IMPROVING THE NETWORK THROUGHPUT WITH CODING THEORY

KHOO ZONG CHEN

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Electrical and Electronic Engineering

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2016

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : _____

ID No. : _____

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"IMPROVING THE NETWORK THROUGHPUT WITH CODING THEORY"** prepared by **KHOO ZONG CHEN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : _____

Date : _____

IMPROVING THE NETWORK THROUGHPUT WITH CODING THEORY

ABSTRACT

In the transmission model, the message symbols may be erased or interfered during the transmission that will eventually cause the receiver fails to receive the original message symbols. The receiver will, by anyhow, ask for the retransmission of the message symbols until that the complete message symbols are received, but this will results in decreased network throughout. In this report, only erasure channel will be considered so that the erasure correcting code can be applied to improve the network throughput. Reed-Solomon codes, which are a fixed-rate erasure codes, can always decode successfully whenever the number of encoded symbols received is *k*, out of *n* total encoded symbols, where *k* is the number of original message symbols. But it is constraint to that the code rate (k/n) has to be predetermined and the erasure probability has to be pre-estimated. Later on, rateless erasure codes such as Luby Transform (LT) codes and Raptor codes are introduced to get rid of the said constrains. The encoder will repeatedly encode and send message packets until the decoder successfully retrieve the original message symbols, and then only the decoder will signal the encoder to stop sending to end the transmission. These codes, however, are advantageous only for long messages. Hence, Random codes are introduced to compromise for the short messages. With $k+10$ encoded symbols, they are able to achieve 99.9% of complete decoding. However, it is still considered inefficient because, for example, for a short message of $k=10$, now it requires a total number of 20 encoded symbols to achieve high probability of complete decoding, which is double the size of the original message symbols. So, the finite field in the Random codes is expanded to a higher order for the improvement i.e. minimum usage of overhead symbols in Random codes while attaining high probability of complete decoding. Finally, the systematic Random codes are also proposed in the report to boost their usefulness.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

k	message symbols
n	total encoded symbols
e	error
i	received bits
j	transmitted bits
q	order or element
C	channel capacity
G	random generator symbol matrix
M	message symbol matrix
X	encoded symbol matrix
I	identity matrix
δ	erasure probability
LT	Luby Transform
BEC	binary erasure channel
GF	Galois field
ASCII	American Standard Code for Information Interchange

CHAPTER 1

INTRODUCTION

1.1 Erasure Channel

In coding theory, erasure channel is a communication channel model, where the encoded data packets sent by the transmitter in most of the cases will get erased partially during transmission, and the decoder at the receiver side will not receive enough encoded packets to recover the original data.

1.1.1 Binary Erasure Channel

A binary erasure channel (BEC) as shown in Figure 1.1, is the common communication channel where losses of data occur in the channel, and it is one of the simplest noisy channels.

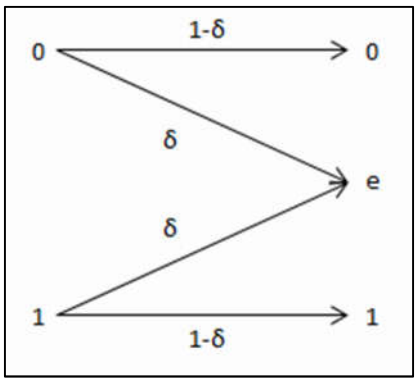

Figure 1.1: Binary Erasure Channel Model

This kind of channel has a binary input, i.e. 0 or 1 and a ternary output, i.e. 0, 1 and error (e). The sender sends a bit, either 0 or 1, and the receiver will either receive that particular bit or error due to the bit being erased during transmission. The chance of the bit being erased in the noisy channel is determined by the erasure probability, δ . On the other hand, the probability of the receiver to receive the bit without error is $1 - \delta$, which is also known as capacity of that particular channel.

Since the channel is memoryless and the receiver will either receive the correct bit or error for each transmission, each time a bit transmitted is equivalent to conducting an independent Bernoulli trial. The desired probability is the same as the probability of i successes out of j trials, where j is the transmitted bits and i is the received bits. Hence, it has a binomial distribution:

$$P(i) = \binom{j}{i}(1 \quad \delta)^i \delta^{(j-i)} \tag{1.1}$$

For most of the cases, the sender has more than two inputs. Figure 1.2 shows the 8-ary erasure channel model.

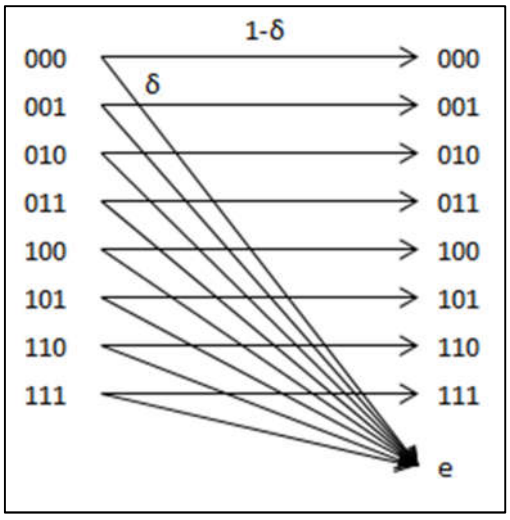

Figure 1.2: The 8-ary Erasure Channel Represented in Binary Form

In this erasure channel, say if the message symbols are chopped into 8 encoded symbols, the binary digits are used to represent these encoded symbols, starting from 000, 001, … to 111 accordingly. During transmission, if the noise occurs in the channel with erasure probability δ , the receiver will fail to receive

some of the bits due to the bits being erased. Hence the original message symbols cannot be recovered. An erasure code is indeed necessary to bring back the missing bits.

1.2 Erasure Codes

Communication over the erasure channel normally requires a feedback channel to link between the sender and receiver so that the feedback channel can indicate for the retransmission due to erased packets detected at the receiver side. The retransmission will stop when all the packets have been received correctly.

This kind of feedback has the advantage that they will always succeed when all the retransmissions end, regardless of the erasure probability δ. But with reference to Shannon theory, this method is wasteful in terms of memory because when the probability δ is large, the number of feedback messages will be large too. It will end up receiving multiple redundant copies of some packets. According to Shannon, with appropriate erasure code applied in the noisy channel, the feedback from the receiver can be potentially minimised.

Since there exists an erasure probability δ that the receiver will fail to receive the message symbols, the erasure code will be used as a technique to control these failures during transmission over the unreliable channel. The central idea is that the sender will encode the message symbols with redundancy in such a way that when the receiver wants to retrieve the original message, it can be recovered from a subset of the total encoded symbols. The redundancy, which is also known as overhead symbol, minimises the number of errors during transmission and increases the decoding probability.

Generally, there are two types of erasure codes. In the following subsections, fixed-rate erasure codes and rateless erasure codes will be discussed accordingly.

1.3 Fixed-rate Erasure Codes

In the fixed-rate erasure codes, the code rate (k / n), which is the proportion of number of message symbols (k) to the total number of encoded symbols (n), has to be pre-determined before any transmission. One of the examples of fixed-rate erasure codes is Reed-Solomon code.

1.3.1 Reed-Solomon Codes

Reed-Solomon codes are fixed-rate erasure codes. The encoding and decoding processes are done in domain GF(2^m), where GF is the Galois Field (finite field) and m is the positive integer that constraint to $m \leq 8$, and the overall presentation indicates its total elements in that finite field. A (n, k) Reed-Solomon code is further constraint to $n \leq 2^m$, where n - k is the number of redundancies.

The algorithm for these codes requires a Vandermonde matrix of n rows and k columns having a property that every row is linearly independent so that any combination of exactly k rows of matrix can be inverted, and then to recover the original data symbols. The general Vandermonde Matrix is shown in Figure 1.3.

$$\begin{bmatrix} 0^0 & 0^1 & 0^2 & \cdots & 0^{(k-1)} \\ 1^0 & 1^1 & 1^2 & \cdots & 1^{(k-1)} \\ 2^0 & 2^1 & 2^2 & \cdots & 2^{(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (2^m-1)^0 & (2^m-1)^1 & (2^m-1)^2 & \cdots & (2^m-1)^{(k-1)} \end{bmatrix}$$

Figure 1.3: General Vandermonde Matrix

For instance, the Vandermonde matrix in GF(2^3) with dimension 8×3 has the following form, shown in Figure 1.4.

