Chapter 5

Reliable Broadcasting via Independent Spanning Trees

Parallel computers offer high speed and throughput needed in solving large and complex problems. Parallel computers consist two main components. These are the interconnection network and processing elements. The secret of high performance computing is the interconnection network, so a faulty interconnection network can result to message losses and/or performance degradation of the system. Hence, it is necessary to tolerate faults in interconnection network and also consider reliability aspects to produce a desired effect.

Fault-tolerant broadcasting for parallel and distributed computing is an important issue for many applications in interconnection networks. Independent Spanning Trees (ISTs) provide a number of advantages in data broadcasting in the networks. Using ISTs one can enhance the fault-tolerance, bandwidth, and security. In this chapter, we study the existence and construction of n ISTs rooted at an arbitrary vertex in $H_n (n \geq 1)$. A parallel algorithm with the time complexity $O(n)$ is proposed to construct n ISTs on H_n, where $n \geq 1$.

Section 5.1 recalls some existing important methods for constructing ISTs. Also, some general definitions are given which is used later. In section 5.2, the broadcasting algorithm is described with the form of virtual roots. The correctness of the algorithm is studied in section 5.3. Experimental results are given in section 5.4 and section 5.5 finally summarizes this chapter.

5.1 Overview

Today's computer systems are based on multi-core processor technology. Actually, the performance of such multi-core systems are depend on the interconnection network connecting these cores. According to google search results, the rate of published papers associated to hypercube networks and their variants averages at about 20 papers annually since 1980. The topology of an interconnection networks can be classified as static and dynamic. Static interconnection networks use direct links which are fixed once built and made of point-to-point direct connections. It includes linear array, ring, chordal ring, tree, star, mesh, torus, hypercubes, cube-connected cycles and k-ary n-cube networks. Dynamic interconnection networks are implemented with switched channels and include bus systems, multistage interconnection networks, and crossbar switch networks. Hypercube multi-computer interconnection network is used for connecting distributed memory machine. A number of distributed memory parallel computers utilized hypercubes; e.g., iPSC/2 [87], iPSC/860 [46], nCUBE [43], SGI Origin [84], and Caltech Hypercube [90]. Traditional measurements such as speedup and throughput are central measures for performance evaluation of multi-computer systems. However, developers insist that evaluating the fault tolerance and reliability, is as other performance measures.

Interconnection networks play an important role in parallel and distributed systems. If any node and/or link in the broadcasting tree on multicomputer systems is faulty, then the source node will not transmit the data and hence the broadcasting will not be successful. Fault tolerance reliable data broadcasting can be achieved by sending the message over multiple ISTs to provide other paths for handling some faulty nodes and/or links. Applying IST based broadcasting on interconnection network is a new solution for more reliability, which is an alternative answer to the previous study done by Ramanathan et al. [102].

Finding a high degree of fault tolerance in broadcasting relies on specific interconnection networks and one of the popular networks is the n-dimensional hypercube (H_n). Broadcasting in a network is sending a message from a given vertex to all the other vertices in the network. A fault-tolerant broadcasting protocol can be designed by means of independent spanning trees. If a spanning tree rooted at the source node in a network is viewed as a broadcast channel for data communication, then fault-tolerant

broadcasting can be achieved by sending n copies of a message along n ISTs on the network provided that there are at most $n-1$ faulty nodes/edges in the network.

An interconnection network can be denoted by a graph $G = (V, E)$, where $V(G)$ is the set of processors and $E(G)$ is the set of communication links in the network. For $x, y \in V(G)$, two paths P and Q from x to y are said to be internally disjoint (vertex/edge disjoint), if they have no common vertex and edge except the end vertices (i.e., $E(P) \cap E(Q) = \phi$ and $V(P) \cap V(Q) = x, y$). A tree T in a graph G is called its spanning tree (an acyclic connected graph) if T contains all vertices of G. Spanning tree is very much useful for data broadcasting in distributed system because it reduces the number of links that the data must to cross to reach all the processes.

