Order Number 9119620

Dynamic security

Chung, Ilyong, Ph.D.

City University of New York, 1991

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A

DYNAMIC SECURITY

by

ILYONG CHUNG

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfilment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1991

The manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

January 30, 1991
Date

Michael Anshel
Chair of Examining Committee

January 30, 1991
Date

K. Wendkos
Executive Officer

Professor Michael Anshel
Professor Dipak Basu
Professor Ted Brown
Professor Michael Conner
Professor Kimyeong Lee
Professor Jacob Rootenberg
Professor Richard Tolimieri

Supervisory Committee

The City University of New York

Abstract

Dynamic Security : Parallel Communication and Secret Routing Algorithms

by

Ilyong Chung

Advisor: Professor Michael Anshel

Data security can be divided into two categories - static security and dynamic security. Static security is the security of the data itself and dynamic security is the security of the route for the data transmitted. If the route is detected by an adversary, it is very likely that the data will be intercepted. Therefore, the route must be protected. To accomplish this, we select an intermediate node secretly and transmit the data using this intermediate node, instead of sending the data to the destination node using the shortest direct path. The above route consisting of two paths - a path from the source to the intermediate node and a path from that intermediate node to the destination - is called a secret route. Furthermore, if we use a number of secret routes from the source to the destination, data security is much stronger since we can transmit partial data rather than the entire data along a secret route. To employ the above idea, the data is dispersed into a number of pieces. In this thesis, the data is dispersed into n

pieces using the technique of information dispersal called the dispersal algorithm using the FFT algorithm(DAF).

We choose the n-dimensional hypercube network as the network model for transmission of the data. Each piece is transmitted to the destination along its own secret route in the n-dimensional hypercube network. In order for all the pieces to reach the destination node in the minimum possible time, all the routes should be disjoint. We show how to construct a set of disjoint paths from the starting to intermediate nodes by employing the modified no same-entry matrix(MNSEM) and then show how to construct a set of disjoint paths from these intermediate nodes to the destination by employing the partial Hamiltonian circuit latin square(PHCLS). Later, This idea is extended to secret routing algorithm for the mixed radix number system(MRNS) network.

Dedicated
To the Memory of My Father
CHUNG, JUNG-TAE
1925-1975

Acknowledgments

I would like to express my gratitude to my thesis committee menbers: Professors Michael Anshel, Dipak Basu, Ted Brown, Michael Conner, Kimyeong Lee, Jacob Rootenberg and Richard Tolimieri.

My first thanks go to Professor Michael Anshel, my thesis supervisor. After taking courses with Professor Anshel for five years, I've learned a tremendous amount about the theory of Computer Science. He introduced me a broad and deep knowledge of computer science and guided me in directions that led me to new interests in this field. When I constructed the routing algorithm for the hypercube network, he encouraged me to generalize my ideas in order to make the new routing algorithm applicable to a larger class of networks. Professor Anshel provided me with all the theoretical background that I needed to design this algorithm. He was always ready and willing to discuss my dissertation - even late at night(12 A.M.) or weekends. He always tried to understand my meaning and intention, which was not always easy because English is not my first language. Also, I am very thankful to Mrs. Anshel. She always made me feel welcome whenever I worked with Professor Anshel at their apartment. I especially appreciate Mrs. Anshel's generosity, her delicious hot tea, and her selection of beautiful classical music.

I am also grateful to Professor Jacob Rootenberg. He advised me to strengthen my knowledge of computer hardware for real-world applications. In addition, he encouraged me to adapt my ideas for more practical applications. I thank him for giving me his comments after reading my thesis carefully.

I thank Professor Ted Brown. He provided me with an office at Queens College so I could concentrate on my studies. In this office, I read a considerable number of papers and completed my dissertation. Since Professor Brown understood clearly the entire content of my doctoral thesis, he was able to provide me with many insightful comments about my work.

I would like to thank Professors Dipak Basu, Michael Conner, Kimyeong Lee and Richard Tolimieri for their participation as committee menbers at my doctoral defense.

I am thankful to Ron Skurnick. In order to edit my dissertation, he sacrificed lots of time during his summer vacation. He sat with me for countless hours and painstakingly modified any grammatical construction within my thesis that he felt might be unprofessional. "Ron, I shall never forget you".

I thank my colleagues and friends: Jaehong Lee, Prashant Joshi, Mohammed Nizauddine, Vasilios Keramaris, Hansoo Na, Seongyong Oh, Heakyeong Na, David Shyu, Kisong Yoon and Ching-Bin Wang for their

support over the years. I especially thank the Computer Science Department at Queens College. This department allowed me to use all the facilities it owns. Last, but not least, I thank my mother, wife, brothers, sisters and relatives for their emotional and financial support, and for their constant prayers for my successful completion of my doctoral degree.

Contents

1

Introduction

When we transmit data in a distributed network, we must consider data security. Data security can be classified into two categories: static security and dynamic security. Static security is the security of data itself. This security problem concerns the following question: How can the original data be encrypted by the sender and decrypted by the receiver so that both the encryption and decryption algorithms are efficient and reliable, and yet secure from passive or active attack by adversaries. Some well-known algorithms for this type of security are the *DES* algorithm[1] and *the RSA* algorithm[2]. The *DES* algorithm, implemented via the classical cryptography techniques of transposition and substitution, is used by banks for money transfers and by federal bureaus for data transfers. The *RSA* algorithm is implemented using arithmetic properties of modular number system. It is still considered powerful because there is no efficient algorithm for large integer factorization.

Dynamic security is the security of the route taken by the data transmitted over the network. If the route is detected by the adversary, it is very likely that the data will be intercepted. Therefore, the route must be protected. To accomplish this, we select an intermediate node randomly and transmit the data using

this intermediate node, instead of sending the data to the destination node by the shortest direct path. The above route consisting of two paths - a path from the starting node to the destination node and a path from that intermediate node to the destination node - is called a secret route. Furthermore, if we disperse the message into pieces and use a number of secret routes from the source node to the desired node, data security is much stronger since we can transmit partial data rather than the entire data along a secret route. However, the time complexity for setting up these routes is greater than the corresponding time complexity for a single secret route.

In addition to secrecy, we are interested in maintaining a short path in order to transmit the data as quickly as possible. A shortest path algorithm may generate a set of disjoint paths, but, a shortest path can not always be good with respect to secrecy because the shortest path can be easily computed in some cases. Therefore, dynamic security requires that we compromise on the shortest path and requires that the data should be transmitted to the destination node quickly and safely.

As described above, the data is sent to the destination along a number of secret routes. At the destination, the data should be reconstructed, even if some problems occur in the network. This leads to the question of network reliability. To solve this problem, computer scientists suggest dispersal algorithms which after the original data is broken into n pieces, are able to

recover the original data using at least (n-k) pieces, where k is the number of missing pieces. These dispersal algorithms increase the size of the original data. An efficient algorithm will make the total size close to the size of the original data.

An important algorithm used to break the data into n pieces is the one proposed by Shamir. Although Shamir's method[3] is designed to share secrecy, it can be used as a dispersal technique. Shamir's method leads to an n-fold increase of the original data size. It becomes efficient as a dispersal algorithm with respect to the time complexity only when Newton interpolation is employed. Shamir's method handles only a single fixed length message at a time. Since numerous data items to be managed in real-world applications, research has continued on the handling of multiple data items at a time. Asmuth and Blakley[5] present an algorithm that employs the Chinese Remainder Theorem[6]. This algorithm has better space efficiency than Shamir's, but the arithmetic computation is much more complicated. Another dispersal algorithm is the information dispersal algorithm(IDA) was developed by Rabin[7]. The IDA employs matrix-theoretic properties. As far as space complexity, this algorithm is better than Asmuth and Blakley's. Also, it makes the computation simpler.

In this thesis, we propose a method called ***the DAF(Dispersal Algorithm using the FFT Algorithm)*** which employs the FFT algorithm[9] and matrix-theoretic properties. Given a file F(a message) and (n x n) matrix M, where M satisfies the conditions needed for the FFT

algorithm, this algorithm using the FFT algorithm disperses F into n pieces. Each piece is transmitted along a secret route. The file F is reconstructed from any (n-k) pieces, where k is the number of missing pieces. For the process of reconstruction, The DAF requires one more step than the IDA. This step for recovering the missing pieces involves generating linear equations using properties of the FFT algorithm(which will be described later) and then solving these equations by the inversion of a matrix. Then, the original file is reconstructed via the FFT algorithm. The total storage requirement is $|F|*n/(n-k)$, and the time complexity is max $(O(n^2*\log_2 n), O(n^2*k))$. This algorithm improves the time complexity and the space complexity as the channel becomes error-free. If a network has error-free links, the time complexity of $O(n*k)$ is negligible. Hence, the time complexity will be $O(n^2*\log_2 n)$.

To apply this dispersal algorithm to the network the transfer time should be considered. If a node receives two pieces of data that are to be forwarded, one of the two pieces will have to wait for its turn. In order to avoid any waiting time, each path should be disjoint from all other paths. An efficient parallel algorithm is developed in order to transmit all pieces of data to the destination in the minimum possible time. A question arises: What kind of machines are suitable for the application of the DAF? There are fundamental decisions that must be made to determine the appropriate architecture of the machines[10]. Generally speaking, the decisions concern the selection of the control strategies, the

switching methodologies, and the network topologies. The parameter space of these machines can be visualized as the cartesian product of the above three sets of design features. In addition to these decisions, our application requires the transmission of multiple data items to their destination along secret routes.

Owing to its structural regularity and its high potential for the parallel execution of various algorithms, hypercube computers have drawn considerable attention in recent years from both academic and industrial communities[14]. The n-dimensional hypercube is composed of concurrent, loosely coupled parallel processors whose coupling is based on a binary n-cube topology. The n-dimensional hypercube consists of 2^n identical processors. Each of the processors, provided with its own sizable memory, is connected through bidirectional, point-to-point communication channels to n different neighbors to form a communication network. Because of its low degree and low diameter, and because of the relative ease in mapping different network configurations (rings[11], linear arrays[11], systolic arrays[12], trees[13], and multi-dimensional meshes[11]) into hypercube networks, the machines based on the n-cube topology have been described as a ideal parallel architecture. Commercial hypercube machines have been built [14,15].

Efficient message routing is a key to the performance of the hypercube network. There are several algorithms using well-known methods, such as the shortest path algorithm(the forward algorithm)

[16], the backward algorithm[16], the spanning tree algorithm[17], and the probabilistic algorithm[18]. These methods provide for only sequential transmission from the source node to the desired node in a short time. However, we would like to look for algorithm that are capable of handling multiple data items simultaneously transmitted from the starting node to the destination node. To find n disjoint paths in a general network is a computationally difficult problem. There are a few algorithms that allow us to locate n disjoint paths such as the Hamiltonian path algorithm[17], the rotation algorithm using tree structure[17], the disjoint path algorithm[19], and the combinatorial algorithms[7].

Algorithms for discovering n parallel paths are computationally difficult in the general case. It has been proved that these paths exist in a specific network[7]. From this fact, combinatorial algorithms have been designed. In particular, Rabin introduces the Hadamard matrix [20] to simplify the route computation of the above combinatorial algorithm. His intention is to send a number of data items to perfectly distributed positions simultaneously. Perfectly distributed positions means that the distance between any two positions is n/2 on the n-dimensional hypercube(for n even). To interpret this routing algorithm, we examine the characteristic of the Hadamard matrix. This characteristic has a relationship with the block design method[21], which is transformed into the class of error correcting codes[22]. Another important property is the symmetric structure of the hypercube. Solution to the assignment problem[23] can be applied to select n different short paths.

Combining these techniques reveals an algorithm which is efficient and whose computational implementation is greatly simplified. Moreover, Rabin's algorithm is an elegant combinatorial algorithm.

Finally, we propose another algebraic approach to the design of secret routing algorithms on hypercube networks and generalize this to MRNS(Mixed Radix Number System) networks. As described before, each piece is transmitted to the destination node along its own secret route in the hypercube network. In order for all pieces to reach the destination node in the minimum possible time, each part of the secret route should be disjoint from the corresponding parts of all other routes. We show how to construct a set of disjoint paths from the starting node to the intermediate nodes. The main idea is to transform a set of vertex-disjoint paths into disjoint sets of bit positions and then to explore these disjoint sets. Later, these disjoint sets with some constraints are used to construct a special class of matrices. One of these matrices is the special latin square[24].

A latin square is a square matrix with n^2 entries of n different elements, none of them occurring twice within any row or column of the matrix. The integer n is called the order of the latin square. Theoretical aspects of the latin square and some associated groups are considered in [25], in which two kinds of special latin squares are designed. One is *the Hamiltonian circuit latin square(HCLS)*, and the other is *the cyclic subsets of order n latin square(CSnLS)*. Applying the special latin square to the parallel routing algorithm of the hypercube network

and of the MRNS network, we can construct a linear-time algorithm. To obtain a set of disjoint paths from these intermediate node to the destination node, *a partial Hamiltonian circuit latin square(PHCLS)* is employed.

This work is organized as follows. Chapter 2 describes the important dispersal algorithms - Shamir's algorithm, Asmuth and Blakley's algorithm, and Rabin's algorithm. Rabin's algorithm is analyzed.

Chapter 3 introduces our method - the DAF(Dispersal Algorithm using the FFT algorithm). This part describes the design and gives an analysis of the DAF, comparing it with the previously mentioned algorithms. An example of the DAF is illustrated.

Chapter 4 explains the network design topology. The fundamental design decisions are discussed. In order to select a particular network, the application demand, transmitting n pieces to the desired node through the n secret paths, should be considered. The switching methodologies and the network topologies will be discussed. The important references for this part are [10] and [16]. The second part of this chapter is a presentation of the topology of the hypercube. The interesting aspects of this structure are examined. This part of the chapter is based on reference[11]. The related combinatorial topics are mentioned.

Chapter 5 describes and analyzes Rabin's algorithm - the simplified routing algorithm employing the Hadamard matrix. This algorithm can be analyzed by introducing the covering problem[27]

and the symmetric structure of the hypercube. The covering radii are symmetrically distributed. Then, the assignment problem is solved and applied to choose n different short paths from n different starting nodes to n different destination nodes.

Chapter 6 is the central contribution of this work. This chapter focuses on special latin squares and their application is to secret routing algorithms for the hypercube network. The theoretical aspects of how to design these special latin squares are discussed. Partial special latin squares are designed and utilized for the design of secret paths. Later, these methods are extended to secret routing algorithms for the MRNS network which arises from the study of finite groups and their Cayley graph. Examples of this design methodology are presented. This chapter concludes with a discussion of simple parallel algorithms for the hypercube and MRNS networks.

2

Analyses of Dispersal Algorithms

Dispersal algorithms break the original file into n pieces and by using any (n-k) pieces, are able to recover the original file, where each piece is not a subset of the original file and k is the number of missing pieces. This chapter is composed of two sections. Section 1 is a review of the important dispersal algorithms. Section 2 provides a further analysis of Rabin's algorithm.

2.1 Review of the Dispersal Algorithms

An important algorithm used to break the file into n pieces is the one proposed by Shamir(see Appendix 1). Although Shamir's method is designed to share secrecy, it can be used as a dispersal technique.Shamir's method employs the polynomial interpolation. From an original file F it creates n pieces $F_1, F_2.., F_i, ...F_n$, each F_i the same size as F, (i.e., length(F) = length(F_i) ($1 \leq i \leq n$)) so that F can be reconstructed from any (n-k) pieces. Furthermore any less provide no information about F. Although the total storage requirement leads to an n-fold increase in size compared with the size of the original data, in terms of the time complexity, this algorithm becomes efficient when Newton interpolation is employed. Shamir's method can handle only a single fixed length of data item

at a time. In real-world applications, it is often necessary to currently process multiple data items. The following dispersal algorithms can handle more than one data item at a time.

Asmuth-Blakley Algorithm

The Asmuth-Blakley algorithm(see Appendix 2) is based on the use of the Chinese Remainder Theorem. Depending on the choice of the coefficients of the special matrix called *coprime*, this approach can require much more complicated computations than Shamir's algorithm. The coprime matrix with positive integer entries satisfies the following condition - every one of its rows consists of pairwise relatively prime entries and every one of its column consists of pairwise relatively prime entries. The original file F is dispersed into n pieces $F_1, F_2..F_i,...F_n$ by the coprime matrix. The size of a piece can vary greatly. Given any (n-k) pieces, F can be reconstructed. As with Shamir's algorithm, any less give no information about F. The total storage of all pieces is at least |F|*n/(n-k), but the storage requirement is dependent on the specific coprime matrix. With respect to the space complexity, this algorithm has better efficiency than Shamir's. The next algorithm is simpler to execute with regard to computation.

Rabin's Information Dispersal Algorithm(IDA)

Rabin has proposed a method called the IDA. He explains that this method can be viewed in the context of error-correcting codes, in which extra bits are added to the message to create a block.

Later, the file can still be reconstructed even if after transmission, any k errors occur within the block. These k errors, changing a bit or erasing it, can occur anywhere in the block. However, he treats a specialized case of the problem and consequently achieves optimal efficiency in data overhead and simplicity in the encoding-decoding algorithm. Before we describe the IDA, one of the data encryption algorithms, called the LTA(Linear Transformation Algorithm or Linear Cipher Algorithm), will be examined. The IDA is very similar to the LTA and understanding the LTA makes it easier to comprehend the IDA.

2.1.1 Examination of the LTA(Linear Transformation Algorithm)

The LTA employs the defining property of invertible matrices, namely, $M * M^{-1} = I$, where M^{-1} is called the inverse matrix of M and I is the identity matrix.

The following three matrices will be denoted.

$A = [a_{ij}]$ encoding matrix

$B = [b_{ij}]$ encoded data matrix

$X = [x_{ij}]$ data matrix

$1 \leq i, j \leq m$, $a_{ij}, b_{ij}, x_{ij} \in Z_p$ for any prime number p, i.e., $Z_p = \{0,1,2,..,p-1\}$.

The data is partitioned into sequences of blocks. Each block has size n. The block becomes a row of the data matrix. Given the data

matrix, the data is encoded and is decoded as follows:

$$A * X_1 = B_1, \quad X_1 = A^{-1} * B_1$$

Example 1: m = 2.

$$A = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

$$X = (x_1\ x_2\ x_3\ x_4) = (X_1\ X_2) \quad \begin{cases} X_1 = (x_1\ x_2) \\ X_2 = (x_3\ x_4) \end{cases}$$

Each column of the encoded data matrix is computed as follows:

$$B_1 = A * \begin{vmatrix} x_1 \\ x_2 \end{vmatrix} = \begin{vmatrix} a_{11}x_1+a_{12}x_2 \\ a_{21}x_1+a_{22}x_2 \end{vmatrix}$$

$$B_2 = A * \begin{vmatrix} x_3 \\ x_4 \end{vmatrix} = \begin{vmatrix} a_{11}x_3+a_{12}x_4 \\ a_{21}x_3+a_{22}x_4 \end{vmatrix}$$

The original data matrix is recovered as follows:

$$X_1 = A^{-1} * \begin{vmatrix} a_{11}x_1+a_{12}x_2 \\ a_{21}x_1+a_{22}x_2 \end{vmatrix}$$

$$X_2 = A^{-1} * \begin{vmatrix} a_{11}x_3+a_{12}x_4 \\ a_{21}x_3+a_{22}x_4 \end{vmatrix}$$

2.1.2 Description of the IDA(Information Dispersal Algorithm)

Given the (m x m) matrix A for the LTA, augment A by adding k more rows and let Å be an (n x m) matrix, where n = m + k. By multiplying each block of X by Å, B will be an (n x m) matrix. Transmit each row of B to the destination node. If the destination node receives at least m rows of B, the original data matrix X can be reconstructed.