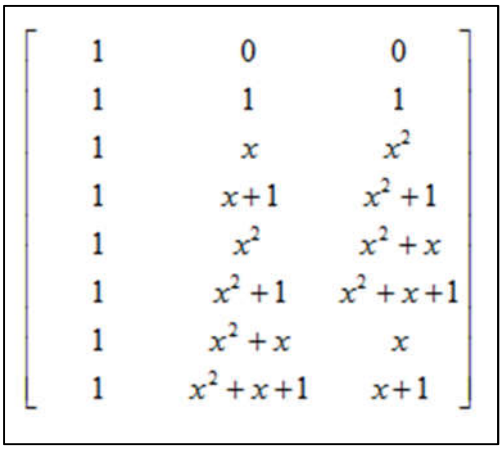

Figure 1.4: The Vandermonde Matrix in GF(2^3)

A (n, k) Reed-Solomon code over a size of 2^m ensures that if any k of the n transmitted encoded symbols is received by the receiver via a noisy channel, the original message symbols can be retrieved, but it is only practical for small k, n and m due to encoding and decoding implementations in Reed-Solomon code have a cost of order k (n - k) $\log_2 n$ packet operations. Erasure probability δ and code rate also need to be pre-determined before transmission. If δ becomes larger than the pre-determined one, the encoded symbols received by the receiver will be less than k, causing the decoder to fail instantly as extra encoded symbols cannot be generated on the fly.

There is a better way to overcome this situation, which is by using the rateless erasure code.

1.4 Rateless Erasure Codes

Rateless erasure code, which is also known as fountain code, is a coding method that limitless encoded symbols can be generated from the message symbols such that the

message can be recovered from any subset of the encoded symbols at the receiver side, provided that the size of the subset is slightly larger than the message size *k*.

From its name, the encoder of the fountain code can be metaphorically said as a fountain that sprays the water continuously and if the number of water droplets collected from this fountain is slightly larger than *k*, the message can be recovered. It is so-called rateless due to the fact that it does not have a fixed code rate. Examples of rateless erasure code are Luby Transform (LT) code and Raptor code. The Raptor code, which is the extended version of LT code, shows high performance with minimal overhead symbols, supports long and short messages and can be generated on the fly. These high performance characteristics are very essential for rateless erasure codes, and the Random codes have the potential to achieve them.

1.4.1 Random Codes

Random code is a rateless erasure code that uses a random generator to create a matrix with distributed binary values. It is the same as any other rateless erasure code that given the message symbols sized *k*, it is then encoded into potentially limitless of encoded symbols. Any extra encoded symbols requested by the decoder can also be generated on the fly. Whenever $k+10$ encoded symbols are received, high probability of complete decoding i.e. 99.9% can be achieved, which means that the encoded symbols can be successfully decoded into original message symbols with high chance.

1.4.2 Systematic Random Codes

Systematic Random codes are modified version of Random codes, where *k* message symbols are embedded as part of the encoded symbols. The receiver will first receive the message symbols sized *k* as the first part of the encoded symbols, and $(k+1)$th onwards later. If the receiver receives the first part of the encoded symbols without

loss, then the original message can be reconstructed instantly. Otherwise, any encoded symbol from ($k+1$)th onwards, which is generated from the random generator, will be sent to the receiver one by one to complete the decoding process of the message.

1.5 Objectives

In this project, the objectives as follows will be carried out.

- To study the finite field for a better understanding in the arithmetic operations
- To increase the order of the finite field so as to reduce the overhead symbols
- To rearrange the Random code into systematic Random code and contrast with the Random code
- To make comparison between Random code, systematic Random code and Reed-Solomon code

1.6 Contribution

The Random code is implemented in GF(2). It is able to produce a high probability of complete decoding when a total number of encoded symbols equals to k+10 is received by the decoder. No matter short or long messages, as long as 10 overhead symbols are added to the message symbols, the original message symbols can be recovered with high probability. But for a short message symbols of length $k = 10$, the receiver has to get 20 encoded symbols which is the double the size of the original message symbols in order to recover the message with high probability. It is considered as inefficient for the receiver as its memory capacity has to be double sized up.

So, in order to minimise the overhead symbols while maintaining a high probability of complete decoding, the order of the finite field can be increased. This

is because the decoding process of the Random code requires the encoded symbols to be independent among themselves. When the field is expanded, the chance of independency increases as every element in every row of the generator matrix is generated from a wider range of elements in the expanded field.

If the expanded finite field used in Random code can improve the decoding probability, then is there any method that can produce a decoding probability of 100% when no loss occurs? Yes, the Random code can be modified in such a way that the first part of the random generator matrix is an identity matrix (size depending on the message size), and the second part of the matrix is its ordinary random matrix. Hence the first part of the encoded symbols will be exactly the same as the message symbols and if no loss occurs, these message symbols can be recovered intact and instantly.

1.7 Outline

In Chapter 2, literature review will be discussed. The evolution of the erasure codes from Reed-Solomon codes to RaptorQ codes and Random codes will be reviewed.

In Chapter 3, introduction to Galois field and the arithmetic operations within the field are explained. General idea of the transmission model is also carried out and followed by the usage of Random codes in transmission model. Improvement to the binary Random codes is also made which is illustrated in matrix operations.

Then, numerical results in graph forms are shown in Chapter 4. Effects of increment of overhead symbols and expansion of finite field to the probability of complete decoding are simulated. The length of the message symbols is also tested in GF(2) and GF(256). Extra encoded symbols needed to complete the decoding process are also found out for different finite fields. Properties of systematic Random codes are observed from the simulation results and comparisons to ordinary Random codes are made.

Finally, conclusion is made with recommendations in Chapter 5, followed by references.

CHAPTER 2

LITERATURE REVIEW

2.1 Review on Erasure Codes

In the transmission of data over the networks, particularly through the Internet which is one of the common erasure channel nowadays, losses of data packets will everlastingly occur. A standard solution is to request for the retransmission of data which is not received, while retaining those which have successfully received. If data is lost again during these retransmissions, another request is made, and so on. Luby, Mitzenmacher, Shokrollahi, Spielman and Stemann (1998) stated that this solution produces unacceptable delays resulted from recurring communication between sender and receiver in real-time transmission.

Thus, erasure codes are indeed necessary to dismiss the retransmissions of lost data. The encoded symbols (n) are generated from the data symbols (k) so that subsets of encoded symbols can successfully recover the data symbols, where $n > k$. The very well-known Reed-Solomon code which was invented by Reed and Solomon (1960), showed that it is able to recover the data symbols as long as the number of encoded symbols received is k out of n encoded symbols. The work is further agreed by Sklar (2002), MacKay (2005) and Chong (2015). It is capable of correcting $n - k$ erasures within the block, and can be designed to have any redundancy. But according to Luby (2002) and MacKay (2005), Reed-Solomon code is only efficient for relative small settings of k and n, which are constrained to $0 < k < n \leq 255$ and the code rate (k / n) must be fixed before transmission. MacKay (2005) further mentioned that the erasure probability δ must also be pre-estimated.

If δ is happened to be larger than estimated, the receiver will receive lesser than k symbols and consequentially result in instant decoder failure as Reed-Solomon code cannot be extended on the fly due to fixed code rate. Although it does not require any overhead symbol to recover the data symbols, it does not support data length that is longer than 256.

In terms of efficiency of encoding and decoding operations, Byers, Luby, Mitzenmacher and Rege (1998) stated that Tornado code is much faster to generate and fix erasures in shorter time compared to Reed-Solomon code. However, it requires slightly more than k encoded symbols in order to reconstruct the data symbols. Luby (2002) further emphasised that Tornado codes analysis requires a technique that will lead to a reception overhead symbol which is inherently at least a constant fraction of the message length. Furthermore, Tornado code is also one of the fixed-rate erasure codes. Extra encoded symbols are prohibited from generating on the fly when the decoder demands.