Two rooted spanning trees, T and T' of a graph G with the same root (r) are said to be independent if for every $x \in V(G)$ the paths from x to the root r in T, and T' are unique. Also, we call a set of rooted spanning trees of G to be independent if they are pairwise independent. The Hamming distance between two nodes in H_n is the number of positions for which the corresponding symbols are different.

The n-dimensional hypercube H_n, suggested first by Sullivan and Bashkow [114], is one of the most popular, versatile and efficient interconnection networks, which possesses many excellent features such as logarithmic number of links per node, logarithmic diameter, high symmetry, recursive structure, linear bisection width and, thus, becomes the first choice for the topological structure of parallel processing and computing systems. As demonstrated in Figure 2.9, an n-dimensional hypercube, H_n, consists of $N = 2^n$ nodes, which are labelled $0, 1, \ldots, 2^{n-1}$; two nodes are adjacent if their labels differ in exactly one bit position. This property highly facilitates the broadcasting of data through the network. In addition, the regular nature of the network provides fault tolerance. The network diameter is defined as the shortest path between most distant nodes and is formulated as $log_2\ N$. Symmetric structure of the network provides the load balancing where every node can become the source of a broadcast as the root of a spanning tree of the network, since the load will always be shared equally in hypercubes.

It is well known that the hypercube network can be considered as a spanning tree in which any vertex could be considered individually as a root node. It is obvious that the difference between parent and child labels in this tree is only in one bit and the maximum number of children of each vertex would be $log_2\ N$. The hypercube topology allows creating an optimal spanning tree in terms of messages sent. It is

always possible to create a spanning tree, starting with any node of the hypercube, so that the maximum number of message transmissions is $N - 1$, and every node receives the message exactly once.

We can see that traditional broadcasting in the hypercube consists of n steps, which can be represented in Figure 5.1 as spanning binomial tree. A 0-level binomial tree B_0 has one vertex. An n-level binomial tree B_n is constructed out of two $(n-1)$-level binomial trees by adding one edge between the roots of the two trees and by making either root the new root. But their data broadcasting pattern has been limited due to components failures.

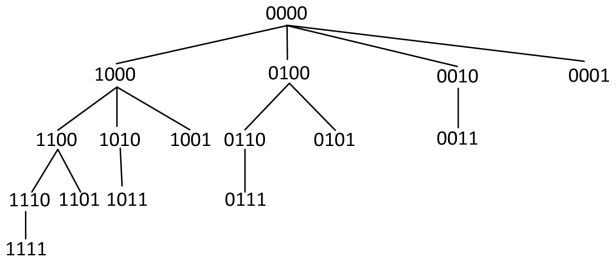

Figure 5.1: The spanning binomial trees with $r = 0$ on H_4.

The study of independent spanning trees has applications in fault-tolerant protocols for distributed computing networks. Itai and Rodeh [65] proved for $n = 2$ that n-connected graph there exist n ISTs rooted at any vertex. For $n = 3$ it was independently proved by Zehavi and Itai [135], and Cheriyan and Maheshwari [29]. In 2006, Curran et al. [35] proved for n-connected graphs with $n = 4$. However, several algorithms are known in some classes of graphs such as product graphs [93], planar graphs [64], De Brujin and Kautz graphs, chordal rings, star graphs, hypercubes, torus [116], folded hypercubes, locally twisted cubes, recursive circulant graphs [131], and Möbius cubes [28], etc.

In 2004, Tang et al. [115] developed an algorithm to construct ISTs on H_n by using recursive feature from H_{n-1}, i.e., it is necessary to construct $n - 1$ trees in advance to obtain the n ISTs. In 2007, Yang et al. [132] presented an algorithm based on Hamming Distance Latin Square (HDLS) distance, which is not recursive like the one presented in [115] and can, therefore, be parallelized.

Marco and Vaccaro [89] considered the problem of broadcasting in the n-dimensional hypercube under the hypothesis that each node can inform in one unit of time all of

its n neighbours and that 1 message transmissions can fail during each time unit. Bao et al. [10] considered the case where all nodes are faultless but links may fail randomly in the n-cube. Bermond et al. [13] describe the neighbourhood broadcasting problem, they consider the node that is broadcasting needs to inform only its neighbours. In our proposed approach, one can broadcast the message at any node to remaining nodes at the same time.