Example 2: m = 2, k = 2.

$$\text{Å} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{vmatrix}$$

$$X = (x_1\ x_2\ x_3\ x_4) = (X_1\ X_2) \quad \begin{cases} X_1 = (x_1\ x_2) \\ X_2 = (x_3\ x_4) \end{cases}$$

Each column of the encoded data matrix is computed as follows:

$$B_1 = \text{Å} * \begin{vmatrix} x_1 \\ x_2 \end{vmatrix} = \begin{vmatrix} a_{11}x_1+a_{12}x_2 \\ a_{21}x_1+a_{22}x_2 \\ a_{31}x_1+a_{32}x_2 \\ a_{41}x_1+a_{42}x_2 \end{vmatrix}$$

$$B_2 = \text{Å} * \begin{vmatrix} x_3 \\ x_4 \end{vmatrix} = \begin{vmatrix} a_{11}x_3+a_{12}x_4 \\ a_{21}x_3+a_{22}x_4 \\ a_{31}x_3+a_{32}x_4 \\ a_{41}x_3+a_{42}x_4 \end{vmatrix}$$

The encoded data matrix is then constructed as follows:

$$B = \begin{vmatrix} a_{11}x_1+a_{12}x_2 & a_{11}x_3+a_{12}x_4 \\ a_{21}x_1+a_{22}x_2 & a_{21}x_3+a_{22}x_4 \\ a_{31}x_1+a_{32}x_2 & a_{31}x_3+a_{32}x_4 \\ a_{41}x_1+a_{42}x_2 & a_{41}x_3+a_{42}x_4 \end{vmatrix}$$

Each row of B is considered to be one piece of data, and it is transmitted to the destination node through the paths. If at least two pieces of data arrive at the destination node, the original data can be reconstructed. Suppose the 1st piece and the 2nd piece arrive. The recovery of the original data goes as follows:

Let $\hat{A}^*$ be the submatrix of $\hat{A}$ and let $(\hat{A}^*)^{-1}$ be the inverse matrix of $\hat{A}^*$.

$$\hat{A}^* = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

$$X_1 = (\hat{A}^*)^{-1} * \begin{vmatrix} a_{11}x_1+a_{12}x_2 \\ a_{21}x_1+a_{22}x_2 \end{vmatrix}$$

$$X_2 = (\hat{A}^*)^{-1} * \begin{vmatrix} a_{11}x_3+a_{12}x_4 \\ a_{21}x_3+a_{22}x_4 \end{vmatrix}$$

$$X = X_1\ X_2$$

2.1.3 An Example of the IDA

Encoding matrix Å

$$Å = \begin{vmatrix} 1 & 2 \\ 2 & 3 \\ 5 & 4 \\ 6 & 5 \end{vmatrix}$$

Original data X = (1 2 3 4) = (1 2)(3 4)

$$X = \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix}$$

Each column of B is computed.

$$B_1 = Å * \begin{vmatrix} 1 \\ 2 \end{vmatrix} = \begin{vmatrix} 5 \\ 8 \\ 13 \\ 16 \end{vmatrix}$$

$$B_2 = Å * \begin{vmatrix} 3 \\ 4 \end{vmatrix} = \begin{vmatrix} 11 \\ 18 \\ 31 \\ 38 \end{vmatrix}$$

The encoded data matrix is constructed.

$$B = \begin{vmatrix} 5 & 11 \\ 8 & 18 \\ 13 & 31 \\ 16 & 38 \end{vmatrix}$$

The encoded data matrix B is broken up into n pieces of data by taking each row of B as one piece of data. D_i, the i^{th} row of B, denotes the i^{th} piece of transmitting data. ($1 \leq i \leq 4$)

$D_1 = (5,11)$

$D_2 = (8,18)$

$D_3 = (13,31)$

$D_4 = (16,38)$

If at least two pieces of data arrive at the destination node, then the original data matrix X can be reconstructed. Suppose D_1 and D_2 arrive, the reconstruction process takes place as follows:

Before reconstructing the data, we assume that the encoding matrix $\hat{A}$ is known. Let $\hat{A}^*$ be the submatrix of $\hat{A}$ that is used to produce D_1 and D_2, and let $(\hat{A}^*)^{-1}$ be the inverse matrix of $\hat{A}^*$, where $\hat{A}^*$ is any (m x m) submatrix of $\hat{A}$.

$$\hat{A}^* = \begin{vmatrix} 1 & 2 \\ 2 & 3 \end{vmatrix}$$

$$(\hat{A}^*)^{-1} = \begin{vmatrix} -3 & 2 \\ 2 & -1 \end{vmatrix}$$

$$X_1 = (\hat{A}^*)^{-1} * \begin{vmatrix} 5 \\ 8 \end{vmatrix} \qquad X_2 = (\hat{A}^*)^{-1} * \begin{vmatrix} 11 \\ 18 \end{vmatrix}$$

$$X = X_1\ X_2 = (\ 1\ 2\ 3\ 4\ )$$

2.2 An Analysis of the IDA

This section deals with two topics. One of them is a necessary condition for reconstructing the original data matrix: ***The determinant of any (m x m) submatrix of the encoding matrix should not be 0.*** The other topic provides an analysis of IDA, comparing it with the other algorithms mentioned previously.

2.2.1 A Necessary Condition for the IDA

A necessary condition for reconstructing the original data matrix is that the determinant of any (m x m) submatrix of the encoding matrix should not be 0. For if it is, the original data matrix cannot be uniquely determined.

Given the matrix $A = [a_{ij}]$ $(1 \le i, j \le n)$, a_{ij} is selected randomly and independently from the set $Z_p = \{0,1,2,..,p-1\}$, where p is any prime number.

$$A = \begin{vmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ & & \cdots\cdots\cdots & & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{vmatrix}$$

Let $a_i = (a_{i1}, a_{i2}, \ldots, a_{in})$ be the i^{th} row of the matrix A. If A is nonsingular, then the row vectors $a_1, a_2, \ldots, a_n$ must be linearly

independent. The number of ways to choose the n linearly independent vectors $a_1, a_2, \ldots, a_n$ is given below:

a_1 : $p^n - 1$ ways

a_2 : $p^n - p$ ways

a_3 : $p^n - p^2$ ways

.....

a_n : $p^n - p^{n-1}$ ways

Remark : a_i can be any vector not in (i-1)-dimension subspace generated by the previously chosen vector(s).

The probability that the matrix A is nonsingular is as follows:

$$(p^n-1)(p^n-p)(p^n-p^2)\ldots.(p^n-p^{n-1})/p^{n*n}$$

$$= (1 - 1/p)(1 - 1/p^2)(1 - 1/p^3)\ldots.(1 - 1/p^n) \; >$$

$$1-(1/p + 1/p^2 + 1/p^3 \ldots + 1/p^n) \to 1 - 1/(p-1) \text{ as } n \to \infty$$

The above calculation depends on the simple algebraic fact that if $x > 0$ and $y > 0$, then $(1 - x)(1 - y) > 1 - x - y$.

Remark : In practice the reader should choose a large prime for p, say > 100.

There are two special matrices that are nonsingular: Cauchy's matrix and Vandermonde's matrix. The reference [7] explains how to design a Cauchy matrix and to construct its inverse. In Appendix 3, we will show how to construct and invert a Vandermonde matrix, and

refer the reader to [8] for further discussion.

2.2.2 The analysis of the IDA

The file is dispersed into n pieces, and reconstructed from any (n-k) pieces by the inversion process. $|F|*n/(n-k)$ is required for the storage. The choice for n depends on the reliability of the network. If the network is not reliable, then the number of missing pieces is large. This means that for an unreliable network, we need to send more pieces to improve the reliability of the network. Furthermore, since more pieces are missing, each piece should share a greater portion of the data. Since, in a reliable network, we can choose n so that $n/(n-k) \rightarrow 1$(i.e., the ratio $n/(n-k)$ can be made close to 1), the IDA is space-efficient. The time complexity for reconstructing of the original data is $O(n^3)$, since the main cost of the complexity is due to the computation of the inverse of a submatrix of the encoding matrix. An efficient way to invert a matrix is introduced in [8]. In this approach, the coefficients of an (m x m) matrix are chosen so that the inversion of this matrix will require just $O(m^2)$ for each element of the inverted matrix. The IDA is better than the Asmuth-Blakley Algorithm with regard to the space complexity. In addition, the IDA is made for simpler arithmetic computations.

In implementing the dispersal algorithm, it is useful to include the coding vector as the header of each piece of data. In this way, there is no need to store the vectors separately. Unlike the

situation involving Shamir's or the Asmuth-Blakley Algorithm, we can obtain partial information, using the IDA, in case fewer than (n-k) pieces of data arrive at the destination node. Note that it is possible that an adversary can obtain some pieces of the file by eavesdropping. Therefore, when applying the IDA, the original file must be in a well-encrypted form.

When examining the time complexity for the reconstruction of the original data, we notice a paradox of information transmission. As mentioned above, the space complexity becomes better as the reliability of the network becomes better. However, the time complexity becomes worse as the network becomes more reliable, because the size of the submatrix(of the encoding matrix) that must be inverted increases as the reliability of the network increases.

3

The Design and Analysis of the DAF

We propose a method called the DAF(Dispersal Algorithm using the FFT algorithm). The IDA uses the FFT algorithm and a matrix-theoretic property: for a nonsingular matrix M, $M * M^{-1} = I$. Let F be a file, that is, let F be a string of characters. Each of these characters may be thought of as an integer which is randomly and independently taken from a certain set {0,1,2,...,p-1}, where p is any prime number. We will employ a special (n x n) matrix M which satisfies the conditions of the FFT algorithm. The IDA disperses F into n pieces using the FFT algorithm. Each piece is transmitted through the channels subject to the given condition that at most k pieces may be lost during transmission. Later, F is reconstructed from at least m pieces, where m = n - k. This chapter consists of three sections. Section 1 discusses the design of the DAF. Section 2 gives an example of the DAF. Section 3 provides an analysis of the DAF.

3.1 The Design of the DAF

Let F be a file. Each character in the file may be thought of as an integer which is taken randomly from a certain set {0, 1, 2, ... ,p-1}, where p is any prime number. The file F is segmented into

sequences of length m as follows:

$$F = (a_1, \ldots, a_m), (a_{m+1}, \ldots, a_{2m}), \ldots\ldots$$

An (m x m) data matrix A is constructed using these sequences.

$$A = \begin{vmatrix} a_1 & a_2 & \ldots & a_m \\ a_{m+1} & a_{m+2} & \ldots & a_{2m} \\ & \ldots\ldots\ldots & & \\ & \ldots\ldots & & a_{mm} \end{vmatrix},$$

where A_{r_i} denotes the i^{th} row of A and A_{c_i} denotes the i^{th} column of A.

Remark : F is formatted but this presents no obstacle.

To describe the DAF, the FFT algorithm and a matrix-theoretic property should be explained. The following definition and theorem, which are needed to design the FFT algorithm, are taken from the reference[6].

Let R be a ring with a primitive N^{th} root of unity w. Let Ω be the (N x N) matrix given by $\Omega_{i,j} = w^{i,j}$, $0 \leq i, j \leq N - 1$. Let $f = (f_0, f_1, \ldots, f_{N-1})^t$ be any column N-tuple with entries in R. The discrete Fourier transform, F(f), of f is the column N-tuple Ωf. Writing $F(f) = (F(f)_0, F(f)_1, \ldots, F(f)_{N-1})^t$, we have

$$F(f)_m = \sum_{j=0}^{N-1} \Omega_{m,j} f_j = \sum_{j=0}^{N-1} f_j w^{m,j} = \sum_{j=0}^{N-1} f_j (w^m)^j$$

[Definition 9.4, [6] p. 445]

Let n be a nonnegative integer. Let $M = 2^{2^n} + 1$ and $N = 2^{n+1}$. Then $w = 2$ is a primitive N^{th} root of unity in Z_M such that $w^{N/2} = -1$. N is invertible in Z_M. Let Ω be the (N x N) matrix of Definition 9.4 given by $\Omega_{i,j} = w^{i,j}$. Then Ω^{-1} is given by $\Omega^{-1}_{i,j} = N^{-1}w^{-i,j}$. [Theorem 9-18, [6], p. 455]

Given the (n x n) encoding matrix Ω

$$\Omega = [\ w^{i,j}\ ] = \begin{vmatrix} w^{0,0} & w^{0,1} & \cdots & w^{0,n-1} \\ w^{1,0} & w^{1,1} & \cdots & w^{1,n-1} \\ w^{2,0} & w^{2,1} & \cdots & w^{2,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & w^{n-1,n-1} \end{vmatrix},$$

where $w^{i,j} = w^{i*j}$.

We apply the FFT algorithm to each column of the data matrix A mentioned on the previous page. Then each column $D_{c_i} = (d_{11}, d_{21}, \ldots, d_{n1})$ of the encoded data matrix D, is generated as follows:

$$D_{c_i} = \Omega * A_{c_i}$$

(D_{c_i} : i^{th} column of D)

(A_{c_i} : i^{th} column of A)

Then, the encoded matrix D can be written as follows:

$$D = \begin{vmatrix} d_{11} & d_{12} & d_{13} & \cdots & d_{1m} \\ d_{21} & d_{22} & d_{23} & \cdots & d_{2m} \\ & & \cdots & & \\ d_{n1} & d_{n2} & d_{n3} & \cdots & d_{nm} \end{vmatrix}$$

Let D_{r_i} denote the i^{th} row of D. Each row of D can be considered to be one piece of data and is transmitted to the destination. During transmission, at most k pieces are lost. Then at least (n-k) pieces arrive at the destination. Let $\tilde{D}$ be the matrix that is constructed as follows.

If D_{r_i} (the i^{th} row of D) arrives at the destination, then the i^{th} row of $\tilde{D}_i = D_{r_i}$, otherwise the i^{th} row of $\tilde{D}_i = (0,0,\ldots,0)$.

$$\tilde{D} = \begin{vmatrix} d_{11} & d_{12} & d_{13} & \ldots. & d_{1m} \\ 0 & 0 & 0 & \ldots. & 0 \\ & & \ldots\ldots\ldots\ldots & & \\ d_{n1} & d_{n2} & d_{n3} & \ldots. & d_{nm} \end{vmatrix}$$

Recovering the lost pieces is *not a difficult problem*. Examining the FFT algorithm, we find that every entry of A_{r_i} (the i^{th} row of A), where $i > m$, equals zero. Apply $(w^{-1})^m, (w^{-1})^{m+1}, \ldots, (w^{-1})^{n-1}$ individually to each column of $\tilde{D}$.

$$\tilde{D}_{c_i}(x) = f^i(x) = d_{ni}x^{n-1} + d_{n-1i}x^{n-2} + \ldots + d_{1i}$$

Let $U = [\, u_{ij} \,]$ denote the (k x n) matrix constructed from the last k rows of Ω', where $\Omega' = [(w^{-1})^{i,j}]$ and let $\tilde{U}$ denote the (k x k) matrix constructed from the columns of U. The selected columns come from the indices of the lost pieces. Let $B_i = [b_{ji}]$ be the (k x 1) matrix, where b_{ji} is the i^{th} element in the lost piece D_j.

Then,

$$U * B_1 = \begin{vmatrix} -f^1((w^{-1})^m) \\ -f^1((w^{-1})^{m+1}) \\ \cdots\cdots \\ -f^1((w^{-1})^{n-2}) \\ -f^1((w^{-1})^{n-1}) \end{vmatrix}$$

Now, apply U^{-1} on the left to each side of this equation.

So,

$$B_1 = U^{-1} * \begin{vmatrix} -f^1((w^{-1})^m) \\ -f^1((w^{-1})^{m+1}) \\ \cdots\cdots \\ -f^1((w^{-1})^{n-2}) \\ -f^1((w^{-1})^{n-1}) \end{vmatrix}$$

The LUP Decomposition[9] or iteration method[28] can be used for inverting U. From the above calculation, the lost pieces of D are recovered.

For reconstructing the original data matrix, a matrix-theoretic property is utilized again. By Theorem 9-18[6], Ω is inverted.

$$\Omega^{-1} = N^{-1} * [w^{-i,j}] = N^{-1} * [(w^{-1})^{i,j}]$$

Given the inverted matrix, each column of the original data matrix can be reconstructed as follows:

$$A_{c_i} = \Omega^{-1} * D_{c_i}$$

(D_{c_i} : i^{th} column of D)

(A_{c_i} : i^{th} column of A)

3.2 An Example of the DAF

Let F be the original data, and let Ω be a matrix that satisfies the conditions of the FFT algorithm. Each element of F is selected randomly from Z_p, where $p = 2^{2^n} + 1$ is a prime number.

Let $n = 1$, $N = 2^{n+1} = 4$, and $p = 5$. 2 is a primitive 4th root of unity in Z_5, $2^{N/2} = 2^2 = 4 = -1$, and $4^{-1} = 4$ in Z_5. Now we apply theorem 9-18 as follows:

$$\Omega = [\ 2^{i,j}\ ] = \begin{vmatrix} 2^0 & 2^0 & 2^0 & 2^0 \\ 2^0 & 2^1 & 2^2 & 2^3 \\ 2^0 & 2^2 & 2^4 & 2^6 \\ 2^0 & 2^3 & 2^6 & 2^9 \end{vmatrix}$$

2 is a primitive 4^{th} root of unity. That is,

$$2^1 = 2 \quad 2^2 = 4 \quad 2^3 = 3 \quad 2^4 = 1$$

Then Ω becomes as follows.

$$\Omega = [\ 2^{i,j}\ ] = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \\ 1 & 3 & 4 & 2 \end{vmatrix}$$

The file F is broken up into sequences of length 2.

$$F = (1\ 3\ 2\ 4) \rightarrow (1\ 3)(2\ 4)$$

The data matrix A can be constructed from the file F as follows:

$$A = \begin{vmatrix} 1 & 3 \\ 2 & 4 \end{vmatrix}$$

The FFT algorithm is applied to each column of A, and this generates each column of the encoded data matrix D.

$$D_{c_i} = \Omega * A_{c_i}$$

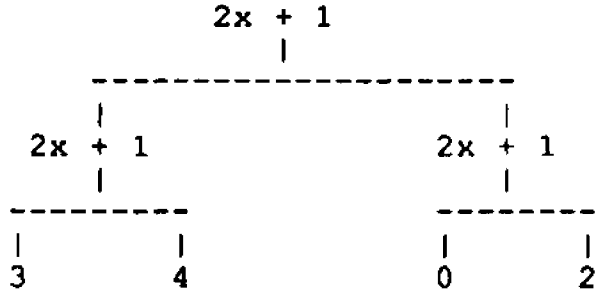

Fig. 1

This figure illustrates how to apply (1 2 0 0) to the FFT algorithm. The first column of D is generated as (3 0 4 2). Likewise, we have (2 1 4 0) for the second column of D.

Therefore, the encoded data matrix is

$$D = \begin{vmatrix} 3 & 2 \\ 0 & 1 \\ 4 & 4 \\ 2 & 0 \end{vmatrix}.$$

All the rows(or pieces) of D are transmitted to the destination node. F can be reconstructed if at least 2 pieces of D are received at the destination node. Let us suppose that D_{r_1} and D_{r_2} arrive. We construct $\hat{D}$ as follows:

$$\hat{D} = \begin{vmatrix} 3 & 2 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{vmatrix}$$

By Theorem 9-18, Ω is inverted.

$$\Omega^{-1} = N^{-1} * [w^{-i,j}] = N^{-1} * [(w^{-1})^{i,j}]$$

So,

$$\Omega^{-1} = 4 * [3^{i,j}] = 4 * \begin{vmatrix} 3^0 & 3^0 & 3^0 & 3^0 \\ 3^0 & 3^1 & 3^2 & 3^3 \\ 3^0 & 3^2 & 3^4 & 3^6 \\ 3^0 & 3^3 & 3^6 & 3^9 \end{vmatrix}$$

3 is also a primitive 4^{th} root of unity. We have

$$3^1 = 3 \quad 3^2 = 4 \quad 3^3 = 2 \quad 3^4 = 1$$

Then Ω^{-1} is given by the following computation:

$$\Omega^{-1} = 4 * \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 4 & 2 \\ 1 & 4 & 1 & 4 \\ 1 & 2 & 4 & 3 \end{vmatrix} = \begin{vmatrix} 4 & 4 & 4 & 4 \\ 4 & 2 & 1 & 3 \\ 4 & 1 & 4 & 1 \\ 4 & 3 & 1 & 2 \end{vmatrix}$$

To compute the values of the elements in D_{c_1}, 3^2 and 3^3 are substituted for x in equations below:

$$f^1(x) = 3$$

Then,

$$\begin{vmatrix} 1 & 4 \\ 4 & 3 \end{vmatrix} * \begin{vmatrix} q_{31} \\ q_{41} \end{vmatrix} = \begin{vmatrix} 2 \\ 2 \end{vmatrix}$$

The LUP decomposition method is applied to these linear equations.

$$q_{31} = 4, \quad q_{41} = 2.$$

We repeat the same process to recover the values of the variables in D_{c_2}.

$$q_{32} = 4, \quad q_{42} = 0$$

Then, all the lost pieces of D are recovered.

$$D = \begin{vmatrix} 3 & 2 \\ 0 & 1 \\ 4 & 4 \\ 2 & 0 \end{vmatrix}$$

Each column of the original data matrix A is now reconstructed using the FFT algorithm.

$$A_{c_i} = \Omega^{-1} * D_{c_i}$$

(D_{c_i} : i^{th} column of D)

(A_{c_i} : i^{th} column of A)

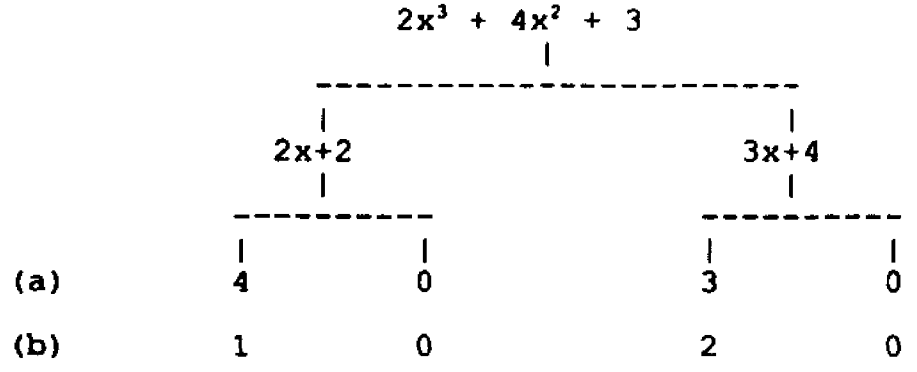

Fig. 2

Note that the values in (b) were obtained by multiplying the values in (a) by 4. Thus, A_{c_1} is (1 2 0 0) and A_{c_2} is (3 4 0 0). The original data matrix A can be written as follows:

$$A = \begin{vmatrix} 1 & 3 \\ 2 & 4 \end{vmatrix}$$

Therefore, the original file F is (1 3 2 4)

3.3 An Analysis of the DAF

The file F is dispersed into n pieces, and reconstructed from

any m pieces, where m = n - k. |F| * n / m is required for the storage. The DAF chooses n, depending on the reliability of the network. If the network is not reliable, then the number of lost pieces is large. This means that for an unreliable network, we send more pieces to improve the reliability of the network. Additionally, each piece should share a greater portion of the data. Since, in a reliable network, we can choose n so that n / m approaches 1, the DAF is space-efficient. Because the file is dispersed using the FFT algorithm, this process of dispersal requires $n*O(n*\log_2 n)$ for the time complexity. The file F is reconstructed using the LUP decomposition method and the FFT algorithm. The time complexity is max $\{n*O(n*\log_2 n), n*k*(n-k)\}$. If a network contains error-free channels, the time complexity of $n*k*(n-k)$ is negligible. Hence, the time complexity, in this case, is $n*O(n*\log_2 n)$. Unlike the IDA, the DAF improves both the time complexity and the space complexity as the channels become error-free. The DAF method discovered by the author who was pleased to discover a very similar algorithm. The Holographic Algorithm already was formulated by Preparata in [26]. We leave to this historian the actual origin of this method which may be cloaked in various archives yet to be released.