Luby Transform (LT) codes which are the first realisation of rateless erasure codes, were introduced by Michael Luby. According to him, they are very efficient when the number of data symbols becomes larger as the reception overhead symbols are an asymptotically vanishing fraction of the data length, but at a cost of slightly higher asymptotic encoding and decoding times. The memory of the LT codes encoder and decoder is proportional to the data length and the decoding time only depends on the data length, irrespective of how many encoded symbols are received. Khisti (2003) had agreed to this and further concluded that it has a higher complexity than Tornado codes.

In contrast, Shokrollahi (2006) introduced Raptor codes which are an extended version of LT codes, that claimed to be with linear time encoding and decoding. Both GF(2) and GF(256) are used in the codes in such a way that the vast majority of symbol operations are over GF(2), and only a small part of the operations are in GF(256) as computation in GF(256) is much more expensive than GF(2) that uses simple XOR operations. They have also shown excellent recovery properties for all range of data symbols lengths. With latest generation of Raptor codes, the RaptorQ codes, according to the technical review of Qualcomm, they are able to

produce 99% of success decoding probability from k reception of encoded symbols, 99.99% with 1 overhead symbols and 99.9999% with only 2 overhead symbols.

While for Random codes, according to Chong (2015), it is able to achieve high probability of complete decoding i.e. 99.9% with $k + 10$ encoded symbols. This is the case for Random code in GF(2). With further works and researches, Random codes have the capability to achieve the performance results similar to RaptorQ codes.

CHAPTER 3

METHODOLOGY

3.1 Galois Field

Galois field or a finite field is a field with a finite number of elements in it. Any operation such as addition, subtraction, multiplication, or division which is defined and done within the Galois field is satisfied to certain basic rules, which is totally different from the standard arithmetic operations. There will be limited number of elements in the field, so any operation performed in the finite field will result in an element within that field also. If a Galois field has q elements, it is denoted as GF(q), in which it is called order q. The field exists if and only if the order q is a prime power.

3.1.1 GF(2)

The smallest useful finite field is GF(2), which consists of two elements and it is usually 0 and 1. The arithmetic operations in GF(2) have some unique properties:

- Addition operation satisfies logical XOR operation
- Multiplication operation satisfies logical AND operation
- Subtraction operation: Since every element x in GF(2) satisfies $x + x = 0$, therefore $-x = x$

- Division operation satisfies identity function due to the only invertible element in GF(2) is 1

In Table 3.1, the addition and multiplication in GF(2) are shown, which are the simplest operations among the finite fields.

Table 3.1: Addition and Multiplication in GF(2)

+	0	1
0	0	1
1	1	0

x	0	1
0	0	0
1	0	1

GF(2) is convenient to represent data in modern computers because they are using binary digits in their very basic language. That is why GF(3), GF(5) and so on are not recommended here. Furthermore, GF(2) can be expanded to arbitrarily large fields by increasing the power of the order 2. Particularly, the benefit is shown in GF(256) as 8 bits can be represented in 1 byte now.

3.1.2 GF(4), GF(8), …, GF(256)

In GF(4), which consists of four elements, has the tables of addition and multiplication as in Table 3.2.

Table 3.2: Addition and Multiplication in GF(4)

+	0	1	x	x+1
0	0	1	x	x+1
1	1	0	x+1	x
x	x	x+1	0	1
x+1	x+1	x	1	0

x	0	1	x	x+1
0	0	0	0	0
1	0	1	x	x+1
x	0	x	x+1	1
x+1	0	x+1	1	x

Addition in GF(4) still follows the rules of XOR logical operation but the multiplication is a bit different now as it requires an irreducible polynomial in the steps.

Addition and multiplication tables of GF(8) are shown in Table 3.3 and Table 3.4 respectively.

Table 3.3: Addition in GF(8)

+	0	1	x	x+1	x^2	x^2+1	x^2+x	x^2+x+1
0	0	1	x	x+1	x^2	x^2+1	x^2+x	x^2+x+1
1	1	0	x+1	x	x^2+1	x^2	x^2+x+1	x^2+x
x	x	x+1	0	1	x^2+x	x^2+x+1	x^2	x^2+1
x+1	x+1	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	x+1
x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	x+1	x
x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	x+1	0	1
x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	x+1	x	1	0

Table 3.4: Multiplication in GF(8)

x	0	1	x	x+1	x^2	x^2+1	x^2+x	x^2+x+1
0	0	0	0	0	0	0	0	0
1	0	1	x	x+1	x^2	x^2+1	x^2+x	x^2+x+1
x	0	x	x^2	x^2+x	x+1	1	x^2+x+1	x^2+1
x+1	0	x+1	x^2+x	x^2+1	x^2+x+1	x^2	1	x
x^2	0	x^2	x+1	x^2+x+1	x^2+x	x	x^2+1	1
x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	x+1	x^2+x
x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	x+1	x	x^2
x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	x+1

From the advancement of GF(2) to GF(8), the tables are expanding. So when it advances to GF(256), the tables will be large that contain 256 elements. Although the addition is a fast operation that using only XOR to operate, the multiplication will be getting more tedious.

Since ASCII is the most commonly used format for text files in computer and on the Internet, GF(256) is the most suitable finite field to represent these ASCII codes. The codes are shown in Table 3.5.

Table 3.5: The Extended ASCII Codes

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	Null	37	%	74	J	111	o	148	ö	185	╣	222	
1	SOH	38	&	75	K	112	p	149	ò	186	║	223	
2	STX	39	‘	76	L	113	q	150	û	187	╗	224	α
3	ETX	40	(	77	M	114	r	151	ù	188	╝	225	ß
4	EOT	41	)	78	N	115	s	152	ÿ	189	╜	226	Γ
5	ENQ	42	*	79	O	116	t	153	Ö	190	╛	227	π
6	ACK	43	+	80	P	117	u	154	Ü	191	┐	228	Σ
7	BEL	44	,	81	Q	118	v	155	¢	192	└	229	σ
8	BS	45	-	82	R	119	w	156	£	193	┴	230	µ
9	TAB	46	.	83	S	120	x	157	¥	194	┬	231	τ
10	LF	47	/	84	T	121	y	158		195	├	232	Φ
11	VT	48	0	85	U	122	z	159	ƒ	196	─	233	Θ
12	FF	49	1	86	V	123	{	160	á	197	┼	234	Ω
13	CR	50	2	87	W	124	|	161	í	198	╞	235	δ
14	SO	51	3	88	X	125	}	162	ó	199	╟	236	∞
15	SI	52	4	89	Y	126	~	163	ú	200	╚	237	φ
16	DLE	53	5	90	Z	127	DEL	164	ñ	201	╔	238	ε
17	DC1	54	6	91	[	128	Ç	165	Ñ	202	╩	239	∩
18	DC2	55	7	92	\	129	ü	166	ª	203	╦	240	≡
19	DC3	56	8	93	]	130	é	167	º	204	╠	241	±
20	DC4	57	9	94	^	131	â	168	¿	205	═	242	≥
21	NAK	58	:	95	_	132	ä	169		206	╬	243	≤
22	SYN	59	;	96	`	133	à	170	¬	207	╧	244	
23	ETB	60	<	97	a	134	å	171	½	208	╨	245	
24	CAN	61	=	98	b	135	ç	172	¼	209	╤	246	÷
25	EM	62	>	99	C	136	ê	173	¡	210	╥	247	≈
26	SUB	63	?	100	d	137	ë	174	«	211	╙	248	°
27	ESC	64	@	101	e	138	è	175	»	212	╘	249	
28	FS	65	A	102	f	139	ï	176		213	╒	250	·
29	GS	66	B	103	g	140	î	177	▒	214	╓	251	√
30	RS	67	C	104	h	141	ì	178	▓	215	╫	252	ⁿ
31	US	68	D	105	i	142	Ä	179	┃	216	╪	253	²
32	Space	69	E	106	j	143	Å	180	┤	217	┘	254	■
33	!	70	F	107	k	144	É	181	╡	218	┌	255	
34	“	71	G	108	l	145	æ	182	╢	219	█		
35	#	72	H	109	m	146	Æ	183	╖	220	▄		
36	$	73	I	110	n	147	ô	184	╕	221	▌		