In this chapter, we propose an efficient and optimal parallel algorithm for constructing ISTs on hypercube through which broadcasting is done. For the generation of parent node, our algorithm uses negation and XOR logical operators. Correctness of our parallel algorithm for broadcasting on H_n will be proved and the generated optimal ISTs complexity will be $O(n)$. The algorithm is easily implemented in parallel and distributed computing systems.

5.1.1 Parent Exchange-based n-IST Optimal Construction

In 2004, Tang et al. [115] modified the algorithm given by Obokata et al. [93] in order to reduce the height from 6 to 5 by performing the parent exchange operation. But, the time complexity of both the algorithm was same.

5.1.2 HDLS-based n-IST Parallel Construction

In 2007, Yang et al. [132] proposed the parallel construction of n optimal ISTs ($r = 0$) on hypercubes. In this approach, the HDLS matrix was used to generate all unique paths from all destination nodes (x) to source node(r). They design an algorithm for describing the parent of every node in each spanning tree T_i. Algorithm 4 requires the HDLS preprocessing in which every node x except 0 computes the corresponding HDLS matrix in parallel. For Algorithm 4, the inputs are all nodes of a hypercube and fix root node to 0 and output is to generate a tree rooted at root node 0.

Algorithm 4 construct a tree T_i, rooted at node $r = 0$ and requires $O(n)$ time to be parallelized on H_n, where $0 \leq i \leq n-1$. Let a binary string of the root r be $(r_{n-1} \ldots r_i \ldots r_1 r_0)$ and the binary string of any node $x(\neq 0)$ be $(x_{n-1} \ldots x_i \ldots x_1 x_0)$. If $x_i = 1$, then HDLS is applied for construction of T_i. The HDLS of x is a Latin square matrix whose entries are rotated from the Hamming distance set $H_n(x) = \{i : 0 \leq i \leq n-1 \text{ and } x_i = 1\}$. Suppose a node $x(\neq 0)$ has Hamming distance $t(\leq n)$ to the root node ($r = 0$) and $H_n(x) = \{i_0, i_1, \ldots, i_{t-1}\}$ such that $i_0 < i_1 < \ldots < i_{t-1}$. From the

Algorithm 4 GEN-PARENTS

```
1: for every node x(≠ 0) ∈ V(H_n) with binary string x = x_{n-1} ... x_i ... x_0 do
2:     for every i(0 ≤ i ≤ n − 1) do
3:         if (x_i = 1) then
4:             parent(T_i, x) = x − 2^{succ(i)}
5:         else if (x_i = 0) then
6:             parent(T_i, x) = x + 2^i
7:         end if
8:     end for
9: end for
```

$\text{HDLS}_n(x)$ matrix, if $i = i_p$ is an element in matrix then the successor of i_p is i_{p+1}. Then, the matrix defined below is:

$$HDLS_n(x) = \begin{bmatrix} i_0 & i_1 & \dots & i_{t-2} & i_{t-1} \\ i_1 & i_2 & \dots & i_{t-1} & i_0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ i_{t-2} & i_{t-1} & \dots & i_{t-4} & i_{t-3} \\ i_{t-1} & i_0 & \dots & i_{t-3} & i_{t-2} \end{bmatrix}$$

Example 5.1.1. Consider node $x = 1011$ in H_4. The matrix defined is given by

$$HDLS_4(11) = \begin{bmatrix} 0 & 1 & 3 \\ 1 & 3 & 0 \\ 3 & 0 & 1 \end{bmatrix}$$

Since the successor of i_p is i_{p+1} where $0 \leq i_p \leq n-1$, in $\text{HDLS}_4(11)$ for $x = 1011$ then successor of 0 is 1, the successor of 1 is 3, and the successor of 3 is 0.