4

The Network Design Topology

In order to design a network, several fundamental elements should be considered to determine the appropriate architecture for the network. The first part of this chapter discusses three of these elements - control strategies, switching methodologies, and network topologies. The choice will depend upon the application. In the second part, we focus our attention on the hypercube network and examine the hypercube topology. This chapter is composed of two sections. Section 1 discusses three fundamental elements of network design. Section 2 gives a description of the hypercube topology.

4.1 Network Design Decisions

Three fundamental elements should be considered in choosing the appropriate architecture of the network. These elements are control strategies, switching methodologies, and network topologies. The space of all networks can be represented by the cartesian product of these three elements. The choice of particular network should depend on the application: to transmit n pieces of data to the destination node along the secret routes simultaneously. In this section, these three elements are examined with regard to the application.

4.1.1 Control Strategies

There are two operations that must be performed to achieve fault-tolerant parallel communication: computation and communication. For the computation, a node(the sender) disperses data into n pieces by using the DAF. For the communication, all the pieces are transmitted to the destination along secret routes. Decisions related to routing are controlled by specific nodes. There are two types of control strategies: distributed control and centralized control. In the first, control is handled by each node individually, while in the second control strategy, control is managed by a centralized controller. At any given time, a node must determine which operation it will perform after receiving a request for that operation. Therefore, distributed control is more suitable for fault-tolerant parallel communication.

4.1.2 Switching Methodologies

In a switching network, data is transmitted from the source station to the destination station through the nodes of the network. The two major switching methodologies are circuit switching and packet switching. In circuit switching, the connection path is established between the source and the destination before data transmission begins. That path is a connected sequence of channels between two stations. In terms of performance, there is a delay prior to data transfer because it

takes time to establish an appropriate route between the two stations. However, once the circuit is established, the data is suitable for bulk transmission. A common example of circuit switching network is the classical telecommunication network. The circuit switching network does not fit our demand of secrecy, since this switching network utilizes only one dedicated path for the entire data transmission. In the other major switching methodology, the packet switching, the data are divided into smaller units and sent out one at a time without establishing a physical connection path prior to data transmission. As described in Chap. 3, the data is dispersed into n pieces by the DAF. Each piece is treated as a unit in a packet switching network. Since our application demand is to transmit all the pieces to the destination via disjoint paths and since the packet switching network has a number of alternative routes between the source and the destination, we will examine the routing in a packet switching network in some detail below.

4.1.2.1 Routing in a Packet Switching Network

The routing function in a packet switching network is more complex and more difficult to explain than the routing function in a circuit switching network. The main reason for this is the relatively undifferentiated nature of the packet switching network. This network tends to be equal rather than hierarchical, since it has a number of alternative routes between the source and

destination and there are no preferred routes. As noted before, the source station breaks the data into numerous packets and sends these packets to the destination station. There are two approaches used to handle this stream of packets. These approaches are known as *datagram* and *virtual circuit*. In datagram, each packet is treated independently. Each packet is transmitted to the destination along its own path. The destination collects and reorders these packets to reconstruct the data. In virtual circuit, a route between two stations is set up before data transfer. This does not mean that there is a dedicated path, as in circuit switching. A packet is still buffered at each node, and queued for output. Unlike the datagram approach, the node need not make a routing decision for each packet. The routing decision for a virtual circuit is made only once for each connection. The datagram approach provides some advantages that other networks do not provide. One advantage is that establishment of paths is avoided. If a station transmits only a few packets, the datagram method will be fast. Another advantage is that the datagram method is more flexible, because it is more primitive. If congestion occurs in one part of the network, incoming datagrams can be routed away from the congestion. A third advantage is that datagram delivery is inherently more reliable. With the use of virtual circuits, if a node fails, all the packets that pass through that node are lost. With datagram delivery, if a node is down, packets may find alternate routes. From the above facts, the datagram delivery is attractive to us. The discussion in the previous paragraph is summarized from Stallings[16] to where

the reader is referred for further elucidation.

In this thesis, we employ one of the advantages of the datagram approach, that is, random routing, for finding secret routes. Our approach provides the source node with the responsibility for determining n secret paths before data transmission. Even though the source node completely decides the path to the destination node, the randomness of the channels selected is similar to that found in datagram delivery.

4.1.3 Network Topologies

A network can be described by a graph in which the vertices represent nodes and the edges represent communication channels. There may be many approaches to examine the topology of the network. In this paper, we explore applications of some permutations and some groups to this topology.

4.1.3.1 Permutations and Groups

Given n starting nodes and n destination nodes, let us suppose the time complexity to be $T(n) = k(n)$ for finding a path from a starting node to a destination node in a specific permutation(Fig. 4-1). In order to save time for finding the path between two nodes, the original permutation is partitioned into 2 smaller permutations (Fig. 4-2). This method is very similar to the "divide-and-conquer"

method(see [9] for discussion). The time complexity will be $T(n) = k(n/2) + f(n)$, where $f(n)$ is the time complexity for combining the two parts. A good example of this comes from the modular number system. To examine how to construct groups using a fixed number of smaller permutation, it is necessary to understand concepts from group theory, such as subgroups and cosets, presented in for computer scientists in [25].

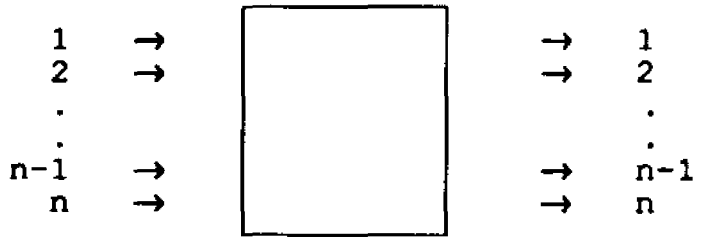

Fig. 4-1

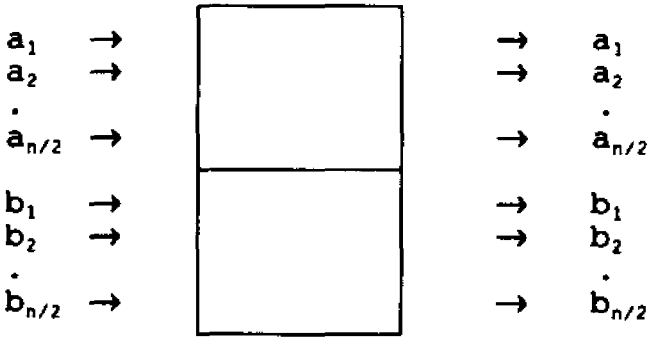

Fig. 4-2

Let us consider the group$(Z_6, +)$ and design the composition table for $(Z_6, +)$. The dashed lines in Fig. 4-3 partition the table into four areas. The vertical dashed line divides the columns into the

two cosets of the group. Similarly, the horizontal dashed line divides the rows into the two cosets of the group. Using the quotient structure, the composition table for a normal subgroup of order 3 is designed in Fig. 4-4.

Composition table for $(Z_6, +)$

	0	2	4	1	3	5
0	0	2	4	1	3	5
2	2	4	0	3	5	1
4	4	0	2	5	1	3
1	1	3	5	2	4	0
3	3	5	1	4	0	2
5	5	1	3	0	2	4

Fig. 4-3

Composition table for normal subgroup of order 3.

	(0 2 4)	(1 3 5)
(0 2 4)	(0 2 4)	(1 3 5)
(1 3 5)	(1 3 5)	(0 2 4)

Fig. 4-4

From the composition table for $(Z_6, +)$ (see Fig. 4-3), Fig. 4-5 is drawn, where (s,+) means "add s to the given number(modulo n)".

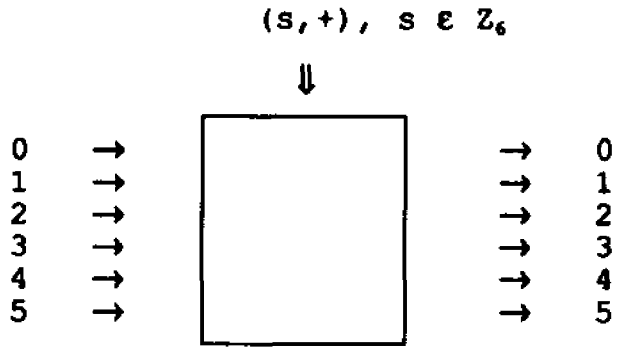

Fig. 4-5

From the composition table in Fig. 4-4, Fig. 4-6 and Fig. 4-7 can be designed, depending on the choice of s.

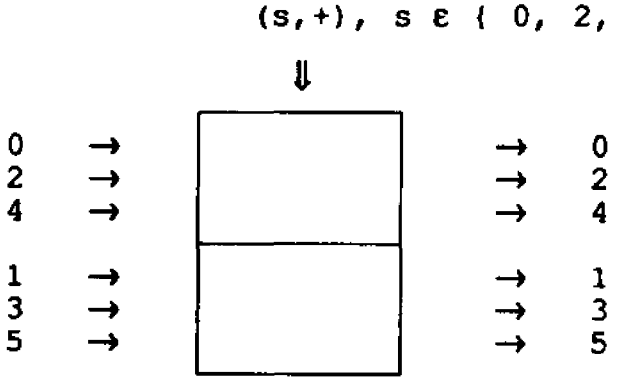

Fig. 4-6

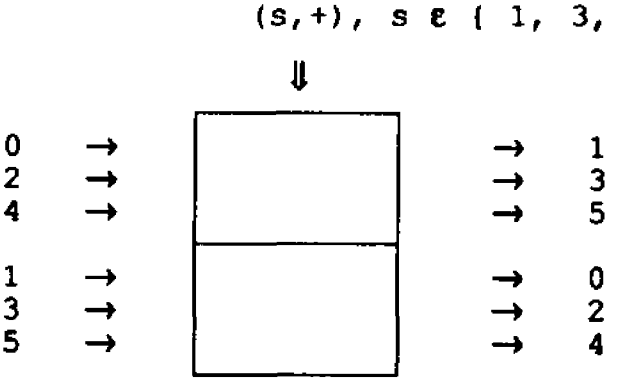

Fig. 4-7

Fig. 4-8 presents a group graph to illustrate a well-defined operation on the quotient structure of the group in Fig. 4-6.

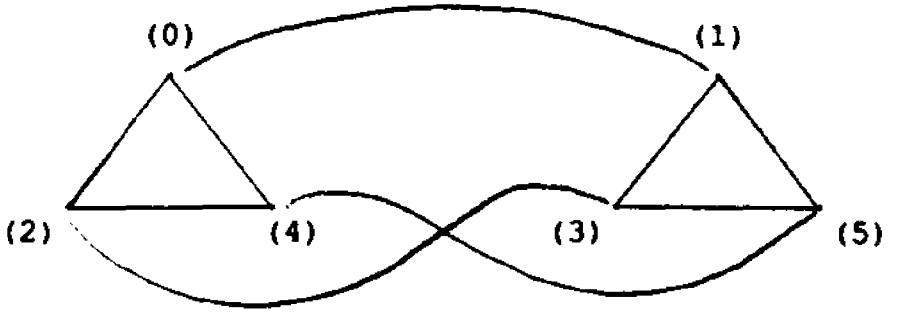

Fig. 4-8

operations
- clockwise : (+ 4)
- counter-clockwise : (+ 2)
- left -> right : (+ 1)
- right -> left : (+ 5)

The group graph in Fig. 4-8 consists of two subgraphs. Each subgraph has the same number of vertices, and edges generated by its operations. From the above facts, we can divide the operations into two categories: global operation and local operations. Global operations are performed between two vertices belonging to different subgraphs. Local operations are performed between two vertices in the same subgraph. Fig. 4-9 is designed as follows:

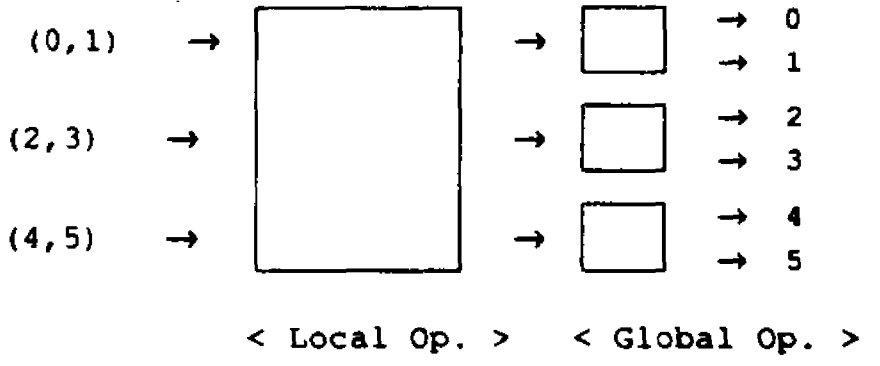

Fig. 4-9

Since the graph in Fig. 4-8 is composed of two isomorphic subgraphs, Fig. 4-10 is presented by introducing the following index scheme.

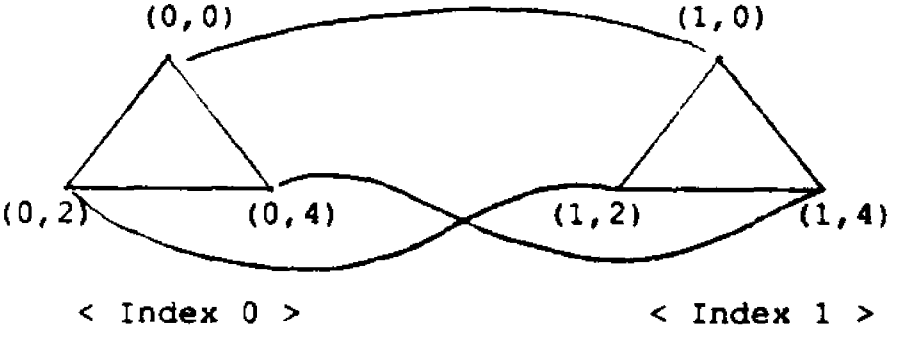

Fig. 4-10

The first coordinate of each ordered pair at each vertex indicates the index of the subgraph that the vertex belongs to. Utilizing this index scheme, we can partition a number of vertices into blocks, each of which contains a fixed number of vertices. To identify the address of a vertex, that vertex is defined by a block number and a physical number in that block. If there are many

blocks, then this set of blocks can be partitioned into segments, each containing a number of blocks. By using the index scheme described above, we can more easily examine the structure of a specific network. One of the networks which has the above structure is the hypercube network. In addition to having this structure, the hypercube network has a reasonable number of channels per each node, relatively short distances between every pair of nodes, and a simple routing function. Therefore, we select the hypercube network for fault-tolerant parallel communications. The next section discusses several characteristics of the hypercube in detail.

4.2 Hypercube Topology

The n-dimensional hypercube consists of highly concurrent, loosely coupled parallel processors based on the binary n-cube topology. The n-cube parallel processors consist of 2^n identical processors. Each of the processors, provided with its own sizable memory, is connected through bidirectional, point-to-point communication channels to n different neighbors to form a communication network. Because of the hypercube's low degree and small diameter, and because of the relative ease in mapping different network configurations (rings, linear arrays, systolic arrays, trees, and multi-dimensional meshes) into the hypercube network, machines based on the hypercube topology have been described as ideal parallel architectures for their powerful

interconnection features.

4.2.1. The Hypercube Graph and Basic Properties

The hypercube is regarded as a graph and we will use the terms vertices and nodes interchangeably for the processors they represent. In addition, we will use the term edges for the channels. We will look at the theoretical properties of the hypercube.

Definition 1: An n-cube graph is an undirected graph consisting of $N = 2^n$ vertices labeled from 0 to N - 1 and such that there is an edge between any two vertices if and only if the binary representations of their labels differ by one and only one bit.

An important property of the n-cube is that it can be constructed recursively from lower dimensional cubes. More precisely, consider two identical (n-1)-cubes whose vertices are numbered from 0 to 2^{n-1}. By joining each vertex in the first (n-1)-cube to its corresponding vertex in the second (n-1)-cube, one obtains an n-cube. It suffices to renumber the nodes of the first cube as $0 \cdot a_i$ and those of the second by $1 \cdot a_i$, where a_i is a binary number representing the two corresponding nodes of the (n-1)-cubes, $0 \leq i \leq 2^{n-1}$. From Definition 1, simple properties of the hypercube are summarized in the following propositions.

Proposition 1: There are n different ways of tearing an n-cube, i.e., of splitting it into two (n-1)-subcubes so that their respective vertices are connected in a one-to-one way.

Given the labeling of 2^n vertices, each tearing corresponds to each bit position: one whose vertex label is 1 in position i and one whose vertex label is 0 in position i.

Proposition 2: There are $n!2^n$ different ways in which the 2^n nodes of the n-cube can be numbered.

Proof : The proof is by induction.

1) If n = 0, then true.

2) Assume true for n-1. According to Proposition 1, the n-cube can be torn into two (n-1)-cubes in n different ways.

$$n[(n-1)!2^{n-1} + (n-1)!2^{n-1}] = n!2^n$$

Therefore, we obtain $n!2^n$ different numberings for the vertices of the n-cube.

Proposition 3: There are no cycles of odd length in an n-cube.

Proof : Consider a cycle $A_1, A_2, \ldots, A_t$, with $A_1 = A_t$. To travel from A_i to A_{i+1}, $1 \leq i \leq t-1$, the $(i+1)^{st}$ position is changed. Since $A_1 = A_t$, there must be an even number of changes, i.e., the length of the cycle is necessarily even.

Proposition 4: The n-cube is a connected graph of diameter n.

Proof : Given two nodes A and B, one way to travel from A to B is to modify one bit of A, which differs from the corresponding bit of B at a time. This means that we select one edge at a time in constructing a path from A to B. This provides a simple way to create a path of length at most n between any two nodes.

4.2.2 Distances and Paths in the Hypercube

Any multiprocessor system should allow for its processors to exchange data between all of its node. Let A and B be any two nodes of the n-cube and consider the problem of sending data from node A to node B. In the previous section, we observed that there exists a path of length at most n between any two nodes (Proposition 4). To travel from A to B, it suffices to cross successively the nodes whose labels are those obtained by modifying the bits of A one by one in order to traverse from A to B.

Definition 2: The routing function of the n-dimensional hypercube network for the i^{th} bit is as follows:

node address $A = (a_{n-1}a_{n-2}..a_i..a_1a_0)$, $0 \leq A \leq 2^n - 1$, $a_i \in \{0,1\}$

$C_i(a_{n-1}a_{n-2}..a_i..a_1a_0) = a_{n-1}a_{n-2}..\bar{a}_i..a_1a_0 \quad 0 \leq i \leq n-1$

At node A, the data are transmitted along the i^{th} channel of the n neighboring channels which A is directly connected to. The node at which the data arrive differs in exactly one bit position from A - i.e., the i^{th} bit position. A path from node A to node B can be constructed from a sequence of nodes in which every two consecutive nodes are physically adjacent to each other. This means that the addresses of every two consecutive nodes differ by exactly one bit. In order to obtain a path from A to B, it is necessary to introduce an exclusive operation on two nodes denoted by binary numbers, through which the relative address between two hypercube nodes can be computed.

Definition 3: The exclusive operation on two binary strings $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$, denoted by $A \oplus B = r_{n-1}r_{n-2}..r_0$, is defined by $r_i = 0$ if $a_i = b_i$ and $r_i = 1$ if $a_i \neq b_i$, for $0 \leq i \leq n-1$.

The relative address of node A to node B can be described by the exclusive operation on the two nodes - $A \oplus B$. Let us apply the relative address of A and B to the routing function.

$$C_i(A) = \begin{cases} a_{n-1}a_{n-2}..\bar{a}_i..a_0 & \text{if } i = 1 \\ A & \text{if } r = 0 \end{cases}$$

Definition 4: Let P be a position sequence $P = (p_1, p_2, \ldots, p_k)$, which is a set of positions along which the data traverse from node A to node B. The traversal of the data is determined by the order

of the elements in the sequence P.

