國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

利用 LT 編碼增進網路通訊系統吞吐量之研究

Throughput Enhancement Using LT Codes in Erasure Network Communications

研究生：胡健甫

指導教授：張錫嘉 教授

中 華 民 國 九 十 七 年 十 二 月

利用 LT 編碼增進網路通訊系統吞吐量之研究

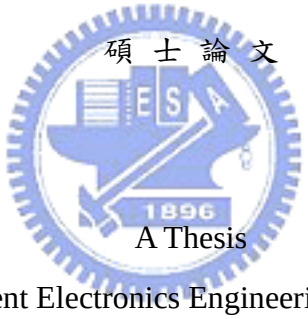
Throughput Enhancement Using LT Codes in Erasure Network Communications

研 究 生： 胡健甫　　Student：Chien-Fu Hu

指導教授： 張錫嘉　　Advisior：Hsie-Chia Chang

國 立 交 通 大 學

電 子 工 程 學 系　電 子 研 究 所 碩 士 班



碩 士 論 文

A Thesis

Submitted to Department Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

In Partial Fulfillment of the Requirements

for the Degree of Master

In

Electronics Engineering

December 2008

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 七 年 十 二 月

利用 LT 編碼增進網路通訊系統吞吐量之研究

學生：胡健甫　　指導教授：張錫嘉

國立交通大學

電子工程學系　　電子研究所碩士班

摘　　要

網路傳輸時所發生的瓶頸於接收端已造成嚴重的吞吐量降低。這篇論文中，我們介紹了一個將LT碼的編碼系統應用於傳輸環境之下來增進吞吐量。所提的LT編碼能改善吞吐量來逼近理論上的最大值。同時也能提供重要的資料保護能力來對抗於通道的封包遺失。在LT編碼系統中，相對於所有傳輸資料量，中間節點所需要的緩沖器變成不重要的因素而能大量地減少。在某些網路中，我們結合LT碼和特定的網路編碼機制來得到更進一步接收端吞吐量的改善，同時也對資料提供強大的保護。在整個傳輸過程，因編碼機制所造成的額外負擔也在一個合理的代價。

所提出的LT碼工作在封包數目從4K到64K，每個封包的大小爲1KB。提出的方法能在兩種網路之下成功的減輕瓶頸：含edge-disjoint edges（EDPs）以及不含EDPs。同時也能給予強大的資料保護在封包遺失比率從0%到20%。我們提及的LT碼能改善20%到30%的吞吐量。於包含EDPs的網路之中，結合的LT-網路編碼能提供更進一步到達約50%的吞吐量改善。最後，編碼運算是在有限場GF(2)而所需的額外負擔爲每一個碼字大約30個exclusive-or運算。

Throughput Enhancement Using LT Codes in Erasure Network Communications

Student：Chien-Fu Hu Advisor：Hsie-Chia Chang

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung Eunversity

Abstract

Bottlenecks occurring during the transmission in the network have caused serious consequences on the throughput degradation in receivers. In this thesis, a coding system that applies low degree LT codes in transmission environment to enhance the throughput is introduced. The proposed LT code can improve the throughput approach to the theoretical maximum. In the meanwhile, it provides a significant data protection ability against the packet loss in erasure channels. The required buffer size in the intermediates in the LT codes system becomes a inconsequential factor that it can be reduced considerably compared to the data size. In some network topology, we combine low degree LT codes with specified network coding mechanism to get further improvement in throughput of every receiver and also give strong protection of data. The overhead of the coding mechanism causes the reasonable computation cost in whole transmission.

For each packet of 1KB, the proposed LT codes work under the entire packet number from 4K to 64K. The proposed method alleviates the bottlenecks successfully in two kinds of topologies, with edge-disjoint paths (EDPs) and without EDPs. It also gives strong protection of data in the erasure channels with different scale of loss rate from 0% to 20%. Our proposed low degree LT code can enhance the throughput in the range from 20% to 30%. In the network with edge-disjoint paths, the combined LT-network code offers advanced improvement up to 50% or so. Finally, the computation is over GF(2) and the coding overhead is about 30 XOR operations per codeword.

誌　謝

時間過得很快，兩年多的研究生活就這樣匆匆而過。回首來時，是許多人的幫助才能讓我順利完成。首先要感謝指導我的張錫嘉老師，在研究上放心地讓我做新的嘗試，並作適時的修正。除此之外，平日也相當照顧學生，尤其在我研究上最低潮的時候，總不停地鼓勵我，十分感動。也很幸運能待在 OASIS 和 OCEAN 這氣氛良好的實驗室。謝謝建青學長教導我使用 LATEX 軟體，彥欽學姐在畢業之後還特地花費時間研讀 Network Coding 的書，充實我的理論基礎。國光不厭其煩地教導我 LT CODE，平日更是一起玩樂的朋友。齊哥耐心地聽我一次次的口試並給予建議，大頭跟阿龍學長也叮嚀我投影片的修改。胖達、義凱、佳瑋、元等學長姐也給我許多指導。大嘴、永裕、QMO、鑫偉更是一起奮鬥的好伙伴，尤其最後半年總是一起討論、互相打氣，沒日沒夜地做實驗、趕論文。而高守、振揚、裕淳、晶今、廷聿等更是不可多得的學弟妹。

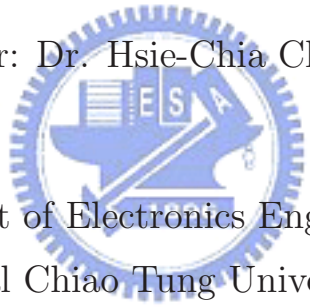
此外，在 316 實驗室的大大、承曄、JUJU、已畢業的 apu、阿俊讓我在實驗之餘，能放鬆心情。感謝 VAN 在我苦悶的碩三生活相互打氣，也謝謝篤雄、老大、acer 總是在我詢問程式問題的時候，非常耐心且仔細地教導、指正我。也謝謝國權、孟琦陪我一起打球鍛鍊體魄。也謝謝遠在美國的包子三不五時跟我哈拉，讓我心情保持愉悅。身為棒球愛好會唯二會員之一並且一起騎腳踏車環島的秀逗，從考試開始到碩三，一路互相扶持砥礪到最後，非常感動。更特別感謝 spice 和俊男兩位朋友從我重考開始無論在研究、生活上一直給予我支持、鼓勵，也一起度過許多美好的時光。

最後，我要謝謝我的父母、姐姐總在生活上給我最大的支柱，給予我正面的能量。謝謝小嗨在我漫長的碩士生活中每天回應我的喜怒哀樂，一路陪伴，當我的避風港。千言萬語，還是只有感謝，再感謝！

Throughput Enhancement Using LT Codes in Erasure Network Communications

Student: Chien-Fu Hu

Advisor: Dr. Hsie-Chia Chang



Department of Electronics Engineering

National Chiao Tung University

Contents

List of Figures

List of Tables

Abstract

Bottlenecks occurring during the transmission in the network have caused serious consequences on the throughput degradation in receivers. In this thesis, a coding system that apply low degree LT codes in transmission environment to enhance the throughput is introduced. The proposed LT code can improve the throughput approach to the theoretical maximum. In the meanwhile, it provides a significant data protection ability against the packet loss in erasure channels. The required buffer size in the intermediates in the LT codes system becomes a inconsequential factor that it can be reduced considerably compared to the data size. In some network topology, we combine low degree LT codes with specified network coding mechanism to get further improvement in throughput of every receiver and also give strong protection of data. The overhead of the coding mechanism causes the reasonable computation cost in whole transmission.

For each packet of $1KB$, the proposed LT codes work under the entire packet number from $4K$ to $64K$. The proposed method alleviates the bottlenecks successfully in two kinds of topologies, with edge-disjoint paths (EDPs) and without EDPs. It also gives strong protection of data in the erasure channels with different scale of loss rate from 0% to 20%. Our proposed low degree LT code can enhance the throughput in the range from 20% to 30%. In the network with edge-disjoint paths, the combined LT-network code offers advanced improvement up to 50% or so. Finally, the computation is over $GF(2)$ and the coding overhead is about 30 XOR operations per codeword.

Chapter 1

Introduction

Network has existed for nearly 30 years, and it plays a significant role in computer communication. Owing to the enhancement of the bandwidth, a large quantity of data are able to deliver from the server to users such as MOD (Multimedia On Demand) that the server multicasts data to the users who request the services. Data transmits from one source to the other destination through some nodes called intermediates during the transmission. In the exiting network system, the tradition method is that every intermediate node just does store-and-forward to pass the information to the next node. We find that the tradition routing, however, can not achieve the max flow proved by the max-flow min-cut theorem, particularly in the multicasting applications. Therefore, network coding theory is proposed to alleviate the intermediate bottleneck and to elevate the utility of the network channel.

Packets flooding in the communication networks suffer the loss due to the disturbance. In the erasure channels, every sink either believe what it receives or get nothing. Now that we can't avoid the loss, we have to facilitate systems to have good capability to fight against error and protect the information. Our goal is to enhance the throughput of the sink and in the meantime, to enable system to establish error protection mechanism to reduce information loss.

The thesis is organized as followed. Firstly, we introduce some basic concepts regarding network coding including some simple examples, mathematical representation, and a innovated method called random network coding which have been announced.

Although the theory confirms the benefit on network coding, the implementation is cumbersome due to the restrict of the true network transmission such as packet loss, network topology, etc. In order to give consideration to both network throughput and error protection mechanism, we propose LT code to apply for the network coding to fulfill our goals. In chapter 3, we introduce LT code, a famous code applied in erasure channel. We will give explicit description concerning coding procedure, decoding procedure and the design of degree distribution.

In chapter 4, we evaluate the method that network coding collaborates with LT code to approach the optimal performance proved by the theory. The construction of simulation environment will be explained. For the sake of simplicity, the acyclic network and single source multicast condition are specifically concerned in our simulation.

We show our simulation results in chapter 5. We compare three different systems including routing, a coding method have been proposed and LT code applied to network according different packet loss rate.

Finally, some conclusions and discussions will show in chapter 6. In the meantime, we point out some experimental experience in the appendix.

Chapter 2

Network Coding

Over the last decade, there has been a large interest in network coding. The concept was firstly propounded by [1] and there is dramatic increase in the number of publications on it. Firstly, we describe the main concept of network coding and the well-known butterfly network will be illustrated. Secondly, We generalize the method in mathematical domain, representing the equivalent network coding. Because of the impractical efficient coefficients assignment of every intermediate, random network coding is proposed [3]. In the end of the chapter, we consider the packet loss during the realistic transmission. Therefore, we propose the cooperative method between network and LT code to enhance the throughput of network, in the meantime, to provide error protection of the data.

2.1 Max-flow Theorem

Before we describe the concept on network coding, we should know how to evaluate the theoretical max flow of a sink in the network to be the target we want to achieve. A network consisting of nodes and edges can be viewed as a kind of graph. And the acyclic network is the network that contains no circle or loop composed by the directed edges. The transmission model is in a condition called single source multicast circumstance that there exists only one

source node to transmit the data to different number of destinations. In graph theory, we know that for a given network topology, we can calculate the max flow of each sink. The max-flow min-cut theorem is described as followed.

Theorem 2.1.1 (Max-flow min-cut Theorem)**.** *If f is a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:*

1. *f is a maximum flow in G.*
2. *The residual network G_f contains on augmenting paths.*
3. *$|f| = c(S, T)$ for some cut (S, T) of G.*

There are some algorithms to calculate the max flow such as Ford-Fulkerson algorithm, Edmonds-Karp algorithm, and the Relabel-to-Front algorithm in [6].

2.2 Main concept on Network Coding

As mentioned before, data packets are delivered from the source to the destination according the path composed of the chain of the intermediate nodes. Every intermediate node receives the data packets from its input link, storing, and then passes the packets to the next node by the output link. In the case that one intermediate node in the path transmit the data toward multiple nodes or destinations, it copies the data from the input link and then pass the same copy to the different output links. In some situation, this store-and-forward method causes that the node receives the same data by the different input links belonged to distinct nodes, decreasing the utility of the bandwidth. Now that the intermediate nodes process data during the transmission, we let them do arithmetic calculation rather than store-and-pass. The packets transmitted during the path become either true information or some combination (linear or non-linear) of the data. Every destination node also called

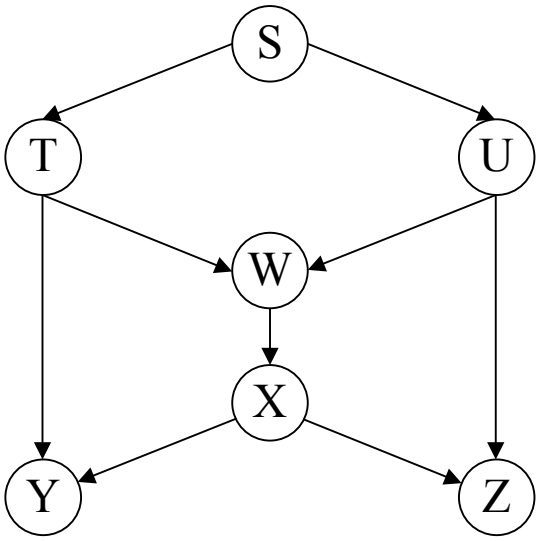


Figure 2.1: Butterfly network

sink receives the sufficient processed packets, decoding them to recover the true information. Network coding aims to resolve the bottleneck of the intermediate node, and to let every sink fulfill its theoretical max flow. The butterfly network is illustrated in Figure A.1.

2.2.1 Butterfly Network (Coding in Intermediates)

Fig A.1 is a communication network represented by the nodes and directed edges (links). Nodes are categorized with three types such as source, intermediate and sink. Node without any incoming edges is called source that transmits the information, and by contrast, node without outgoing edges is called sink which is the destination of the messages. Node which is neither source nor sink is called intermediate. In the figure, the node labeled S is source, the nodes labeled Y and Z are sinks, and the other nodes labeled T, U, W, X are intermediates. The directed edge represents the direction of the lossless packet transmission channel and each one has its own capacity per unit time. Each edge capacity of the example is set to 1.

The network is said to be acyclic if there exits no directed circle in the whole network

topology. The multicast condition is that the source wants to transmit the data to all the sinks in the network. In the example, the source S multicasts the data to both the sinks, node Y and Z.

First, we consider the traditional store-and-forward method. In the first transmission, S sends data b_1 and b_2 to the T and U by the edges ST and SU respectively. And then every intermediate sends the data it receives to the next node. Obviously, we can find that the node W has 2 incoming edges TW and UW but only one edge WX. Therefore, node W chooses either b_1 or b_2 passing to the X. Assume that W choose b_1 and b_2 in order, node Z will receive both b_1 and b_2 but node Y will only receive the data b_1. That is to say, we need extra transmission to let W send data b_2 to Y through the edge WX and XY. The equivalent throughput of the entire network for node Y and Z is 1 meaning that every sink receive one data per transmission.

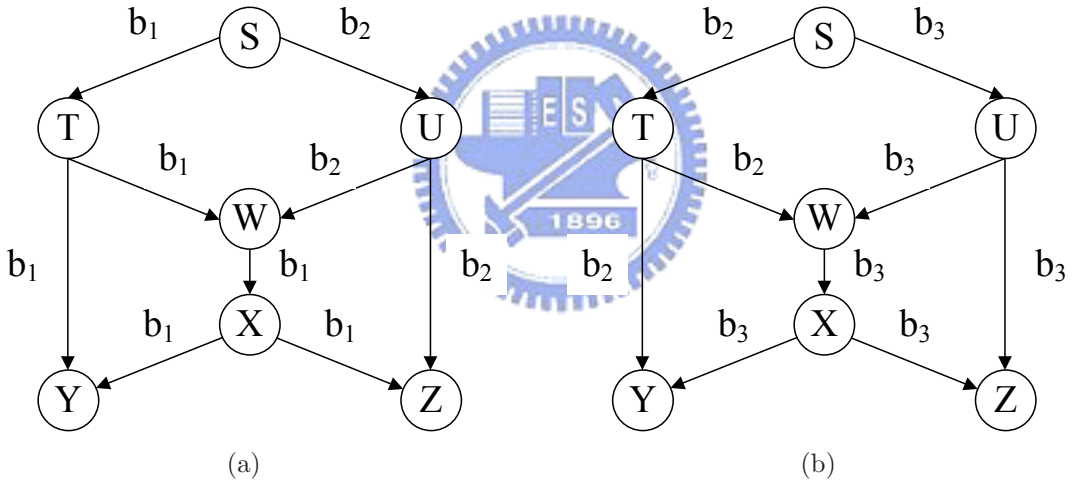


Figure 2.2: Tradition routing in butterfly network

There is one modification in store-and-forward. The optimal method is that in the first time S send two data b_1 and b_2 to node T and U, and W sends the data b_1. After the first run, the outcome is the same as mentioned above. Y receives the only data b1 and Z receives

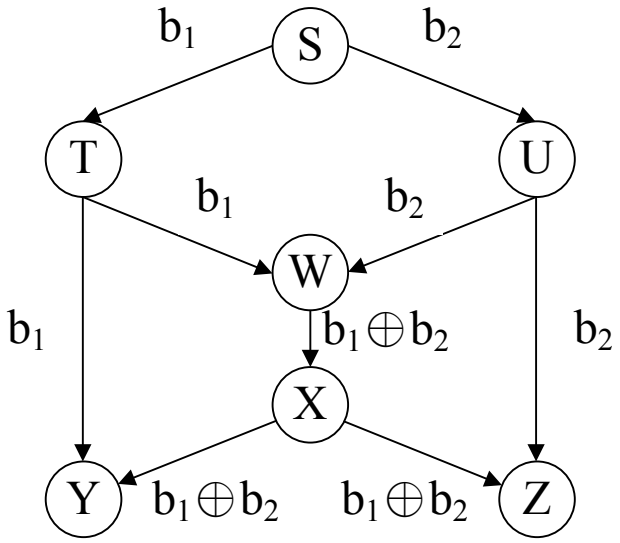


Figure 2.3: Network coding in butterfly network

both b_1 and b_2. The different is that in the second time, S sends the data b_2 and b_3 to the node T and U, and node W chooses the b_3 passing to the X. When the second transmission ends, we find that every sink has three data b_1, b_2 and b_3 respectively. Hence, every sink obtains 3 data in 2 transmission, the throughput of each sink enhances to 1.5. The procedure is shown in Fig 2.2.

Based on the Theorem 2.1.1, the theoretical max throughput of the sink in the example is 2. Unfortunately, the tradition method is 1 and even the improved method is 1.5, which can not reach the max flow.