Example 5.1.2. If $x = 11$, by applying Algorithm 4 on H_4 provides the parent node of x. For construction of T_0, the $parent(T_0, 11) = 11 - 2^1 = 9$. In T_1, the $parent(T_1, 11) = 11 - 2^3 = 3$. In T_2, the $parent(T_2, 11) = 11 + 2^2 = 15$. In T_3, the $parent(T_3, 11) = 11 - 2^0 = 10$.

In this way, we can construct n ISTs which requires n iterations. For H_4, four ISTs are constructed as depicted in Figure 5.2.

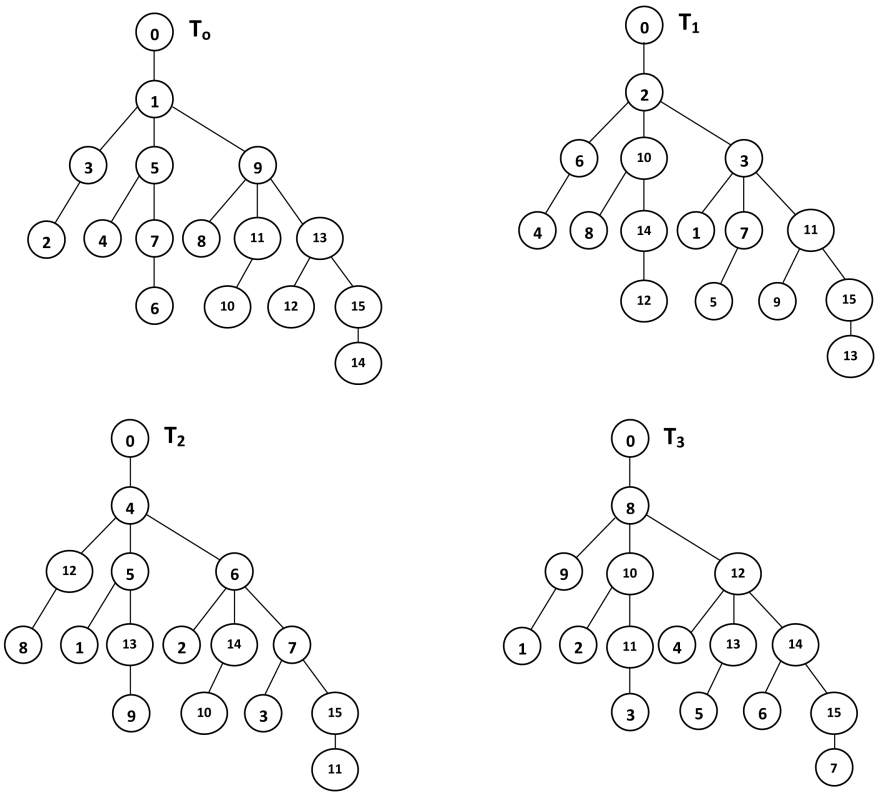

Figure 5.2: Four pairwise independent spanning trees of H_4 rooted at the vertex 0.

5.2 Constructing Independent Spanning Trees on Hypercube

In this section, an efficient parallel algorithm is given which construct optimal n ISTs for reliable broadcasting on the hypercube (H_n). The time complexity of our algorithm is optimal in terms of height and space. Since our algorithm uses $H_{(r,x)}$ set to generate unique paths with time $O(n)$ and space $\leq 2n(N-1)$. we can also broadcast the message from a dynamic root $r = 0, 1, 2, \ldots$, or $2^n - 1$.

Section 5.2.1 defines the virtual roots and construction of n ISTs. Next, section 5.2.2 presents the efficient parallel preprocessing for generating n ISTs and then, section 5.2.3 elaborate the efficient construction of parallel n ISTs with the help of examples.