Example 1:

Given node A : (0010), P = <0,1,2,0>.

The traversal of the data is

(0011) → (0001) → (0101) → (0100).

Remark : The above notation is certainly simpler than $C_0(C_2(C_1(C_0(A))))$, where A = the starting node.

A position sequence should be constructed from the bit positions of all the 1's in the relative address of A and B, while the bit positions of the 0's may or may not be included. If we include one of the 0's, then the relative address of that position becomes 1, and then that position must be traversed again later. To find out the shortest path from A to B, the Hamming distance will be introduced.

Definition 5: The Hamming distance between two nodes with addresses $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$ in the hypercube network is defined as

$$H(A,B) = \sum_{i=0}^{n-1} h(a_i,b_i),$$

$$\text{where} \quad h(a_i,b_i) = \begin{cases} 1, & \text{if } a_i \neq b_i \\ 0, & \text{if } a_i = b_i \end{cases}$$

Proposition 5: The minimum distance between the nodes A and B is equal to the number of bits that differ between A and B, i.e., to the Hamming distance H(A,B).

Proposition 6: Let A and B be any two nodes in the n-dimensional hypercube and assume that H(A,B) < n. Then there are H(A,B) parallel paths(i.e. disjoint paths) of length H(A,B) between the nodes A and B.

The proof of Proposition 6 is given Chapter 6. The idea of the proof is to construct parallel paths from the H(A,B) different bit positions at which A and B differ. Once there are known the computation of the parallel paths is automatic.

Proposition 7: Let A and B be any two nodes of an n-dimensional hypercube and assume that H(A,B) < n. Then there are n parallel paths between A and B. The length of each path is at most H(A,B)+2.

Proof : Let H(A,B) = k. Then the different bit positions at which A and B differ are $\{a_1, a_2, .., a_k\}$. In addition to the k parallel paths mentioned in Proposition 6, we consider the other (n-k) remaining paths. Each of these additional paths starts at a bit position at which A and B do not differ. Then correct bits a_1 through a_k by choosing one of the k paths from the previous proposition. Finally, change the starting bit back to its original bit position. The (n-k) additional paths will never cross each

other, since these paths select different positions at the first and the last steps, and select the other positions by Proposition 6 for the remaining steps. They also will not intersect any of the k parallel paths because of the different lengths of the paths. Moreover, the length of each additional path is k + 2.

5

An Analysis of Rabin's Routing Algorithm(RRA)

The application of algebraic methods to the parallel routings on the hypercube network is discussed by Rabin[7] and Lyuu[30]. Rabin's algorithm employs an (n x n) Hadamard matrix in which every two rows differ from each other by exactly n/2 positions. Considering each row of this matrix as the relative address of the starting node with respect to some destination node, n different destination nodes are generated by applying the routing function to the starting node and each relative address. The Hamming distance between any two nodes is n/2. From this fact, we conclude that these n nodes are placed in well-distributed positions. This chapter examines Rabin's Routing Algorithm(RRA). This chapter is composed of three sections. In Section 1 the Hadamard matrices are discussed. Section 2 outlines the RRA and in Section 3 we conclude with an analysis of the RRA.

5.1 Examination of the Hadamard Matrix

The Hadamard matrix $H = [\ d_{ij}\ ]$ satisfies the following conditions:

1) $d_{ij} \in \{1,-1\}, \quad 0 \le i,j \le n-1$

2) $d_{i0}d_{j0} + d_{i1}d_{j1} + \ldots + d_{in-1}d_{jn-1} = 0 \quad$ for $i \neq j$.

The above conditions imply that for any two rows, there are exactly n/2 positions containing different bits. Also, If a matrix is Hadamard, then permuting the rows and the columns of the matrix does not change the property. Multiplying a row or a column by -1 does not change the property either. A Hadamard matrix is said to be normalized if its first row and first column consist entirely of 1's. By normalizing a Hadamard matrix, we shall mean multiplying those rows containing a -1 as the leftmost entry and those columns containing a -1 as the topmost entry by -1 so that the resultant matrix is normalized. In Sec. 5.3.1 the connection of a normalized Hadamard matrix and the block design method is discussed. One surprising thing about Hadamard matrices is the method of construction. The following theorem enables us to construct Hadamard matrices of higher order from Hadamard matrices of lower order. Therefore, we can construct Hadamard matrices in a recursive manner.

The ***Kronecker product*** (see Appendix 4) of two Hadamard matrices is a Hadamard matrix. [Theorem 14-12, [21] p. 381]

The discussion in the previous paragraph is summarized from Liu[21] to which the reader is referred for further discussion. In order to apply Hadamard matrices to the routing algorithm, each element of this matrix is transformed into a binary number as follows.

Definition 1: Let v_i be the i^{th} Hadamard node, $v_i \in (0,1)^n$.

$$\begin{cases} v_i[j] = 1 & \text{if } d_{ij} = 1, \\ v_i[j] = 0 & \text{otherwise,} \end{cases}$$

where $v_i[j]$ is the j^{th} element of v_i.

5.2 Outline of the RRA

The packet P_x emanating from a node x is broken up into n pieces - $P_{x1}, P_{x2}, \ldots, P_{xn}(|P_{x1}| = |P_{xj}|)$. Each piece P_{xi} is routed independently in the n-dimensional hypercube network. The distinct n intermediate nodes are selected randomly. Each piece is transmitted to an intermediate node via the chosen Hadamard node (see Definition 1). At the intermediate node, the piece is sent to the destination node via the selected Hadamard node.

cobegin (for the information P_x, node x and node y)

(1) Split P_x into $P_{x1}, P_{x2}, \ldots, P_{xn}$

(2) Choose randomly n nodes $R_1(x), R_2(x), \ldots, R_n(x)$

(3) Send each piece P_{xi} from x to $x + v_i$

(4) Send each piece P_{xi} from $x + v_i$ to $R_i(x)$ along the shortest path

(5) Send each piece P_{xi} from $R_i(x)$ to $y + v_i$ along the shortest path

(6) Send each P_{xi} from $y + v_i$ to y

coend.

5.3 Analysis of the RRA

The RRA is assumed in considering the *covering problem* and the *assignment problem*. The covering problem refers to the situation in which each node in the hypercube network can reach at least one of n different Hadamard nodes within the fixed distances - the shortest distance is best. This problem is attacked by employing sphere packing methods(see [27] for further discussion) in conjunction with symmetric property of the hypercube. The assignment problem is to select n paths from n different starting nodes to the n different destination nodes under the condition that two starting nodes must choose pairwise different destination nodes. This problem is solved by the Hungarian method(see [29] for discussion).

5.3.1 Covering Problem

Given a normalized Hadamard matrix M, each row of this matrix is called a Hadamard node. Then all the nodes in the hypercube network can reach one of the Hadamard nodes within Hamming distances of n/2. Applying the sphere packing method and symmetrical structure of the hypercube, the Hamming distance is distributed symmetrically by the parameter - the radius of the sphere packing. This distribution can be interpreted by error-correcting distance and the symmetrical structure of the hypercube. To explain the error-correcting distance, the block design method

is described in Definition 2.

Definition 2: A balanced incomplete block design(BIBD) is called a (b,v,r,k,λ)-configuration.

1) b is a set of blocks.
2) v is a set of objects.
3) Each object appears in exactly r of the b blocks.
4) The size of each block is k.
5) Every two objects appear simultaneously in exactly λ of the b blocks.
6) $k < v$

Example 1: A BIBD.

$$v = \{x_1, x_2, x_3, x_4\}, \quad b = 6, \quad r = 3, \quad k = 2, \quad \lambda = 1$$

$$b_1 = \{x_1, x_2\} \quad b_2 = \{x_1, x_3\} \quad b_3 = \{x_1, x_4\} \quad b_4 = \{x_2, x_3\}$$

$$b_5 = \{x_2, x_4\} \quad b_6 = \{x_3, x_4\}$$

A BIBD can be described by the incidence matrix Q, which is a (v x b) matrix with 0's and 1's as entries. The rows of the matrix correspond to the objects $x_1, x_2, \ldots, x_v$, and the columns of the matrix correspond to the blocks $b_1, b_2, \ldots, b_b$. The entry in the i^{th} row and the j^{th} column of Q is a 1 if the object x_i is in the block b_j, and is a 0 otherwise.

Example 2: The incidence matrix Q of the BIBD in example 1.

	b_1	b_2	b_3	b_4	b_5	b_6
x_1	1	1	1	0	0	0
x_2	1	0	0	1	1	0
x_3	0	1	0	1	0	1
x_4	0	0	1	0	1	1

In communication, the digits 0 and 1 can be transmitted along the channels. The distance messages represented by the sequences of o's and 1's are called code words. A block code consists of a set of code words of fixed length. The distance between two code words is the number of digits at which the two code words differ. The distance of a block code is defined as the minimum distance between all pairs of code words in the code. This distance is a significant characteristic of the code since it determines the number of erroneous digits that can be tolerated in the transmission of a code word. From Definition 2, Corollary 1 is described.

Corollary 1: Given a (b,v,r,k,λ)-configuration, there is a set of v codewords with distance between every two codewords being $2 * (r - \lambda)$

Corollary 2 is inferred from $2e + 1 \leq 2 * (r - \lambda)$, where e is the maximum error-correcting distance.

Corollary 2: Given a distance 2 * (r - λ), the maximum error-correcting distance e is r - λ - 1.

We describe a special BIBD in Definition 3.

Definition 3: A BIBD(Balanced incomplete block design) is called a (v,k,λ)-configuration(SBIBD) if b = v and r = k.

To construct a SBIBD(Symmetric Balanced Incomplete Block Design) is important because a BIBD can be derived from a given SBIBD[21]. The following theorem relates a normalized Hadamard matrix to a SBIBD.

Theorem 14-11[21]: A normalized Hadamard matrix of order n is equivalent to a (v,k,λ)-configuration with v = n - 1,k = (n/2) - 1, and λ = (n/4) - 1. [[21] p. 380]

From Corollary 2 and Theorem 14-11[21], we obtain Corollary 3.

Corollary 3: Given a normalized Hadamard matrix of order n, the maximum error-correcting distance e is (n/2) - (n/4) - 1.

Definition 4: d(x) is a minimum distance from a node x to a Hadamard node a_i.

$$d(x) = \min_{1 \le i \le n} H(x, a_i)$$

From Corollary 3 and the symmetrical aspect of the hypercube, we show the relationship between the covering radius d(x) and the number of nodes corresponding to that radius in Fig. (1). Fig (2) indicates this relationship in the 8-dimensional hypercube(n = 8).

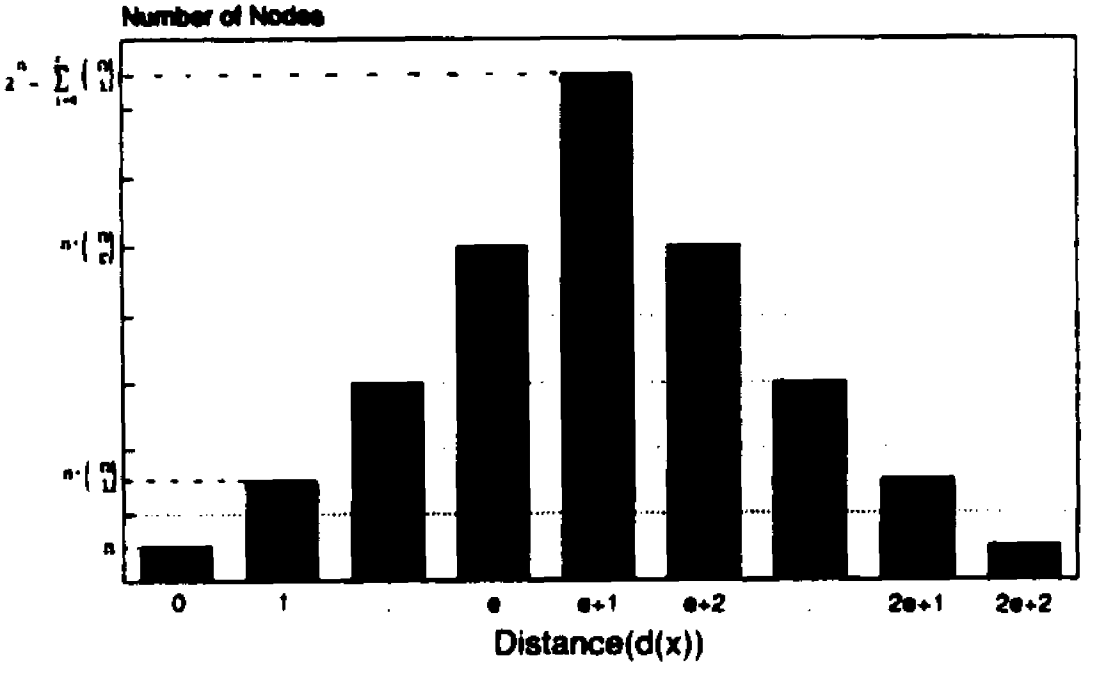

Fig. 1 Distribution of Distance

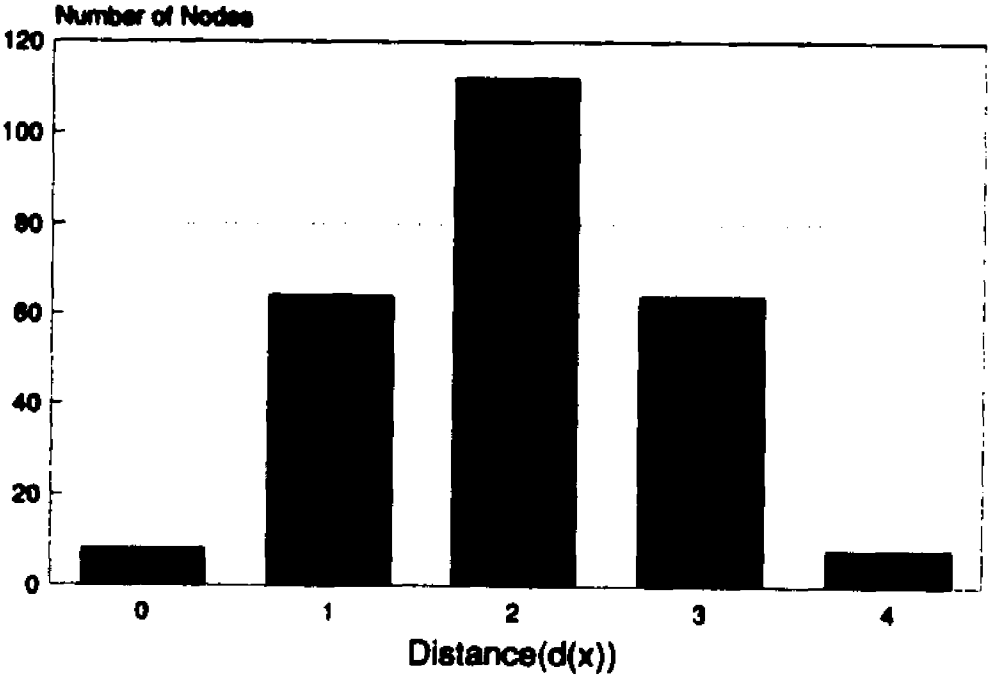

Fig. 2 Distribution of Distance for n=8

5.3.2 Assignment Problem

This problem is to choose n paths from n Hadamard nodes to the n different destination nodes. No two Hadamard nodes should select the same destination node.

Hadamard Matrix M, each row a_i represents a Hadamard node.

$$M = \begin{array}{|cccccccc|c} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & a_0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & a_1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & a_2 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & a_3 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & a_4 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & a_5 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & a_6 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & a_7 \end{array}$$

Destination Matrix D, each row b_i represents a destination node.

$$D = \begin{array}{|cccccccc|c} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & b_0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & b_1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & b_2 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & b_3 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & b_4 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & b_5 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & b_6 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & b_7 \end{array}$$

To find optimal paths from n Hadamard nodes to n destination nodes (algorithm in 5.2;(4) & (5)), the assignment problem will be employed.

The effectiveness Matrix E_0 describes the relation between a_i and b_j ($0 \leq i,j \leq 7$). Each coefficient of this matrix is the length of

the path from a_i(row) to b_j(column), i.e., $H(a_i,b_j)$.

$$E_0 = \begin{vmatrix} 7 & 6 & 6 & 5 & 6 & 5 & 5 & 4 \\ 5 & 4 & 4 & 3 & 4 & 3 & 3 & 2 \\ 3 & 2 & 4 & 3 & 2 & 1 & 3 & 2 \\ 3 & 2 & 4 & 3 & 4 & 3 & 5 & 4 \\ 5 & 4 & 6 & 5 & 6 & 5 & 7 & 6 \\ 5 & 4 & 4 & 3 & 6 & 5 & 5 & 4 \\ 5 & 4 & 6 & 5 & 4 & 3 & 5 & 4 \\ 3 & 2 & 2 & 1 & 4 & 3 & 3 & 2 \end{vmatrix}$$

Applying the Hungarian method, first find the smallest number in each row. So, corresponding to row number 0,1,2,3,4,5,6,7 select the coefficients in column numbers 7,7,5,1,1,3,4,3, respectively. Then, subtract each smallest number from every coefficient belonging to that row. This produces the matrix E_1 below:

$$E_1 = \begin{vmatrix} 3 & 2 & 2 & 1 & 2 & 1 & 1 & 0 \\ 3 & 2 & 2 & 1 & 2 & 1 & 1 & 0 \\ 2 & 1 & 3 & 2 & 1 & 0 & 2 & 1 \\ 1 & 0 & 2 & 1 & 2 & 1 & 3 & 2 \\ 1 & 0 & 2 & 1 & 2 & 1 & 3 & 2 \\ 2 & 1 & 1 & 0 & 3 & 2 & 2 & 1 \\ 2 & 1 & 3 & 2 & 1 & 0 & 2 & 1 \\ 2 & 1 & 1 & 0 & 3 & 2 & 2 & 1 \end{vmatrix}$$

If there is a 0 in each column and in each row, then we are done. Otherwise, we should make at least one 0 in each row and each column. In E_1, there is no 0 in columns 0,2,4 and 6. To construct E_2, choose the smallest coefficient in each column not containing a 0, and subtract that smallest coefficient from every coefficient in that column.

$$E_2 = \begin{vmatrix} 2 & 2 & 1 & 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 \\ 1 & 1 & 0 & 0 & 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 2 & 2 & 1 & 1 \end{vmatrix}$$

Now, in E_2, there is a 0 in all columns and in all rows. In each row, we select one zero coefficient satisfying the condition that no two selections can be in the same column. We select the zeros from column numbers 7,6,4,0,1,2,5, and 3, respectively. This means that a_0 selects b_7, a_1 selects b_6, ... , and a_7 selects b_3. Because there are many possible selections, the selecting process is much more complicated. This process considers the position of each 0, the shortest distance from a_i to b_j, and the better distribution of distances. The better distribution means that we select the smallest of the maximum distances of the chosen n paths from all the alternative routes. Even if the total distance for a certain route is the shortest, it is still possible that one of the paths in that route may have a length > n/2. In this case, the above process may not be an efficient algorithm for communication in the hypercube network, since the packets which have arrived at the destination node already must wait until the packet having the longest distance arrives at the destination node. This cause a delay in reconstructing the original data. Hence, "distribution" is the most important factor. From the point of view of the probabilistic algorithm, the algorithm which uses the Hadamard

matrix is efficient since the information can reach the destination node within a distance of n/2 in most cases.

Example 3: From E_2, we can select the zero coefficients in two ways: from columns 7,6,4,0,1,2,5,3, respectively and columns 6,7,4,0,1,2,5,3, respectively. The total lengths of these two paths is the same, but the maximum length for each is different. The first selection has a maximum length of 4 and the second has a maximum length of 5. Therefore, the first selection is better in terms of the distribution of distances.

However, RRA cannot handle the worst case when large numbers of Hadamard nodes have only one or two destination nodes within a distance of n/2. Given a Hadamard node, how many destination nodes can this node reach within a distance of n/2? If there are k destination nodes within a distance of n/2 from a given Hadamard node, then it can be considered that there are k possible connections from this Hadamard node to each destination node within a distance of n/2.

Example 4: Given Hadamard matrix M(see Section 5.3.2), node u_1 = (10101010), u_2 = (01111111) in the 8-dimensional hypercube network, $H(a_i,u_1)$ is 4,2,4,2,4,4,2,6 for i = 0,1,2,..,7, respectively. This means that the seven different Hadamard nodes can reach node u_1 within a distance of n/2. The path from a Hadamard node to node u_1 can be chosen among the seven paths. $H(a_i,u_2)$ is 1,5,5,5,5,5,5,5 for

$i = 0,1,2,..,7$, respectively. This means that only one Hadamard node arrives at node u_2 within a distance of n/2. There is only one path from a Hadamard node to u_2; namely, from the first Hadamard node to node u_2.