3.1.3 Addition in GF(256)

GF(256) follows the rules of XOR in addition operation. For example, the addition between x^6+x^4+x+1 and $x^7+x^6+x^3+x$ are shown below:

		1	0	1	0	0	1	1	(x^6+x^4+x+1)
+	1	1	0	0	1	0	1	0	$(x^7+x^6+x^3+x)$
	1	0	0	1	1	0	0	1	$(x^7+x^4+x^3+1)$

3.1.4 Multiplication in GF(256)

Multiplication in GF(256) is to be done in three steps which are shown in the algorithm below:

Step 1: Multiplication follows by XOR operation

$(x^7 + x^5 + x^3 + x)(x^5 + x^4 + x^3 + x^2 + 1)$
$= (x^{12} + x^{11} + \mathbf{x^{10}} + \mathbf{x^9} + \mathbf{x^7}) + (\mathbf{x^{10}} + \mathbf{x^9} + \mathbf{x^8} + \mathbf{x^7} + \mathbf{x^5}) + (\mathbf{x^8} + x^7 + \mathbf{x^6} + \mathbf{x^5} + \mathbf{x^3}) + (\mathbf{x^6} + x^5 + x^4 + \mathbf{x^3} + x)$
$= x^{12} + x^{11} + x^7 + x^5 + x^4 + x$
$= 1100010110010_2$

Step 2: Find the suitable irreducible polynomial for that particular field (polynomial that cannot be further factored out)

Irreducible polynomial in GF(256) = $x^8 + x^4 + x^3 + x + 1 = 100011011_2$

Step 3: Answer in Step 1 modulo irreducible polynomial

```
1 1 0 0 0 1 0 1 1 0 0 1 0   mod   1 0 0 0 1 1 0 1 1
1 0 0 0 1 1 0 1 1
-----------------------------
  1 0 0 1 0 0 0 0 0 0 1 0
  1 0 0 0 1 1 0 1 1
  -----------------------------
        1 1 1 0 1 1 0 1 0
        1 0 0 0 1 1 0 1 1
        -----------------------
          1 1 0 0 0 0 0 1
```

The remainder = $x^7 + x^6 + 1$ is the final answer for the multiplication between $(x^7 + x^5 + x^3 + x)$ and $(x^5 + x^4 + x^3 + x^2 + 1)$ in GF(256). Since it involves three steps in multiplication that intensify the decoding complexity, a lookup table is suggested for the Random code whenever GF(256) is used.

3.2 Transmission Model

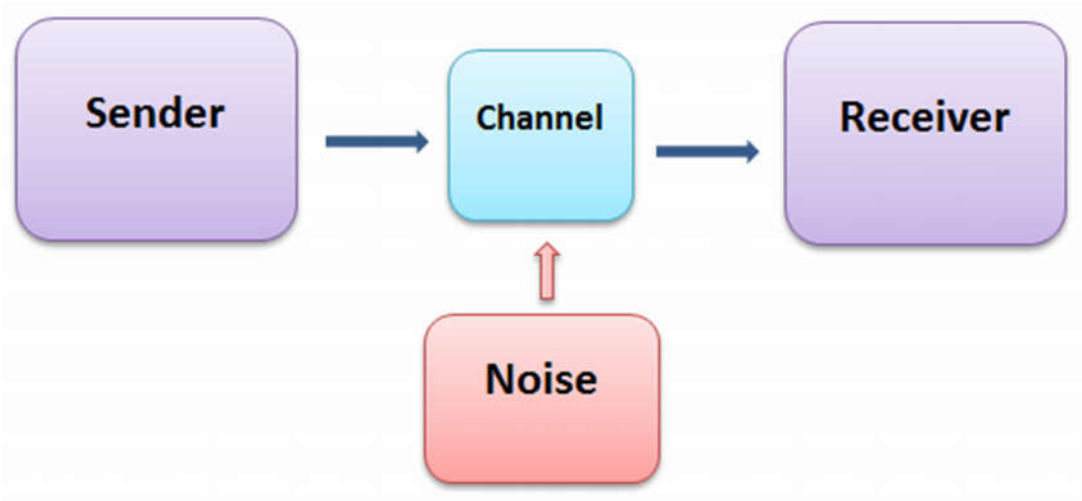

Figure 3.1: The Transmission Model

Figure 3.1 shows the transmission model. In a network, when data is being sent down to the receiver via a transmission channel, more or less it is interfered by the noise. When the noise exists in the transmission channel, it causes error to the data symbols or any of the data symbols is dropped eventually the receiver will not receive a complete original data. According to Shannon theorem which states that every

channel has a channel capacity of C, and if the transmission rate is less than C, there are some codes which are able to achieve arbitrarily low probability of decoded error. Assuming C is always larger than the transmission rate, the Random code is applied in this transmission channel.

3.3 Usage of Random Code in Transmission Model

Before the message is sent to the receiver via a transmission channel, a random generator is utilised to generate encoded symbols from the message symbols. All the calculation steps will be expressed in the matrix forms.

3.3.1 Implementation of Random Code in GF(2)

An example is given in the following to ease the explanation of the steps.

$$G \times M = X$$

$$\Downarrow$$

Step (1): Multiplication

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\Downarrow$$

$$[G \ \vdots \ X]$$

$$\Downarrow$$

Step (2): Augmentation

$$\begin{bmatrix} 0 & 0 & 1 & \vdots & 1 \\ 0 & 1 & 0 & \vdots & 0 \\ 1 & 0 & 1 & & 0 \end{bmatrix}$$

$$\Downarrow$$

Step (3): Gaussian Elimination $\left[\begin{array}{ccc:c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right]$

$$\Downarrow$$

$$[I \ \vdots \ M]$$

where,

G = Random generator symbol matrix

M = Message symbol matrix

X = Encoded symbol matrix

I = Identity matrix

This illustration is computed in GF(2). In Step (1), a random generator matrix is multiplied with the message matrix to create the encoded symbols matrix. Then, the random generator matrix is augmented with the encoded symbols matrix in Step (2). Finally, this augmented matrix will be sent through the transmission channel to the receiver to be decoded where Gaussian elimination will be done there as in Step (3). The final matrix will be in row-echelon form, where an identity matrix is augmented with the original message matrix, but there are cases where identity matrix cannot be formed which lead to decoding failure.

Although sometimes the message symbols are formed as the output after the Gaussian elimination, it is considered as decoder failure as the decoder could not get the confirmation whether the message is correct or wrong. The confirmation can be done through the existance of the identity matrix.

3.4 Improvement for Random codes

When the decoder fails, the message cannot be recovered. So there must be some ways to assist this Random code so that it is able to achieve high probability of success decoding.

3.4.1 The Increment of Redundancies

Redundancies or overhead symbols are added to the code to improve the probability of complete decoding. Let the dimension of the random generator matrix be $n \times k$ and that of the message matrix be $k \times l$, provided that $n \geq k$. Let $n = 7$, $k = 3$, and $l = 1$. Now, n is 4 more than k, and the number of overhead symbols will be 4. The example is shown in the following.

$$G \times M = X$$

$$\Downarrow$$

Step (1): Multiplication

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Downarrow$$

$$[G \ \vdots \ X]$$

$$\Downarrow$$

Step (2): Augmentation

$$\left[\begin{array}{ccc:c} 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

$$\Downarrow$$

Step (3): Gaussian Elimination

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

The probability of complete decoding is higher than the previous example which using the random generator with dimension of $k \times k$, as shown in the simulation results in the next chapter. This is because the number of encoded symbols as in Step (2) is 4 more than the message symbols ($k = 3$), so the decoder now has 4 more choices to choose for the decoding. Thus, probability increases.

3.4.2 The Expansion of the Finite Field

Another method used to improve the Random code is to expand the finite field. Once the number of elements in the finite field is increased, more bits or more information can be represented in a single symbol. As shown in Figure 3.2, given a lengthy message in binary form, each bit of the message has to be represented by 1 GF(2) symbol because the highest value that can be represented by the element in GF(2) is 1. When the field is increased to GF(4) as shown in Figure 3.2, 2 bits from the message can be represented by 1 GF(4) symbol, thus reducing the total number of symbols needed to represent the whole message.