5.2.1 Virtual Roots for Reliable Broadcasting

The total number of ISTs on H_n are n ISTs $(T_0, T_1, \ldots, T_i, \ldots, T_{n-1})$. The root node $r = r_{n-1} \ldots, r_i \ldots r_1 r_0$ of each T_i has a unique connection $v = r_{n-1} \ldots, \bar{r_i} \ldots r_1 r_0$, which is the one of the n virtual roots of the n ISTs. When $x_i = r_i$, the proposed algorithm generates unique paths from $(N/2) - 1$ leaf nodes for each T_i and when $x_i \neq r_i$, the proposed algorithm generates unique paths from $(N/2)$ non-leaf nodes for each T_i. We can get the leaf node (x) of the longest path by $x = \bar{v}$. Clearly, in each spanning tree the root node has exactly one child by inverting each binary bit in every possible dimension (for example in H_4, if root node is 0000 then inverting bit in 1^{st} dimension gives $0001 : 1$, in 2^{nd} dimension gives $0010 : 2$, in 3^{rd} dimension gives $0100 : 4$, and in 4^{th} dimension gives $1000 : 8$) and so the set of the independent spanning trees can be denoted by $\{T_i, 0 \leq i \leq n-1\}$. The message passing through broadcasting via n-ISTs have the advantages for providing n-degree fault tolerance.

Example 5.2.1. A 4-dimensional hypercube H_n with their four ISTs (T_0, T_1, T_2, T_3) as shown in Figure 5.2. For root node $r = 0 : 0000$, the virtual roots be $v = 1 : 0001, 2 : 0010, 4 : 0100, 8 : 1000$. Let's suppose, if a node 13 and/or a link $(9, 13)$ is faulty during broadcasting of T_0, then nodes $12, 13, 14$ and 15 will not receive the message. However, at the same time the message will be forwarded to those nodes through the T_1, T_2 and T_3.

5.2.2 Parallel Preprocessing for Generating n ISTs

Parallel preprocessing is needed for unique parent-child connection during construction of n ISTs $(T_0, T_1, T_3, \ldots, T_{n-1})$. Our algorithm uses $H_{(r,x)}$ (Hamming distance) to set corresponding parent nodes. The $H_{(r,x)}$ preprocessing is applied for getting unique paths in an arbitrary $r = 0, 1, 2, \ldots, 2^n - 1$ for constructing n ISTs. On each $\{T_i, 0 \leq i \leq n-1\}$, if $x_i = r_i$ or $x_i \neq v_i$ then for leaf nodes with respect to the root r, parent nodes (p) will be evaluated by setting $p_i = \bar{x_i}$. If $x_i \neq r_i$, then for non-leaf nodes the corresponding $H_{(r,x)}$ are applied for evaluating parent-child relation from x to r. The parent p of each node x except root node r is made out by $p_j = \bar{x_j}$, where j is the next position of different bits between x and r.

5.2.3 Algorithm

In this section, we propose a simple algorithm for generating optimal independent spanning trees rooted at any vertex in the graph G. Algorithm 5 uses n iterations for generating n ISTs $(T_i, i = 0, 1, 2, n-1)$ on H_n. For generating T_i in parallel, every node $x = x_{n-1} \dots x_i \dots x_1 x_0$ except $r = r_{n-1} \dots r_i \dots r_1 r_0$ determines its parent $p = p_{n-1} \dots p_i \dots p_1 p_0$. Two functions are required when determining p of x in each T_i and they are:

1. One for $N/2 - 1$ leaf nodes
2. Another for $N/2$ non-leaf nodes

If $x_i = r_i$, then for each leaf nodes (x) their parent (p) has all bits similar to x except at bit position i and if $x_i \neq r_i$, then for each non-leaf nodes x their parent (p) has all similar to x except at bit position j (where $j = succ(i)$).

To generate spanning tree T_i, the following algorithm determines the parent of each vertex $x(= x_{n-1}x_{n-2} \dots x_i \dots x_1 x_0)$ other than the root r. The algorithm depends only on the Hamming distance between x and r, and so it can be easily implemented in parallel or distributed systems. The following algorithm is used to generate the ISTs. It is optimized to execute using logic operators such as negate and XOR, whose task is to compare vertices indexes and avoid edge repetitions. For Algorithm 5, the inputs are all nodes of a hypercube with condition $x \neq r$ and output is to generate a tree rooted at root node.