Let the Hadamard nodes be the original nodes and let the distorted nodes be the destination nodes within the framework of the error-correcting code. If a Hadamard node can reach k destination nodes, each within a distance of n/2, one of the k destination nodes is thought to be distorted from that original node. As mentioned earlier, the worst case occurs when a large number of Hadamard nodes have only one or two destination nodes within a distance of n/2 from them. However, for the worst case, a large number of distorted nodes can be recovered from the corresponding original nodes with high probability. From the viewpoint of routing, this recovery can be regarded as designing the shortest path in a specific space produced by the radius of sphere packing. The distance of this path is $\leq (n/2)-(n/4)-1$ by Corollary 3. Finally, RRA does not explain how to design disjoint paths. It only mentions that the randomly selected portions of the paths are almost edge-disjoint paths with high probability. The specifications of these paths will be the topic of Chapter 6.

6

The Design of Secret Routes

When we transmit the data in a distributed network, we must consider dynamic security. Dynamic security is the security of the route taken by the data transmitted over the network. If the route is detected by the adversary, the probability is high that the data are lost or the data can be intercepted by the adversary. Therefore, the route must be protected. To accomplish this, we select an intermediate node secretly and transmit the data using this intermediate node, instead of sending the data to the destination node using the shortest direct path. The above route consisting of two paths - the first path from the source node to the intermediate node and the second path from that intermediate node to the destination node, is called a secret route. Furthermore, if we use a number of secret routes from the starting node to the destination node, data security is much stronger since we can transmit partial data rather than the entire data along a secret route. To employ the above idea, the data is dispersed into n pieces by the DAF(Dispersal Algorithm using the FFT algorithm) in Chapter 3. Then, each piece is transmitted simultaneously to the destination along its own secret route in the n-dimensional hypercube network. For this routing, we must find these n secret routes from the starting node to the destination node. Also, all

the pieces should arrive at the destination in the minimum possible time. Therefore, each secret route should be disjoint from all other secret routes.

Finding a set of disjoint paths in a general network is a computationally difficult problem. However, researchers have proved that there exist sets of disjoint paths in specific kinds of networks. From these proofs, they have designed combinatorial algorithms for finding a set of disjoint paths. Unfortunately, these combinatorial algorithms require much time for obtaining these paths. Some approaches have been tried to reduce the time complexity for these algorithms. As noted in Chapter 5, Rabin has applied an error-correcting code method to the parallel routing algorithms for the hypercube network. Rabin's algorithm employs an (n x n) Hadamard matrix, every two different rows of which differ in exactly n/2 positions. This algorithm is central to the current studies in dynamic security and we continue his work below.

We have two methods for designing n secret paths from the starting node to the intermediate nodes. One of them is to select the intermediate nodes secretly, and then to find the disjoint paths from the starting node to the intermediate nodes. The other is to make the disjoint paths secretly and then to randomly select the nodes on these paths as the intermediate nodes. Since these paths are made secretly and the nodes on these paths are selected randomly, each node in this network is chosen secretly. Rabin's algorithm follows the first method. Valiant's algorithm(see Appendix 5), which involves finding one secret path, may be adopted

to yield an example of the second method. Our method, as yet unexplained, is designed based on the second method.

We propose an alternative algebraic method for secret routing on the hypercube network and on the MRNS(Mixed Radix Number System) network. For the first part of the secret routes from the source node to intermediate nodes, our method is to transform a set of vertex-disjoint paths into disjoint sets and then to investigate these disjoint sets. Later, these sets with some constraints are used to construct a special class of matrices, where each set is used as an element of a matrix. We construct two kinds of special latin squares, HCLS(Hamiltonian Circuit Latin Square) and CSnLS(Cyclic Subsets of order n Latin Square), each of which belongs to this class of matrices, from which the first part of the secret routes are designed. For the second part of the secret routes from these intermediate nodes to the destination node, the PHCLS(Partial Hamiltonian Circuit Latin Square) is employed using the HCLS. The reader is referred to [24] for extensive discussion of latin squares and their applications.

This chapter is composed of the following three sections. Section 1 describes the design of a matrix called ***the special latin square***. Section 2 gives an application of this matrix to the secret routing algorithms for the hypercube network and for the MRNS network. Section 3 provides an application of the parallel routing algorithms of the hypercube network and of the MRNS network.

6.1 The Design of the Special Latin Square

To construct disjoint paths we consider four types of disjoint paths: vertex-disjoint paths, edge-disjoint paths, dynamical vertex-disjoint paths, and dynamical edge-disjoint paths. For constructing the two kinds of dynamical disjoint paths, the "time" factor should be considered. The dynamical vertex-disjoint(edge-disjoint) paths choose distinct vertices(edges) each time, while the vertex-disjoint(edge-disjoint) paths must not repeat any vertex(edge) during the whole transmission. These four disjoint paths are defined as follows:

Definition 1: Let $pos(d_i,t_j)$ be the function that locates the position of d_i at time t_j, where d_i = the i^{th} piece of data. The piece d_i traverses the link $(pos(d_i,t_j),pos(d_i,t_j+1))$.

A set of paths is

i) vertex-disjoint if for $\forall\ t_1,t_2$, $pos(d_i,t_1) \neq pos(d_j,t_2)$,

ii) edge-disjoint if for $\forall\ t_1,t_2$, $(pos(d_i,t_1),pos(d_i,t_1+1)) \neq (pos(d_j,t_2),pos(d_j,t_2+1))$,

iii) dynamical vertex-disjoint if for $\forall\ t_k$, $pos(d_i,t_k) \neq pos(d_j,t_k)$,

iv) dynamical edge-disjoint if for $\forall\ t_k$, $(pos(d_i,t_k),pos(d_i,t_k+1)) \neq (pos(d_j,t_k),pos(d_j,t_k+1))$,

where $i \neq j$ in each case.

Of the four types of disjoint paths mentioned above, the vertex-

disjoint paths require the strongest conditions for their construction. In fact, all vertex-disjoint paths are edge-disjoint paths, dynamical vertex-disjoint paths, and dynamical edge-disjoint paths. Here, we examine the vertex-disjoint paths.

Given n vertex-disjoint paths for n pieces of data, each path of length m, we can easily determine which dimension in n-dimensional hypercube network the i^{th} piece of data traverses each time. Then, these m dimensions taken from the i^{th} path are transformed into m sets, each consisting of the dimension(s) that the i^{th} piece of data has already traversed. To accomplish this, we introduce a special matrix, called the MGVDP, which is constructed from the above (n*m) sets.

Definition 2: Call matrix M the MGVDP(Matrix for Generating Vertex-Disjoint Paths). No two entries of this matrix are the same. This matrix satisfies the following conditions.

(n x m) matrix $M = [\ a_{i,j}\ ]$, $a_{i,j} \subset Z_n^{j+1}$, $0 \le i \le n-1$, $0 \le j \le m-1$.
$a_{i,j}$ = the set of all dimensions that the i^{th} piece of data has already traversed at time j+1.

i) $|a_{i,j}| = j + 1$

ii) $a_{i,j} = a'_{i,j}$, where $a'_{i,j}$ = {elements in $a_{i,j}$ that occur an odd number of times in $a_{i,j}$}.

iii) $a_{i,j} \neq a_{p,q}$, if $(i \neq p)$ or $(j \neq q)$, $0 \le p \le n-1$, $0 \le q \le m-1$.

iv) $a_{i,j} \subset a_{i,j+1}$

This matrix, described by Definition 1, has interesting properties. If each element of M indicates a "relative address" of the starting node with respect to a given intermediate node, then each element of M represents a unique relative address, since no two entries of M are the same. Definition 2_{iii} should be required in a certain case. If a piece traverses the same dimension an even number of times, then the numerical value(whether 0 or 1) of that dimension is not changed(see Definition 2 in Section 4.2.2). Definition 2_{iv} proposes a relationship between consecutive elements of each row, namely, that is only one path between two intermediate nodes along the dimension $(a_{i,j+1} - a_{i,j})$. In order to use each element of M as the relative address of the starting node with respect to a given intermediate node in the n-dimensional hypercube network, each element of M is transformed into a string of binary numbers of length n.

Definition 3: Let M be the TBM(Transformed into Binary-set Matrix).

$[a_{i,j}] \subset$ MGVDP, $[b_{i,j}] \subset$ TBM.

$f : \{ a_{i,j} \mid a_{i,j}[k] \in Z_n,\ 0 \le k \le j \} \rightarrow \{0,1\}^n$,

where $a_{i,j}[k]$ is the k^{th} element of $a_{i,j}$.

$$f_{a_{i,j}}(b_{i,j}) = \begin{cases} b_{i,j}[k] = 1 & \text{if } k \in a_{i,j},\ 0 \le k \le n-1 \\ b_{i,j}[k] = 0 & \text{otherwise,} \end{cases}$$

where $b_{i,j}[k]$ is the k^{th} element of $b_{i,j}$.

Example 1: A set of vertex-disjoint paths, the MGVDP and the TBM, where m, n = 5.

A set of vertex-disjoint paths

00000 → 00010 → 00110 → 01110 → 11110 → 11111
00000 → 00100 → 01100 → 11100 → 11101 → 10101
00000 → 01000 → 11000 → 11001 → 11011 → 01011
00000 → 10000 → 10001 → 10011 → 10111 → 10110
00000 → 00001 → 00011 → 00111 → 01111 → 01101

MGVDP

(1)	(1,2)	(1,2,3)	(1,2,3,4)	(1,2,3,4,0)
(2)	(2,3)	(2,3,4)	(2,3,4,0)	(2,3,4,0,3)
(3)	(3,4)	(3,4,0)	(3,4,0,1)	(3,4,0,1,4)
(4)	(4,0)	(4,0,1)	(4,0,1,2)	(4,0,1,2,0)
(0)	(0,1)	(0,1,2)	(0,1,2,3)	(0,1,2,3,1)

TBM

(00010)	(00110)	(01110)	(11110)	(11111)
(00100)	(01100)	(11100)	(11101)	(10101)
(01000)	(11000)	(11001)	(11011)	(01011)
(10000)	(10001)	(10011)	(10111)	(10110)
(00001)	(00011)	(00111)	(01111)	(01101)

Remark : Note that each entry of the MGVDP is a set of bit positions which have been complemented with respect to the starting node.

Given the starting node and an element of the TBM as a relative address of the starting node with regard to some intermediate node, each piece traverses to its unique intermediate node. Therefore, n vertex-disjoint paths are constructed. Each path has length m.

This author believes that designing the MGVDP is a computationally difficult problem. In order to decrease the difficulty, we construct a subclass of the MGVDP, adding two constraints to the conditions of the MGVDP. The first constraint is that all the elements in each $a_{i,j}$ be distinct. This means that the data cannot traverse the same dimension more than once. The second constraint is that $m = n - 1$. This means that the length of each path is (n-1). The following definition introduces a special class of matrices which belongs to the MGVDP.

Definition 4: Call matrix M the NSEM(No Same-Entry Matrix). No two entries of this matrix are the same. This matrix satisfies the following conditions.

$(n \times (n-1))$ matrix $M = [\ a_{i,j}\ ]$. $0 \le i \le n-1$, $0 \le j \le n-2$.
$a_{i,j} \subset Z_n$.

i) $|a_{i,j}| = j + 1$

ii) $a_{i,j} \neq a_{k,j}$, if $i \neq k$

iii) $a_{i,j} \subset a_{i,j+1}$

In this thesis, our main goal is to design the NSEM. This author thinks that this goal is a computationally difficult problem. To decrease the difficulty, we construct a subclass of the NSEM, adding one constraint to the conditions of the NSEM. The constraint is that no two pieces traverse the same dimension. To accomplish this, we introduce a method that transforms special latin squares

into modified special matrices which are NSEMs. *A latin square(see Appendix 6)* is a square matrix with n^2 entries of n different elements, none of them occurring twice within any row or column of the matrix. The integer n is called the order of the latin square. Two special latin squares are described below. One is *the cyclic subsets of order n latin square(CSnLS)*, and the other is *the Hamiltonian circuit latin square(HCLS)*. Later, a modified special latin square is introduced.

To generate the CSnLS, we investigate relations between elements in the same row, and in the same column. This means that each element is derived from another element and an operation that are predetermined. To explore the relation between elements in the same column and the operation, Definition 5 is introduced.

Definition 5:

Let $S_0 \subset Z_n$, $S_1 = f_k(S_0) = \{ k \cdot s \mid s \in S_0 \}$, $S_{n+1} = f_k(S_n)$. The order of S_0 is the smallest i such that $S_i = f_k^i(S_0) = f_k^j(S_0)$, $0 \le j < i$. Let $<S> = \{ S_0, S_1, S_2, \ldots, S_{i-1} \}$. Then $(<S>, f_k)$ is called the cyclic subsets of order i generated by f_k.

Example 2: $S_0 \subset Z_8$, $S_0 = \{ 1,3,4 \}$, f_k means " multiply by 2".

$S_0 = \{ 1,3,4 \}$
$S_1 = \{ 2,6,0 \}$
$S_2 = \{ 4,4,0 \}$
$S_3 = \{ 0,0,0 \}$
$S_4 = \{ 0,0,0 \}$

S_4 is equal to S_3. The order of S_0 is 4.

Also, we examine the relation between elements in the same row and an operation. Definition 6 is given below.

Definition 6:

Let $g_0 \in Z_n$, $g_1 = f_k(g_0) = k \bullet g_0$, $g_{m+1} = f_k(g_m)$.
The order of f_k is the smallest i such that $f_k^i(g_0) = f_k^j(g_0)$, $0 \leq j < i$. If the order of $f_k = n$, $f_k \in S_{(\bullet)}$,
where $S_{(\bullet)}$ = { operations of order n on g_0 }

From Definition 6, the size of $S_{(\bullet)}$ = {"add k" of order n | $k \in Z_n$} can be computed.

Proposition 1: Let f_k be "add k". Then, $S_{(\bullet)}$ is the set of numbers in Z_n that are relatively prime to n; ($S_{(\bullet)} = \{s \in Z_n \mid \gcd(n,s)=1\}$).

Using the above definitions and Proposition 1, the CSnLS is defined below.

Definition 7 : The CSnLS M is defined as follows.

(n x n) matrix $M = [a_{i,j}]$, $a_{i,j} \in Z_n$, $a_{0,p} \neq a_{0,q} (p \neq q)$, $0 \leq i,j,p,q \leq n-1$. $a_{i,j} = a_{0,j} + i*k \pmod n$, $f_k \in S_{(*)}$ (Proposition 1). Let $A_{0,k}$ be $(a_{0,0}, a_{0,1}, \ldots, a_{0,k})$, $0 \leq k \leq n-2$. We require that the order of $A_{0,k}$ (see Definition 5) is n, $0 \leq k \leq n-2$.

Theorem 1 : The CSnLS defined in Definition 7 is a latin square matrix.

Proof : Using Definition 6, each column of M is generated by the first element in that column and f_k. Since each element in the first row is distinct, each element in the remaining rows is also distinct. Therefore, Theorem 1 is proved.

Example 3: The CSnLS.

Let n = 6, k = 1, and the first row of M be (0,2,1,3,5,4).

$$M = \begin{vmatrix} 0 & 2 & 1 & 3 & 5 & 4 \\ 1 & 3 & 2 & 4 & 0 & 5 \\ 2 & 4 & 3 & 5 & 1 & 0 \\ 3 & 5 & 4 & 0 & 2 & 1 \\ 4 & 0 & 5 & 1 & 3 & 2 \\ 5 & 1 & 0 & 2 & 4 & 3 \end{vmatrix}$$

We should examine whether M satisfies the following conditions of CSnLS.

	subset	order
(1)	$A_{0,1} = \{0,2\}$	6
(2)	$A_{0,2} = \{0,2,1\}$	6
(3)	$A_{0,3} = \{0,2,1,3\}$	6
(4)	$A_{0,4} = \{0,2,1,3,5\}$	6

Since the order of $A_{0,k}$ is n, this matrix M belongs to the CSnLS. From the CSnLS, we now design *the MCSnM(Modified Cyclic Subsets of order n Matrix)*.

Theorem 2. The MCSnM satisfies the conditions of the NSEM.

Proof: A CSnLS matrix M is defined in Definition 7, using k = 1.

$$M = \begin{vmatrix} a_0 & a_1 & .. & a_{n-2} & a_{n-1} \\ a_0+1 & a_1+1 & .. & a_{n-2}+1 & a_{n-1}+1 \\ a_0+2 & a_1+2 & ... & a_{n-2}+2 & a_{n-1}+2 \\ & & \cdots & & \end{vmatrix}$$

The CSnLS matrix M is transformed into the MCSnM L as follows:

$$L_{i,j} = \{ a_0+i, a_1+i, \ldots, a_j+i \}, \quad 0 \le i \le n-1, \quad 0 \le j \le n-2.$$

The matrix L satisfies Definition 4_{ii} and 4_{iii} by the designing strategy of this matrix. The second condition of Definition 4 is automatically satisfied by the condition of Definition 7; the order of subset $L_{0,j}$ is n. This implies that $L_{0,j}, L_{1,j}, \ldots,$ and $L_{n-1,j}$ are distinct. Therefore, the matrix L satisfies the conditions of NSEM.

Example 4: (4 x 4) CSnLS and (4 x 3) MCSnM

CSnLS

1	0	3	2
2	1	0	3
3	2	1	0
0	3	2	1

MCSnM

(1)	(1 0)	(1 0 3)
(2)	(2 1)	(2 1 0)
(3)	(3 2)	(3 2 1)
(0)	(0 3)	(0 3 2)

Corollary 1: The row complete latin square(see Appendix 8) is a subset of the CSnLS.

Another method to generate the special latin square is to design the HCLS and then to construct *the MHCM(Modified Hamiltonian Circuit Matrix)* from the HCLS. The HCLS is defined below.

Definition 8: The HCLS matrix M is constructed as follows:

Let M be an (n x n) square matrix. Given the Hamiltonian circuit $a_0 \rightarrow a_1 \rightarrow \ldots \rightarrow a_{n-2} \rightarrow a_{n-1} \rightarrow a_0$, each row of M is obtained from the Hamiltonian path starting at any position $a_j (0 \leq j \leq n-1)$ under the condition that no two rows begin at the same position.

Example 5: (n x n) HCLS M

$$M = \begin{vmatrix} a_0 & a_1 & \ldots a_{n-2} & a_{n-1} \\ a_1 & a_2 & \ldots a_{n-1} & a_0 \\ & \ldots\ldots\ldots\ldots & & \\ a_{n-1} & a_0 & \ldots\ldots & a_{n-2} \end{vmatrix}$$

Theorem 3 : The HCLS is a latin square.

Proof: Since each row is taken from a the Hamiltonian path, each element in given row is distinct. Each element in given column is also distinct, because each row begins at a different position of the Hamiltonian circuit.

From the HCLS, *the MHCM(Modified Hamiltonian Circuit Matrix)* is constructed below.

Theorem 4 : The MHCM satisfies the conditions of the NSEM.

Proof : Let M be an HCLS. The HCLS M is transformed into the MHCM L as follows:

$L_{i,j} = \{ a_i, a_{i+1}, \ldots, a_{i+j} \}$, $0 \le i \le n-1$, $0 \le j \le n-2$.

The matrix L automatically satisfies the first and the third conditions of Definition 4 by the designing strategy. The second condition of Definition 4 will be proved. Suppose that two subsets $L_{i,j}$ and $L_{k,j}$ $(i \neq k)$ start at a different element and that these subsets are the same. This implies that the size of each subset

must be n. However, this is a contradiction, since the size of $L_{1,j}$ is j+1 and j ≤ n-2. Therefore, the matrix L satisfies the second condition of Definition 4.

Corollary 2: The Cayley table(see Appendix 7) of cyclic group of order n is a subset of the HCLS.

Proposition 2: The total number of ways to generate the HCLS of order n is n!*(n-1)!.

Proof: Given n numbers, the number of choosing the first row is n!. From Definition 6, the second row can start at any position among (n-1) positions. The third row starts at any position out of (n-2) positions. The construction for the rest rows of the HCLS follow the above process. Therefore, the total number of ways to generate the HCLS is n!*(n-1)*(n-2)*···*1 = n!*(n-1)!

Example 6: (4 x 4) HCLS and (4 x 3) MHCM

HCLS

1	2	3	0
2	3	0	1
3	0	1	2
0	1	2	3

MHCM

(1)	(1 2)	(1 2 3)
(2)	(2 3)	(2 3 0)
(3)	(3 0)	(3 0 1)
(0)	(0 1)	(0 1 2)

We now modify the NSEM so that we may apply the routing function. The basic assumption in designing the NSEM(see Definition 3) is that given a set of vertex-disjoint paths, each path has the same length. However, in most cases, these vertex-disjoint paths do not actually have the same length. In the hypercube network, it takes one time unit to transmit one piece to one of the neighboring nodes. In order to satisfy the basic assumption that each path has the same length, we add time unit(s) to some paths which has shorter lengths. This means that each piece which arrives at its destination node earlier must wait at the destination node until other pieces arrive at their respective destination nodes. Then, we can say that all the paths have the same length. Considering this factor, the MNSEM(Modified No Same-Entry Matrix) is defined as follows.

Definition 9: Call matrix M the MNSEM(Modified No Same-Entry Matrix). No two entries of this matrix are the same. This matrix satisfies the following conditions.