In the example above, we locate that the the network obstacle is node W. The number of incoming edges is 2, whereas that of outgoing edges is 1, causing that one of the two data packets needs to be stored in the buffer awaiting the extra delivery. The bottleneck of W can be resolved by the network coding skill. Let W do exclusive-or operation of two packets from incoming edges. The edge WX will transmit the data $b_1 \bigoplus b_2$ to node X. We see that Y will receive the data b_1 and $b_1 \bigoplus b_2$, and Z will receive the data b_2 and $b_1 \bigoplus b_2$. Both of

them can recover the true information b_1 and b_2 by doing the XOR operation of two packets they receive as shown in Fig 2.3. The equivalent throughput is elevated to 2, the theoretical maximum.

2.2.2 One-source Three-Sinks Network (Coding in Source)

Fig 2.4 is another network topology. Capacity of every edge is also set to 1. Source S needs to multicast data to all the destinations, node X, Y, and Z. The max flow of every sink is 2.

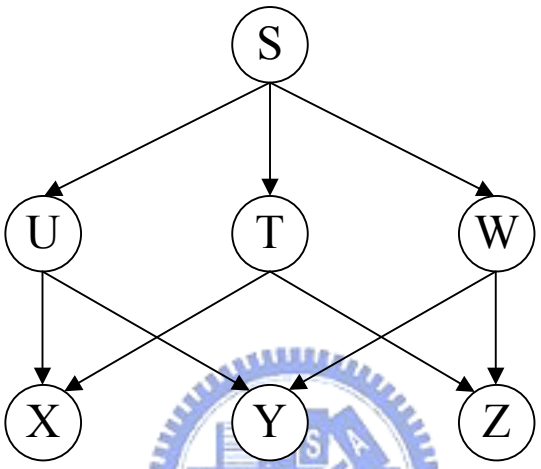


Figure 2.4: One-source three-sinks network

The tradition routing is shown in Fig 2.5. Firstly, node S sends data b_1, b_2, b_3 to their adjacent nodes, U, T and W. After first transmission, each sink receives two data as shown in 2.5(a). Then, we allocate the data by shifting them to different edges clockwise (shifting counterclockwise gets the same outcome) as shown in 2.5(b). Each sink can get the third data from one edge of two, and get the repeated data from the other one. The efficient throughput of each sink is 1.5.

How can we apply the network coding method in this network topology? In this situation, all we do is let source S do coding. S sends data b_1 to U by edge SU and data b_2 to T by edge ST as routing does. However, S node doesn't send the data b_3 but $b_1 \bigoplus b_2$. Obviously,

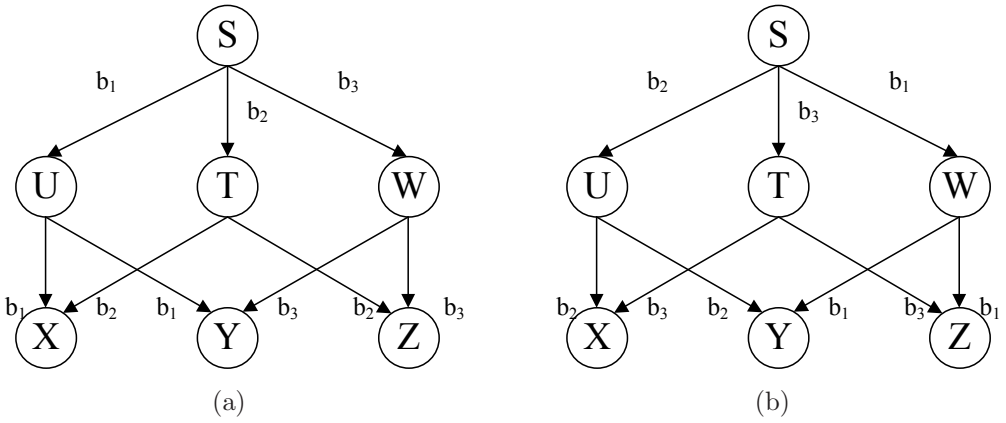


Figure 2.5: Tradition routing in one-source three-sinks network

by this alteration applying coding technique, each sink X, Y and Z will receive two data (b_1, b_2), $(b_1, b_1 \bigoplus b_2)$, and $(b_2, b_1 \bigoplus b_2)$ in turn. Every sink is capable of get two data in one time. Hence, the throughput of every sink is approach to 2, the theoretical maximum.

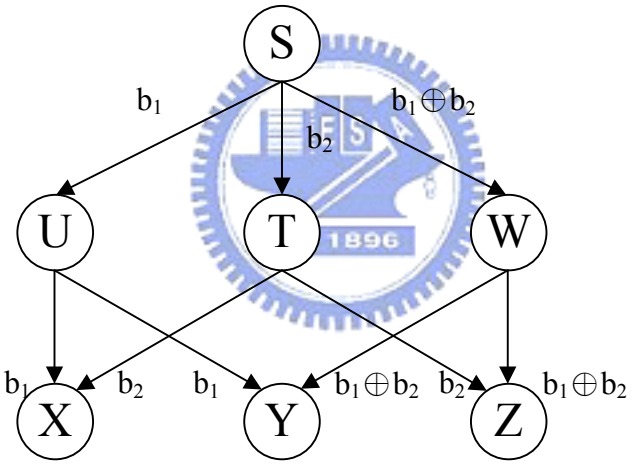


Figure 2.6: Network coding in one-source three-sinks network

The benefit of the network coding is illustrated in two examples above. We claim that network coding can enlarge throughput by letting intermediates or sources do some simple operations. However, it is insufficient to elucidate the delicacy of network coding by merely indicating the specified operation in some nodes to convince that the result will get

reformed coincidentally. In the following, we will formulate the method mathematically and theoretically.

2.3 Mathematical Representation

We exemplify two cases how network coding apply to multicasting system in different network and gain the improvement compared to the method nowadays. However, we can't foresee and control every operation in every node intuitively, expecting our straightforward innovation work successfully. In this subchapter, we will formulate the mathematical model to generalize the network coding issue. Adhere to this formulation, we can analyze and resolve the problem systematically.

Network coding is proposed to enhance the flow in the network by doing some computation of original data either in sources or intermediates. Every data packet flooding in network can be regarded as one combination of all intrinsic data. (Here, only the linear operation is discussed for the implementation simplicity.) We find that the original data spans one space, and packets in every edge span another. That is to say, there exists one mapping in every edge between two spaces. The functionality of every node becomes to map the entire received symbols from its incoming edges to a symbol for outgoing edges. Network coding can be converted to the mechanism for encoding process of every edge.

For the clarification, the definition and symbol notations used in our mathematical Representation are listed as followed. The notation is quoted by [2].

Notations

- Source: A node without any truly incoming edges.
- Every edge in graph represents channel with capacity data unit per unit time.

- In(T)/Out(T): The set of incoming/outgoing edges of node T.
- In(S): a set of imaginary edges without originating nodes.
- ω: The number of the imaginary edges.
- data unit: An element of $GF(F)$.
- message x: A ω-dimension row vector $\in F^{\omega}$.
- A network code is in $GF(F)$ and ω dimension.

Definition 2.3.1. *A network consists of a local encoding mapping*

$$\tilde{k_e} : F^{|In(T)|} \to F$$

for each node T in the network and each channel $e \in Out(T)$.

By Definition 2.3.1, we construct the the transform between the incoming and outgoing edges in one node. Since the acyclic network provides the upstream to downstream procedure, data is transmitted by the path composed of edges. The mapping of each edge is equivalent to continual transforming by the passed edges before. Hence, we give another definition to represent the outcome of the processing of the recursive mapping.

Definition 2.3.2. *A network consists of a local encoding mapping $\tilde{k_e} : F^{|In(T)|} \to F$ and a global encoding mapping $\tilde{f_e} : F^{\omega} \to F$ for each edge e in the network such that:*

- *For every node T and edge $e \in Out(T)$, $\tilde{f_e}(x)$ is uniquely determinded by $(\tilde{f_d}(x), d \in In(T))$, and $\tilde{k_e}$ is the mapping via*

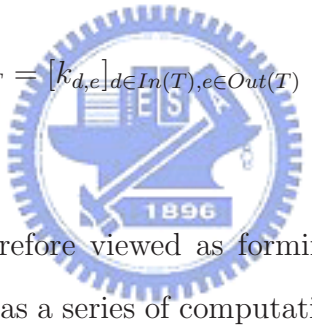
$$(\tilde{f_d}(x), d \in In(T)) \longmapsto \tilde{f_e}(x)$$

- *The mapping $\tilde{f_e}$ are the natural projections from the space F^{ω} to the ω different coordinates,respectively.*

Considering the physical implementation, it is desirable that the fast computation and simple circuit in the node. Therefore, the linear transformation is involved. If the encoding mapping $\tilde{f}_e(x)$ is linear, there exists a corresponding column vector f_e with ω dimension such that the product $x \cdot f_e$ is equal to $\tilde{f}_e(x)$, where x is the ω-dimensional row vector data generated from the source. Similarly, there exists $|In(T)|$-dimensional column vector k_e such that $y \cdot k_e = \tilde{k_e}(y)$, where $y \in F^{|In(T)|}$ represents the symbol received in the node T. Since every edge has its own mapping column vector, we can formulate the operation in the node of every edges connected in one node. If a pair of edge (d, e) is linked by one node T with $d \in In(T)$ and $e \in Out(T)$, we call these two edges an adjacent pair. Therefore, we can formulate the coding process by matrix form in every node.

Definition 2.3.3. *Network consists of a scalar $k_{d,e}$, called the local encoding kernel, for every adjacent pair (d,e). Meanwhile, the encoding kernel at the node T means the $|In(T)| \times |Out(T)|$ matrix*

$$K_T = [k_{d,e}]_{d \in In(T), e \in Out(T)}$$



The network coding can be therefore viewed as forming the effective matrix of every node, and every edge can be viewed as a series of computation of the column vector in every matrix of the node that data passes. Note that the structure of matrix assures the order of linked edges.

Definition 2.3.4. *A network consists of a scalar $k_{d,e}$, for every adjacent pair (d,e)in the network as well as an ω-dimensional column vector f_e for every channel e such that:*

- $f_e = \sum_{d \in In(T)} k_{d,e} f_d$, *where* $e \in Out(T)$.
- *The vector f_e for the ω imaginary channels $e \in In(S)$ form the natural basis of the vector space F^ω.*

- *The vector f_e is called global encoding kernel for the channel e.*

2.3.1 Butterfly Network over $GF(2)$

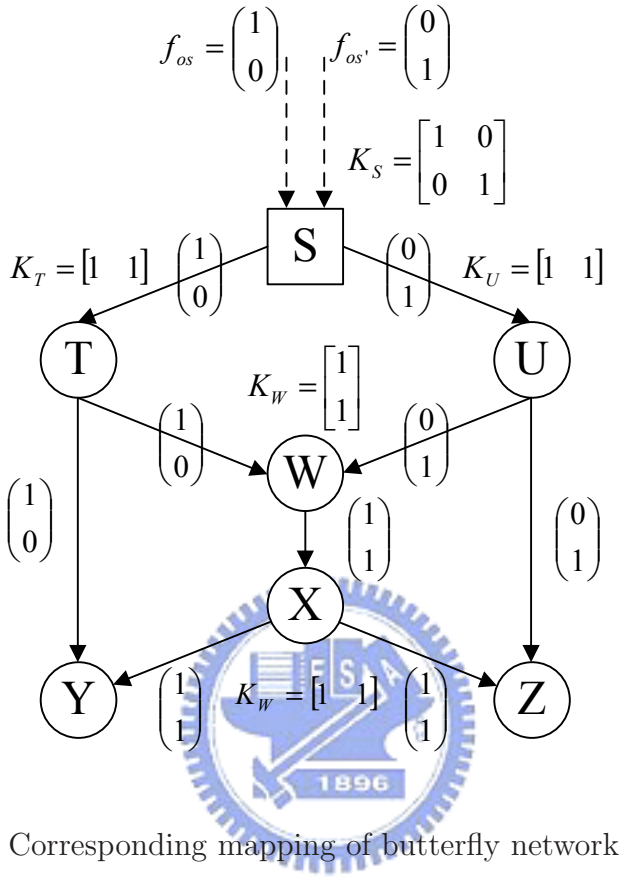


Figure 2.7: Corresponding mapping of butterfly network in Fig 2.3

The corresponding edge mapping and operation matrix of every node in Fig 2.3 is showed in Fig 2.7. The imaginary edges of source S is two, and global encoding kernel of two edges, f_{os} and $f_{os'}$, represent the mapping of the original data to produce the information data b_1 and b_2. The exclusive-or operation means the computation is in $GF(2)$. According the matrix of every node, we can calculate the global coding kernel f_e of every edge.

We give some examples to derive the global encoding kernel in Definition 2.4. Observing the source matrix K_S with 2 incoming and 2 outgoing edges, the element of matrix represents the scalar of two specified linked edge. Based on the definition2.4, we can finds that

the equivalent global encoding kernel is the summation of the global encoding kernel with corresponding scalar in node matrix.

$$f_{ST} = \sum_{d \in In(S), e \in Out(S)} k_{d,e} f_e = k_{11} f_{OS} + k_{21} f_{OS'} = 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 0 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$f_{WX} = \sum_{d \in In(W), e \in Out(W)} k_{d,e} f_e = k_{11} f_{OS} + k_{21} f_{OS'} = 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 1 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Fig 2.7 is the special case that the chosen finite field F is 2. However, scalars in every matrix and computations are done in the $GF(F)$, and it can be generalized in Fig 2.8.

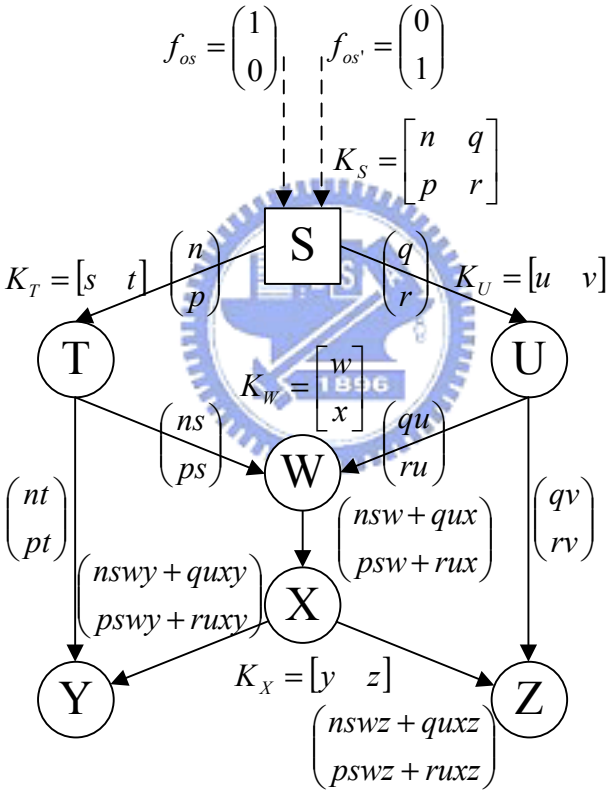


Figure 2.8: General mapping modified of butterfly network in Fig 2.7

2.3.2 Butterfly Network over $GF(F)$

In Fig 2.8, each global kernel can be calculated by the same steps described above. The design parameters are the scalars in every matrix such as $n, p, q, r, \ldots, z$. The assignment of all scalars influences the efficiency of the network utility. Concerning to sink Y , if we want to approach the theoretical maximum, 2, the global kernel f_{TY} and f_{XY} should be linear independent ,namely , the space spanned by these two vector should also be 2. The condition of another sink Z is the same. If the two vectors are linear dependent, the sink will suffer the flow decreasing.Therefore, we can remark that when the source transmits a message of ω data units into the network, a receiving node T obtains sufficient information to decode the message if and only if $dim(V_T) = \omega$, of which a necessary prerequisite is that maxflow$(T) \leq \omega$. The prerequisite assures the necessity to applying network coding to enhance utility of the network. If maxflow$(T) > \omega$, the entire network is capable of affording the whole being transmitted data. There exists no bottleneck in the network and transmission will certainly accomplished without difficulty.

We convert linear network coding to matrix forming, and comprehend that the key to enhance the throughput and decode information successfully is the well designed coefficients in every matrix of each node in whole network. However, it is difficult to implement this concept directly, and the random coding mechanism is recommended in the next subchapter.

2.4 Random Network Coding

We have derived the identical method for network coding from the inference above. Designing a effective linear network code is equivalent to finding out adequate coding matrix K of every node in the network to guarantee that every sink receive enough information to decode the data. However, in the real communication network is enormous and complex that we

we should avoid the cumbersome and inefficient task such as detecting the entire network. Observing the example in Fig 2.8, the number of design parameters in the network with seven nodes is twelve, and it will increase explosively with the total number of network grows.

Since the coefficients assignment of each node is time-consuming and exhausted, we let each node produce the coefficients randomly rather than appoint them. Linear random network coding provides the method that every node independently and randomly select linear mapping from inputs to outputs from some finite field. By doing this, we don't need to design the coefficients tiresomely and the key factor becomes how to choose the linear combinations effectively. Coefficients are chosen uniformly or more generally, based on a distribution. We can regard uniformly choice as a special case that every candidates are selected with equal probability. Regarding the sink, it receives the packets which is the linear combination of the intrinsic information and recover data from them. If the distribution performs outstandingly, every sink is able to recover the original data after it receives N data ,where N is the total number of data in source. Namely, the dimension of the packets originated from the linear random coding should span N-dimensional space equivalent to the space spanned by N data.

Random network coding offers a coding mechanism by statistic property instead of deterministic structure. However, we ought to know that designing a well performed encoding matrix in every intermediate is difficult but not impossible. If we try hard to find efficient encoding matrix in every intermediate, we can therefore get the optimal solution as two examples mentioned above. In the meantime, assigning every encoding matrix varies significantly due to the network topologies. Namely, we have to design the specified encoding matrix whenever we meet different networks. Random network coding let intermediate encode independently regardless of the topologies. The performance should be basically not as good as the well-designed optimal structure for specified networks. However, if we can

find a good mechanism to combine packets effectively, the outcome can be approach to the optimal solutions.

Another factor we should concern is the filed size we choose. If the computation is under a insufficient finite filed, the combinations of data will be easily dependent with each other. It shortens the codeword space which should be as large as the data space and therefore degrade the network utility. The innovated random network coding and some theorems can be found in [3] and [4].

2.5 Summary

The discussion above is on the basis that packets are delivered in lossless communication channel. However, in real system, the packet will suffer loss from the unsteady and noised environment. It causes that even the well designed random method performs poorly due to the packet loss. In order to work against the packet loss during the transmission, we request the coding mechanism with the following properties.