Algorithm 5 Efficient parallel ISTs

```
1: for every node x(≠ r) ∈ V(H_n) with binary string x = x_{n-1} ... x_i ... x_0 do
2:     for every i(0 ≤ i ≤ n − 1) do
3:         for virtual root invert the bit of x in every possible dimension
4:         if (x_i = r_i) then
5:             parent(T_i, x) = x̄_i
6:         else if (x_i ≠ r_i) then
7:             parent(T_i, x) = x ⊕ 2^{succ(i)}
8:         end if
9:     end for
10: end for
```

In our algorithm, for finding each unique path of T_i, if $x_i = r_i$ then bitwise negation is performed on a leaf node x at position i with parent$(T_i, x) = \bar{x_i}$ and if $x_i \neq r_i$ then each of its corresponding non-leaf nodes to the root r at positions $succ(i)$ with parent$(T_i, x) = x \oplus 2^{succ(i)}$, according to a sequence of locations ($x_i \neq r_i$) with the Hamming distance between r and x at index i.

Example 5.2.2. Let the binary address of r be $r_3r_2r_1r_0 = 0000$ and $x = x_3x_2x_1x_0$. Table 5.1, Table 5.2, Table 5.3, and Table 5.4 shows the computation of all $p =$ parent$(T_i, x) = p_3p_2p_1p_0$ for constructing T_0, T_1, T_2 and T_3 with respect to r.

Table 5.1 is used for constructing T_0 with $i = 0$ and virtual root $v = 0001$. For leaf nodes with condition $x_0 = r_0$, there are seven nodes $x = \{2, 4, 6, 8, 10, 12, 14\}$ and their corresponding parents (by applying $p_0 = \bar{x_0}$) are $p = \{3, 5, 7, 9, 11, 13, 15\}$, connecting along the first dimension ($i = 0$).

For non-leaf nodes with condition $x_0 \neq r_0$, there are eight nodes $x = \{1, 3, 5, 7, 9, 11, 13, 15\}$. In this case, their parents (T_i, x) are identified by using the corresponding $H_{(r,x)}$ tables in order to find position $succ(0)$ (i.e., parent$(T_i, x) = x \oplus 2^{succ(i)}$), defined as follows:

- Along dimension 1, for every x with $succ(0) = 0$, which is $x = \{0001 : 1\}$, then parent$(T_i, x) = 1 \oplus 2^0 = 0000 : 0$.
- Along dimension 2, for every x with $succ(0) = 1$, which are $x = \{0011 : 3, 0111 : 7, 1011 : 11, 1111 : 15\}$, then parent$(T_i, x) = 3 \oplus 2^1 = 0001 : 1, 7 \oplus 2^1 = 0101 : 5, 11 \oplus 2^1 = 1001 : 9, 15 \oplus 2^1 = 1101 : 13$.
- Along dimension 3, for every x with $succ(0) = 2$, which are $x = \{0101 : 5, 1101 : 13\}$, then parent$(T_i, x) = 5 \oplus 2^2 = 0001 : 0, 13 \oplus 2^2 = 1001 : 9$.
- Along dimension 4, for every x with $succ(0) = 3$, which is $x = \{1001 : 9\}$, then parent$(T_i, x) = 9 \oplus 2^3 = 0001 : 1$.

Similarly, one can construct the trees for $T_1(i = 1), T_2(i = 2)$ and $T_3(i = 3)$. Figure 5.2 depicts the construction of four ISTs (T_0, T_1, T_2, T_3) for $H_4(n = 4)$ with root 0.