(n x (n-1)) matrix $M = [\ a_{i,j}\ ]$. $0 \le i \le n-1$, $0 \le j \le n-2$.
$a_{i,j} \subset \{\ Z_n \cup s\ \}$. s means "stay at the current node".

i) $|a_{i,j}| = j + 1$

ii) $a_{i,j} \neq a_{k,j}$, if $i \neq k$

iii) $a_{i,j} \subset a_{i,j+1}$

iv) $a_{i,j+1} = a_{i,j} \cup \{s\}$, if $s \notin a_{i,j}$

Example 7: A set of vertex-disjoint paths and (5 x 4) MNSEM.

A set of vertex-disjoint paths

```
00000 → 00010 → 00110 → 01110 → 11110
00000 → 00100 → 01100 → 11100 → 11101
00000 → 01000 → 11000
00000 → 10000 → 10001 → 10011
00000 → 00001 → 00011
```

MNSEM

(1)	(1,2)	(1,2,3)	(1,2,3,4)
(2)	(2,3)	(2,3,4)	(2,3,4,0)
(3)	(3,4)	(3,4,s)	(3,4,s,s)
(4)	(4,0)	(4,0,1)	(4,0,1,s)
(0)	(0,1)	(0,1,s)	(0,1,s,s)

Now, we can apply the routing function to the MNSEM above to transmit all the pieces to the destination nodes in the hypercube network.

6.2 The Application of the Special Matrix(MNSEM) to Secret Routing Algorithms

The special latin squares, the CSnLS(Cyclic Subsets of order n Latin Square) and the HCLS(Hamiltonian Circuit Latin Square), are applied to the secret routing algorithms of the hypercube network and of the MRNS(Mixed Radix Number System) network. The routing algorithms are composed of two phases. The first phase involves transmission of each piece to a randomly chosen intermediate node along a secret path. The second phase involves sending each piece

from that intermediate node to the destination node along a secret path. This section of Chapter 6 consists of two parts, 6.2.1 presents three routing algorithms of the hypercube network, 6.2.2 describes two routing algorithms of the MRNS network.

6.2.1 Three Routing Algorithms of the Hypercube Network

For the n-dimensional hypercube network, let x and $\pi(x)$ be the starting node and the destination node, respectively, where x, $\pi(x)$ ε $\{0,1\}^n$, $\pi:\{0,1\}^n \rightarrow \{0,1\}^n$ is a permutation. In this paper, the packet P_x residing in node x will be dispersed into n pieces P_{x1}, P_{x2}, ... , P_{xn}, so that any (n - k) pieces suffice to reconstruct P_x, where k is the number of missing pieces.

In the transfer of P_x from x to $\pi(x)$, a vector T_{x1} will be "attached" to P_{x1}, which indicates all the dimensions that P_{x1} shall traverse. Each vector T_{x1} has the same length, the length of T_{x1} is $\leq$ (2n-1), and T_{x1} is chosen so that all the paths are disjoint. Then, P_{x1} will be transmitted independently of all other $P_{xj}(i \neq j)$.

Say, P_{x1} resides at some node j at time t. If $T_{x1}[t] \varepsilon Z_n$, then P_{x1} traverses along dimension $T_{x1}[t]$ to the node that node j is physically connected to. Otherwise, if $T_{x1}[t] \notin Z_n$, let $T_{x1}[t] = s$, which means that P_{x1} remains at node j at time t.

As mentioned earlier, the routing process is composed of two phases. In the first phase, we design n secret paths by secretly making n disjoint paths, and then by choosing a node randomly on each path. A special latin square(either CSnLS or HCLS) is employed

to secretly make n disjoint paths. In order to select nodes randomly, the lengths of the paths are chosen randomly since a node is identified by the length of its path. Given the maximum length of the path = (n-1) (see Definition 3) and the random selection of the length of each path, every node (except for two of them) has the same probability of being chosen. The two exceptional nodes are nodes x_1 and x_2, where x_1 = the starting node, and $H(x_1, x_2) = n$. In order for n pieces to start simultaneously on the second phase, we add time unit(s) to some T_{x_i}'s. If $|D_i|$ is the maximum length of a path in the first phase (where D_i is the path for the i^{th} piece of data, $1 \leq i \leq n$), we add $(|D_i| - |D_j|)$ s's to the end of the $T_{x_j}[|D_j|]$. Then, the MNSEM is designed.

In the second phase, we can easily determine bit positions that differ between the intermediate node and the destination node for each piece. However, we do not know which dimension the piece traverses each time. Using this information, we will construct the MNSEM.

For the above communication model, the special latin square HCLS is employed. The secret routing algorithm for transmission of the data P_x from node x to node $\pi(x)$ is implemented as below.

Routing Algorithm 1.

cobegin $x \in \{0,1\}^n$

1) Split P_x into n pieces $P_{x1}, P_{x2}, .., P_{xn}$ using the DAF, P_x is the data at node x.
2) Randomly choose a sequence from the n! permutations of $<0,1,2,...,n-1>$.
3) Design an (n x n) HCLS and construct the (n x (n-1)) matrix by randomly choosing (n-1) columns of the HCLS.
4) Randomly choose k_i from the set $\{1,2,..,n-1\}$ for the length of the i^{th} path, and design pairwise vertex-disjoint paths $D_1, D_2, .., D_n$, $|D_i| = k_i (1 \le i \le n)$, from x to $R_1, R_2, .. R_n$. Each R_i is the intermediate node of the i^{th} path.
5) Construct a set of different bit positions of R_i and $\pi(x)$ for the i^{th} piece, $1 \le i \le n$.
6) Design the MNSEM from the above sets.
7) Using the above MNSEM, make pairwise vertex-disjoint paths $E_1, E_2, .., E_n$ from $R_1, R_2, .. R_n$, respectively, to the destination node $\pi(x)$. Each path E_i has length $\le n$.
8) Attach the i^{th} routing path (D_i, E_i) to P_{xi}.

coend.

Example 8: Let n = 4, x = (0000) and $\pi(x)$ = (1100). Then, Routing Algorithm 1 is executed as follows:

From (1)

The data at node x is dispersed into n pieces using the DAF.

From (2)

Select a sequence (2 1 3 0).

From (3)

Design the HCLS, and construct the matrix by randomly choosing (n-1) columns.

the HCLS	(n x (n-1)) matrix
2 1 3 0	2 1 3
1 3 0 2	1 3 0
3 0 2 1	3 0 2
0 2 1 3	0 2 1

From (4)

Randomly choose 2,2,3,2 for the lengths of the paths D_1, D_2, D_3, D_4, respectively.

D_1 : (0000) -> (0110) : (2,1)
D_2 : (0000) -> (1010) : (1,3)
D_3 : (0000) -> (1101) : (3,0,2)
D_4 : (0000) -> (0101) : (0,2)

For the synchronization, we add ($|D_3|$ - $|D_1|$) s's to the end of $T_{x1}[|D_i|]$, $1 \le i \le 4$.

D_1 : (2,1,s)
D_2 : (1,3,s)
D_3 : (3,0,2)
D_4 : (0,2,s)

From (5)

E_1 : (0110) -> (1100) : different bit positions = (1,3)
E_2 : (1010) -> (1100) : different bit positions = (1,2)
E_3 : (1101) -> (1100) : different bit position = (0)
E_4 : (0101) -> (1100) : different bit positions = (0,3)

For the synchronization, we add ($|E_1|$ - $|E_i|$) s's to the end of $T_{xi}[|E_i|]$, $1 \leq i \leq 4$.

E_1 : (1,3)
E_2 : (1,2)
E_3 : (0,s)
E_4 : (0,3)

From (6) & (7)

Find the MNSEM. s means "stay at the current node".

(1)	(1 3)	
(2)	(2 1)	
(0)	(0 s)	
(3)	(3 0)	

From (8)

(P_{x1},2,1,s,3,1)
(P_{x2},1,3,s,1,2)
(P_{x3},3,0,2,s,0)
(P_{x4},0,2,s,0,3)

This routing algorithm requires a lot of time to find the vertex-disjoint paths from the intermediate nodes to the destination node(see the 6th step in Routing Algorithm 1). A simplified algorithm for finding n secret paths should be developed. Now, we apply the partition method to Routing Algorithm 1: This partition method is used to design a number of "small" MNSEMs based on the size of the set of bit positions that differ between each intermediate node the destination node, rather than

only one "large" MNSEM. This method definitely reduces the time complexity for designing n secret paths.

Routing Algorithm 2.

cobegin $x \in \{0,1\}^n$

1) Split P_x into n pieces $P_{x1}, P_{x2}, .., P_{xn}$ using the DAF, P_x is the data at node x.
2) Randomly choose a sequence from the n! permutations of $<0,1,2,...,n-1>$.
3) Design an (n x n) HCLS and construct the (n x (n-1)) matrix by randomly choosing (n-1) columns of the HCLS.
4) Randomly choose k_i from the set $\{1,2,..,n-1\}$ for the length of the i^{th} path, and design pairwise vertex-disjoint paths $D_1, D_2, .., D_n$, $|D_i| = k_i (1 \le i \le n)$, from x to $R_1, R_2, .. R_n$. Each R_i is the intermediate node of the i^{th} path.
5) Construct a set of different bit positions of R_i and $\pi(x)$ for the i^{th} piece, $1 \le i \le n$.
6) Collect the sets that have the same size, design the NSEM from these sets.
7) Using the above NSEMs, make dynamical vertex-disjoint paths $E_1, E_2, .., E_n$ from $R_1, R_2, .. R_n$, respectively, to the destination node $\pi(x)$. Each path E_i has length $\le n$.
8) Attach the i^{th} routing path (D_i, E_i) to P_{xi}.

coend.

Example 9: Let n = 4, x = (0000) and $\pi(x)$ = (1100). Then, Routing Algorithm 2 is executed as follows:

From (1)

The data at node x is dispersed into n pieces using the DAF.

From (2)

Select a sequence (2 1 3 0).

From (3)

Design the HCLS, and construct the matrix by randomly choosing (n-1) columns of the HCLS.

the HCLS	(n x (n-1)) matrix
2 1 3 0	2 1 3
1 3 0 2	1 3 0
3 0 2 1	3 0 2
0 2 1 3	0 2 1

From (4)

Choose(randomly) 2,2,3,2 for the lengths of the paths D_1, D_2, D_3, D_4, respectively.

D_1 : (0000) -> (0110):(2,1)
D_2 : (0000) -> (1010):(1,3)
D_3 : (0000) -> (1101):(3,0,2)
D_4 : (0000) -> (0101):(0,2)

For the synchronization, we add ($|D_3|$ - $|D_1|$) s's to the end of $T_{x_i}[|D_i|]$, $1 \le i \le 4$.

D_1 : (2,1,s)
D_2 : (1,3,s)
D_3 : (3,0,2)
D_4 : (0,2,s)

From (5)

E_1 : (0110) -> (1100) : different bit positions = (1,3)
E_2 : (1010) -> (1100) : different bit positions = (1,2)
E_3 : (1101) -> (1100) : different bit position = (0)
E_4 : (0101) -> (1100) : different bit positions = (0,3)

From (6)

Collect the sets that have the same size, find the NSEM from these sets. s means "stay at the current node".

Size 1 : (0)

Size 2 : (1 3), (1 2), (3,0)

Find the NSEM for the sets of size 2.

| (3) (3 1) |
| (1) (1 2) |
| (0) (0 3) |

For the synchronization, we add one s to the end of E_3.

From (7).

$(P_{x1}, 2, 1, s, 3, 1)$
$(P_{x2}, 1, 3, s, 1, 2)$
$(P_{x3}, 3, 0, 2, 0, s)$
$(P_{x4}, 0, 2, s, 0, 3)$

Proposition 3: In Routing Algorithm 2, the n pieces take n dynamical vertex-disjoint paths from the intermediate nodes to the destination node.

Proof: Each piece, whose set of bit positions has size k ($k < n$) traverses along a path which is vertex-disjoint from the paths of all other pieces whose sets of bit positions also have size k. If the set for P_{xi} has size z_1 and the set for P_{xj} has size z_2, where $z_1 \neq z_2$, then the two pieces take dynamical vertex-disjoint paths from the intermediate nodes to the destination node because the two paths taken have different lengths. Therefore, the second phase of

Routing Algorithm 2 generates n dynamical vertex-disjoint paths.

The above algorithm is much better than Routing Algorithm 1 in terms of the time complexity. However, Routing Algorithm 2 should also specify how to design the NSEM as in Routing Algorithm 1(see the 6^{th} step in Routing Algorithm 2). The design of the NSEM belongs to a class of purely combinatorial problems. This means that the problem of finding vertex-disjoint paths from the intermediate nodes to the destination node $\pi(x)$ belongs to a class of purely combinatorial problems.

Generally, dynamical edge-disjoint paths have the following property: given a set of nodes, each node transmits at most one packet via the distinct link each time. In some sense, finding dynamical edge-disjoint paths is more complicated than finding vertex-disjoint paths, since the vertex-disjoint paths do not care about the above property. If the paths are vertex-disjoint paths, each node automatically transmits one piece of data. To design dynamical edge-disjoint paths, we consider an intelligent distribution of traversing positions. The question is how to distribute these positions intelligently for all the paths.

We introduce the masking function and the partial latin square to implement the above idea. Later, we propose a new routing algorithm and illustrate an example of this algorithm.

Definition 10: Let F be a masking function.

$$(n \times n) \text{ HCLS } M = [a_{i,j}],\ a_{i,j} \in Z_n,\ 0 \le i,j \le n-1.$$

$$F_{r_0,r_1,\ldots,r_{n-1}}(a_{i,j}) = \begin{cases} a_{i,j} & \text{if } a_{ij} \in r_i,\ r_i \subseteq Z_n \\ s & \text{otherwise} \end{cases}$$

Definition 11: An (n x n) incomplete or partial latin square is an (n x n) array such that in some subset of the n^2 cells of the array each of the cells is occupied by an integer from the set 0,1,2,..,n-1 and such that no integer from 0,1,2,..,n-1 occurs in any row or column more than once.

The PHCLS(Partial Hamiltonian Circuit Latin Square) is constructed according to Definition 11 employing the masking function specified by Definition 10.

Definition 12: Given an (n x n) HCLS, this matrix is transformed into an incomplete or partial latin square by the masking function. Then this transformed matrix is called the PHCLS.

Theorem 5: There exist n dynamical edge-disjoint paths - $E_1, E_2, .., E_n$ - from the intermediate node $R_1, R_2, .., R_n$, respectively, to the destination node $\pi(x)$. Each path E_i has length $\le n$.

Proof : There is a unique set s_i of different bit positions between the intermediate node R_i and the destination node $\pi(x)$ for

the i^{th} piece P_{xi}. Let k be the Hamming distance between R_i and $\pi(x)$. The piece P_{xi} will be sent exactly k times to other nodes. The PHCLS is constructed using the masking function in Definition 10 - let M be an HCLS, and let r_i be a unique set of bit positions. The length of each path E_i is at most n. At time t, let A and B be two sets whose elements are bit positions that two pieces have already traversed. We must consider two cases.

Case 1 : If A = B, then the two pieces stay at two different nodes since they started at two different nodes.

Case 2 : If A ≠ B, then there are two possibilities. Either the two pieces arrive at two different nodes at time t, or they both arrive at the same node at time t. If they arrive at two different nodes at time t, we have nothing to prove. However, we must consider the possibility that they both arrive at the same node y at time t. There is a unique path between each intermediate node and the destination node for each piece. At time t+1, it is impossible for node y to send two pieces to the same node because of the bit positions traversed are determined by the latin square.

The following algorithm is the same as Routing Algorithm 2 except for the second part of the route. In this algorithm, the second part of the route is developed by the PHCLS.

Routing Algorithm 3.

cobegin $x \in (0,1)^n$

1) Split P_x into n pieces $P_{x1}, P_{x2}, .., P_{xn}$ using the DAF, P_x is the data at node x.
2) Randomly choose a sequence from the n! permutations of $<0,1,2,...,n-1>$.
3) Design an (n x n) HCLS and construct the (n x (n-1)) matrix by randomly choosing (n-1) columns of the HCLS.
4) Randomly choose k_i from the set $\{1,2,..,n-1\}$ for the length of the i^{th} path, and design pairwise vertex-disjoint paths $D_1, D_2, .., D_n$, $|D_i| = k_i (1 \le i \le n)$, from x to $R_1, R_2, .. R_n$. Each R_i is the intermediate node of the i^{th} path.
5) Construct a set of different bit positions of R_i and $\pi(x)$ for the i^{th} piece, $1 \le i \le n$.
6) Randomly choose a sequence from the n! permutations of $<0,1,2,...,n-1>$ and make an (n x n) HCLS.
7) From (5) and (6), we design the PHCLS, and then make dynamical edge-disjoint paths $E_1, E_2, .., E_n$ from $R_1, R_2, .. R_n$, respectively, to the destination node $\pi(x)$. Each path E_i has length $\le n$.
8) Attach the i^{th} routing path(D_i, E_i) to P_{xi}.

coend.

Example 10: Let n = 4, x = (0000) and $\pi(x)$ = (1100). Then, Routing Algorithm 3 is executed as follows:

From (1)

The data at node x is dispersed into n pieces using the DAF.

From (2)

Select a sequence (2 1 3 0).

From (3)

Design the HCLS, and construct the matrix by randomly selecting (n-1) columns of the HCLS.

the HCLS	(n x (n-1)) matrix
| 2 1 3 0 |	| 2 1 3 |
| 1 3 0 2 |	| 1 3 0 |
| 3 0 2 1 |	| 3 0 2 |
| 0 2 1 3 |	| 0 2 1 |

From (4)

Randomly choose 2,2,3,2 for the lengths of the paths D_1, D_2, D_3, D_4, respectively.

D_1 : (0000) -> (0110) : (2,1)
D_2 : (0000) -> (1010) : (1,3)
D_3 : (0000) -> (1101) : (3,0,2)
D_4 : (0000) -> (0101) : (0,2)

For the synchronization, we add ($|D_3|$ - $|D_1|$) s's to the end of $T_{x_i}[|D_i|]$, $1 \le i \le 4$.

D_1 : (2,1,s)
D_2 : (1,3,s)
D_3 : (3,0,2)
D_4 : (0,2,s)

From (5)

E_1 : (0110) -> (1100) : different bit positions = (1,3)
E_2 : (1010) -> (1100) : different bit positions = (1,2)
E_3 : (1101) -> (1100) : different bit positions = (0)
E_4 : (0101) -> (1100) : different bit positions = (0,3)

From (6)

Select a sequence (0 3 1 2)

```
| 0 3 1 2 |
| 3 1 2 0 |
| 1 2 0 3 |
| 2 0 3 1 |
```

From (7)

```
| 0 3 1 2 |     | s 3 1 s |       | (s) (s 3) (s 3 1) (s 3 1 s) |
| 3 1 2 0 | ->  | s 1 2 s |  ->   | (s) (s 1) (s 1 2) (s 1 2 s) |
| 1 2 0 3 |     | s s 0 s |       | (s) (s s) (s s 0) (s s 0 s) |
| 2 0 3 1 |     | s 0 3 s |       | (s) (s 0) (s 0 3) (s 0 3 s) |
```

From (8)

$(P_{x1}, 2, 1, s, s, 3, 1, s)$
$(P_{x2}, 1, 3, s, s, 1, 2, s)$
$(P_{x3}, 3, 0, 2, s, s, 0, s)$
$(P_{x4}, 0, 2, s, s, 0, 3, s)$

Considering the aspect of fault-tolerant communication, we will analyze the above routing algorithm, in which for a given permutation $\pi \in S_N$, packet P_x is dispersed into n pieces and these pieces are routed simultaneously from x to $\pi(x)$ for every $x \in \{0,1\}^n$, with the following comments.

Comment 1: Every $m = \lfloor 5n/6 \rfloor$ of the pieces $P_{x1}, \ldots, P_{xn}$ suffice to reconstruct P_x. Later on, for Theorem 2, we shall take $m = \lfloor n/2 \rfloor$.

Comment 2: At any time $1 \leq t \leq 2n-1$, if a buffer BF(y) at a node y receives more than 5n pieces P_{x1}, then the overflow above its capacity of 6*L(the equivalent of six original packets), will be rejected and lost, where L = the size of the original packet.

The following two theorems discuss the probability of all packets arriving at the destination under the conditions : 1) a fixed size for the queue of each node; 2) a fixed number of links randomly fail. These theorems are taken from [7].

Theorem 1[7]: Under any routing algorithm, for any given permutation π, the probability of all packets reaching their destination is at least $1 - (1/N^4)$.

Theorem 2[7]: Assume that within a transmission round, fewer than N/n of the links randomly fail. If we break each packet P_x into n pieces so that any $m = \lfloor n/2 \rfloor$ pieces suffice for reconstruction of P_x, and if we employ a routing algorithm with buffer size $|BF(y)| = p^*|P_x|$ large enough to make the probability of buffer-overflow negligible, then the probability of all packets P_x, for every $x \in \{0,1\}^n$, reaching their respective destination is at least $1 - 2 * N * (4*e/n)^{0.25n}$.

From Routing Algorithm 3, we calculate the number of n dynamical edge-disjoint paths that can be generated.

Proposition 4: A set of n dynamical edge-disjoint paths constructed from Routing Algorithm 3 is selected from among $n!*(n-1)^n*((n-1)!)^2*n$.