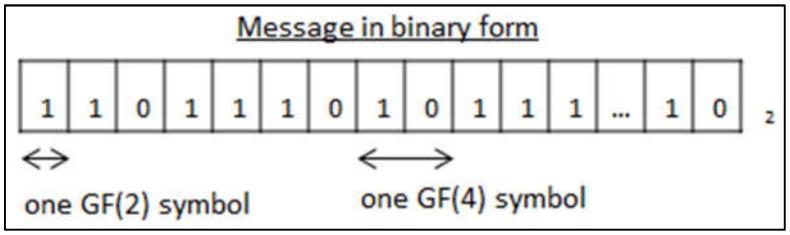

Figure 3.2: Message Symbols to be Represented by GF(2) and GF(4) Symbols

Other than that, there is another significant improvement when higher finite field is used in the Random code. Since the decoding part of the Random code is done by Gaussian elimination, and reducing to the row echelon form, it is crucial to make the row i.e. symbol to be linear independent to others. When the number of elements is increased, the random generator will have wider range of elements to generate from. Thus the row of symbols after the augmentation will be more different from the other rows in terms of probability, resulting it to have higher chance to be one of the independent rows. Put it in other words, the symbols in the row of matrix after the augmentation will hardly be the same as in another row with the same column when the number of elements increases.

$$G \times M = X$$

$$\Downarrow$$

Step (1): Multiplication $\begin{bmatrix} x & x+1 & x \\ 0 & x+1 & x \\ x+1 & x & x+1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ x \\ 0 \end{bmatrix}$

$$\Downarrow$$

$$[G \ \vdots \ X]$$

$$\Downarrow$$

Step (2): Augmentation $\left[\begin{array}{ccc:c} x & x+1 & x & 0 \\ 0 & x+1 & x & x \\ x+1 & x & x+1 & 0 \end{array}\right]$

$$\Downarrow$$

Step (3): Gaussian Elimination $\left[\begin{array}{ccc:c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right]$

The example shown above uses GF(4) random generator. The rows of the random matrix are now having higher chance to be independent among themselves due to dispersion of the elements in GF(4) as compared to the one that uses GF(2).

3.5 Systematic Random Code in Transmission Model

The systematic Random code is also one of the rateless erasure code based on Random code. The random generator matrix is transformed into identity matrix so that it is able to produce the original message symbols as the encoded symbols.

$$G \times M = X$$

$$\Downarrow$$

Step (1): Multiplication

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$\Downarrow$$

$$[G \ \vdots \ X]$$

$$\Downarrow$$

Step (2): Augmentation

$$\begin{bmatrix} 1 & 0 & 0 & & 1 \\ 0 & 1 & 0 & \vdots & 0 \\ 0 & 0 & 1 & & 1 \end{bmatrix}$$

When the random matrix, which is now an identity matrix, is multiplied with the message matrix, the encoded symbols matrix will be the same as the message matrix. So after the augmentation and being sent to the decoder, the matrix is already in row echelon form and does not need to undergo Gaussian elimination. One step is waived and the original message symbols can be recovered intact.

What if there exists noises in the transmission channel and causes interference to the encoded symbols? In systematic Random code, the random generator matrix can be extended in such a way that the random matrix is stacked below the identity matrix so that extra encoded symbols can be generated from this random matrix.

CHAPTER 4

NUMERICAL RESULTS

4.1 Effects of Overhead Symbols Increment to Complete Decoding Probability

In the simulation, the message symbols matrix with dimension of $k \times l$ is set to 10×1 so that it contains 10 symbols with size 1 bit. A random generator matrix is created with dimension $k \times k$ i.e. 10×10. The random generator matrix is multiplied with the message symbols matrix to produce an encoded symbols matrix. The random generator matrix is then augmented with the encoded symbols matrix, and their rank is found. If the rank equals to the message length i.e. 10, then the counter will be incremented by 1 (initial = 0), and that means the message can be recovered. The steps are repeating for 10000 times so that the average probability of complete decoding can be obtained.

Then, extra 1 row (overhead symbol) is attached to the random generator matrix so that the dimension becomes 11×10, and the remaining steps are the same as above. The extra row is increased 1 by 1 until 10, and the result is plotted in Figure 4.1.

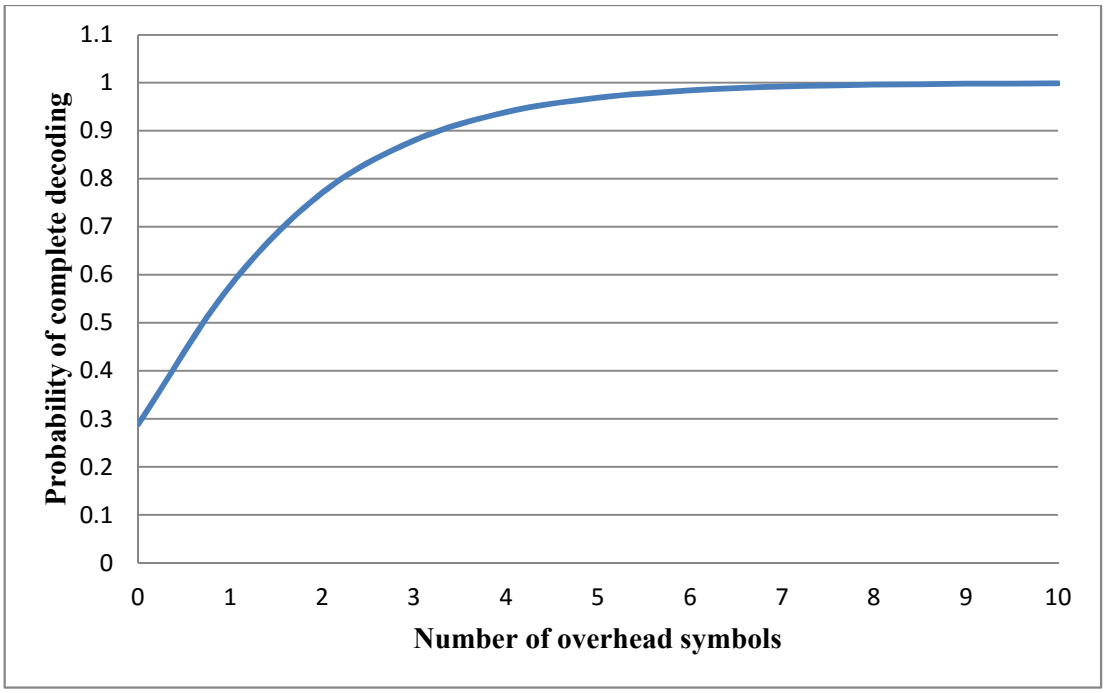

Figure 4.1: Probability of Complete Decoding against Number of Overhead Symbols for GF(2)

From Figure 4.1, when no overhead symbol is applied to the encoder, the probability of complete decoding is approximately 0.3 only. When the number of overhead symbol increases, the probability of complete decoding also increases. When the overhead symbols equals to 10, the probability is already very close to 1 i.e. 0.999. From this point, it can be concluded that Random code in GF(2) is able to achieve high probability of complete decoding with $k+10$ encoded symbols.

4.2 Effects of Finite Field Expansion to Complete Decoding Probability

For this simulation, no overhead symbol will be used. The dimensions of the message symbols matrix and the random generator matrix are fixed to 10×1 and 10×10 respectively. The order of the finite field is increased with powers of 2, starting from GF(2) until GF(256). Each order is repeated for 10000 times so that the average probability of complete decoding can be obtained. The result is shown in Figure 4.2.