- Coding is based on distribution.
- Simple encoding operation.
- Good protection of data.

First property continues the random network coding method, and second one simplify hardware implementation and operation complexity. The final property is involved to protect the data due to the inevitable loss. Hence , we bring up a method that applying LT code to network coding to fulfill the demands and enhance the throughput of every sink during entire transmission.

Chapter 3

LT Code

LT code (Luby Transform code) is a sparse random linear fountain code designed by Michael Luby with a outstandingly cheap computation for decoding algorithm. It especially outperforms in the communication for channels with erasures, such as the internet. Every receiver collects any N packets to recover the original data, where N is slightly greater than the original files size K. The computation complexity is astonishingly small, growing linearly with the file size K.

The chapter is organized as followed. Firstly, we introduce the main concept on fountain code. Secondly, we specifically discuss the LT code including encoding process, decoding process , and code structure regarding to distribution. Finally, we focus on how LT code applying to network coding to enhance the efficient flow quantity to fulfill our goal.

3.1 Fountain Code

Fountain code is the one kind of rateless code for erasure channel that packets are either received correctly or lost. Packets passing during the erasure channels gets loss with the probability, causing the sink receives incomplete data, asking retransmission instruction for the erasure parts. The retransmission mechanism results inefficiency of the utility of

network. The situation becomes worse when multicasting or broadcasting applied in the system. Thus, the need for the erasure correct code is needed to avoid the retransmission. The concept of Fountain code is that the source produces considerable quantities encoded packets, limitless potentially. Comparatively, the sink is respected to receives a slightly larger quantities compared to the total size of data to recover successfully. The encoding method is randomly picks of file with size N. Every encoded packet is a randomly linear summation under modulo 2. If the process continues, it forms a generator matrix of infinite length. However, sink only receives packets of size N due to the erasure channels. The received N packets and the K file forms another generator matrix G. Every element G_{nk} is set to 1 to represent that source and encoded packets is connected, otherwise represent no connection. Supposed that we know the matrix G, we can therefore decode the whole data without retransmission. If $N < K$, there can be no opportunity to decode successfully because of the insufficient information. If the number of received packets is exactly K, we need the $K \times K$ matrix is invertible, meaning spanning the same space of the file space. The key of the performance that N is closer to K depends whether any subspace of $K \times N$ be capable of forming isomorphism space of file space. More detail is introduced in [7].

3.2 LT code

LT code is introduced in [9], and is the first realization of a class of the random linear fountain codes which is the record-breaking sparse-graph code for erasure channels. It substantially reduce encoding and decoding complexity.

3.2.1 Encoding

Consider source file $s_1, s_2, s_3, \ldots, s_k$ of size K, the encoded packet t_n is produced as follows:

1. Randomly choose the degree d_n of the packet from a degree distribution $\rho(d)$; the appropriate choice of ρ depends on the source file size K, as we will discuss later.

2. Uniformly choose d_n distinct input packets, and set t_n equal to the bitwise sum of these d_n packets. The equivalent operation can be done by continuously exclusive-or-ing the packets until d_n times.

After the encoding process, source defines a bipartite graph categorized by source information and encoded symbols. Connection structure between two groups depends on the degree distribution significantly. Degree d_n means the number of distinct source information connected to an encoded symbol. If the mean degree $\bar{d}$ is extremely smaller than K, the graph is spare. We can regard the produced code as an irregular low-density generator-matrix code.

In order to decode successfully, sinks have to know the the information including connected degree and the members of the connected source information of the received symbols. There are two method for the source to communicate code information with the sinks. First is relied on the synchronized clocks. We can use the random number generator which is seeded by the clock to decode every encoded symbol based on random degree and each connection members of this symbol. Another is to carry the information with the packets. However, the overhead is significantly depended on the max degree of distribution and the size of the index bits to assign identical number of each source information. The cost is tiny if the size of packet is much longer than these carried information.

3.2.2 Decoding

Decoding process is easily in the erasure channel. All that a decoder need to do is to solve the equivalent function $t = Gs$ to recover s from t, where s are source information and t

are received symbols. Since the channel is erasure, we receive the certainly correct symbols or get nothing due to lost. The simple method to decode is by message-passing, using the complete certain symbols to recover those with uncertainly. The decoding procedure are described as follows.

1. Find a check node t_n is degree 1 (only connected with one source packet s_k). (If there exists no such check node, this decoding algorithm stops right now, and fails to decode all the source packets).

   (a) Set $s_k = t_n$.

   (b) Add s_k to all check nodes $t_{n'}$ that are connected to s_k

   $$t_{n'} := t_{n'} + s_k$$

   for all n' such that $G_{n'k} = 1$.

   (c) Remove all the edges connected to the source packet s_k.

2. Repeat 1 until all s_k are decoded.

A simple example is illustrated in Fig 3.1. There are three source information $(S_1, S_2, S_3)$and four encoded check symbols (t_1, t_2, t_3, t_4). At the beginning, only t_1 is connected merely to S_1. We set $S_1 = t_1 = 1$ and cancel the edge between them after the first iteration as showed in b. Then, we add the S_1 to all connected check nodes, deleting the edges between S_1 and its connected group. In second iteration, we find that t_4 is only connected to S_2, and we can recover S_2 from t_4. Similarly, we delete the edge between S_2 and t_4. Repeat the iteration, we can finally recover all three source packet successfully.

In our example above, total data can be decoded. However, if we find that there exist no check nodes with degree one, decoding procedure will stop, meaning that the process

crashes. Namely, we need to receive extra symbols to decode the remaining to recover source information.

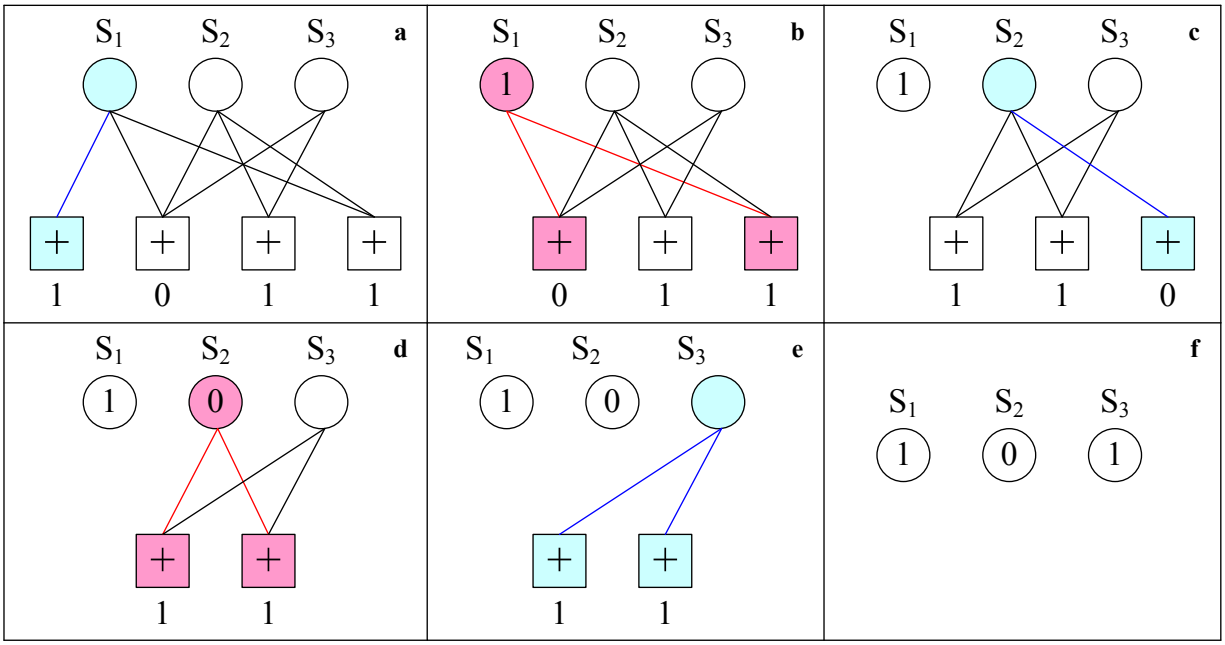


Figure 3.1: LT decoding procedure cited from Fig.4 in [7]

3.2.3 Distribution Design

We have described coding process and decoding process in the case that the operation is based on a determined distribution $\rho(d)$. In the following, we discuss how to design a distribution to performs well.

In the decoding process, we discover that decoding procedure fails if there exists no symbol with degree one. If we want decoding continues, there must also exist some symbols with lower degree that have chance to become a new degree one symbol to let process keep going. At the same time, if the max degree of distribution is too low, there may exists some source information that are connected to none of the encoded symbol and therefore causes tremendous loss. Thus, the performance is vitally depended on the designed distribution. In order

to fulfil the desired requirements, ideal soliton distribution is derived from mathematical theory.

Ideal soliton distribution defines $\rho(d)$ as follows:

$$\rho(d) = \begin{cases} 1/K & \text{for } d = 1 \\ \frac{1}{d(d-1)} & \text{for } d = 2, 3, \ldots, K \end{cases} \tag{3.1}$$

The expected value of degree is roughly $log_e K$.

Soliton distribution works poorly in the real transmission. Because when we obey on this distribution, it gets high probability that there exist no degree one check symbol during the decoding process. Thus, robust soliton distribution modifies the degree distribution.

Robust soliton distribution defines extra two parameters, c and δ.

c : a constant determined by the designer.

δ : the probability that the decoding fails to decode completely after a certain number K' of symbols have been received.

The modified terms are

$$\tau(d) = \begin{cases} \frac{s}{K}\frac{1}{d} & \text{for } d = 1, 2, \ldots, (K/S) - 1 \\ \frac{s}{K} log(S/\delta) & \text{for } d = K/S \\ 0 & \text{for } d > K/S \end{cases} \tag{3.2}$$

S is a constant calculated by

$$S \equiv c \cdot log_e(K/\delta)\sqrt{K} \tag{3.3}$$

Add the modified terms to ideal soliton distribution and normalizes, we get the robust soliton distribution $\mu(d)$

$$\mu(d) = \frac{\rho(d) + \tau(d)}{Z} \tag{3.4}$$

where

$$Z = \sum_{d} \rho(d) + \tau(d) \tag{3.5}$$

Regarding to the additional distribution $\tau(d)$ summed to the ideal solition $\rho(d)$, the max degree d is extended to K/S. The spike at $d = K/S$ ensures whole source information connected with higher probability during the encoding process. The max degree required is proportional to the size of files, with inverse proportion to S calculated by the tuning parameter c and δ. δ can be viewed as the probability of decoding failure, and if we want to lower the failure probability, we have the higher corresponding max degree, which fits in with the straightforward intuition.

If we want to decode the source information completely after receiving the whole symbols with a probability $(1 - \delta)$ at least, then the required number of total received packets is $K' = KZ$. It is obvious that Z will slightly larger than 1, and equal to 1 for the optimal solution we look forward.

Robust soliton distribution offer two designed parameters c and δ to design distribution. The number of each degree depends on c, δ, K essentially. The more representative factor is Z in equation 3.5, the excess quantity of necessitated symbols. A good distribution can be tuned to the result that the needed overhead is usually about 5 to 10 percentage. And the constant c is usually chosen smaller than to 1 to get better performance.

3.3 Summary

Recall the requirements we desired to enhance the throughput of the network utility.

- Coding is based on distribution.
- Simple encoding operation.
- Good protection of data.

We can easily discover that LT code can meet these desired requirements. Therefore, we propose a method to apply LT code for network coding to accomplish our targets. More detail will be illustrated in the next chapter.

Chapter 4

Cooperative Network Coding with LT Code

We have introduced the network coding theoretically in chapter 2 and the LT code in chapter 3. In this chapter, we will demonstrate the innovative ideas how to gather them to achieve our targets, high throughput and prominent error protection. The flow chart below shows the simulation procedure, cutting into several partitions. We will explain the work of each section according the Fig 4.1.

The chapter is organized as followed. First, we will specify the network and calculate the max flow quantity, the goal we pursuit, of every sink. Due to the coding, every packet must carry the extra index of combined information, which causes the fragmentation. After the initialization, the packets will be encoded, transmitted, received, buffered, and decoded recursively until the sink decodes the whole information source delivered. Simulation ends if every node recover the total information. The whole simulation environment is using C++ and the detail will be discussed in every section.

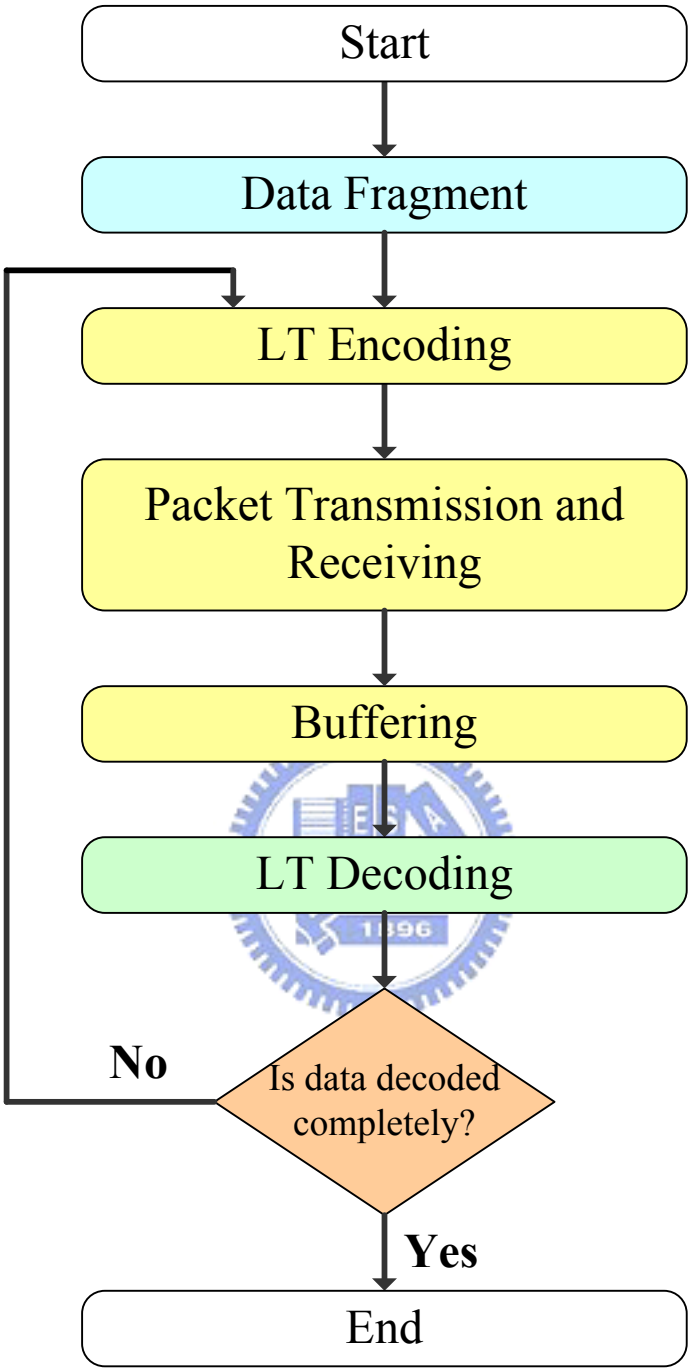


Figure 4.1: Simulation flow chart

4.1 Network Topology Specification

In the simulation, the acyclic network is specifically concerned. The transmission model is in single source multicast communication and no edges are connected between sinks meaning that sinks receive the data from either sources or intermediates, or both. The conditions are listed below.

- Acyclic network.
- Single source multicast.
- No shared content among sinks.

Based on these conditions above, two network topologies exemplified in Fig 2.1 and Fig 2.4 are particularly discussed whose max flow is 2 unit capacities in every sink.

4.2 Data Fragment

We apply LT code for network coding. Thus, every packet includes two parts. One part is the outcome by series of exclusive-or operations of the original information. The other is the overhead that records the indices of all original information involved during the encoding process. It is intuitive that additional overhead will lengthen if the total transmitted data enlarges, and we will need more number of bits to record the data correspondingly. We can evaluate by the following formula.

$$F + D \times I = B$$

The parameters are listed in Table 4.1 .

We explain every parameter below.

Table 4.1: Design parameters

M	total files
B	unit capacity
D	max degree of LT code
F	fragment information size
I	data index

M : The total files to be transmitted. When M enlarges, the number of transmission time increases.

B : The unit capacity of the edge. We set this by finding out the greatest common divisor(G.C.D) of all edges. The capacity is the multiple of B. In reality, it should be the last guaranteed bandwidth of network.

D : The max degree of the LT Code, meanwhile, is the largest number allowed to combine the information.

F : The actual information size of the encoded symbol. If network coding is not executed, F will be the same with B.

I : The number of bits to assign the number of every fragment for identification. It can be calculated by $I = log_2(\frac{M}{F})$.

We find that the real information carried in a packet shortens due to the overhead of encoding information. The efficiency drops out after the segmentation, causing extra transmission compared to simple routing. (In existing system, there exists particular headers to record the information of transmitted packet.) We define some notations as followed.

- $N_{original}$. Total number of packets a sink should receive in routing method.
- N_{coding}. Total number of packets a sink should receive in coding method.

- $M = N_{original} \times B = N_{coding} \times F$.
- O_{frag}. Normalized overhead after the fragmentation.