Proof: From the 2nd step in Routing Algorithm 3, we choose a

sequence from n! sequences. From the 3rd step, we design n different (n x (n-1)) matrices from the given the HCLS of order n. From the 4th step, $(n-1)^n$ paths can be constructed. From the 6th step, the number of ways to generate the PHCLS of order n is calculated in Proposition 3. That is n!*(n-1)!. Therefore, the number of n dynamical edge-disjoint paths is $n!*(n-1)^n*((n-1)!)^2*n$.

In the proof of the above proposition, Routing Algorithm 3-(2)&(3) can produce n!*(n-1)! distinct HCLS's. However, the number of non-equivalent HCLS's for the purpose of Routing Algorithm 3 is (n-1)!.

The cost of the hardware involved in implementing Routing Algorithm 3 is very high. The main source of this expense is the fact that each node has to contain n processors in the n-dimensional hypercube network. The total number of processors is $2^n * n$. At this point, a question arises: If each node has only one processor, can we construct a set of disjoint paths? The answer is "Yes". If we modify the order of the travelling times $t_1, t_2, \ldots, t_n$ in order to avoid the possibility that a single node may transmit more than one piece at a time, we can produce a set of dynamical vertex-disjoint paths. Given the sets defined in Routing Algorithm 3-(5), after transmission time unit t_i has passed, the bit position that was traversed at time t_i is deleted from each of the given sets. Then, we compare the new sets that appear as a result of this deletion. If k of the these new sets are the same at time t, this means that k pieces arrive at the same node n' at time t. If m of

these k pieces ($m \leq k$) travel simultaneously from node n^* at time (t+1), then n^* will transmits m pieces along m distinct links at the same time (t+1). In order to avoid this case, we change the order of the travelling times for (m-1) of the m pieces. This ensures that each node transmits only one piece of data at each travelling time.

6.2.2 Two Routing Algorithms of the MRNS Network

The MRNS network is constructed from the mixed radix number system(MRNS). The routing algorithms of the MRNS network are similar to those of the hypercube network. Each algorithm is composed of two phases. The first phase is to transmit the piece to a randomly chosen intermediate node through the secret route. The second phase is to send the piece from this intermediate node to the destination node along the secret path. This section provides the definition of the MRNS, gives a description of the MRNS network, and presents two routing algorithms of the MRNS network.

6.2.2.1 A Mixed Radix Number System(MRNS)

Let N be the total number of nodes of the MRNS network and let N be represented as a product of m_i's, $m_i > 1$ for $0 \leq i \leq n-1$.

$$N = m_{n-1} * m_{n-2} * \cdots * m_1 * m_0$$

Then, each node u between 0 and N-1 can be represented as an n-tuple $(u_{n-1}u_{n-2}\cdots u_1u_0)$ for $0 \le u_i \le (m_i - 1)$.

$$u = \sum_{i=0}^{n-1} u_i * w_i$$

$$w_i = \prod_{j=0}^{i-1} m_j = m_{i-1} * m_{i-2} * \cdots * m_0, \quad w_0 = 1$$

Example 11: Let N = 24.

$$24 = 4 * 3 * 2.$$

$$m_0 = 2, \ m_1 = 3, \ m_2 = 4$$

$$w_0 = 1, \ w_1 = 2, \ w_2 = 6.$$

Then, $u = (u_2u_1u_0)$, $0 \le u_0 \le 1$, $0 \le u_1 \le 2$, $0 \le u_2 \le 3$ for any u in the range 0 - 23. $0_{10} = (000)$, $23_{10} = (321)$ in this mixed number system.

6.2.2.2 Description of the MRNS Network

Each node $u = (u_{n-1}u_{n-2}\cdots u_i\cdots u_1)$ is connected to nodes $(u_{n-1}u_{n-2}\cdots u'_i\cdots u_1)$ for all $1 \le i \le n$, where u'_i can be any integer from $(0,1,..,m_i-1)$ except u_i itself. Given n-dimensions with m_i number of nodes in the i^{th} dimension, the following facts are described.

(1) The total number of links per node is $\ell = \sum_{i=0}^{n-1} (m_i - 1)$.

(2) The total number of links in the MRNS network is N/2 * ℓ, where N is the total number of nodes.

(3) Each dimension is constructed as a complete graph. This means that for the i^{th} dimension, the total number of vertices is m_i and the total number of links is $m_i * (m_i - 1)/2$. Then, the link(p,q) is represented as (i,j), where $j = \sum_{k=0}^{p-1} (m_i - k - 1) + (q - p)$, $p < q$, $p,q \in Z_{m_i}$.

(4) The n-dimensional MRNS is a connected graph of diameter n.

Now, we examine the flexibility of designing the MRNS network. Given N nodes(N ≠ prime number), more than one kind of MRNS network can be designed based on considerations such as the dynamic security, the volume of data to be transmitted, and the cost of the hardware. If the network is more secure and has a large volume of data, then the network can be constructed with more links. However, the cost for constructing the network is a primary consideration, so the network should be designed with as few links as possible.

Example 12: N = 24

Given 24 nodes, four kinds of MRNS networks, NK_1, NK_2, NK_3 and NK_4, can be designed.

$$NK_1 = Z_2 \times Z_{12}$$

$$NK_2 = Z_3 \times Z_8$$

$$NK_3 = Z_4 \times Z_6$$

$$NK_4 = Z_2 \times Z_3 \times Z_4$$

From 6.2.2.2-(2), the total number of links are 144, 108, 96, 72 for NK_1, NK_2, NK_3, NK_2, respectively.

6.2.2.3 Two Routing Algorithms of the MRNS Network

The routing algorithms of the MRNS network are similar to those of the hypercube network. For the MRNS network, the number of channels is determined by the modular number for each dimension, while the modular number of each dimension in the hypercube network is always 2. Considering the structure of the MRNS network, the following two propositions are described.

Proposition 5: Let A and B be any two nodes in the MRNS network and assume that $H(A,B) < n$. Then there are $H(A,B)$ parallel paths of length $H(A,B)$ between the nodes A and B.

Proof : Let $H(A,B) = k$. Then the bit positions that differ between A and B are $\{a_1, a_2, .., a_k\}$. We can write k permutations of this set, indicating the different bit positions for k parallel paths of length k. These k permutations are used to design the special latin square matrix. Using this special latin square matrix, k parallel paths are obtained automatically.

Proposition 6: Let A and B be any two nodes of an n-dimensional MRNS network and assume that $H(A,B) < n$. Then there are ℓ parallel paths between A and B, where $\ell = \sum_{i=0}^{n-1} (m_i - 1)$. The length of each

path is at most H(A,B) + 2.

Proof : In addition to the k parallel paths mentioned in Proposition 5, we consider the other (ℓ - k) different paths. There are two types of paths. The length of the path for the first type is k + 1. The length of the path for the other type is k + 2. For the first type, each of the additional paths starts at some bit position *d* at which A and B differ. The piece is sent initially along an ***incorrect link*** in some dimension *d*; that is, the address of the node where the piece arrives differs from the address of the destination node by exactly the same number of bit positions as the original node does. Then, the piece is sent along the ***correct links*** in all dimensions other than *d*, as in Proposition 6. Finally, we send the piece along the link in dimension *d* that takes the piece to the destination node. This link can be found easily from the fact that each dimension of the MRNS network is constructed as a complete graph.

For the latter type, each of the additional paths starts at a bit position at which A and B do not differ. Then, correct bits a_1 through a_k by choosing one of the k paths from the previous proposition. Finally, change the starting bit back to what it was originally. The additional paths will never cross each other, since these paths select different links at the first and the last steps, and select the links for the remaining steps by Proposition 5. These ℓ paths are disjoint because each set of paths has a

different length.

Given an n-dimensional MRNS network, the following routing algorithm describes how to construct n secret routes. The structure of each dimension is described as a complete graph of given nodes. The n secret paths of the MRNS network are more secure than those of the hypercube network, since the choice of the link at each dimension in the MRNS network is flexible.

Routing Algorithm 4.

cobegin

1) Split P_x into n pieces $P_{x1}, P_{x2}, .., P_{xn}$ by the DAF, P_x is the data at node x.
2) Randomly choose a sequence from the n! permutations of <0,1,2,...,n-1>.
3) Design an (n x n) HCLS, and construct the (n x (n-1)) matrix by randomly choosing (n-1) columns of the HCLS.
4) Randomly choose k_1 for the length of each path, $k_1 \in \{1,2, ...,n-1\}$ and randomly choose k_2 from the set of all possible links, $k_2 \in \{c_1, c_2, .., c_{m-1}\}$, where c_j is the j^{th} link the current node is connected to. Using k_1 and k_2, construct pairwise vertex-disjoint paths $D_1, D_2, .., D_n$ from x to $R_1, R_2, .. R_n$, respectively, each length is at most (n-1).
5) Construct a set of different bit positions of R_i and $\pi(x)$ for the i^{th} piece, $1 \leq i \leq n$.

6) Randomly choose a sequence from the n! permutations of <0,1,2,...,n-1> and design an (n x n) HCLS.

7) From (5) and (6), we design the PHCLS, and then make dynamical edge-disjoint paths $E_1, E_2, .., E_n$ from $R_1, R_2, .. R_n$, respectively, to the destination node $\pi(x)$. Each path E_i has length $\leq n$.

8) Attach the i^{th} routing path (D_i, E_i) to P_{xi}.

coend.

Example 13: Let x = (0000) and $\pi(x)$ = (1100), and let m_i $(0 \leq i \leq 3)$ be 5. The MRNS network is $Z_5 \times Z_5 \times Z_5 \times Z_5$. The total number of nodes $N = 5^4$, and the total number of links ℓ per node is 16. Then, Routing Algorithm 5 is executed as follows:

From (1)

The data at node x is dispersed into n pieces using the DAF.

From (2)

Select a sequence (2 1 3 0).

From (3)

Design the HCLS, and construct the matrix by randomly selecting (n-1) columns of the HCLS.

the HCLS	(n x (n-1)) matrix
2 1 3 0	2 1 3
1 3 0 2	1 3 0
3 0 2 1	3 0 2
0 2 1 3	0 2 1

From (4)

Choose 1,2,3,2 for the lengths of the paths, respectively, and choose {3} for dimension 2 in the first path, {1,4} for dimensions 1 and 3 in the second path, {1,2,3} for dimensions 3, 0 and 2 in the third path, and {1,4} for dimensions 0 and 2 in the fourth path.

D_1 : (0000) → (0300) : (2,3)
D_2 : (0000) → (0010) → (4010) : ((1,1)(3,4))
D_3 : (0000) → (1000) → (1002) → (1302) : ((3,1)(0,2)(2,3))
D_4 : (0000) → (0001) → (0401) : ((0,1)(2,4))

For synchronization, we add ($|D_3|$ - $|D_1|$) s's to the end of $T_{x1}[|D_1|]$.

D_1 : ((2,3) s)
D_2 : ((1,1) (3,4) s)
D_3 : ((3,1) (0,2) (2,3))
D_4 : ((0,1) (2,4) s)

From (5)

E_1 : (0300) -> (1100) : diff. positions = ((3,1),(2,6))
E_2 : (4010) -> (1100) : diff. positions = ((1,1),(2,1),(3,7))
E_3 : (1302) -> (1100) : diff. positions = ((0,2),(2,6))
E_4 : (0401) -> (1100) : diff. positions = ((0,1),(2,6),(3,1))

From (6)

Select a sequence (0 3 1 2)

| 0 3 1 2 |
| 3 1 2 0 |
| 1 2 0 3 |
| 2 0 3 1 |

From (7)

| 0 3 1 2 |
| 3 1 2 0 |
| 1 2 0 3 |
| 2 0 3 1 |

->

| s (3,1) s (2,6) |
| (3,7) (1,1) (2,1) s |
| s (2,6) (0,2) s |
| (2,6) (0,1) (3,1) s |

->

(s)	((s (3,1))	((s (3,1) s))	((s (3,1) s (2,6))
(3,7)	((3,7)(1,1))	((3,7)(1,1)(2,1))	((3,7)(1,1)(2,1) s)
(s)	(s (2,6))	(s (2,6)(0,2))	(s (2,6)(0,2) s)
(2,6)	((2,6)(0,1))	((2,6)(0,1)(3,1))	((2,6)(0,1)(3,1) s)

From (8) & (9)

(P_{s1}	(2,3)	s	s	s	(3,1)	s	(2,6))
(P_{s2}	(1,1)	(3,4)	s	(3,7)	(1,1)	(2,1)	s)
(P_{s3}	(3,1)	(0,2)	(2,3)	s	(2,6)	(0,2)	s)
(P_{s4}	(0,1)	(2,4)	s	(2,6)	(0,1)	(3,1)	s)

The following algorithm describes how to construct ℓ parallel routes for the n-dimensional MRNS network, where $\ell = \sum_{i=0}^{n-1} (m_i - 1)$. Unlike other algorithms, this algorithm uses the HCLS to make partitions, each consisting of secret paths. Paths belonging to the same partition utilize the same dimensions at fixed times for the pieces, if necessary. Since the source node has ℓ channels, the data is dispersed into ℓ pieces and all pieces of the data are transmitted to the neighboring nodes. These ℓ channels are determined by two factors - the dimension that the HCLS assigns to each partition, and the link of each dimension that each partition assigns to the piece. By considering the structure of each dimension in the MRNS network, Proposition 6 and Proposition 7, the second part of the route is determined. To construct disjoint paths in each partition, we use the property of the NSEM. But it is hard to check that each element(which represents a link) in the NSEM is distinct from all other elements. Instead of examining all the elements, we just look at the element in the first last columns. For the first phase of the route, suppose that all the elements in

the first column are distinct. Then, the paths represented by the rows of the matrix, will be disjoint, even if the elements in all columns other than the first are the same. For the second phase of the route, if two elements in the last column are the same, change one of the elements in the first column(in the same row as one of the elements in the last column), and compensate for this by adding an extra step at the end of that path.

Routing Algorithm 5.

cobegin

1) Split P_x into ℓ pieces $P_{x1}, P_{x2}, .., P_{x\ell}$ using the DAF, P_x is the data at node x.
2) Randomly choose a sequence from the n! permutations of $<0,1,2,...,n-1>$.
3) Design an (n x n) HCLS, and construct the (n x (n-1)) matrix by randomly choosing (n-1) columns of the HCLS.
4) Design an (ℓ x (n-1)) matrix by randomly choosing the link from all possible links the current node is connected to.
5) Randomly select k_i from the set (1,2,..,n-1) for the length of the i^{th} path, and make pairwise vertex-disjoint paths $D_1, D_2, .., D_\ell$ from x to $R_1, R_2, .. R_\ell$, $|D_i| = k_i (1 \le i \le n)$, each of length at most n-1.
6) Construct a set of different bit positions of R_i and $\pi(x)$ for the i^{th} piece, $1 \le i \le \ell$.

7) Randomly choose a sequence from the n! permutations of <0,1,2,...,n-1> and design an (n x n) HCLS.

8) From 6) and 7), we design the PHCLS, and then make dynamical edge-disjoint paths $E_1, E_2, .., E_\ell$ from $R_1, R_2, .. R_\ell$, respectively, to the destination node $\pi(x)$. Each path E_i has length $\leq n+1$.

9) Attach the i^{th} routing path (D_i, E_i) to P_{xi}.

coend.

Example 14: Let x = (0000) and $\pi(x)$ = (1100), and let $m_i = 3$, where $(0 \leq i \leq 3)$. The total number of nodes $N = 3^4$, and the total number of links ℓ per node is 8. Then, Routing Algorithm 5 is executed as follows:

From (1)

The data at node x is dispersed into ℓ pieces using the DAF.

From (2)

Select a sequence (2 1 3 0).

From (3)

Design the HCLS, and construct the matrix by randomly selecting (n-1) columns of the HCLS.

the HCLS	(n x (n-1)) matrix
2 1 3 0	2 1 3
1 3 0 2	1 3 0
3 0 2 1	3 0 2
0 2 1 3	0 2 1

From (4)

Construct the (8 x 3) rectangular matrix.

(2,1)	(1,2)	(3,1)
(2,2)	(1,1)	(3,2)
(1,1)	(3,1)	(0,2)
(1,2)	(3,2)	(0,1)
(3,1)	(0,2)	(2,2)
(3,2)	(0,1)	(2,1)
(0,1)	(2,1)	(1,2)
(0,2)	(2,2)	(1,1)

From (5)

Choose 1,2,3,2,3,2,2,1 for the lengths of the paths, respectively.

D_1 : (0000) → (0100) : (2,1)
D_2 : (0000) → (0200) → (0210) : ((2,2) (1,1))
D_3 : (0000) → (0010) → (1010) → (1012) : ((1,1) (3,1) (0,2))
D_4 : (0000) → (0020) → (2020) : ((1,2) (3,2))
D_5 : (0000) → (1000) → (1002) → (1202) : ((3,1) (0,2) (2,2))
D_6 : (0000) → (2000) → (2001) : ((3,2) (0,1))
D_7 : (0000) → (0001) → (0101) : ((0,1) (2,1))
D_8 : (0000) → (0002) : (0,2)

For synchronization, we add ($|D_3|$ - $|D_i|$) s's to the end of $T_{xi}[|D_i|]$, $1 \le i \le 8$.

D_1 : ((2,1) s s)
D_2 : ((2,2) (1,1) s)
D_3 : ((1,1) (3,1) (0,2))
D_4 : ((1,2) (3,2) s)
D_5 : ((3,1) (0,2) (2,2))
D_6 : ((3,2) (0,1) s)
D_7 : ((0,1) (2,1) s)
D_8 : ((0,2) s s)

From (6)

E_1 : (0100) -> (1100) : diff. position = ((3,1))
E_2 : (0210) -> (1100) : diff. positions = ((1,1),(2,3),(3,1))
E_3 : (1012) -> (1100) : diff. positions = ((0,2),(1,2),(2,1))

E_4 : (2020) -> (1100) : diff. positions = ((1,2),(2,1),(3,3))
E_5 : (1202) -> (1100) : diff. positions = ((0,2),(2,3))
E_6 : (2001) -> (1100) : diff. positions = ((0,1),(2,1),(3,3))
E_7 : (0101) -> (1100) : diff. positions = ((0,1),(3,1))
E_8 : (0002) -> (1100) : diff. positions = ((0,2),(2,1),(3,1))

From (7)

Select a sequence (0 3 1 2)

| 0 3 1 2 |
| 3 1 2 0 |
| 1 2 0 3 |
| 2 0 3 1 |

From (8)

Design the (8 x 5) rectangular matrix.

s	(3,1)	s	s	s
(0,1)	(3,1)	(1,1)	(2,3)	(0,1)
s	(1,2)	(2,1)	(0,2)	s
(3,3)	(1,2)	(2,1)	s	s
s	(2,3)	(0,2)	s	s
s	(2,1)	(0,1)	(3,3)	s
s	(0,1)	(3,1)	s	s
(2,2)	(0,2)	(3,1)	s	(2,3)

From (9) & (10)

(P_{x1}, (2,1) s s s (3,1) s s s)
(P_{x2}, (2,2) (1,1) s (0,1) (3,1) (1,1) (2,3) (0,1))
(P_{x3}, (1,1) (3,1) (0,2) s (1,2) (2,1) (0,2) s)
(P_{x4}, (1,2) (3,2) s (3,3) (1,2) (2,1) s s)
(P_{x5}, (3,1) (0,2) s s (2,3) (0,2) s s)
(P_{x6}, (3,2) (0,1) s s (2,1) (0,1) (3,3) s)
(P_{x7}, (0,1) (2,1) s s (0,1) (3,1) s s)
(P_{x8}, (0,2) s s (2,2) (0,2) (3,1) s (2,3))

6.3 The Parallel Algorithms of the Hypercube Network and of the MRNS Network

Let us find n parallel paths in the n-dimensional hypercube. Then, the special latin square of order k is applied to design k parallel paths according to proposition 6 in Chapter 4, where k is the number of bit positions that differ between the source node and the destination node. The remaining (n-k) paths are constructed by Proposition 7 in Chapter 4.

Example 15: Let x = (001100) and $\pi(x)$ = (111101). Then, six parallel paths from (001100) to (111101) are designed as follows.

Step 1.

Locate the bit positions that differ between the two nodes. The different bit positions are 0,4, and 5.

Step 2.

Design the special latin square of order 3 for constructing three parallel paths. The row r_i of the matrix is employed as the path P_i for the piece D_i, where $1 \le i \le 3$.

0	4	5
4	5	0
5	0	4

Step 3.

Design 3 parallel paths by Proposition 7 in Chapter 4. The row r_j of the matrix is employed as the path P_{3*j} for the piece D_{3*j}, where $1 \le j \le 3$.

1	0	4	5	1
2	4	5	0	2
3	0	5	4	3

From the above design, there are 6 parallel paths - each of three paths has length 3 and each of three paths has length 5. The path P_i has the sequence of distinct nodes as follows:

P_1 : x → (001101) → (011101) → (111101)
P_2 : x → (011100) → (111101) → (111101)
P_3 : x → (101100) → (101101) → (111101)
P_4 : x → (001110) → (001111) → (011111) → (111111) → (111101)
P_5 : x → (001000) → (011000) → (111000) → (111001) → (111101)
P_6 : x → (000100) → (000101) → (100101) → (110101) → (111101)

Finally, we construct the parallel paths of the MRNS network. The construction process is similar to that of the hypercube network. Considering the structure of the MRNS network, each of its dimensions is formed as a complete graph. Using Propositions 6 & 7, the following routing algorithm is designed.