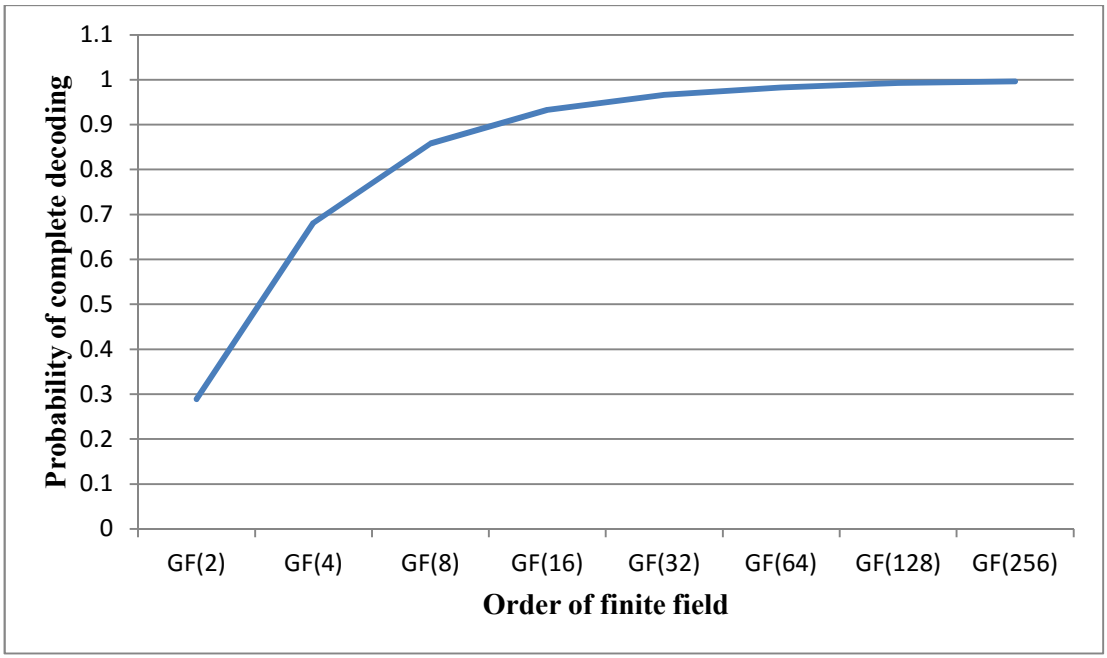

Figure 4.2: Probability of Complete Decoding against Order of Finite Field

From Figure 4.2, when the order of Galois field is increased from 2 to 256, the probability of complete decoding increases gradually. Random code is able to achieve the probability of 0.9965 in GF(256), without any overhead symbols.

4.3 Effects from the Combination of Overhead Symbols Increment and Finite Field Expansion

Continue from previous simulation, each order of the finite field is run with increasing overhead symbols from 0 to 10. As in Figure 4.3, only 4 finite fields are particularly chosen to show the distinction between them.

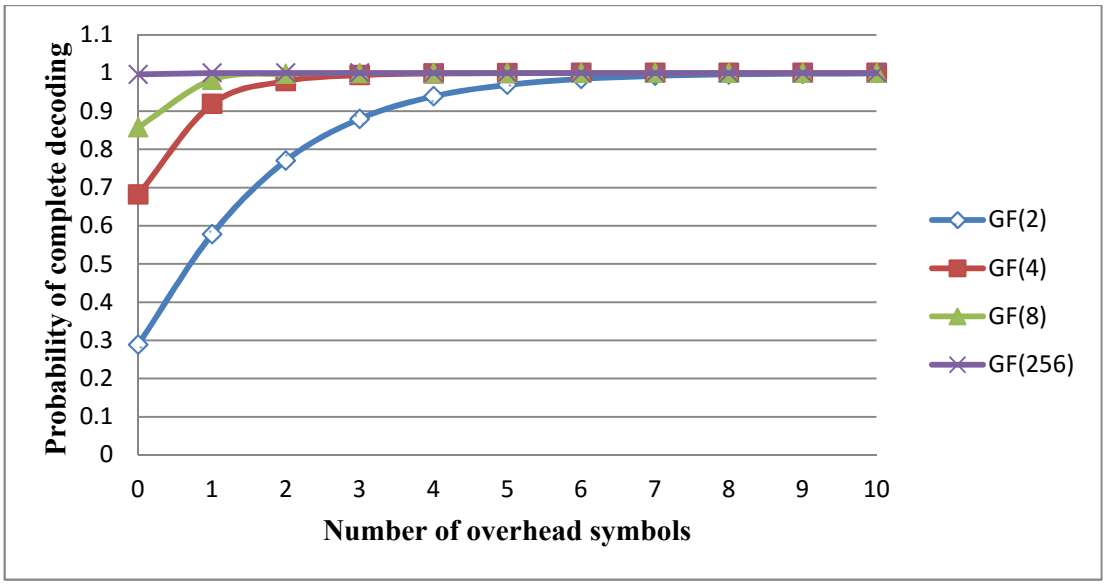

Figure 4.3: Probability of Complete Decoding against Number of Overhead Symbols for 4 Different Finite Fields

In Figure 4.3, it shows the probability of complete decoding against the number of overhead symbols for 4 different Galois fields, namely GF(2), GF(4), GF(8) and GF(256). It has been tested for 100000 times that Random code in GF(256) with 10 overhead symbols is able to decode successfully for 100%. But this is not really necessary as with $k+1$ encoded symbols it is already able to decode with 0.99997 probability.

4.4 Effects of Finite Field Expansion to the Extra Encoded Needed for Complete Decoding

First, $k \times k$ matrix in GF(2) is created and packet loss in proportion to k is introduced to the matrix. If the rank of the matrix is less than k (message fails to be decoded), extra row of random encoded symbols is added to the matrix and the rank is found again and so on until rank equals to k is achieved. Note that the extra encoded symbols (rows) are counted after the number of rows is more than k. For instance, if $k = 50$ and packet loss probability is 0.3, that means 15 rows will be lost in the transmission, receives only 35 rows. Now the encoder will be sending extra symbols to the decoder so that rank of 50 can be achieved to indicate the complete decoding.

If, in average, extra encoded symbols sent are 16.6, adding back the attained 35 symbols just now, it will become 51.6. Hence, it requires $k + 1.6$ encoded symbols. This 1.6 is the extra encoded symbols that will be recorded. The finite field is then expanded to GF(4), GF(8), …, and GF(256) Random codes and the results are shown in Figure 4.4.

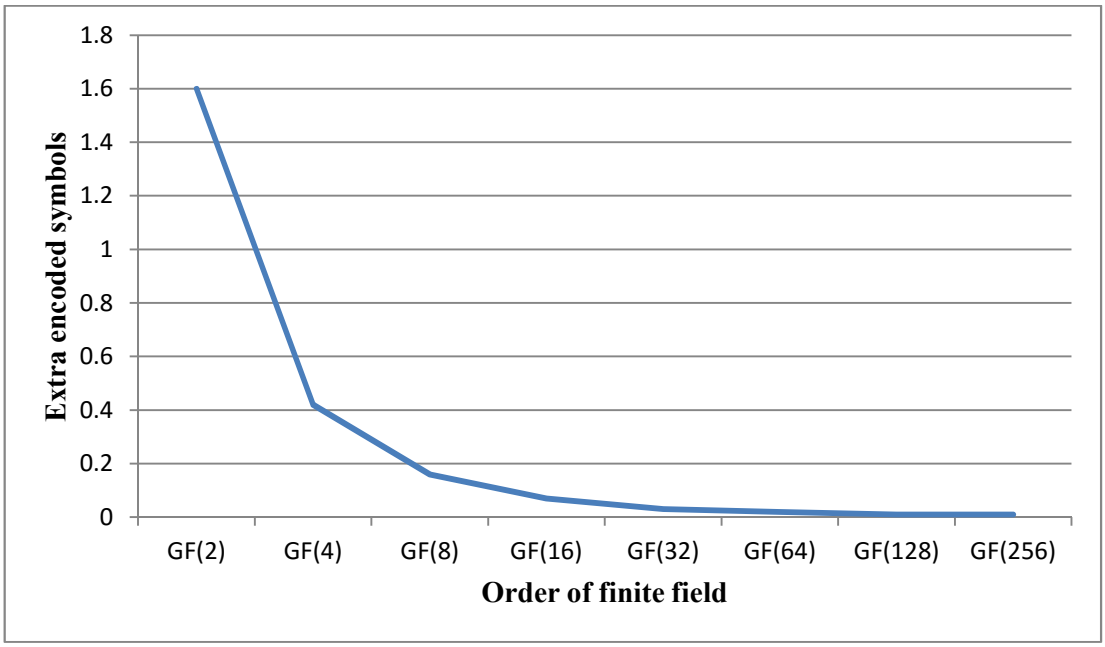

Figure 4.4: Extra Encoded Symbols Needed against Order of Finite Field

If the finite field is expanded from GF(2) to GF(256), the number of extra encoded symbols needed to achieve rank k become smaller. If GF(2) is used in Random code, it needs 1.6 extra encoded symbols in average to complete the decoding, whilst if GF(256) is used, it rarely needs any extra encoded symbols due to the fact that it has high independency results from the dispersion of elements that occurs randomly in the symbols.