The loss can be calculated by

$$O_{frag} = \frac{N_{coding}}{N_{original}}$$

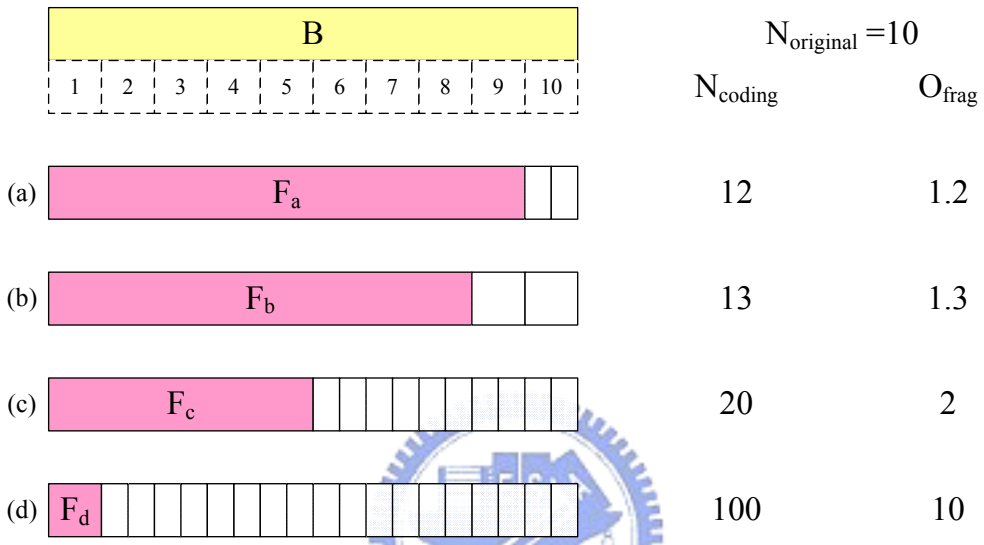


Figure 4.2: Basic concept of fragmentation

We describe the basic concept discussed above in Fig 4.2. Assume the unit capacity of the transmission channel is B, and the number of times needed to delivery is $N_{original}$, which is 10 without any coding mechanism. Every small block labeled from 1 to 10 is helpful to show the ratio of every fragment in different examples. Fig 4.2-(a) shows the fragment F_a is 90% of B and max codeword degree D is 2. The required transmission times is therefore increased to N_{coding}, 12. That is to say, we have two extra transmissions due to the fragmentation, and the overhead O_{frag} is 1.2. Compared (a) and (b), the difference is that the number of bits to index each fragment in (b) is twice as large as that in (a). The quantity of true information a packet can carry is from 90% to 80% of B, causing one extra transmission. Considering

two examples in (c) and (d), the max degree D increases to 10 and 18, weighting 50% and 90% of a packet. When true information weights lower percentage in a packet, it results in a huge quantity of transmission times. In examples (c) and (d), N_{coding} are increased to 20 and 100 which are much more than $N_{original}$. That means we must design a outstanding coding mechanism to make up for additional transmission due to fragmentation. The accurate calculation is showed below.

Table 4.2: Example of the parameters setup

M	D	B	$F(bits)$	$D \times I$	O_{frag}
$512KB$	2	$1KB = 8192_b$	8172	20_b	1.0024
		$4KB = 32768_b$	32752	16_b	1.0005
		$8KB = 65536_b$	65522	14_b	1.0002
$5MB$	2	$1KB = 8192_b$	8164	28_b	1.0034
		$4KB = 32768_b$	32744	24_b	1.0007
		$8KB = 65536_b$	65514	22_b	1.0003
$384MB$	2	$1KB = 8192_b$	8152	40_b	1.0049
		$4KB = 32768_b$	32732	36_b	1.0011
		$8KB = 65536_b$	65502	34_b	1.0005
$384MB$	2	$4B = 32_b$	×	×	×
		$16B = 128_b$	76	52_b	1.6842
		$64B = 512_b$	462	50_b	1.108
$384MB$	30	$1KB = 8192_b$	7592	600_b	1.0790
	60		6992	1200_b	1.1716
	120		5792	2400_b	1.4144

Table 4.2 is the illustration of the relation of the designed parameters. We can find that O_{frag} reach 1 closely if the unit bandwidth is not extremely small, meaning the overhead is slight after fragmentation. However, if the bandwidth is quite small compared to the total data, it causes vital overhead due to the significantly large quantity of index bits carried on. If the case happens, such as the × sign in the table (implying that the required number of index bits is larger than the unit capacity can afford), we recommend the tradition routing

method. Another factor, the max degree of the LT code D, also influences overhead. We have to control the overhead to be adequate or reasonable for fear that even the well performed LT code can't compensate for the fragmentation overhead, and consequently lowers utility of whole network.

The required index bits can be derived from the formula introduced above. It varies due to different numbers of the transmitted packets. We provide two modes that index bits to record each fragment is either 16_b or 32_b. The former permits $2^{16} - 1$ transmitted fragments in total and the later permits $2^{32} - 1$. If we consider the case that total file size is $32MB$, unit capacity is $1KB$, and max degree is 10, the required index bits is 13_b, total overhead is 130_b, and the corresponding O_{frag} is 1.016. If we apply 16_b mode, the required overhead is 160_b, and O_{frag} is 1.02.

When data are fragmented, we cut the total file into pieces. At the same time, we assign the number to each slice as the packet ID. Thus, every non-coding packet can be viewed as a encoded symbol of degree one. Fig 4.3 is the example.

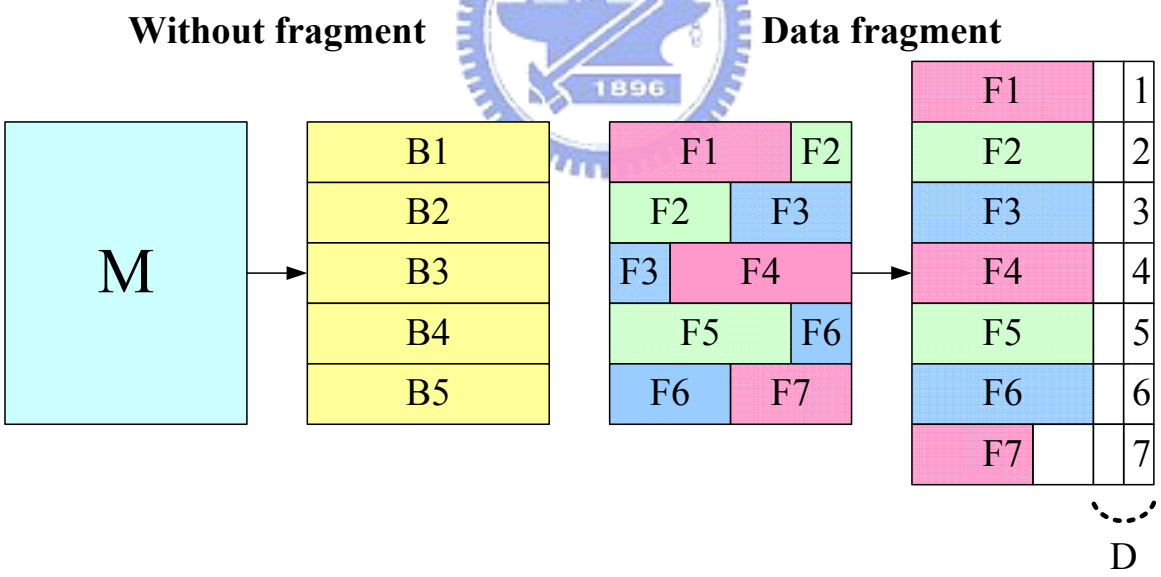


Figure 4.3: Data fragment

4.3 LT Encoding

Coding process is completely the same with the operations of the LT code encoding. A slight different is that since the packets contain the indices of the combined fragments, we need to reallocate the indices after finishing the encoding process. The detail discussion will be presented below.

Source node possesses total fragments also called symbols with degree one, and it can easily encode any symbol with requested degree. Encoding process is described as followed.

1. Randomly choose the degree d of this coding based on distribution.
2. Randomly choose the index of the N_{coding} uniformly in d iterations.
3. Do XOR operations of the fragments.
4. Reallocate the indices of the chosen fragments in order.
5. Repeat 3 and 4 until the degree of packet is d.

Figure 4.4: Illustration of an encoding symbol

Fig 4.4 is the example of an encoded symbol whose degree is 6. The max degree D is 10, and we set the residual indices to 0 in the unused index positions. In the same time, the index number we use is started from 1, not 0. The indices sequence is ordered from low to high during the encoding process.

4.4 Packets Transmission / Receiving

Packets are transmitted from nodes to nodes by edges with packet loss rate L. Since we transmit the data in the erasure channel, meaning that the we either lose the packets or believe every value we receive, we set the packet loss mechanism to point whether packet is lost. If mechanism occurs no loss, the packets will be sent to the adjacent nodes by the edges successfully, otherwise, the adjacent nodes will receive zero packets, representing the null transmission due to the loss. We create data to be all zeros to represent loss occurrence. Simulation result will be shown to compare the throughput and the ability of error protection in every sink with different packet loss.

4.5 Buffering

Buffer are used to store the packets from the incoming edges. It provide the temporary storage to preserve the data especially useful when the total incoming flow is larger than the total outgoing flow. The buffering method is obeyed on FIFO, fist in first out mechanism. In traditional routing, we should choose adequate size of buffer size for fear that the when bottleneck occurs in some intermediate, there will be considerable packet loss if the buffer is too small. When we apply LT code on network, the entire packets flooding the network are encoded symbols. If receiving rate is higher than transmitting rate of a certain node, some packets are definitely discarded. Entire packets are encoded symbols, obeying the designed distribution and therefore, even we lose some encoded symbols, causing the difference of desired distribution a source should conform, the difference will be subtlety tiny. That is to say, the buffer size will no longer a ignoring problem because of the coding mechanism and we can use smaller size of buffer to save the hardware cost, achieving well performed throughput as well.

4.6 LT Decoding

Decoding process resembles the LT decoding. A sink receives symbols continuously, meanwhile, activate the decoding procedure. We will receive and decode recursively until whole data are covered completely. The decoded symbol is one segment of the original data, therefore, we have to recover the true information by extra de-fragmenting operation. The procedure is listed below corresponding to a received symbol.

1. Check the symbol degree. If degree is one, step 2 ,else step 4.
2. Check whether this degree one symbol has been decoded before. If so, step 6, else step 3.
3. Defragment the new symbol, labeling it in decoded index list.
4. Check whether there exists the same indices between the symbol and the decoded index list. If yes, step 5, else, step 6.
5. XOR operation of two matched codewords, erasing the same index of the symbol. Return to step 1.
6. Finish.

The decoding process can be shortly summarized as searching the same indices among the decoded index list, XOR operation, and erasing the computed index to decrease the degree iteratively. Since the property of the LT decoding is vitally dependent on the degree one symbol, we should check two conditions to start the decoding. First condition is the new decoded index, and the other is the same degrees between this received symbol and the list of decoded indices. If the received symbol can not fit in with these two conditions, no decoding process is started up. In the meantime, after one coding process, there should exist

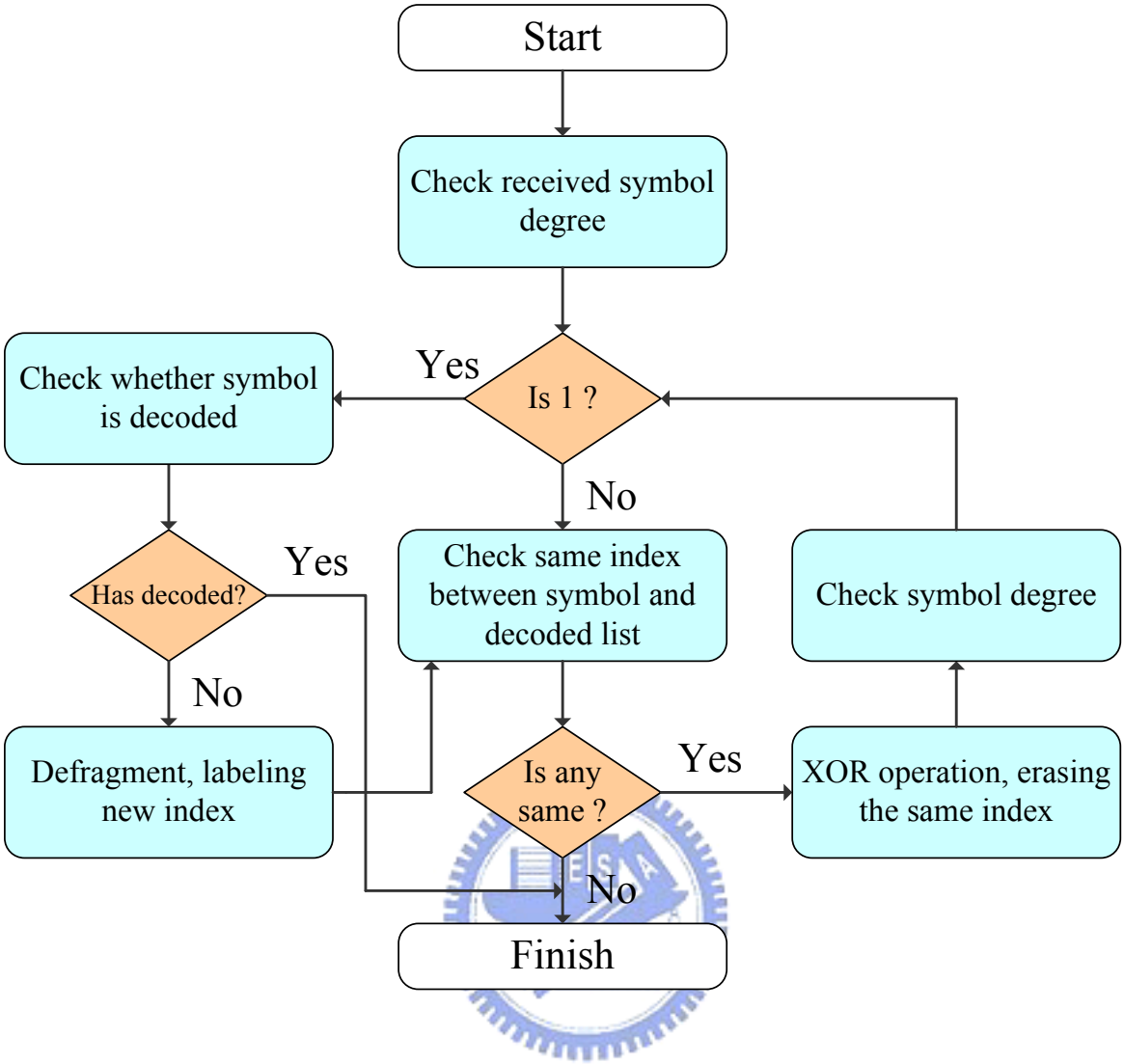


Figure 4.5: Decoding flow chart

no indices the same with any index in decoded list. That is to say, the decoding procedure stops if there is no valuable information among the un-decoded symbols. The flow chart is in Fig 4.5.

Decoding flow chart describes the decoding process when receiving one encoded symbol. Firstly, we need to check the degree of the new arrived symbol. If the degree is not one, we have to examine the index members of this symbol to see if we can reduce its degree by exclusive-or-ing the same index in the decoded list. If not, the decoding is finished leafing

the symbol which can not be decoded. If we find any symbol with index matched to any index member of the symbols, we extract its information, reducing the degree and return the degree check condition. If we discover the degree of received symbol is one, we firstly examine whether we have decoded this symbol before. If so, it represents this symbol is helpless for us to get more information and we stops the decoding procedure right away for fear that we spend much time searching whether there exists same index of this useless symbol. We should avoid the meaningless check. If this symbol is newly decoded, we have to search symbols with this new information to reduce the degrees of those un-decoded symbols to help the procedure go on.

4.7 Degree Distribution Analysis

The discussion above is established on the known well designed degree distribution. In chapter 3, We have studied how to obtain ideal soliton distribution and the robust soliton distribution modified by two additional parameter c and δ. Also, we realize the ratio of the degree one and degree max relevantly influence the decoding performance. If we want to get distribution by robust soliton distribution, the example below shows the corresponding procedure to find distribution with following parameters.

Table 4.3: Parameters of illustrated distribution

M	$10000KB$
B	$1KB$
$N_{original}$	10000
I	16_b
c	0.2
δ	0.05
F	
D	

We have cut file into 10000 pieces, and $N_{original}$ is 10000. We have to design the dis-

tribution and realize the max degree D can be decided by the K/S, the max degree used in robust soliton distribution. Assume we choose $c = 0.2$ and $\delta = 0.05$, we can therefore calculate out $S = 244$, $K/S = 41$, and $Z \simeq 1.33$. The corresponding distribution of $\rho(d)$ and $\tau(d)$ is showed in Fig 4.6. In Fig 4.6, we find that the modified term $\tau(d)$ adds the weighting mostly in max degree K/S, and in degree one. Weighting of every degree in distribution is the sum of $\rho(d)$ and $\tau(d)$. After the calculation, we get max degree D is 41, and the fragment size will be $1KB - 41 \times 16_b = 7356_b$. N_{coding} is therefore increased to 11137. Since LT code concerns the number of actual transmitted symbols, we have to put $N_{coding} = 10871$ to the calculation to get another required max degree and distribution, obtaining another new fragment size. The process will go on iteratively till the outcome converges. The final design will be $N_{coding} = 10917$ and $D = 43$.

If we obey the soliton distribution, we can tune a adequate one in sufficient tries. In our experience, we find that what we do care is the equivalent throughput in the sink. If we attempt to elevate the probability of successful decoding and reduce parameter Z with smaller transmission times, we must pay a high max degree for the coding system. It fatally decreases our ability to carry true information of one packet. The well designed distribution is usually hard to compensate for the N_{coding}, the actual transmission times.

Now that we concern most is the throughput, we hope to let N_{coding} is closer to $N_{original}$ as possible as we can. The intuitive thought is that the quantity of information carried on one packet should be larger. Our methodology is to limit the max degree D and we tune the distribution below our restriction. The goal we want to reach is quite the same, the difference is the way approach to it. If the degree we set up is too small, it is almost impossible for us to tune a adequate distribution. Therefore, we have to enlarge out max degree and tune it again. The process continues till the overall outcome of throughput get enhanced.