Example 16: Let the MRNS network = (Z_2 x Z_3 x Z_3 x Z_4). Let x = (0000) and π(x) = (0223). Then, eight parallel paths are designed as follows:

Step 1.

Locate the bit positions that differ between the two nodes. Then, the different bit positions are 0,1, and 2.

Step 2.

Design the special latin square of order 3 for constructing

three parallel paths. The row r_i of the matrix is employed as the path P_i for the piece D_i, where $1 \leq i \leq 3$.

(0,3)	(1,2)	(2,2)
(1,2)	(2,2)	(0,3)
(2,2)	(0,3)	(1,2)

Step 3.

Design parallel paths using Proposition 5. The row r_j of the matrix is employed as the path P_{3*j} for the piece D_{3*j}, where $1 \leq j \leq 3$.

(3,1)	(0,3)	(1,2)	(2,2)	(3,1)
(0,1)	(1,2)	(2,2)	(0,5)	
(0,2)	(1,2)	(2,2)	(0,6)	
(1,1)	(2,2)	(0,3)	(1,3)	
(2,1)	(0,3)	(1,2)	(2,3)	

From the above design, there are 8 parallel paths - each of three paths has length 3, each of four paths has length 4, and the remaining path has length 5. The path P_i has the sequence of distinct nodes as follows:

P_1 : x → (0003) → (0023) → (0223)
P_2 : x → (0020) → (0220) → (0223)
P_3 : x → (0200) → (0203) → (0223)
P_4 : x → (1000) → (1003) → (1023) → (1223) → (0223)
P_5 : x → (0001) → (0021) → (0221) → (0223)
P_6 : x → (0002) → (0022) → (0222) → (0223)
P_7 : x → (0010) → (0210) → (0213) → (0223)
P_8 : x → (0100) → (0103) → (0123) → (0223)

Conclusion

This thesis presents fault-tolerant parallel communication and secret routing algorithms in the n-dimensional hypercube network and in the MRNS(Mixed Radix Number System) network. Three topics are involved in this thesis - fault-tolerant communication, parallelism, and data security. For the aspect of fault-tolerant communication, we employ a dispersal algorithm that breaks the data into n pieces of data, transmits all the pieces along the channel of the network, and recovers the original data by using any (n-k) pieces of data, where k is the number of missing pieces. We design a dispersal algorithm called the DAF(Dispersal Algorithm using the FFT algorithm), which improves the time and space complexity, compared to previously published dispersal algorithms. For the aspect of parallelism, we construct a set of vertex-disjoint paths in the hypercube network and in the MRNS network employing the special matrices, the modified Hamiltonian circuit matrix(MHCM) and the modified cyclic subsets of order n matrix(MCSnM). These matrices are constructed from disjoint sets, which a set of vertex-disjoint paths are translated into. Our algorithm for constructing n parallel paths in the n-dimensional hypercube network requires only O(n) for the time complexity, while other algorithms, such as Rabin's Routing Algorithm, need more than O(n). For the aspect of data security, this thesis introduces the topic of dynamic

security. Dynamic security is the security of the route taken by the data transmitted over the network. If the route is detected by the adversary, the probability is high that the data will be intercepted. In order to receive the data safely at the destination node, the route must be protected. To accomplish this, we select the intermediate nodes secretly and transmit the data via these intermediate nodes to the destination node. For this routing, the the PHCLS(Partial Hamiltonian Circuit Latin Square) constructed from the HCLS and the special matrix(MHCM or MCSnM) are employed.

This thesis contributes to the following two major areas. First, it introduces dynamic security, and implements this security by designing secret routes. Second, it constructs n parallel paths from the starting node to the destination node in the n-dimensional hypercube network. Later, the maximum number of parallel paths in the corresponding MRNS network is constructed.

Important extentions of this research would involve designing an efficient information dispersal algorithm better than the DAF, constructing secret routes in other network models, and investigating the entropy of secret routings.

Appendix 1

Given a polynomial U of degree $\leq n$, it takes on the respective values $y_0, y_1, .., y_n$ at the n+1 distinct point $x = x_0, x_1, \ldots, x_n$. Using Lagrange's interpolation polynomial of order n, we can write

$$\begin{aligned} U^{[n]}(x) = \quad & y_0 * [\{(x-x_1) * (x-x_2) * \ldots * (x-x_n)\} / \\ & \{\{(x_0-x_1) * (x_0-x_2) * \ldots * (x_0-x_n)\}] \quad + \\ & y_1 * [\{(x-x_0) * (x-x_2) * \ldots * (x-x_n)\} / \\ & \{\{(x_1-x_0) * (x_1-x_2) * \ldots * (x_1-x_n)\}] \quad + \\ & \ldots\ldots \\ & y_n * [\{(x-x_0) * (x-x_1) * \ldots * (x-x_{n-1})\} / \\ & \{\{(x_n-x_0) * (x_n-x_1) * \ldots * (x_n-x_{n-1})\}] \end{aligned}$$

This can be simplified from the following idea:

$$U^{[n]}(x) - U^{[n-1]}(x) = 0 \text{ for } x = x_0, x_1, \ldots x_{n-1}.$$

Thus, $U^{[n]}(x) - U^{[n-1]}(x)$ is a multiple of $(x-x_0) \cdots (x-x_{n-1})$.

$$U^{[n]}(x) = a_n (x-x_0)(x-x_1) \cdots (x-x_{n-1}) + U^{[n-1]}(x),$$

where a_n is a constant.

This simple idea leads to Newton's Interpolation Formula.

$$\begin{aligned} U^{[n]}(x) = \quad & a_n (x-x_0)(x-x_1) \ldots (x-x_{n-1}) + \ldots + \\ & a_2 (x-x_0) * (x-x_1) + a_1 (x-x_0) + a_0, \end{aligned}$$

where a_i and x_i are constants.

This formula is convenient for calculation when we apply Horner's rule

$$U^{[n]}(x) = ((\ldots (a_n (x-x_n) + a_{n-1})(x-x_{n-1}) \ldots)(x-x_0) + a_0)$$

Now we will show Shamir's method. Suppose the data D is transmitted to the destination node. We randomly choose k numbers for the coefficients of a k^{th} degree polynomial equation. Then, we calculate n values for the corresponding points as follows:

$$U(x) = a_0 + a_1*x + a_2*x^2 + \ldots\ldots + a_k*x^k \text{ with } a_0 = D.$$

$$a_0 = 5431,\ a_1 = 7,\ a_2 = 2,\ M = 5441,\ (x \bmod M)$$

$$\begin{aligned}
D(0) &= 5431 \\
D(1) &= 5431 + 7 + 2 = (5440 \bmod M) = 5440 \\
D(2) &= 5431 + 14 + 8 = (5453 \bmod M) = 12 \\
D(3) &= 5431 + 21 + 18 = (5470 \bmod M) = 29 \\
D(4) &= 5431 + 28 + 32 = (5491 \bmod M) = 50 \\
D(5) &= 5431 + 35 + 50 = (5516 \bmod M) = 75
\end{aligned}$$

Each point (i, D(i)) is considered to be the i^{th} piece of data, where $1 \leq i \leq 5$. We send all the pieces to the destination node. The original data can be reconstructed if at least three pieces of data arrive at the destination node. Suppose that D(1),D(3) and D(5) arrive at the destination node. The original data is reconstructed as follows:

$$\begin{array}{lll}
1 & 5440 & \\
 & & ((29-5440+M) \bmod M)/(3-1)=15 \\
3 & 29 & (23-15)/(5-1)=2 \\
 & & ((75-5470+M) \bmod M)/(5-3)=23 \\
5 & 75 &
\end{array}$$

$$U(x) = 2(x-3)(x-1) + 15(x-1) + 5440$$

$$= 2x^2 + 7x + 5431$$

Appendix 2

This algorithm is designed using the Chinese Remainder theorem, which will be described at the end of Appendix 2. Let A = original data, and let M be an (n x m) matrix which is called coprime. Each element of M is taken from the set Z_p, where p is any prime number. The matrix M satisfies the following conditions.

$$\gcd(M[i,j],M[i,k]) = 1 \quad (j \neq k)$$

&

$$\gcd(M[j,i],M[k,i]) = 1 \quad (j \neq k)$$

This algorithm breaks up the original data A into n pieces and then these pieces of data are transmitted to the destination node. The original data A can be reconstructed using any m pieces of data, where m < n.

Example:

$$A = (15,25,35)$$

$$M = \begin{vmatrix} 2 & 3 & 5 \\ 3 & 5 & 2 \\ 5 & 2 & 3 \\ 7 & 11 & 13 \end{vmatrix}$$

Unit vectors for $Z_2 \times Z_3 \times Z_5$ are calcaulated.

$$f^{-1}(1,0,0) = 15$$
$$f^{-1}(0,1,0) = 10$$
$$f^{-1}(0,0,1) = 6$$

Now, unit vectors for $Z_7 \times Z_{11} \times Z_{13}$ are calculated.

$f^{-1}(1,0,0) = 715$
$f^{-1}(0,1,0) = 364$
$f^{-1}(0,0,1) = 924$

The n pieces of data are constructed using modular arithmetic and a unit vector for each row of M.

D(1) = ((15 mod 2)(25 mod 3)(35 mod 5)) = (1,1,0)
D(2) = ((15 mod 3)(25 mod 5)(35 mod 2)) = (1,0,0)
D(3) = ((15 mod 5)(25 mod 2)(35 mod 3)) = (1,2,0)
D(4) = ((15 mod 7)(25 mod 11)(35 mod 13)) = (1,3,9)

D(1) = (1,1,0) = (1,0,0)*1 + (0,1,0)*1 = 15 + 10 = 25
D(2) = (1,0,0) = (1,0,0)*1 = 15
D(3) = (1,2,0) = (1,0,0)*1 + (0,1,0)*2 = 15 + 20 = 35
D(4) = (1,3,9) = (1,0,0)*1 + (0,1,0)*3 + (0,1,1)*9
= 715 + 1092 + 8316 = 10123

All the pieces of data are transmitted to the destination node. Suppose at least 3 pieces arrive. The original data A is reconstructed as follows:

A(1) = ((25 mod 2)(15 mod 3)(35 mod 5)) = (1,0,0) = 15
A(2) = ((25 mod 3)(15 mod 5)(35 mod 2)) = (1,1,0) = 25
A(3) = ((25 mod 5)(15 mod 2)(35 mod 3)) = (1,2,0) = 35

Therefore, A = (15,25,35)

< Chinese Remainder Theorem >

Let N be an integer at least equal to 2. Write N as

$$N = Q_1 * Q_2 * \ldots * Q_k$$

with each Q_i a power of a different prime. Then $(Z_N, +)$ is isomorphic to $(Z_{Q_1} * Z_{Q_2} * \cdots * Z_{Q_k})$ via the map f that takes m in Z_N to its remainder after division by $Q_1, Q_2, .., Q_k$.

Appendix 3

Let M be a Vandermonde matrix $[a_{ij}]$.

$a_{ij} = x^i_j$, $x_1, x_2, \ldots, x_n$ are distinct integers.

$$M = \begin{vmatrix} x^1_1 & x^1_2 & \ldots & x^1_n \\ x^2_1 & x^2_2 & \ldots & x^2_n \\ \ldots & \ldots & \ldots & \ldots \\ x^n_1 & x^n_2 & \ldots & x^n_2 \end{vmatrix}$$

This matrix is inverted as follows:

$$M^{-1} = [b_{ij}]$$

$$b_{ij} = (-1)^{j+1} \sum_{[c1]} (x_{k1}\, x_{k2} \ldots x_{k(n-j)} \;/\; (x_i * \prod_{[c2]} (x_k - x_i))$$

$$[c1] = \begin{cases} 1 \le k_1 \le k_2 \le \ldots \le k_{n-j} \le n \\ k_1, k_2, \ldots, k_{n-j} \neq i \end{cases}$$

$$[c2] = 1 \le k \le n, \quad k \neq i$$

Example:

$$M = \begin{vmatrix} 1 & 2 & 3 \\ 1 & 4 & 9 \\ 1 & 8 & 27 \end{vmatrix}$$

$$M^{-1} = \begin{vmatrix} 3 & -2.5 & 0.5 \\ -1.5 & 2 & -0.5 \\ 1/3 & -0.5 & 1/6 \end{vmatrix}$$

$b_{11} = 2*3/((3-1)*(2-1)*1)$

$b_{12} = -(2+3)/((1*(3-1)*(2-1))$

$b_{13} = 1/((3-1)*(2-1)*1)$

Appendix 4

Let A_1 and A_2 be matrices of orders n and n′, respectively. The Kronecker product of the two matrices denoted by $A_1 \times A_2$ is an nn′ matrix defined by

$$A_1 \times A_2 = \begin{vmatrix} a^1_{11}A_2 & a^1_{12}A_2 & \cdots & a^1_{1n}A_2 \\ a^1_{21}A_2 & a^1_{22}A_2 & \cdots & a^1_{2n}A_2 \\ \cdots & \cdots & \cdots & \cdots \\ a^1_{n1}A_2 & a^1_{n2}A_2 & \cdots & a^1_{nn}A_2 \end{vmatrix},$$

where $a^1_{ij}A_2$ is an n′ x n′ matrix in which the entry in the k^{th} row and the 1^{th} column is $a^1_{ij}a^2_{kl}$.

Appendix 5

This is a routing algorithm of the hypercube network.

Let V = {0,1,2,...,N-1}, N = 2^n, where V is a set of nodes.

cobegin for s ε V

(1) Choose a node randomly using two random number generators

   1) choose one of the elements in a position set {1,2,..,n} and delete that element from that set.

   2) choose an action of {stay, move}. If choosing "move", the data is transmitted along the channel corresponding to the element chosen in 1). Otherwise, stay at the current node.

(2) Construct a set of bit positions at which the intermediate node and the destination node differ.

(3) Choose randomly one of the elements from the set constructed in 2), delete that element from that set, and send the data along the channel corresponding to that element.

coend.

Appendix 6

Definition 1: A latin square is a square matrix with n^2 entries of n different elements, none of them occuring twice within any row or column of the matrix.

Definition 2: A set S is called a quasigroup if there is a binary operation $(\cdot)$ defined in S and if, when any two elements a,b of S are given, the equations $ax = b$ and $ya = b$ each have exactly one solution.

Theorem 1[24]: The multiplication table of a quasigroup is a latin square.

Proof: Let $a_1, a_2, \ldots, a_n$ be the elements of the quasigroup and let its multiplication table be as shown in Fig. A-1, where the entry a_{rs} which occurs in the r^{th} row of the s^{th} column is the product $a_r a_s$ of the elements a_r and a_s. If the same entry occurred twice in the r^{th} row, say in the s^{th} and t^{th} columns so that $a_{rs} = a_{rt} = b$ say, we would have two solutions to the equation $a_r x = b$ in contradiction to the quasigroup axioms. Similarly, if the same entry occurred twice in the s^{th} column, we would have two solutions to the equation $ya_s = c$ for some c. We conclude that each element of the quasigroup occurs exactly once in each row and once in each column, and so unbordered multiplication table(which is a square array of n^2 rows and n^2 columns) is a latin square.

	a_1	a_2	..	a_r	..	a_s	..	a_n
a_1	a_{11}					.		
a_2						.		
.						.		
.						.		
a_r	.	.	.	.	.	a_{rs}		
.								
.								
a_n								

Fig. A-1

Example 1: The multiplication table of the quasigroup, where the operation a * b = 2a + b + 1 (mod 3).

(*)	0	1	2
0	1	2	0
1	0	1	2
2	2	0	1

More generally, the operation a * b = ha + kb + l, where addition is modulo n and h,k and l are fixed integers with h and k prime to n, defines a quasigroup on the set Q = {0,1,...,n-1}

Appendix 7

Theorem 1.2.1[24]: The Cayley table of a finite group G(with its bordering elements deleted) has the following properties:

(1) It is a latin square.

(2) The quadrangle criterion holds, which means that, for any indices i, j, k, ... , it follows from the equations $a_{j,k} = a_{j1,k1}$, $a_{i,k} = a_{i1,k1}$, $a_{i,1} = a_{i1,11}$ that $a_{j,1} = a_{j1,11}$. [[24] p. 18]

Appendix 8

Definition 1: The latin square with elements 1,2,3, .. , n is called row complete if, for any ordered pair of elements α, β ($1 \leq \alpha, \beta \leq n$, $\alpha \neq \beta$), there exists a row of the latin square in which α and β appear as adjacent elements.

Theorem 2.3.1[24]: Let $n = 2m$ be any even positive integer and let an ($n \times n$) latin square L be formed whose first row is 0, 1, 2m-1, 2, 2m-2, 3, ... , m+1, m, where the integers 0 to 2m-1 are regarded as residues modulo n, and whose following rows are formed by the rule that the elements of row k are each one greater than the elements of row k-1. Then L is a row complete latin square.
[[24] p. 82]

References

[1] National Bureau of Standards, "Data Encryption Standard," *Fed. Inf. Process. Stand. Publ. 46*, Jan. 1977

[2] Rivest, R.L., Shamir, A., and Adleman, L., "A Method for obtaining Digital Signatures and Public-Key Cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120-126, Feb. 1978

[3] Shamir, A., "How to Share a Secret," *Commun. ACM*, vol. 22, no. 11, pp. 612-613, Nov. 1979

[4] Knuth, D.E., *The Art of Computer Programming, Vol 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 1983

[5] Asmuth, C.A., and Blakley, G.R., "Pooling, splitting and restituting information to overcome total failure of some channels of communication," in *Proceedings of the 1982 Symposium on Security and Privacy*. IEEE, New York, pp. 156-169, 1982

[6] Laufer, H.B., *Discrete Mathematics and Applied Modern Algebra*. Prindle, Weber & Schmidt, Boston, MA, 1984

[7] Rabin, M.O., "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," *J. ACM*, vol. 36, no. 2, pp. 335-348, Apr. 1989

[8] Knuth, D.E., *The Art of Computer Programming, Vol 1: Fundamental Algorithms*. Addison-Wesley, Reading, MA, 1983

[9] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974

[10] Hwang, K., and Briggs, F.A., *Computer Architecture and Parallel Processing*. McGraw-Hill, New York, 1984

[11] Saad, Y., and Schultz, M.H., "Topological Properties of Hypercubes," *IEEE Trans. Comput.*, vol. 37, no. 7, pp. 867-872, July 1988

[12] Ibarra, O.H., and Sohn, S.T., "On Mapping Systolic Algorithm onto the Hypercube," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 1, pp. 48-63, Jan. 1990

[13] Wu, A.Y., "Embedding of Tree Networks into Hypercubes," *J. Parallel Distrib. Comput.*, vol. 2, pp. 238-249, 1985

[14] Seitz, C.H., "The Cosmic Cube," *Commun. ACM*, vol. 28, no. 1, pp. 22-33, Jan. 1985

[15] NCUBE Corp., "NCUBE/ten: An overview," Beaverton, OR, 1986

[16] Stallings, W., *Data and Computer Communications*. Macmillan, New York, 1985

[17] Johnsson, S.L., and Ho, C.-T., "Optimum Broadcasting and Personalized Communication in Hypercube," *IEEE Trans. Comput.*, vol. 38, no. 9, pp. 1249-1268, Sep. 1989

[18] Valiant, L.G., "A Scheme for Fast Parallel Communication," *SIAM J. Comput.*, vol. 11, no. 2, pp. 350-361, May 1982

[19] Bhuyan, L.N., and Agrawal, D.P., "Generalized Hypercube and Hyperbus Structures for a Computer Network," *IEEE Trans. Comput.*, vol. 33, no. 4, pp. 323-333, Apr. 1984

[20] Hall, M., *Combinatorial Theory*, 2nd Ed. Wiley, New York, 1986

[21] Liu, C.L., *Introduction to Combinatorial Mathematics*. McGraw-Hill, New York, 1968

[22] Welsh, D., *Codes and Cryptography*. Oxford University Press, New York, 1988

[23] Gillett, B.E., *Introduction to Operations Research*. McGraw-Hill, New York, 1976

[24] Denes, J. and Keedwell, A.D., *Latin Squares and Their Applications*. Academic Press, New York, 1974

[25] Stone, H.S., *Discrete Mathematical Structures and Their Applications*. SRA, Chicago, IL, 1973

[26] Preparata, F.P., "Holographic Dispersal and Recovery of Information," *IEEE Trans. on Information Theory*, vol. 35, no. 5, pp. 1123-1124, Sep. 1989

[27] Conway, J.H., and Sloane, N.J.A., *Sphere Packings, Lattices, and Groups*. Springer-Verlag, New York, 1988

[28] Kellison, S.G., *Fundamentals of Numerical Analysis*. Richard D. Irwin, INC, Homewood, IL, 1975

[29] Kuhn, H.W., "The Hungarian Method for the Assignment Problem," *Naval Res. Logist. Quart.*, vol. 2, pp. 83-97, 1955

[30] Lyuu, Y.-D., "Fast Fault-Tolerant Parallel Communication and On-Line Maintenance Using Information Dispersal," in *Proceedings of the 1990 Symposium on Parallel Algorithms and Architectures*. ACM, New York, pp. 378-387, 1990

[31] Papadimitriou, C.H., and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982