4.4.1 Further Proof for 1.6 Extra Encoded Symbols

To prove for average 1.6 extra encoded symbols from Figure 4.4, frequency of the overhead symbols to achieve complete decoding is recorded. The simulation is run for 100000 times and the results are shown in Figure 4.5.

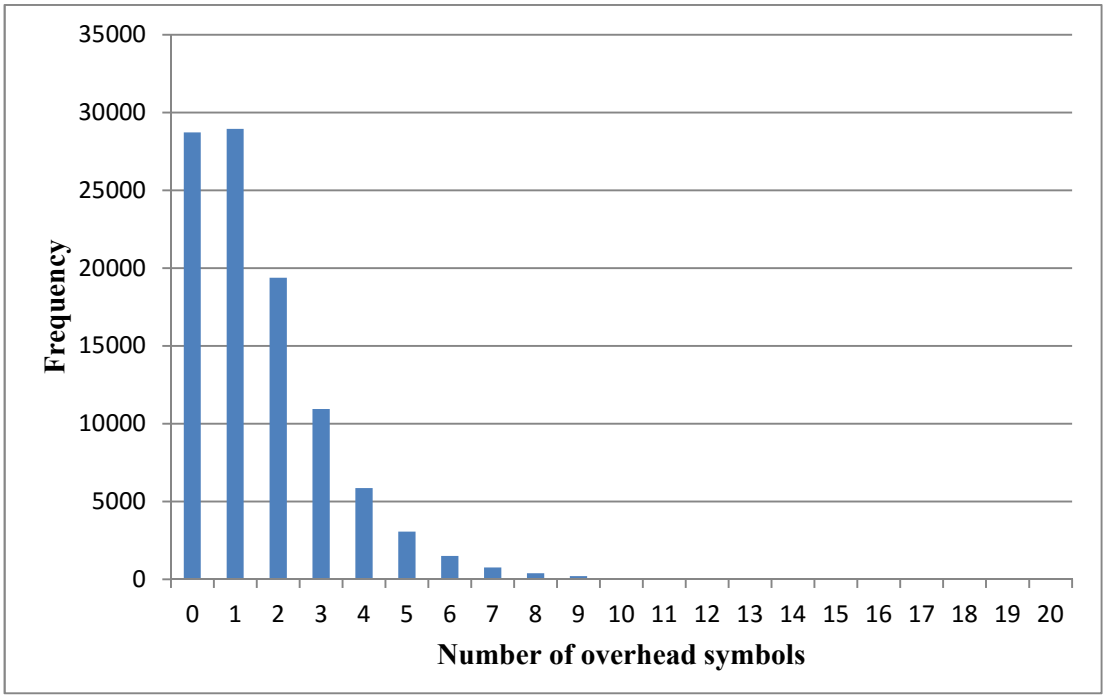

Figure 4.5: Frequency against Number of Overhead Symbols in GF(2)

From Figure 4.5, the numbers of overhead symbols of 0 and 1 are of high frequencies and then the frequency of number of overhead symbols decreases exponentially. So when average number of overhead symbols is counted, it is found to be 1.6 approximately.

4.5 Test for Data Length in GF(2) and GF(256)

The simulation steps from Section 4.4 are continued to test for message lengths of k = 10, 50, 100, 200 and 500 in GF(2) Random codes, with packet loss probability 0 to 0.9, increment by 0.1. These steps are applied to simulations in GF(256) Random codes also. Results are shown in Figure 4.6.

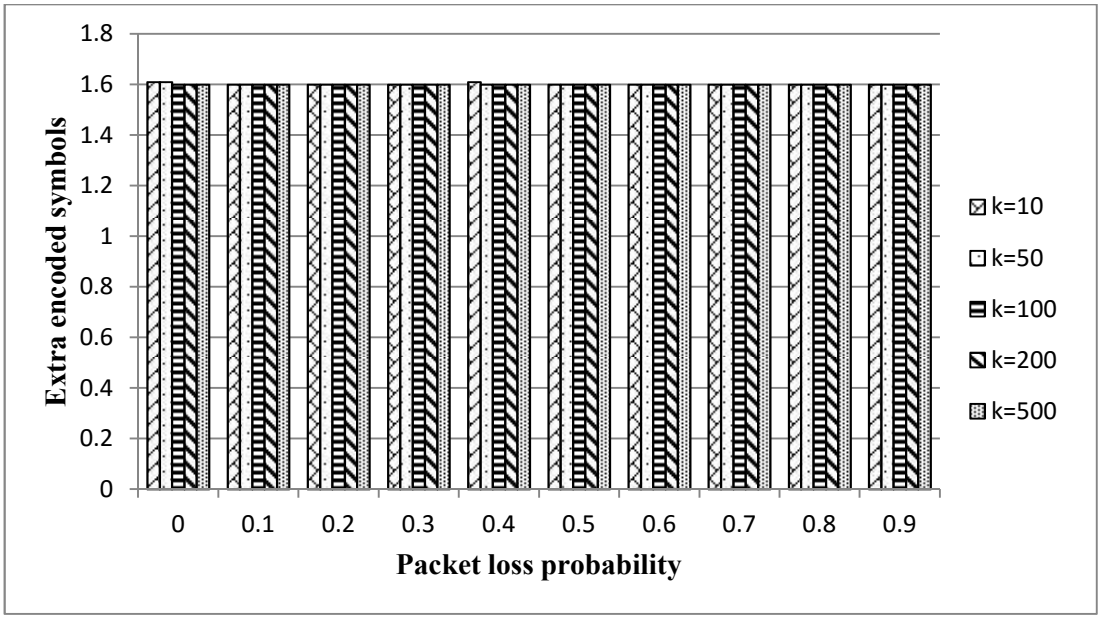

Figure 4.6: Extra Encoded Symbols Needed When Packet Loss Probability Increases for Different Data Length in GF(2)

From Figure 4.6, for GF(2) Random codes, no matter what the packet loss probability is, the decoder will always need extra encoded symbols of about 1.6 irrespective of message lengths. While the simulation steps are done for GF(256) Random codes, every data length shows the same results. For 100000 trials, GF(256) Random codes with any data length very rarely need any extra encoded symbols to decode completely.

4.6 Comparison between Systematic Random Code and Random Code

A systematic Random code is created to compare with the ordinary Random code. An identity matrix is generated and multiplied with the message matrix to produce the encoded matrix, and its rank is found and compared with *k*. If they are the same, no extra encoded symbol is needed. If they are not, encoded symbol will be added and the rank is found again until it is the same with *k*. It is simulated in GF(2), GF(4), … and GF(256) for increasing packet loss probability from 0 to 0.9.