In our tries, we change the relative weighting of every degree. The most importance is

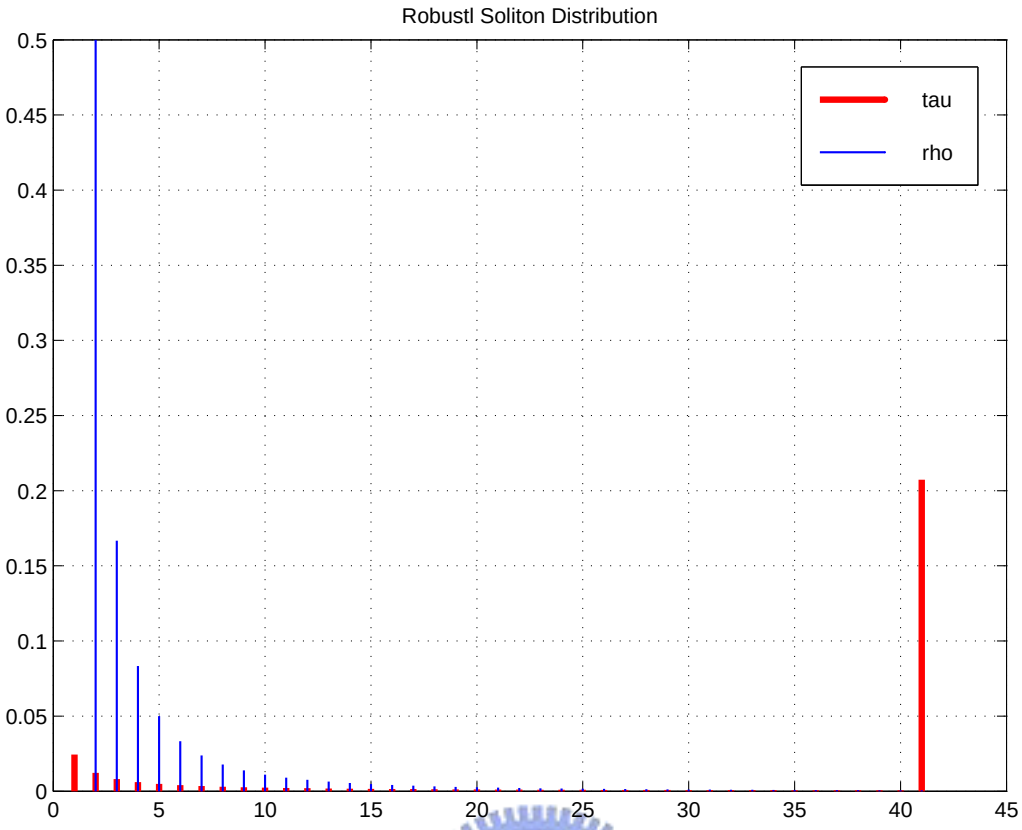


Figure 4.6: Robust Soliton Distribution with two components ρ and τ

to enlarge the distribution of degree one because it is the key to get decoding keep going by belief propagation. On the other hand, we lower the weighting of degree two because if we don't have sufficient degree one codewords, we have more degree two codewords in vain. The third modification is to lower the weighting the distribution of max degree. The max degree functions as the trunk of the code structure, seizing the total information. We let this task distributed to other degrees, and expect that we can alleviate the risk to be unable to decode because of the too many codewords with max degree. Hence, we get much lower distribution of max degree and let the weighting transferred to other degrees.

Chapter 5

Comparison and Simulation Result

In this chapter, we will show our simulations to verify our proposed method. We take two kind of network topologies we have introduced in chapter 2. Every network is operated in multicasting environment. The chapter will be separated to several parts according to different topics.

Every section contains the comparisons with different methods including traditional routing, coding method proposed in chapter 2, and proposed LT code applied to network coding. We simulate in the environment with packet loss rate 0%, 5%, 10%, and 20%. We compare every throughput of different methodology and calculate the total overhead of the coding system such as coding operations and decoding operations.

5.1 Throughput

In this section, we focus on the throughput of the network by different transmission mechanisms. We show the results of two different kinds of networks in Fig 5.1. Since the capacity of all channels is quite the same, we use $1KB$ as the unit capacity. On the other hand, LT code performs variously owing to different numbers of transmitted packets, we set up different size of file M from $4MB$, $8MB$, $16MB$, $32MB$, to $62.5MB$. Number of bits a

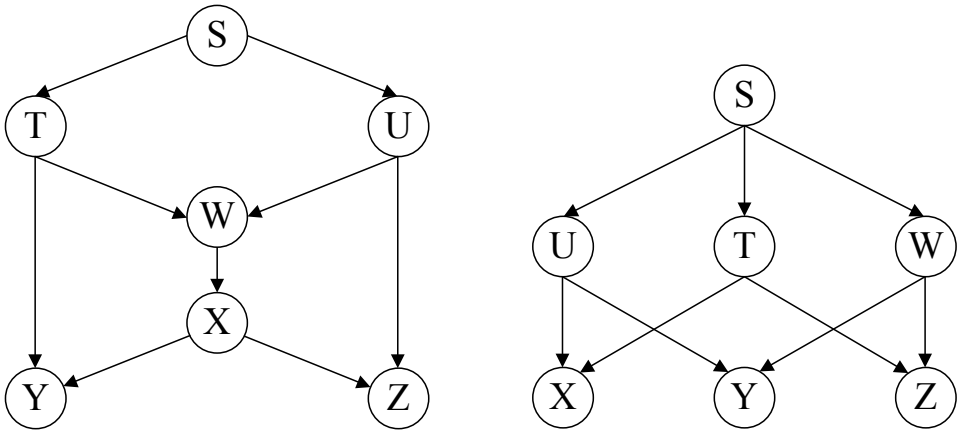


Figure 5.1: (a) Butterfly. (b) One-source three-sinks.

packet index uses is 16_b. The max degree we design here is 10 so that the overhead due to the fragmentation will be substantially tiny. The corresponding $N_{original}$ to each size of files is $4K$, $8K$, $16K$, $32K$ and $64K$. Buffer size of each intermediate is set to $1K$. Parameters and the relations between $N_{original}$ and N_{coding} are summarized as followed in Table 5.1 and Table 5.2.

Table 5.1: Parameters of simulation

M	$4MB$, $8MB$, $16MB$, $32MB$, $62.5MB$
B	$1KB$
$N_{original}$	$4K$, $8K$, $16K$, $32K$, $62.5K$
I	16_b
D	10
Buffer size	$1K$
L (loss rate)	0%, 5%, 10% , 20%

In our simulations, we compare three different methods. The first one is traditional routing method existed in the current system. The second one is network coding proposed in [1]. The third one is our proposed LT code applied to network environment. Since we do care is the utility of network of every sink, that is to say, if we use less cycles to let all sinks

Table 5.2: Relations between $N_{original}$ and N_{coding}

File Size M	$4MB$	$8MB$	$16MB$	$32MB$	$62.5MB$
$N_{original}$	$4K$	$8K$	$16K$	$32K$	$62.5K$
$N_{original}$	4178	8356	16711	33421	65275
O_{frag}	1.02	1.02	1.02	1.02	1.02

receive the data, the system has higher efficient utility. Results are showed in Table 5.3 to Table 5.6.

Table 5.3 and Table 5.4 show the average required run cycles in Fig 5.1(a) according to different levels of completeness in different loss rate. As the tables show, run cycles of routing is close to the required times a source should transmit when edges occur no loss. And we can also find that in lossless environment, results of the methoed proposed in [1] outperforms quite a lot compared to routing and proposed LT. It should be no surprise because this code is a specified design for this particular network so that it can performs outstandingly. However, when it suffers from different level of packet loss, we observe that routing mechanism and the specified design perform from bad to worse sharply. It means that the ability to protect packets from loss is quite insufficient so that we can not recover the lost packets from what we received, therefore, we require more cycles to transmit data to sinks. Regarding proposed LT, we find that required cycles to decode data are also increasing with severe loss. Since proposed LT code offer mechanism against packet loss, we can still decode a certain part of information when some packets are lost. Run cycles of proposed LT are significantly smaller than all the others especially when sinks receive 90% and 95% of entire data. Namely, if you can bear data loss that is from 5% to 10% or so, proposed LT supply a well constructed coding system to enhance throughput in the erasure channels that the worst loss rate scale up to 20%.

The result of Fig 5.1(b) is showed in Table 5.5 and Table 5.6. In this network, we get

similar outcomes displayed in former example above. Proposed LT can give more vigorous protection against more serious packet loss compared to other methods. The difference of run cycles among three methods are much larger than the result above. The reason causes such vital phenomenon is the structure of the network. We will discuss in detail later.

The tables listed above show the run cycles required to let every sink receive data according the different level of completeness. In order to comprehend how close we are to the theoretical upper bond, we normalizes results to show the equivalent throughput. Fig 5.2 shows the results.

We illustrate the throughput of the two networks with parameters that file size is $4MB$, and buffer size of every intermediate is $1K$. Numbers labeled form 10 to 100 in x-axis are completeness percentage of entire data and numbers in y-axis are the corresponding normalized throughput, meaning the equivalent decoded data per run cycle. We should note that when we calculate the equivalent throughput, we have to take the overhead due to the fragmentation into consideration. The required number for the real transmission enlarges because extra degree components occupy a certain part of every packet. When we calculate the effective throughput, we have to divide the throughput in the LT code system by the overhead to get normalized one. Because construct LT codes with a very small scale of degree, the overhead is therefore tiny, and the normalized throughput gets smaller in a tiny difference. In Fig 5.2-(a), we observe that the throughput is about 1.3 at most when during 90% to 95% of data are recovered. It gets 0.3 enhancement compared to store-and-forward system. The performance drops with the increase of the packet loss without surprise. Compared to the routing system, we find that the loss in every different level of packet loss is decreased smoothly, meaning decreased throughput due to packet loss is tolerable. We get the enhanced performance and degrades obviously only when we have to obtain the entire data. This is because when LT code get a sufficient data, un-decoded weights a significantly

small part of the data. Since the LT code encodes randomly merely based on distribution, we have no ability to control the system to send the combinations symbols with the un-decoded data . Another reason is that we use the smaller max degree in the code structure, it causes that the probability to carry different information what we need in our last decoding run cycle to obtain 100% data. Therefore, we have to transmit more time to finish the tail data to accomplish our task.

Results in Fig 5.2-(b) is quite similar to that in Fig 5.2-(a). The throughput gets upgraded steadily to the max 1.8 or so and also decreases when total data are recovered.

5.2 Buffer Size

Intermediates have adequate buffer to store incoming packets temporarily. The size of the buffer differs with the different application. In the traditional routing, when the data rate of the incoming data is higher than the outgoing data, intermediate can not afford for this congestion and suspend the request or lose some parts of data. If we don't want to lose any packet, we have to offer several times scale of total data according to the ratio of the total incoming rate to total outgoing rate. It is quite impossible and dispensable to fit with this requirement and we hope to have smaller needed size of buffer to lower the hardware cost. In our simulation, we find that when we apply LT code for network coding, the size of buffer affects performance slightly. In Fig 5.3, we show compare the throughput in buffer network with parameters that file size is $8MB$ and packet loss is 0%. The buffer size is from 32 to $16K$. We observe that the performance is quite similar in such a different buffer size. Buffer size with $16K$ is twice as the size of entire data, assuring that the encoded packet in the node that causes bottleneck can be obtained to delivery to the adjacent edges. The extremely small buffer size with 16 causes data overwritten in a considerable level. However, the entire system play a role as LT coding system so that even some encoded symbol drop,

the whole packets roughly abide by the LT code. LT code applied for network coding can lower the hardware cost.

5.3 Coding Overhead Analysis

We apply coding mechanism in the network to gain the benefit in throughput. In the same time, we have additional process such as encoding and decoding compared to the routing system. We show these overhead in Table 5.7 and Table 5.8. In these two tables, we find that the xor operation per codeword in [1] is 1 because they merely combine two packets in together. Decoding is increasing when the loss rate is higher. The xor operation per codeword in LT is approximately closer to the expatiation of the degree distribution minus one because we get degree d in $d-1$ xor operations. Xor operations in the decoding is about 24 to 30 or so. It is also increased due to the loss rate roughly. It varies because we calculate this when the total data is finished and there exists probability to get completeness in a extremely large cycle, increasing abnormal decoding operation. The total operation per codeword are the summation of two showed below.

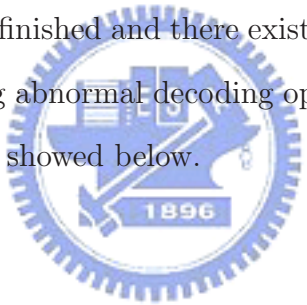


Table 5.3: Average run cycles of file size $4MB$, $8MB$, $16MB$ in butterfly network

$4MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Pass-and-Forward	1921	2770	3696	3899	4104
	NC in [1]	1441	1647	1852	1955	2059
	Proposed LT	∗2852	∗2857	2862	2985	6696
5%	Pass-and-Forward	3691	4027	4305	4940	11904
	NC in [1]	2990	3472	3915	4602	10025
	Proposed LT	∗3051	∗3070	3132	3417	6921
10%	Pass-and-Forward	3716	4286	4841	5945	16152
	NC in [1]	3228	3680	4493	4990	12558
	Proposed LT	∗3421	∗3424	3458	3586	8248
20%	Pass-and-Forward	3853	4479	6134	8305	22683
	NC in [1]	3728	4399	5676	7275	20487
	Proposed LT	∗4175	∗4196	4213	4423	11130
$8MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Pass-and-Forward	3840	5027	6874	7797	8208
	NC in [1]	2880	3291	3703	3908	4115
	Proposed LT	∗5660	∗5666	5672	6039	13870
5%	Pass-and-Forward	5743	6853	8135	8581	25304
	NC in [1]	4625	5157	6660	7971	18703
	Proposed LT	∗6142	∗6203	6264	6466	20752
10%	Pass-and-Forward	5965	7254	8527	11381	32196
	NC in [1]	4907	6278	7742	8506	24569
	Proposed LT	∗6889	∗6908	6926	7236	23584
20%	Pass-and-Forward	7794	8888	12964	15616	42774
	NC in [1]	7308	8391	11451	13970	41241
	Proposed LT	∗8344	∗8451	8487	8952	27910
$16MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Pass-and-Forward	7678	10055	13236	15082	16416
	NC in [1]	5759	6581	7404	7815	8227
	Proposed LT	∗11245	∗11271	11297	11991	45629
5%	Pass-and-Forward	9719	12274	15231	16628	48569
	NC in [1]	7785	8694	12352	14704	43935
	Proposed LT	∗10898	∗11728	12285	12989	46335
10%	Pass-and-Forward	10732	14258	16542	21469	62036
	NC in [1]	8352	12338	15396	16785	55055
	Proposed LT	∗13667	∗13699	13730	14681	64070
20%	Pass-and-Forward	13834	16379	24370	30632	103112
	NC in [1]	12996	15547	20809	25642	83602
	Proposed LT	∗16917	∗16937	16958	17767	64912

[1] Number labeled ∗ is the value calculated by interpolation.

Table 5.4: Average run cycles of file size $32MB$, $62.5MB$ in butterfly network

$32MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Pass-and-Forward	15353	20109	26471	29652	32832
	NC in [1]	11515	13160	14805	15628	16451
	Proposed LT	∗22624	∗22644	22665	23966	93450
5%	Pass-and-Forward	17816	24464	30013	32539	115502
	NC in [1]	14172	15982	24160	29450	85504
	Proposed LT	∗24711	∗24804	24556	26291	122334
10%	Pass-and-Forward	21061	27128	33378	44135	134121
	NC in [1]	16471	22367	29602	33502	107365
	Proposed LT	∗27365	∗27423	27481	28973	138472
20%	Pass-and-Forward	26704	33091	47781	61709	230788
	NC in [1]	24815	30309	41238	50706	168448
	Proposed LT	∗33526	∗33422	33319	34893	142634
$62.5MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Pass-and-Forward	29984	34267	38551	40693	42384
	NC in [1]	22489	25702	28914	30521	32128
	Proposed LT	∗44344	∗44370	44396	46715	183967
5%	Pass-and-Forward	32573	36938	41317	54754	208561
	NC in [1]	27528	31054	45031	56027	192734
	Proposed LT	∗47767	∗47803	47840	50595	188459
10%	Pass-and-Forward	35231	39696	55741	74609	262800
	NC in [1]	31289	41682	56760	63385	244616
	Proposed LT	∗52658	∗52722	52787	55651	206357
20%	Pass-and-Forward	40605	56119	80482	107524	432963
	NC in [1]	47602	58983	81493	100596	366815
	Proposed LT	∗64119	∗64348	64578	68200	247981

[1] Number labeled ∗ is the value calculated by interpolation.

Table 5.5: Average run cycles of file size $4MB$, $8MB$, $16MB$ in one-source three-sinks network

$4MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Store-and-Forward	1920	2194	2468	2605	2472
	NC in [1]	1440	1646	1851	1954	2057
	Proposed LT	∗2110	∗2121	2132	2269	7683
5%	Store-and-Forward	3002	3279	3558	4165	10265
	NC in [1]	2573	2794	3055	4180	6428
	Proposed LT	∗2304	∗2330	2356	2490	8293
10%	Store-and-Forward	3061	3351	3992	5381	16238
	NC in [1]	2712	3001	4060	4523	8846
	Proposed LT	∗2579	∗2608	2619	2739	8595
20%	Store-and-Forward	3255	4118	5394	7632	20094
	NC in [1]	3251	3951	4507	5298	12604
	Proposed LT	∗3275	∗3270	3284	3461	9999

$8MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Store-and-Forward	3839	4387	4935	5209	5484
	NC in [1]	2879	3290	3702	3907	4113
	Proposed LT	∗4206	∗4220	4235	4473	12993
5%	Store-and-Forward	4951	5511	6068	7752	16441
	NC in [1]	4150	4593	5723	7335	14687
	Proposed LT	∗4681	∗4692	4702	4935	17473
10%	Store-and-Forward	5071	5645	7638	9644	25652
	NC in [1]	4404	5338	7131	8030	19076
	Proposed LT	∗5172	∗5177	5182	5467	18680
20%	Store-and-Forward	5419	8085	10173	13376	36002
	NC in [1]	5579	7592	8705	10506	29786
	Proposed LT	∗6457	∗6476	6495	6811	23040

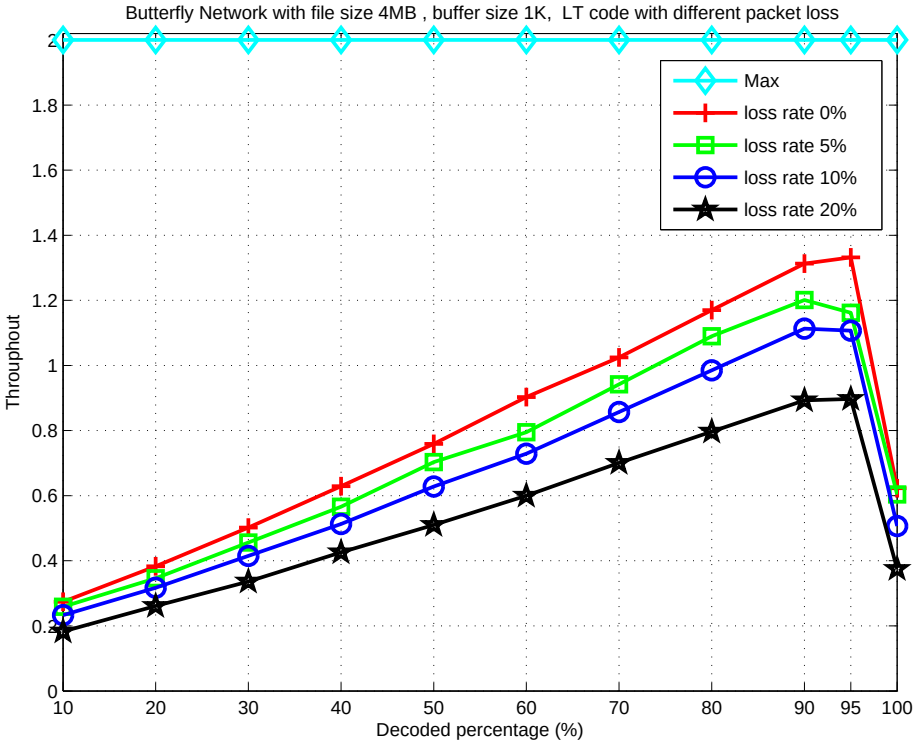
$16MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Store-and-Forward	7677	8773	9870	10418	10966
	NC in [1]	5758	6580	7403	7814	8225
	Proposed LT	∗8405	∗8439	8474	8881	25803
5%	Store-and-Forward	8862	9980	11098	14433	45436
	NC in [1]	7284	8176	10487	13725	29536
	Proposed LT	∗9368	∗9389	9410	9816	28895
10%	Store-and-Forward	9082	10229	14928	19191	65454
	NC in [1]	7868	9683	14013	15818	36915
	Proposed LT	∗10340	∗10373	10406	11014	31975
20%	Store-and-Forward	10545	14763	20603	27343	101599
	NC in [1]	10884	13769	16454	20152	64317
	Proposed LT	∗12838	∗12841	12844	13506	39112

[1] Number labeled ∗ is the value calculated by interpolation.