Example 5.2.3. The construction of T_4 for $H_5(n = 5)$ from Figure 4.5, one of five ISTs $(T_0, T_1, T_2, T_3, T_4)$, rooted at $r = 8 : 01000$. For computing T_4 in parallel, Table 5.5 illustrates $H_{(r,x)}$, $succ(i)$, and all parents (p) with respect to r.

x	$x_3x_2x_1x_0$	$H_{(r,x)}$	$succ(0)$	$2^{succ(0)}$	parent(T_0, x)
0	0000				
1	0001	$\langle 0 \rangle$	0	1	0000 : 0
2	0010	$\langle 1 \rangle$	–	–	0011 : 3
3	0011	$\langle 0, 1 \rangle$	1	2	0001 : 1
4	0100	$\langle 2 \rangle$	–	–	0101 : 5
5	0101	$\langle 0, 2 \rangle$	2	4	0001 : 1
6	0110	$\langle 1, 2 \rangle$	–	–	0111 : 7
7	0111	$\langle 0, 1, 2 \rangle$	1	2	0101 : 5
8	1000	$\langle 3 \rangle$	–	–	1001 : 9
9	1001	$\langle 0, 3 \rangle$	3	8	0001 : 1
10	1010	$\langle 1, 3 \rangle$	–	–	1011 : 11
11	1011	$\langle 0, 1, 3 \rangle$	1	2	1001 : 9
12	1100	$\langle 2, 3 \rangle$	–	–	1101 : 13
13	1101	$\langle 0, 2, 3 \rangle$	2	4	1001 : 9
14	1110	$\langle 1, 2, 3 \rangle$	–	–	1111 : 15
15	1111	$\langle 0, 1, 2, 3 \rangle$	1	2	1101 : 13

Table 5.1: Parent of a node $x \in H_4$ in $T_0(r = 0), i = 0$

Let the binary string of r be $r_4r_3r_2r_1r_0 = 01000$ and x be $x_4x_3x_2x_1x_0 = \{00000, 00001, 00010, \ldots, 10000, \ldots, 11111\}$. For generating T_4, the virtual root of r is $v = 24 : 11000$.

For leaf nodes ($x_4 = r_4$), there are 15 nodes $x = \{0 : 00000, 1 : 00001, 2 : 00010, 3 : 00011, 4 : 00100, 5 : 00101, 6 : 00110, 7 : 00111, 9 : 01001, 10 : 01010, 11 : 01011, 12 : 01100, 13 : 01101, 14 : 01110, 15 : 01111\}$ and their corresponding parent nodes (by applying $p_4 = \bar{x_4}$) are $p = \{16 : 10000, 17 : 10001, 18 : 10010, 19 : 10011, 20 : 10100, 21 : 10101, 22 : 10110, 23 : 10111, 25 : 11001, 26 : 11010, 27 : 11011, 28 : 11100, 29 : 11101, 30 : 11110, 31 : 11111\}$.

For non-leaf nodes $x_4 \neq r_4$, there are 16 nodes $x = \{16 : 10000, 17 : 10001, 18 : 10010, 19 : 10011, 20 : 10100, 21 : 10101, 22 : 10110, 23 : 10111, 24 : 11000, 25 : 11001, 26 : 11010, 27 : 11011, 28 : 11100, 29 : 11101, 30 : 11110, 31 : 11111\}$, of which parents ($p$) are identified by applying the $H_{(r,x)}$ tables in order to find position j and by setting $p_j = \bar{x}_j$, where $j = succ(i), i = 4$. For every x with $succ(4) = 0$, which are $x = \{17 : 10001, 19 : 10011, 21 : 10100, 23 : 10111, 25 : 11001, 27 : 11011, 29 : 11101, 31 : 11111\}$ and their corresponding parent nodes are $p = \{16 : 10000, 18 : 10010, 20 :$

x	$x_3x_2x_1x_0$	$H_{(r,x)}$	$succ(1)$	$2^{succ(1)}$	parent(T_1, x)
0	0000				
1	0001	$\langle 0\rangle$	–	–	0011 : 3
2	0010	$\langle 1\rangle$	1	2	0000 : 0
3	0011	$\langle 1, 0\rangle$	0	1	0010 : 2
4	0100	$\langle 2\rangle$	–	–	0110 : 6
5	0101	$\langle 0, 2\rangle$	–	–	0111 : 7
6	0110	$\langle 1, 2\rangle$	2	4	0010 : 2
7	0111	$\langle 1, 2, 0\rangle$	2	4	0011 : 3
8	1000	$\langle 3\rangle$	–	–	1010 : 10
9	1001	$\langle 0, 3\rangle$	–	–	1011 : 11
10	1010	$\langle 1, 3\rangle$	3	8	0010 : 2
11	1011	$\langle 1, 3, 0\rangle$	3	8	0011 : 3
12	1100	$\langle 2, 3\rangle$	–	–	1110 : 14
13	1101	$\langle 2, 3, 0\rangle$	–	–	1111 : 15
14	1110	$\langle 1, 2, 3\rangle$	2	4	1010 : 10
15	1111	$\langle 1, 2, 3, 0\rangle$	2	4	1011 : 11

