td>
</tr>
<tr>
<td></td>
<td>63 156 206 284 354 433 0</td>
</tr>
<tr>
<td></td>
<td>46 111 234 295 419 508 0</td>
</tr>
<tr>
<td></td>
<td>27 151 217 334 401 462 0</td>
</tr>
<tr>
<td></td>
<td>57 139 241 277 420 429 0</td>
</tr>
<tr>
<td></td>
<td>62 108 245 269 421 446 0</td>
</tr>
<tr>
<td></td>
<td>44 98 246 316 379 498 0</td>
</tr>
<tr>
<td></td>
<td>35 63 156 206 284 354 433 0</td>
</tr>
<tr>
<td></td>
<td>264 264 402 507 0</td>
</tr>
<tr>
<td></td>
<td>508 62 108 245 269 421 446 0</td>
</tr>
<tr>
<td></td>
<td>498 44 98 246 316 379 498 0</td>
</tr>
<tr>
<td></td>
<td>354 35 63 156 206 284 354 433 0</td>
</tr>
<tr>
<td></td>
<td>433 264 264 402 507 0</td>
</tr>
<tr>
<td></td>
<td>402 508 62 108 245 269 421 446 0</td>
</tr>
<tr>
<td></td>
<td>498 44 98 246 316 379 498 0</td>
</tr>
<tr>
<td></td>
<td>354 35 63 156 206 284 354 433 0</td>
</tr>
<tr>
<td></td>
<td>433 264 264 402 507 0</td>
</tr>
<tr>
<td></td>
<td>402 508 62 108 245 269 421 446 0</td>
</tr>
<tr>
<td></td>
<td>498 44 98 246 316 379 498 0</td>
</tr>
<tr>
<td></td>
<td>354 35 63 156 206 284 354 433 0</td>
</tr>
<tr>
<td></td>
<td>433 264 264 402 507 0</td>
</tr>
<tr>
<td>18</td>
<td>112</td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>20</td>
<td>146</td>
</tr>
<tr>
<td>82</td>
<td>127</td>
</tr>
<tr>
<td>76</td>
<td>150</td>
</tr>
<tr>
<td>11</td>
<td>96</td>
</tr>
<tr>
<td>75</td>
<td>122</td>
</tr>
<tr>
<td>74</td>
<td>172</td>
</tr>
<tr>
<td>2</td>
<td>158</td>
</tr>
<tr>
<td>20</td>
<td>146</td>
</tr>
<tr>
<td>82</td>
<td>127</td>
</tr>
<tr>
<td>76</td>
<td>150</td>
</tr>
<tr>
<td>11</td>
<td>96</td>
</tr>
<tr>
<td>75</td>
<td>122</td>
</tr>
<tr>
<td>74</td>
<td>172</td>
</tr>
<tr>
<td>2</td>
<td>158</td>
</tr>
<tr>
<td>20</td>
<td>146</td>
</tr>
<tr>
<td>82</td>
<td>127</td>
</tr>
<tr>
<td>76</td>
<td>150</td>
</tr>
<tr>
<td>11</td>
<td>96</td>
</tr>
<tr>
<td>75</td>
<td>122</td>
</tr>
<tr>
<td>74</td>
<td>172</td>
</tr>
<tr>
<td>2</td>
<td>158</td>
</tr>
<tr>
<td>20</td>
<td>146</td>
</tr>
<tr>
<td>82</td>
<td>127</td>
</tr>
<tr>
<td>76</td>
<td>150</td>
</tr>
<tr>
<td>11</td>
<td>96</td>
</tr>
<tr>
<td>75</td>
<td>122</td>
</tr>
<tr>
<td>74</td>
<td>172</td>
</tr>
<tr>
<td>2</td>
<td>158</td>
</tr>
<tr>
<td>20</td>
<td>146</td>
</tr>
</tbody>
</table>

**Appendixes**

(512, 3, 2) CRT-based


193
(512, 3, 3) CRT-based

Appendices