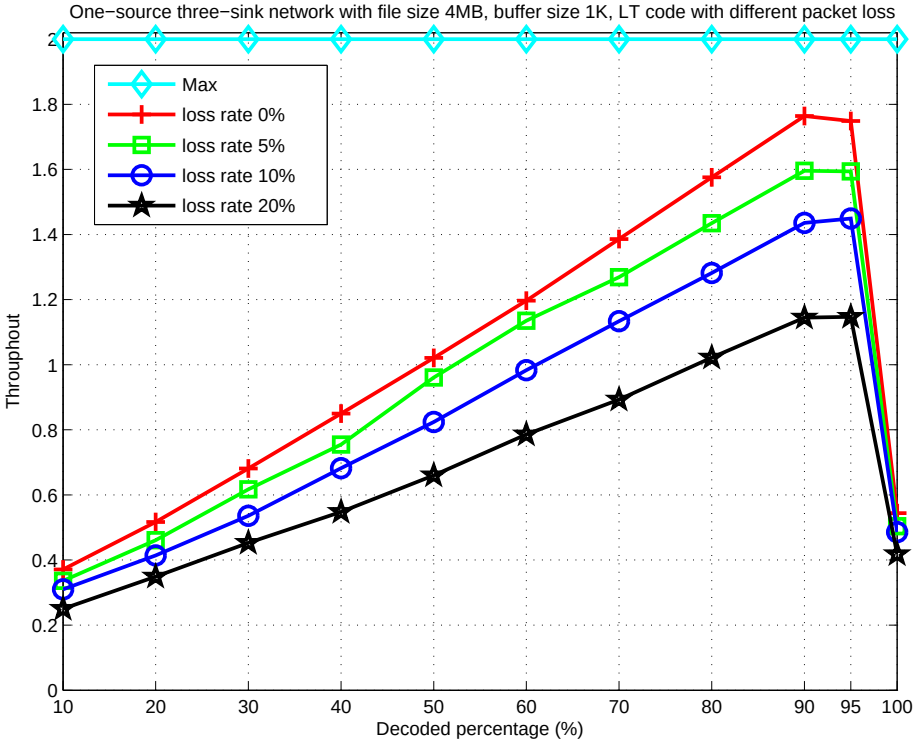
Table 5.6: Average run cycles of file size $32MB$, $62.5MB$ in one-source three-sinks network

$32MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Store-and-Forward	15352	17545	19739	20835	21932
	NC in [1]	11514	13159	14804	15627	16449
	Proposed LT	∗16909	∗16924	16940	17755	61314
5%	Store-and-Forward	16704	18934	21169	28400	73766
	NC in [1]	13549	15342	19664	27342	66295
	Proposed LT	∗14630	∗16671	18711	19639	68529
10%	Store-and-Forward	18059	20344	28236	37478	122955
	NC in [1]	15639	19168	26791	30655	85904
	Proposed LT	∗20833	∗20860	20886	21858	75871
20%	Store-and-Forward	20865	28737	39864	48351	177511
	NC in [1]	21767	26477	32586	39536	139863
	Proposed LT	∗25817	∗25841	25865	27212	94598
$62.5MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	Store-and-Forward	29984	34267	38551	40693	42834
	NC in [1]	22488	25701	28913	30520	32126
	Proposed LT	∗33088	∗33107	33126	34858	128781
5%	Store-and-Forward	32573	36938	41317	54754	208561
	NC in [1]	26445	29949	38746	51962	119185
	Proposed LT	∗36277	∗36389	36501	38532	156108
10%	Store-and-Forward	35231	39696	55741	74609	262800
	NC in [1]	29491	36567	51571	59501	185834
	Proposed LT	∗40502	∗40541	40580	42515	166623
20%	Store-and-Forward	40605	56119	80482	107524	432963
	NC in [1]	41650	51688	63119	77028	278323
	Proposed LT	∗49976	∗50009	50043	53043	211449

[1] Number labeled ∗ is the value calculated by interpolation.



(a)



(b)

Figure 5.2: Normalized throughput of two networks

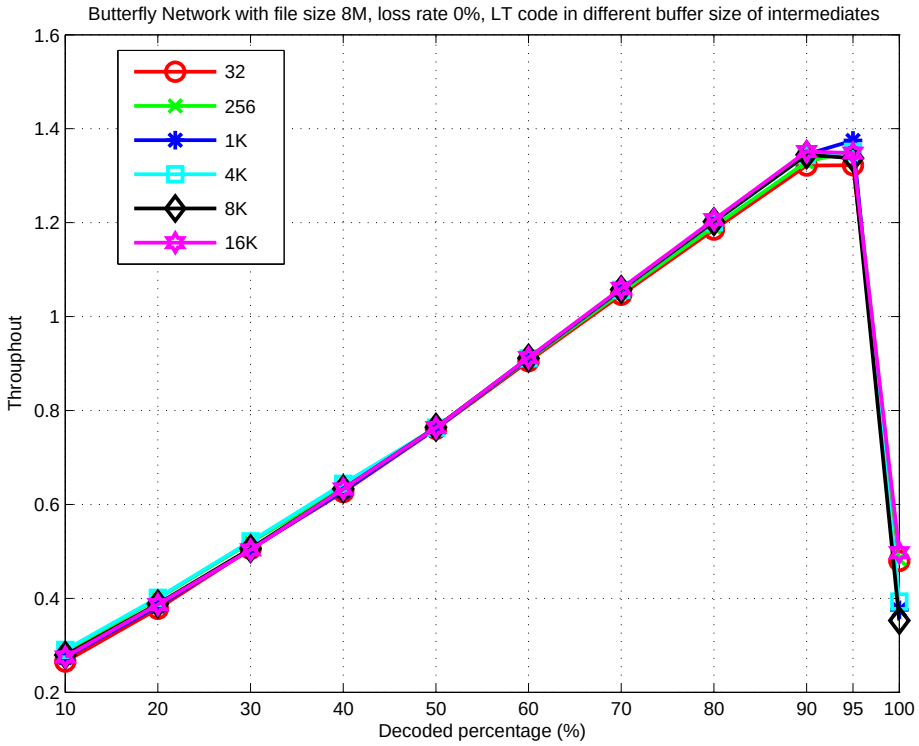


Figure 5.3: Butterfly network with different buffer size

Table 5.7: XOR operations of coding systems in Fig 5.1(a)

Items	Method	File Size is $4MB$ Loss Rate 0%	5%	10%	20%	File Size is $8MB$ Loss Rate 0%	5%	10%	20%	File Size is $16MB$ Loss Rate 0%	5%	10%	20%
Encoding	Proposed in [1]	1	1	1	1	1	1	1	1	1	1	1	1
	LT	2.43	2.42	2.42	2.40	2.40	3.41	2.40	2.42	2.40	2.41	2.40	3.40
Decoding	Proposed in [1]	1	4.09	4.32	5.06	1	3.87	4.32	5.57	1	4.57	4.88	5.35
	LT	21.20	21.74	23.83	22.67	25.02	27.94	22.24	21.18	24.05	23.05	28.16	27.92
Total	Proposed in [1]	2	5.09	5.32	6.06	2	4.87	5.32	6.57	2	5.57	5.88	6.35
	LT	23.63	24.16	26.25	25.07	27.42	31.35	24.64	23.60	26.45	25.46	31.56	31.32

Items	Method	File Size is $32MB$ Loss Rate 0%	5%	10%	20%	File Size is $62.5MB$ Loss Rate 0%	5%	10%	20%
Encoding	Proposed in [1]	1	1	1	1	1	1	1	1
	LT	2.41	2.40	2.39	2.40	2.40	2.40	2.40	2.40
Decoding	Proposed in [1]	1	4.44	4.79	5.44	1	5.19	5.60	6.07
	LT	24.42	39.43	30.84	24.65	25.86	25.11	26.12	28.47
Total	Proposed in [1]	2	5.44	5.79	6.44	2	6.19	6.60	7.07
	LT	26.83	41.83	33.23	27.05	28.26	27.51	28.52	30.87

[1] Each value is the average exclusive-or operations per codeword.

Table 5.8: XOR operations of coding systems in Fig 5.1(b)

Items	Method	File Size is $4MB$ Loss Rate 0%	5%	10%	20%	File Size is $8MB$ Loss Rate 0%	5%	10%	20%	File Size is $16MB$ Loss Rate 0%	5%	10%	20%
Encoding	Proposed in [1]	1	1	1	1	1	1	1	1	1	1	1	1
	LT	2.40	2.40	2.41	2.41	2.40	2.41	2.40	2.40	2.41	2.40	2.40	2.40
Decoding	Proposed in [1]	1	2.84	3.55	4.18	1	3.20	3.61	5.05	1	3.20	4.34	5.53
	LT	25.00	24.04	25.18	27.65	24.96	25.73	26.50	29.35	29.13	26.35	23.81	26.52
Total	Proposed in [1]	2	3.84	4.55	5.18	2	4.20	4.61	6.05	2	4.20	5.34	6.63
	LT	27.40	26.44	27.59	30.06	27.36	28.14	28.90	31.75	31.54	28.75	26.21	28.92

Items	Method	File Size is $32MB$ Loss Rate 0%	5%	10%	20%	File Size is $62.5MB$ Loss Rate 0%	5%	10%	20%
Encoding	Proposed in [1]	1	1	1	1	1	1	1	1
	LT	2.40	2.40	2.40	2.40	2.40	2.40	2.40	2.40
Decoding	Proposed in [1]	1	3.20	4.34	5.53	1	3.32	4.18	5.27
	LT	24.77	38.97	28.01	26.52	25.59	33.63	33.33	34.67
Total	Proposed in [1]	2	4.20	5.34	30.68	2	4.32	5.18	6.27
	LT	27.17	41.37	30.41	33.08	27.99	36.03	36.73	37.07

[1] Each value is the average exclusive-or operations per codeword.

Chapter 6

Conclusion and Discussion

6.1 Conclusion

In the thesis, we propose LT code applied for network coding in the multicast network. The proposed method enhances network throughput by 20% to 30% improvement at most when transmission completeness is in the range from 90% to 95%. LT code can also perform well against the loss rate form 0% to 20% with compared to the routing system. Our proposed LT-Network codes alleviate the required buffer size in the intermediates to maintain good performance such that the buffer size becomes negligible factor regardless the data size. Finally, the additional coding overhead for encoding and decoding is about 30 XOR operations per codeword.

6.2 Discussion

In our simulation, we observe what varies significantly in two different networks is the gap of throughput between the real and the theoretically max flow. In butterfly network, throughput enhances but falls behind the theoretical max in a large scale. However, the result in another network is much closer the theoretical bound. The reason causes such fatal difference is that there exists edge disjoint paths in the transmission network. Edge disjoint paths is

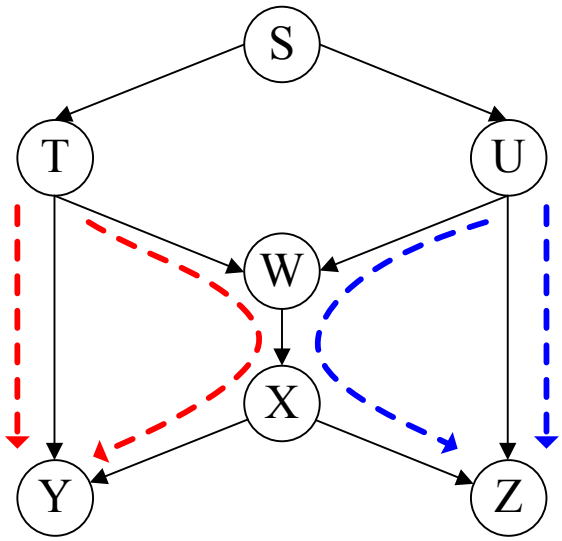


Figure 6.1: Edge disjoint paths in butterfly network

defined as that two paths from node u to v have no common internal edge. In the Fig 6.1, we find that there exist two paths from node T to Y, path TY and path $TWXY$. The sink Y get packets from these two path. Because we allow intermediates pass and forward, node T send two copies what it receives to both edges in the starting of two paths. When sink Y receives a packet from T, after a certain cycle runs, it will get the same packets from XY which originate from the node T. Although it will get some partial of additional information from the path $UWXY$, it still can not avoid receiving the repeated packets what it has obtained. The same condition is displayed in the node Z because of the edge disjoint paths UZ and $UWXZ$. The utility of network is therefore degraded due to this phenomenon.

Regarding to another network, we observe there exists no edge disjoint paths in the network. Therefore, efficiency of the LT code can gain high enhancement approach to the theoretical max flow. Since throughput degradation is due to the repeated codewords, we propose a combined LT-Network to let node W do exclusive-or operations of two codewords it receives. By doing this, every receiver has opportunity to get additional information from

the re-combined codewords instead of the completely same codeword it has received. Re-combined codewords bear more useful information to help each node get higher probability to process decoding procedure, therefore, to decode successfully. When we apply LT-Network code we should notice that we design the LT code with the max degree D. If the combination of two codewords in node W results a codeword with degree larger than the max degree, we drop this combination to pass the codewords in the buffer. The result is showed below.

We find that the required run cycles get further decreased if we apply LT-Network Codes in this network. However, this LT-Network Codes is designed for this specified network. In our simulation, we give a startup for LT-Network codes in erasure channel to exemplify the improvement and good ability to protect data. The further research can focus on how to get a well combined LT-Network in generic situation.

Table 6.1: Average run cycles with LT-network codes of file size from $4MB$ to $62.5MB$ in butterfly network

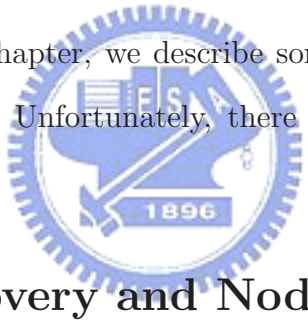
$4MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	LT	∗2852	∗2857	2862	2985	6696
	LT-Network Code	∗2670	∗2671	2672	2673	11953
10%	LT	∗3051	∗3070	3132	3417	6921
	LT-Network Code	∗2912	∗2929	2945	2961	10681
20%	LT	∗3421	∗3424	3458	3586	8248
	LT-Network Code	∗3257	∗3273	3290	3353	11514
30%	LT	∗4175	∗4196	4213	4423	11130
	LT-Network Code	∗4235	∗4265	4296	4326	12812
$8MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	LT	∗5660	∗5666	5672	6039	13870
	LT-Network Code	∗5269	∗5279	∗5289	5300	34565
5%	LT	∗6142	∗6203	6264	6466	20752
	LT-Network Code	∗5770	∗5781	5792	5848	24726
10%	LT	∗6889	∗6908	6926	7236	23584
	LT-Network Code	∗6500	∗6503	∗6506	6512	25962
20%	LT	∗8344	∗8451	8487	8952	27910
	LT-Network Code	∗8276	∗8297	8319	8341	28631
$16MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	LT	∗11245	∗11271	11297	11991	45629
	LT-Network Code	∗10380	∗10432	∗10484	10536	41372
5%	LT	∗10898	∗11728	12285	12989	46335
	LT-Network Code	∗11826	∗11853	∗11881	11908	50941
10%	LT	∗13667	∗13699	13730	14681	64070
	LT-Network Code	∗13059	∗13076	∗13093	13110	63194
20%	LT	∗16917	∗16937	16958	17767	64912
	LT-Network Code	∗16550	∗16580	∗16611	16642	70597
$32MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	LT	∗22624	∗22644	22665	23966	93450
	LT-Network Code	∗20735	∗20807	∗20879	20951	98520
5%	LT	∗24711	∗24804	24556	26291	122334
	LT-Network Code	∗23319	∗23373	∗23427	23482	78833
10%	LT	∗27365	∗27423	27481	28973	138472
	LT-Network Code	∗25905	∗25973	∗26040	26108	153647
20%	LT	∗33526	∗33422	33319	34893	142634
	LT-Network Code	∗32970	∗33034	∗33090	33146	101303
$62.5MB$ Loss Rate	Method	Percentage of Decoded Data 70%	80%	90%	95%	100%
0%	LT	∗44344	∗44370	44396	46715	183967
	LT-Network Code	∗41122	∗41129	∗41135	41141	254453
5%	LT	∗47767	∗47803	47840	50595	188459
	LT-Network Code	∗45268	∗45436	∗45604	45772	217532
10%	LT	∗52658	∗52722	52787	55651	206357
	LT-Network Code	∗51307	∗51333	∗51359	51385	197169
20%	LT	∗64119	∗64348	64578	68200	247981
	LT-Network Code	∗64552	∗64611	∗64569	64708	268342

[1] Number labeled ∗ is the value calculated by interpolation.

Appendix A

Several Ideas to Transmit Packets Efficiently

We have discussed the repeated codewords phenomenon in some edges disjoint paths started from some node and ended at sinks in the network. It fatally decreases the throughput because that sinks get some packets received before at certain ratio, carrying no beneficial information for decoding. In this chapter, we describe some thoughts that we attempt to resolve this troublesome problems. Unfortunately, there is no any breakthrough for this topic.



A.1 Drawback Discovery and Node Analysis

Network with edge disjoint paths ended at sink node will causes harmful influence on throughput regarding coding mechanism. In the previous discussion, we find that even we apply LT code on network to let the whole system paly as a coding mechanism, if we does not let intermediates participate in encoding process, and merely pass-and-forward as traditional routing does, the efficiency will be restricted by the network topology. The way to avoid this limitation is to let intermediates encode as the source does during transmission.