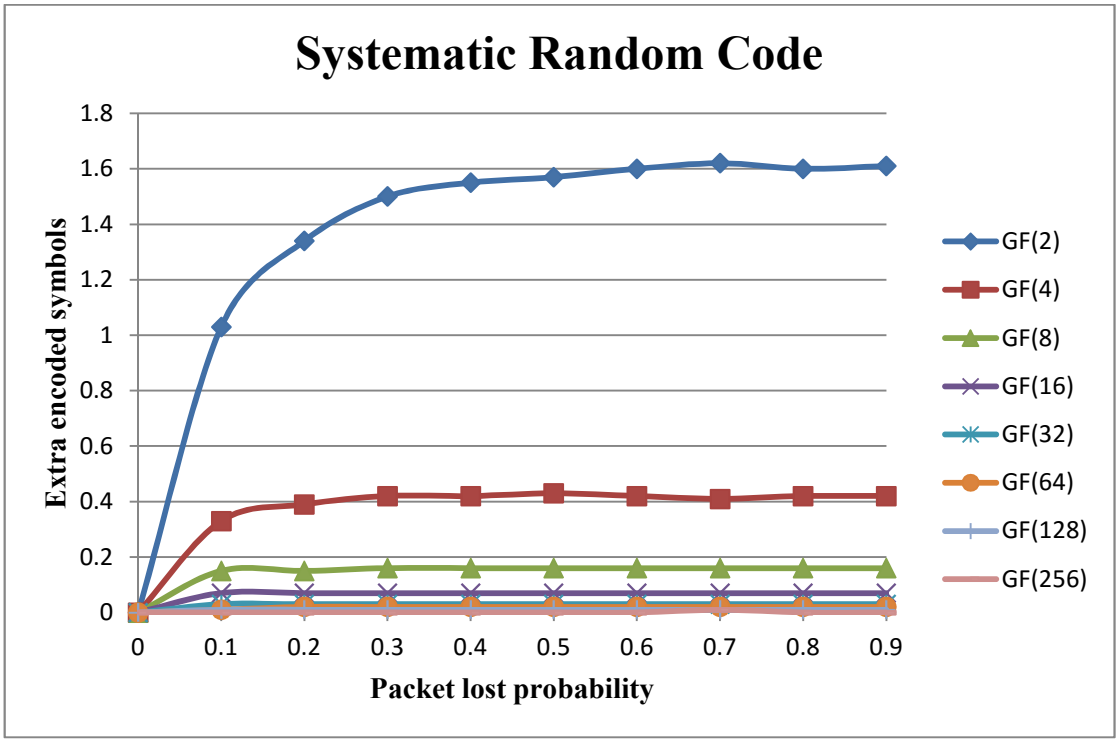

Figure 4.7: Extra Encoded Symbols Needed to Achieve Full Rank Matrix in Systematic Random Code against Packet Lost Probability

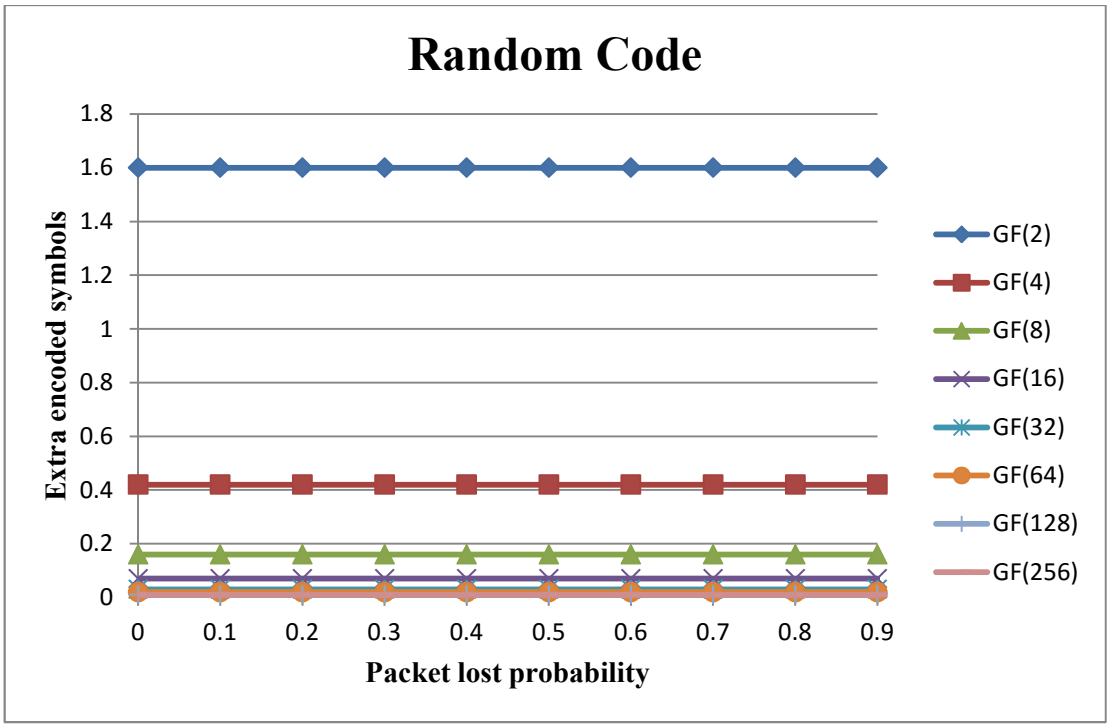

Figure 4.8: Extra Encoded Symbols Needed to Achieve Full Rank Matrix in Random Code against Packet Lost Probability

From Figure 4.7, it is common to all finite fields that systematic Random codes will be able to decode without any extra encoded symbols when totally no packet loss occurs. Comparing between Figure 4.7 and Figure 4.8, it is more advantageous to use GF(2), GF(4) and GF(8) in systematic Random code than normal Random code because lesser extra encoded symbols are needed when packet lost probability is low. When the packet lost probability further increases, it tends to require the same amount of encoded symbols as the Random code. If the finite field is expanded to 256 elements, both codes show similar results, which very minimal extra encoded symbols are needed throughout the packet lost probability.

CHAPTER 5

CONCLUSION

5.1 Summary

Random codes which are implemented in GF(2) are able to achieve high probability i.e. 99.9% of complete decoding with $k+10$ encoded symbols. If the encoded symbols have to be minimised, the order of the finite field can be increased, particularly to GF(256) as it is able to produce 99.65% probability of complete decoding without any overhead symbol. With $k+1$ encoded symbols received, it decodes with 99.997% success probability. In average, 1.6 extra encoded symbols are needed for GF(2) Random codes. But if the field of Random codes is expanded to GF(256), the GF(256) Random codes rarely need any overhead symbols for decoding. Either for short or long messages, the Random codes are able to maintain similar results of high decoding probability. To improve the code, they are modified into systematic Random codes. Without loss introduced to the transmission channel, it will decode with 100% probability.

5.2 Recommendations

To further improve the probability of complete decoding for GF(256) Random codes, micro symbols can be introduced into the codes, albeit this will intensify the computational complexity as a trade-off. Each message symbols is decomposed into smaller symbols of equal size and these micro symbols will be fed into encoder to

generate infinite encoded micro symbols. Then the encoded micro symbols are regrouped before sending to the receiver.

Computation in GF(256) takes relative long time mainly due to the multiplication operations that take several steps before answer. The lookup table of GF(256) is suggested to be built into the encoder and decoder so that it will only take 1 step to look for the answer of the multiplication between 2 elements in the finite field.

REFERENCES

Chong, Z., Goi, B., Ng, B., Ohsaki, H. and Ewe, H. (2015). Probability of complete decoding of random codes for short messages. *Electronics Letters*, 51(3), pp.251-253.

Chong, Z., Bryan Ng, C., Goi, B., Ewe, H. and Ohsaki, H. (2015). Improving the probability of complete decoding of random code by trading-off computational complexity. *IET Communications*, 9(18), pp.2281-2286.

Chong, Z., Goi, B., Ohsaki, H., Ng, B. and Ewe, H. (2015). Systematic rateless erasure code for short messages transmission. *Computers & Electrical Engineering*, 45, pp.55-67.

Khisti, A. (2003). *Tornado codes and Luby Transform codes*. [online] Available at: http://web.mit.edu/6.454/www/www_fall_2003/khisti/tor_summary.pdf [Accessed 7 Feb. 2016].

Luby, M. (2002). LT codes. *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp.271 - 280.

MacKay, D. (2005). Fountain codes. *IEE Proceedings - Communications*, 152(6), pp.1062-1068.

Shokrollahi, A. (2006). Raptor codes. *IEEE Transactions on Information Theory*, 52(6), pp.2551-2567.

Sklar, B. (2001). Reed-Solomon codes. *Digital Communications: Fundamentals and Applications, Second Edition*.

Westall, J. and Martin, J. (2010). *An introduction to Galois fields and Reed-Solomon coding*. [online] Available at: https://people.cs.clemson.edu/~westall/851/rs-code.pdf [Accessed 14 Jan. 2016].