Before we start, we should check whether every intermediate need coding. If not all the

intermediates are necessary, we can save the extra coding overhead such as computations and time. At this subchapter, we will clarify the necessity of coding for any intermediate. Considering the butterfly network in Fig A.1, we observe there exist two edge disjoint paths

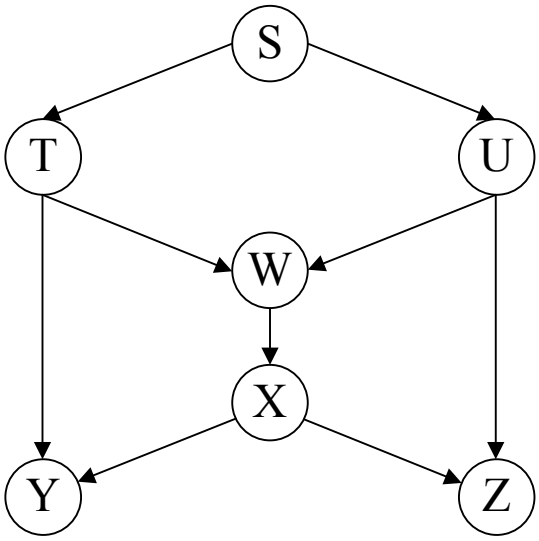


Figure A.1: Butterfly network

from node T to sink Y, path TY and path $TWXY$. Node T has one unit capacity of all incoming edges, and two unit capacity of all outgoing edges. Number of received packets can not keep up with that needed for transmission. In this situation, we have to copy what we receive to delivery for separate edges, causing sink Y receive a certain part of repeated packets. On the other hand, if one intermediate have more packets than what it transmits, there is no need for coding because of the sufficient information even it cause edge disjoint paths.

The conditions we judge whether a intermediate needs coding are based on relations between total incoming packets and outgoing edges and if it causes edge disjoint paths. However, it is tiresome to find edge disjoint paths and indicate such nodes. This task is NP-hard so that we have to modify the methods to decide coding or not.

We conclude the necessity for coding in intermediates as followed.

- A intermediate needs coding if total capacity of incoming edges is larger than that of outgoing edges.

Based on this condition, we find intermediates require coding are node T, U, and X. Node W can do simply pass and forward contracted to original method. We can also observe that node X have to do coding even if it causes no edge disjoint paths. By this constraint, every node can decide coding or passing regardless to the whole network.

A.2 Intermediates Coding

A.2.1 Thought

Since we can decide every work a node should follow, we discuss how to do coding on intermediates. Remember that we apply LT code in transmission system, the entire environment can be viewed as a LT coding structure. If we let intermediates code, we should follow the LT distribution as well. So before we start transmission, we should send information of LT distribution to them.

Intermediates get information from the incoming edges and can not obtain entire data as a source does. Coding can be merely depends on what we store in the buffer. So, we start coding mechanism after receiving some packets. Encoding procedure is the same with that in the source.

A.2.2 Phenomenon

The result is not as we expected. We check degree of every packet received in sinks and find that the distribution differs seriously compared what we design in the lossless circumstance.

That is because intermediates code according to desired distribution and we have to encode the packets by the picked degree chosen according to distribution. We set a high stop criteria until the degree is what we need in this encoding cycle. If we can't produce the codeword with desired degree, we randomly choose a packet in the buffer to transmit. Since the degree of packets in the buffer is biased, we can not encode freely causing a quite large difference between true distribution and designed distribution and the result performs poorly.

A.2.3 Note

Let intermediate encode as possible as it can in a finite stop criteria causes considerable computations and disturbs the degree distribution.

A.3 Codeword Cache

A.3.1 Thought

We encounter significant disorder of degree distribution due to the coding in intermediates. Therefore, our task is to avoid the intermediate being probably not to encode the desired degree in a high opportunity.

At every encoding, we choose a degree randomly $d_{desired}$ based on distribution. If we can't make it, procedure will continues till the the desired one shows up. During this long encoding loop, there exists a codeword that its degree is not what we need. The quantity of these codewords will increase if we can't produce codeword with degree d_{coding} matched to $d_{desired}$. If we save these codewords in the buffer, we obtain packets with various degrees except $d_{desired}$. When the next encoding cycle runs, we can firstly search in the buffer to find whether any packet in buffer with the degree matched to this new degree $d_{desired}$.

We store codewords during coding loop and we search the buffer firstly in the next

encoding cycle. The recommended way to allocate the packets in the buffer is cut buffer into separate parts according the degree distribution. Therefore, we can save searching time to find the degree.

Table A.1: Buffer allocation

Degree	1	2	3	4	...	18	19	20
Distribution	5	4	1	1	1	1	1	5
Ratio	0.3			0.5			0.2	
Buffer store	0 to 14			15 to 38			38 to 47	

Table A.1 is a illustration. Max degree is 20, and every value in distribution row is the weight of particular degree. Ratio shows the percentage of total degree in three region. Buffer size is 48 and we cut the buffer into three parts according to the percentage of the degrees in each region. We find that degrees one to three contain 30% of the distribution, therefore we cut 30% of buffer labeled from 0 to 14 to store those degrees. When we want to store a new created codeword in the buffer, we need to check the degree to put it in the relative position. The method that deal with the data is still FIFO mechanism, meaning that if this region is filled with the codewords, we overwrite it from the first position. The received data is stored by the higher priority. By doing so, packets in the buffer are updated to obtain the newly incoming information received in the buffer. Fig A.2 demonstrate the encoding flow chart.

Note that when we search the degree of packets, we choose the first one whose degree is equal to $d_{desired}$. Since we use FIFO in each sub buffer, we can lower the opportunity to send the same packet we have transmitted before. In the meantime, we create another table attached to every packets in the buffer to label whether this codeword is transmitted before. If this packet has been transmitted, we labels it so that if the same degree is required in the next time, this packet will be skipped. At the same time, when we can not encode what require, we still have to choose one packet to deliver. In this condition, we will uniformly

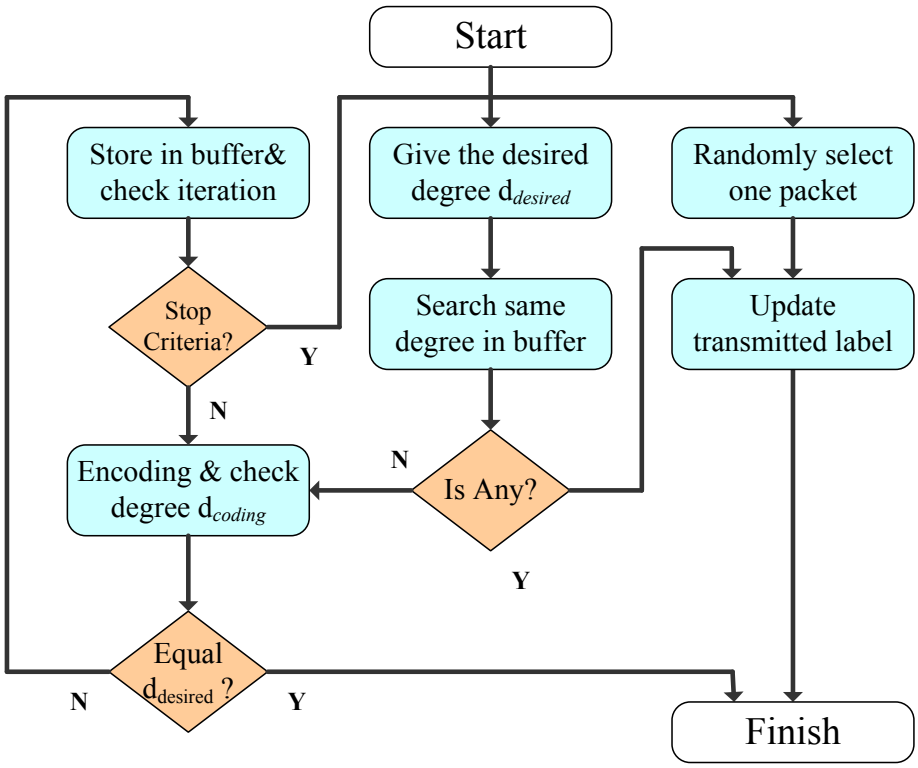


Figure A.2: Encoding flow chart of encode and store mechanism

select one packet in the buffer and label it as identification. We illustrate labeling in Fig A.3.

Assume that node 1 has three outgoing edges and the selected $d_{desired}$ is 2 for each edge with capacity 1. That means we have to send three codewords with degree 2 during the coding in node 1. B_trs_flag is used for labeling and 1 represents the transmitted packet. Table in the left side shows the information of buffer before coding. Number from 0 to 47 means the position of buffer. The second column is the labeling identification. The third column shows degree of every codeword in the buffer. Considering the coding procedure in the node 1, we need to transmit three packets. In the first cycle, $d_{required}$ is 2 and we search the degree finding that there exists one packet with degree 2 in position 1 is not transmitted. We choose this packet to transmit through edge to node 2, labeling it 1. In the next cycle, $d_{required}$ is 2 again. Since packet in position 1 is labeled transmitted, we have to look for

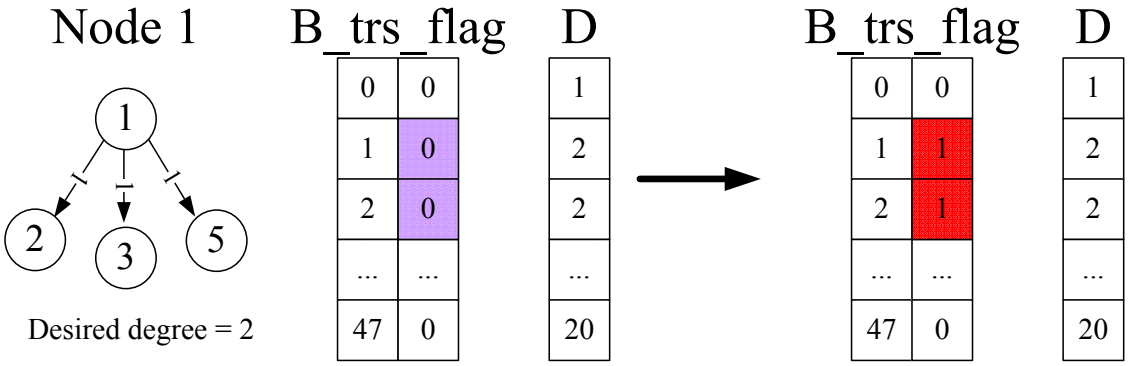


Figure A.3: Illustration of labeling operation

the next one and we find that packet in position 2 fits in with the conditions, selecting and labeling it. In the final cycle, the $d_{required}$ is still 2 and we can not find any packet after the searching so that we have to do coding to produce the codeword we desire.

A.3.2 Phenomenon

By this modification, we observe that the exclusive-or operations is decreased dramatically. When we does not store the new encoded codewords, the number xor operations times is up to the stop bound we setup closely. It shows how tough to encode one codeword in limited information. After our modification, the average xor times to encode one codeword is down less than 10 with a constrain that stop criteria equals 5. Stop criteria means that if we can't encode the desired codeword in 5 cycles, we quit encoding procedure and randomly chose one packet from the whole buffer.

This modification does not work. Distribution we get in the sink gets a slightly improved but its improvement is insufficient to resolve the problems to enhance the throughput. We analyze the distribution and we see that we get considerable packets with degree from D to $D-2$ where D is the max degree and extraordinarily less packets with degree 1 and 2. Since we use belief propagation that number of packets with degree 1 is the vital key to decode

the whole information. We slightly balance the overall distribution but the most valuable part is still biased.

Besides, we find a fatal condition that we obtain a large quantity of the same codewords after our modification. There are two kind of steps to create repeated codewords during the coding procedure.

1. Same operation.
    - Choose a set that cause equivalent combination.
    - This condition occurs apparently when the buffer is spare of packets.
2. Inverse operation.
    - The operation procedure is equivalent to the inverse operation of any past operations.

We describe them separatively.

Fig A.4 explains the same operation. The left table is a tail part of packet recorded the relation how packets combine and we call these information degree list. Number labeled from 0 to 9 is the buffer index. Recall that we store packets based on its degree. The relation between them is in Table A.2

Table A.2: Buffer cut of example

Degree	1	2	3	4	...	6	7	8
Buffer store	0 to 1		2 to 7				8 to 9	

We demonstrate equivalent combinations after two xor operations in a encoding cycle. Assume $d_{desired}$ for this cycle is 3 and we firstly initialize the elements to 0 in degree list. In the first run, we get 0 and 4 to do computation and we get a new codeword with degree 5.

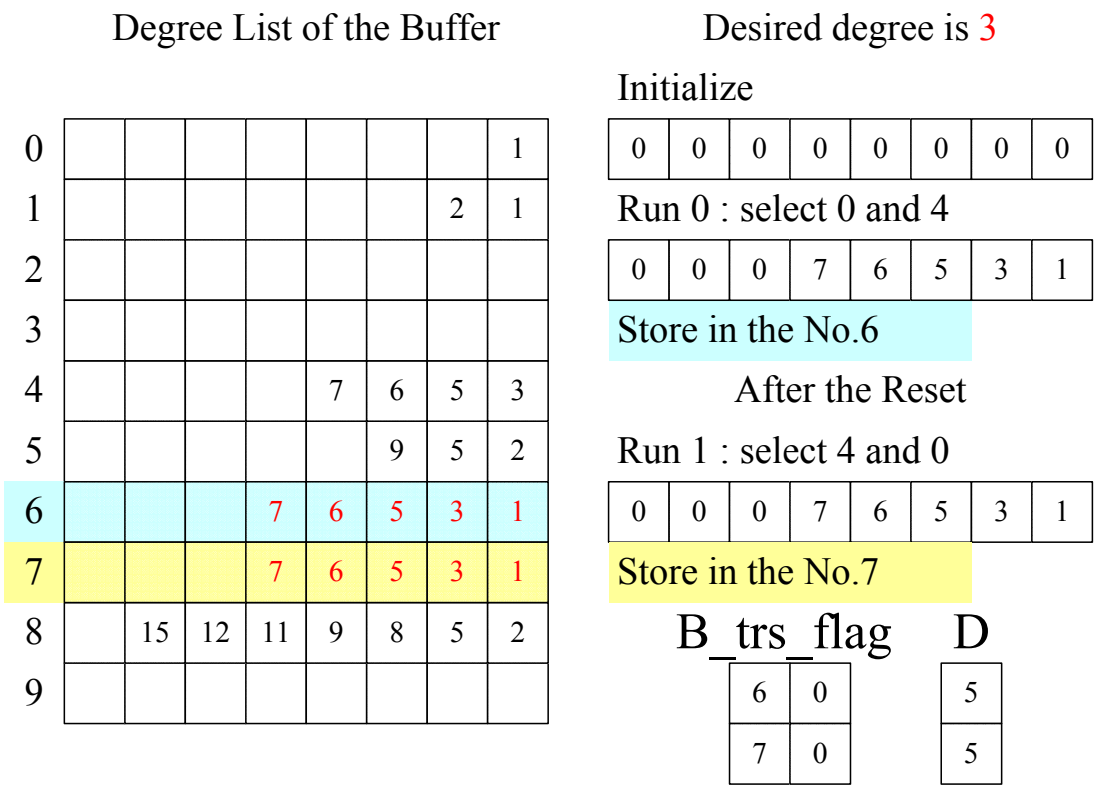


Figure A.4: Same operation while coding

Since 5 is not our desired degree, we store it in the buffer index 6 and continues encoding till codeword degree is larger than max degree or we get required one. Suppose that packet in the buffer index 6 is not overwritten by others and this coding run ends due to degree explosion. We reset the degree to 0, starting another run. Unfortunate, we choose buffer indices 4 and 0 that the outcome is quite the same with we encode in the last run. We store it in another empty buffer indexed 7. Obviously, there are two same packets in the buffer.

We shows another case in Fig A.5.

The basic assumption is the same with the case in Fig A.4. The only difference is the chosen sets of two coding run. In the first run, we still choose no.0 and no.4 to compute and the result is stored in no.6. In the next run, if the chosen indices are no.0 and no.6, we find

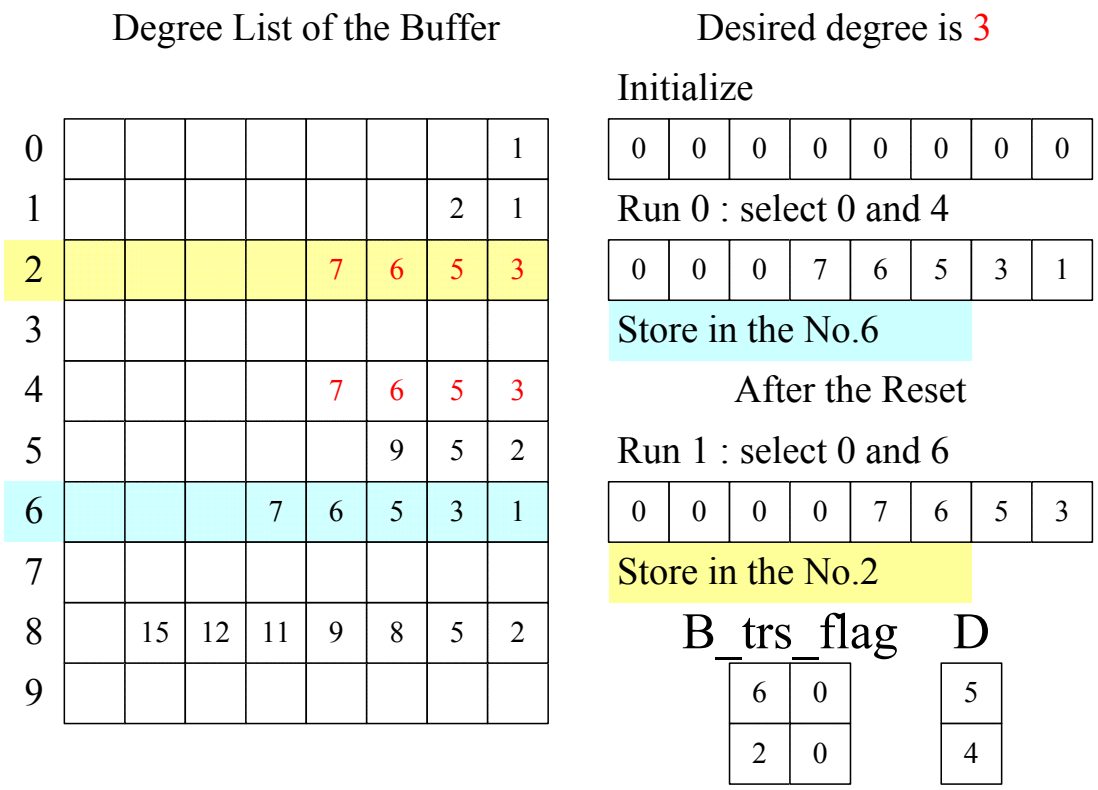


Figure A.5: Inverse operation while coding

that the outcome is the same with that in no.4, a codeword we have already obtained. In such a case, we contain the same packets again.

These two kinds of operations cause serious result. It is probably that the combinations of codewords in the buffer are occupied by some particular codewords. If one repeated codeword is stored in the buffer, the opportunity that this codeword is picked up when we uniformly choose one for encoding is much higher than others. If the outcome of the operations involving these repeated codewords result in more repeated codewords unfortunately, packets in the whole buffer will obtain a certain specified information, causing tragedy. We illustrate in Fig A.6.

Fig A.6(a) show the degree of each packets in the buffer. We observe that there exists

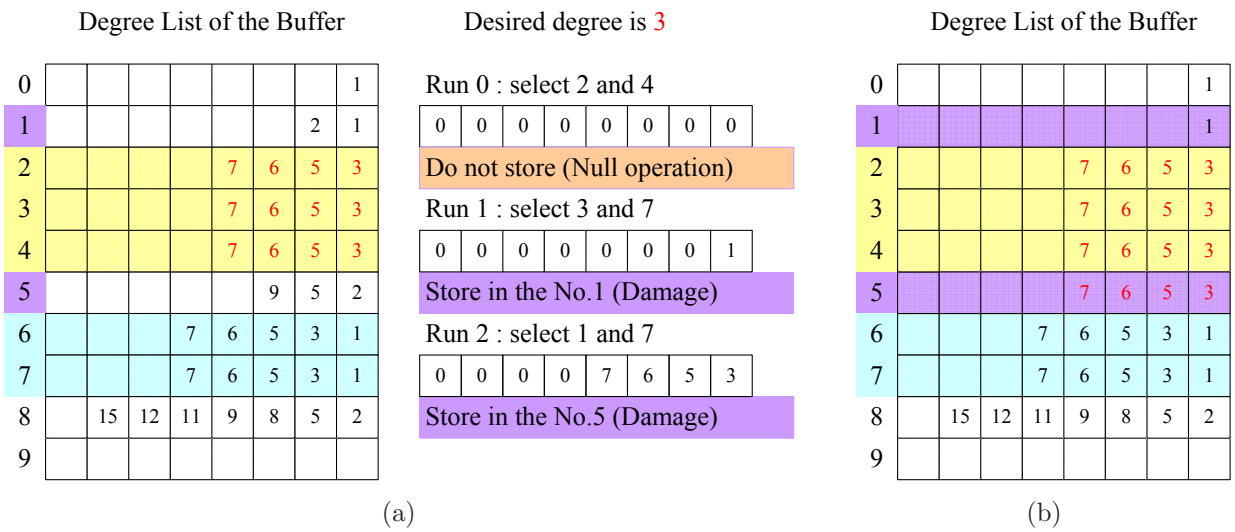


Figure A.6: (a) Step I. (b) Step II.

two kinds repeated degree combinations stored in no.2 to no.4, and no.6 to no.7. If we have three coding runs marking in the right. The first causes the null operation because the outcome is zero due to the two packets with the same degree list. It costs our coding time for this useless operation. In run 2, since packet from no.2 no.4 is the same, we have higher probability to select this codeword since it weights a ratio of 3/9 of total buffer. The outcome after operation in run 1 should be stored in the first region of buffer. If the FIFO mechanism in this region points it to store in no.1, we will overwrite the existed codeword. The condition after in run 2 resembles that in run 1. After the two runs, we find that the variety of the degree list is significant reduced, meaning the information loss. Now that number of the degree category is decreasing from 6 down to 4. The ability to encode a codeword with different degree list degrades vitally. Namely, we will encode more same packets. Once a certain number of repeated are stored in the buffer, the higher probability we choose them during coding, causing more same packets in the buffer due to two kinds of operation described above. The situation goes from bad to worse quickly, hence, we call this avalanche damage.

A.3.3 Note

Labeling the transmitted packet in the buffer can not avoid avalanche damage, causing repeated codewords as well.

A.4 Repeated Codeword Table

A.4.1 Thought

The repeated codewords can not be solved if we just control in the buffer because the same codewords may appear again even we label it in the buffer. In order to eliminate this drawback, we retain that in every transmission.

What bothers fatally the same codewords lowering the utility. We have to assure every packet sent into network possesses different degree components. Thus, we create a look up table to record what we have transmitted and we have to check the codeword degree list in every transmission. That is, we set up a guard to manage every packet outgoing to the network.

At the same tine, LT code places importance on the distribution and it does not care much about whether the codeword of a particular degree is encoded early or late as long as the distribution a sink receives the same with what design. This property gives us a opportunity to lower the xor operation times, speeding up the encoding. Fig A.8 illustrate the modified encoding flow chart.

Compared to Fig A.2, we add three blocks including Check d_{coding} in distribution table, Update distribution table, and Update repeated LUT.

Distribution table is used to record the difference between the desired degree distribution and real coded one. We explain the functionality in Fig A.7.

1	2	3	4	5	⋯	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0

Desired degree = **5** ; Real degree = **3**

1	2	3	4	5	⋯	16	17	18	19	20
0	0	**+1**	0	**-1**	0	0	0	0	0	0

Figure A.7: Distribution table

The top table in the figure is the initiation with max degree 20. If the desired degree of one encoding run is 5 and that of the real transmission is 3. We will update the table by adding 1 to degree 3 and subtracting 1 to degree 5. Thus, the positive number represents the extra quantity that we transmit at this degree compared to the desired one. Similarly, negative number represents lack and 0 represents even. The previous method to treat the packet that d_{coding} is not equal to $d_{desired}$ is merely to store it. If we are short of this kind of packet that we should have transmitted before, we have to compensate for that in some transmission somehow to modify the distribution. So, when we encode a degree that is not match to $d_{desired}$, we should check the table to see if we should transmit this codeword to resupply. If we find number of d_{coding} in the distribution table, we can therefore transmit it and update it by adding one in d_{coding} and subtracting one in $d_{desired}$. Although the desired degree is discarded after this transmission, there still have chance to compensate for it in another coding procedure. By doing this, we can save the operation time and balance the distribution as well.

On the other hand, if we can not create the codeword and find that there is no loss in the distribution table, we still have to choose a packet in the buffer. In this situation, we also have to update table.

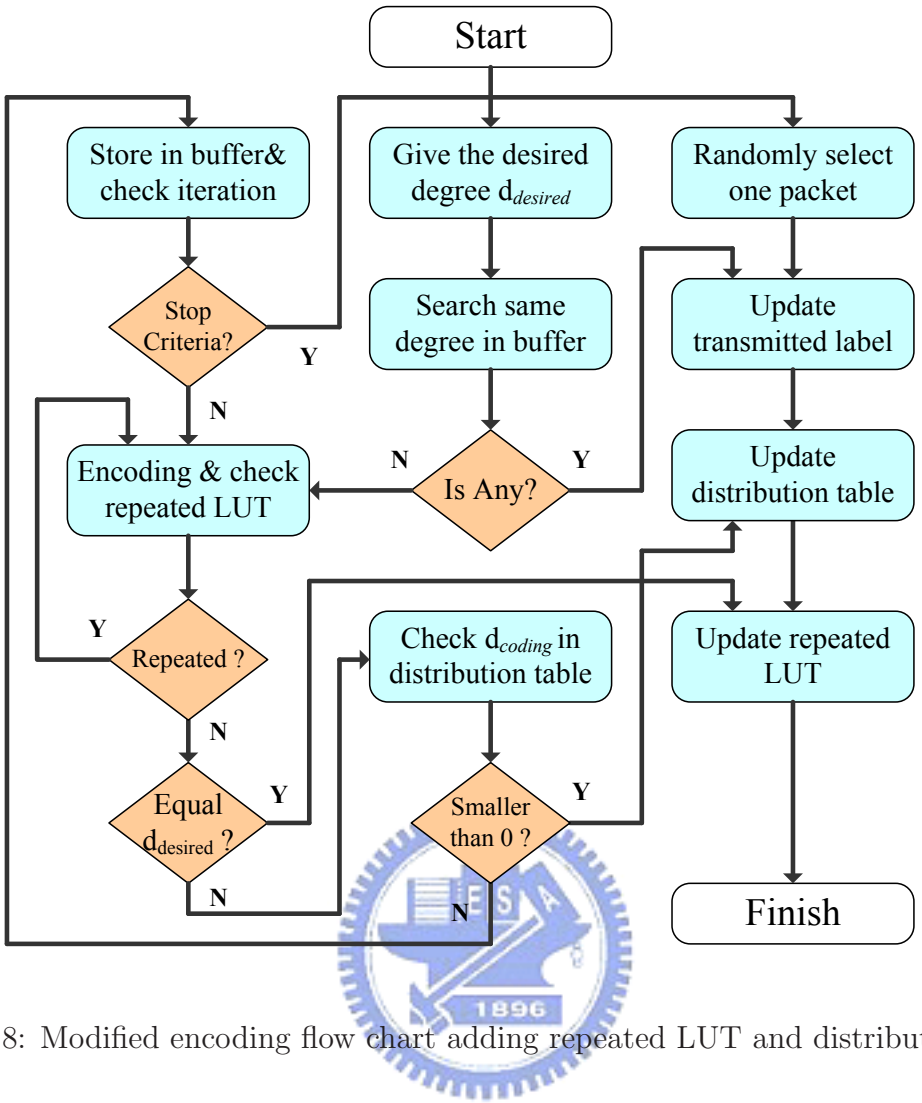


Figure A.8: Modified encoding flow chart adding repeated LUT and distribution table

The repeated look up table is used to record every degree list we have transmitted up to now. As long as we decide the packet we want to transmit at this coding cycle, we have to record it. Another condition is that we have to check repeated LUT when a new codeword is created. If we find this codeword is in LUT, we will discard it and encode again. According to this additional restriction, we can avoid any repeated codeword in the buffer.

Because we have no idea when will the transmission ends, we have to use a dynamical storage to record repeated codeword. Besides, in order not to waste lots of time in checking repeated LUT, we arrange the degree lists in the LUT in the order based on degree. And

we will record the boundary of each degree to indicate the start point and end point for checking.

A.4.2 Phenomenon

After this modification, we find that the result still doesn't get improved. We find that the distribution of every sink get tiny difference proving that the distribution performs well and the codewords are not repeated. Why the result don't get enhanced?

We look back to the concept of the relation between original data and encoded codewords. Since the coding can be viewed as the linear mapping between two space. However, we hope that the dimension of codeword space should be as large as the original data that we can recover them by some computation. However, when we coding only in the source, we find that the packets we create get pretty high opportunity to span a space as original data. Of course, the efficiency get max if your encoded space is identical to the original space, meaning the encoded codewords are linearly independent. When we apply coding in the intermediates, we can simply encode based on the received symbols. The scale of space we can span will enlarge with the more received packets in the buffer. However, the buffer size is limited and we may do coding times to times in a coding cycle, causing the packets in the buffer resemble to each other. Even we can encode non-repeated codewords, the packets we send are too similar that we can not offer enough information for sinks to recover data.

A.4.3 Note

Coding in the intermediates cause that the whole packets are with higher dependence and can not offer enough information for sinks.

A.5 Constraints Alteration on Intermediates

During the discussion above, we do coding in the intermediates that the total incoming capacity is larger than total outgoing capacity. We discover that we can not span out larger space to enhance the throughput. Now that we want to carry more information, we change the condition to do coding in the intermediates where the bottlenecks occurs. At the meantime, we try another case that we let every intermediate do coding in the whole system during transmission. The corresponding results are showed below.

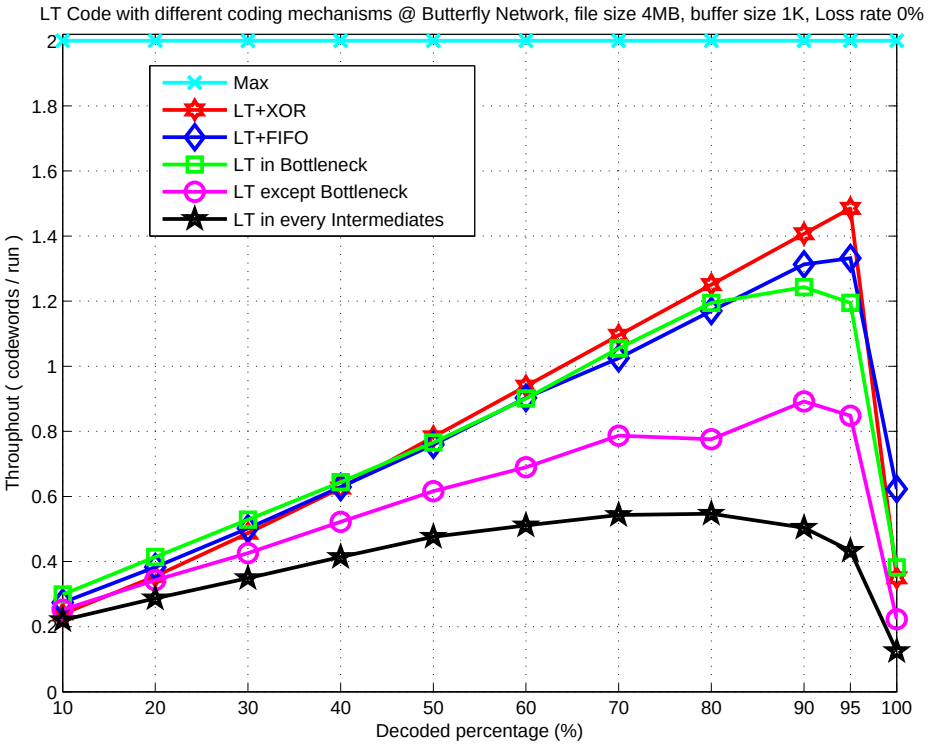


Figure A.9: LT-network code with different coding mechanisms

We show four kinds of coding mechanisms in Fig A.9. Every method we propose here is that if the intermediates need coding, they will obey the LT degree distribution we design. The first FIFO+LT is the method let node W do sample XOR operations between two received codeword symbols, and we can see that the equivalent throughput is the best com-

pared to other mechanisms. Only FIFO mechanism let intermediates do pass-and-forward as routing does. Coding in the bottleneck is that we do coding only in the node W where there are two incoming edges but only one outgoing edge. Coding except bottleneck let intermediates do coding except the node W. The final mechanism let every intermediate do LT encoding during the process. We find that if we do coding except node W in order to avoid the repeated codewords delivery due to the edge disjoint paths, the throughput doesn't get enhanced but lower. The result seems get slightly improved in the case that we do coding in the node W when the completeness of every sink is in the range from 0% to 80%. However, it drops down eventually and can not upgrade the throughout approach to the theoretical maximum. In the last method, we let every intermediate do coding. We find that the result get much worse than we expected. The reason we conjecture is that the encoding process in the intermediates causes the code structure mixed up so that what the sink receive will become some pieces of the shorten and split code structure. The connection of the codewords weakens so that even the received codewords in the sinks differ a tiny minority, the damaged LT code can not achieve high throughput.

A.6 Summary

We have tried hard to resolve the repeated codewords problems to hope we can give a generic solution to enhance throughput and work in vain. The described method above are offered as the experience. We point out the problems to degrade the throughput while applying LT code of whole network in our experiments. We give some direction for further study on this issue. Any discussion and command is welcome.

Bibliography

[1] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, Network information flow, *IEEE Trans. Inf. Theory,* vol. 46, no. 4, pp. 1204-1216, Jul. 2000.

[2] R. W. Yeung, S.-Y. R. Li, N. Cai, Z. Zhang, *Theory of Network Coding* .

[3] T. Ho, R. Koetter, M. Médard, D. R. Karger and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," *IEEE International Symposium on Information Theory,* 2003.

[4] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast, "*IEEE Transactions on Information Theory,* vol. 52, no.10, pp.4413-4430, 2006.

[5] D. B. West, *Introduction To Graphy Theroy,* -2nd ed, pp. 176-180, N.J.:Prentice-Hall, 2001.

[6] T. H. Cormen, C. E. Leisersin, R. L. Rivest, and C. Stein, *Introduction to Algorithm* -2nd ed, pp.651-691, NcGraw-Hill, 2001.

[7] D. J. C. MacKay, "Fountain codes",*IEEE Proc.-Commun,* vol.152, no.6 , pp.1062-168, Dec 2005.

[8] A. Shokrollahi, "Raptor Codes," *IEEE Transactions on Information Theory,* vol.52, no.6 pp. 2551-2567, Jun. 2006.

[9] M. Luby and "LT Codes," *Proc. 43th IEEE Symposium on Foundations of Computer Science,* pp.271-282, Nov. 2002.

[10] R. Koetter and M. Médard, "An Algebraic Approach to Network Coding," *IEEE/ACM Transactions on Networking,* vol. 11, no. 5, pp.782-7952, Oct. 2003.

[11] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network coding," *IEEE Transactions on Information Theory,* vol. 49, no. 2, pp. 371-381, Feb. 2003.

[12] C. Fragouli and E. Soljanin, "A connetcion Between Network Coding and Convolutional Codes," *IEEE Transactions on Information Theory,* vol. 2 pp. 661-666, Jun. 2004.

[13] C. Fragouli and E. Soljanin, "Information Flow Decomposition for Network Coding," *IEEE Communications Society,* vol. 52, no. 3, pp. 829-828, March. 2006.

[14] C. Fragouli, E. Soljanin, and A. Shokrollahi, "Network Coding as a coloring Problem," *Proc. Conf. Information Sciences and Systems,* Mar. 2004.

[15] H. Wang and C.-C. J. Kuo, "Robust video multicast with joint network coding and AL-FEC," *Proc. IEEE International symp. Circuits and Systems,* May. 2008.

[16] H. Zhu, C. Zhang and J. Lu, "Designing of Fountain Codes with Short Code-Length," *Proc. 3rd International Workshop. Signal Design and Its Applications in Communications,* pp. 65-68, Spet. 2007.

[17] M. Langberg, A.Sprintson and J. Bruck, "Network Coding: A Computational Perspective," *40th Information Sciences and Systems Conference,* March. 2006.

[18] Z. Li, B. Li, D. Jiang and L. C. Lau, "On achieving optimal throughput with network coding," *Proc. IEEE Joint Conf. IEEE Computer and Communications Societies,* vol. 3, pp. 2184-2194, Mar.2005.