Table 5.2: Parent of a node $x \in H_4$ in $T_1(r = 0), i = 1$

$10100, 22 : 10110, 24 : 11000, 26 : 11010, 28 : 11100, 30 : 11110\}$. For every x with $succ(4) = 1$, which are $x = \{18 : 10010, 22 : 10110, 26 : 11010, 30 : 11110\}$ and their corresponding parent nodes are $p = \{16 : 10000, 20 : 10100, 24 : 11000, 28 : 11100\}$. For every x with $succ(4) = 2$, which are $x = \{20 : 10100, 28 : 11100\}$ and their corresponding parent nodes are $p = \{16 : 10000, 24 : 11000\}$. For every x with $succ(4) = 3$, which is $x = \{16 : 10000\}$ and their corresponding parent node is $p = \{24 : 11000\}$. For every x with $succ(4) = 4$, which is $x = \{24 : 11000\}$ and their corresponding parent node is $p = \{8 : 01000\}$.

5.3 Correctness

Lemma 5.3.1. All $T_i(T_0, T_1, \ldots, T_{n-1})$ share the same root node (r) on H_n.

Proof. A root node $r : r_{n-1}r_{n-2} \ldots r_i \ldots r_1r_0$ has n child nodes $v = v_{n-1}v_{n-2} \ldots v_i \ldots v_1v_0$ due to inverting each binary bit in every possible dimension. Since every root node r is connecting with n virtual roots of T_i, where $0 \leq i \leq n - 1$. The

x	$x_3x_2x_1x_0$	$H_{(r,x)}$	$succ(2)$	$2^{succ(2)}$	parent(T_2, x)
0	0000				
1	0001	$\langle 0\rangle$	–	–	0101 : 5
2	0010	$\langle 1\rangle$	–	–	0110 : 6
3	0011	$\langle 0,1\rangle$	–	–	0111 : 7
4	0100	$\langle 2\rangle$	2	4	0000 : 0
5	0101	$\langle 2,0\rangle$	0	1	0100 : 4
6	0110	$\langle 2,1\rangle$	1	2	0100 : 4
7	0111	$\langle 2,0,1\rangle$	0	1	0110 : 6
8	1000	$\langle 3\rangle$	–	–	1100 : 12
9	1001	$\langle 3,0\rangle$	–	–	1101 : 13
10	1010	$\langle 3,1\rangle$	–	–	1110 : 14
11	1011	$\langle 3,0,1\rangle$	–	–	1111 : 15
12	1100	$\langle 2,3\rangle$	3	8	0100 : 4
13	1101	$\langle 2,3,0\rangle$	3	8	0101 : 5
14	1110	$\langle 2,3,1\rangle$	3	8	0110 : 6
15	1111	$\langle 2,3,0,1\rangle$	3	8	0111 : 7

Table 5.3: Parent of a node $x \in H_4$ in $T_2(r = 0), i = 2$

child node (v) of the root (r) in that T_i is obtained by inverting the bit r_i in every dimension (i.e., inverting the bit in 1^{st} dimension $r_{n-1}r_{n-2}\ldots r_i\ldots r_1\bar{r_0}$ gives T_0, inverting the bit in 2^{nd} dimension $r_{n-1}r_{n-2}\ldots r_i\ldots \bar{r_1}r_0$ gives T_1, inverting the bit in i^{th} dimension $r_{n-1}r_{n-2}\ldots \bar{r_i}\ldots r_1r_0$ gives T_i, inverting the bit in $(n-1)^{th}$ dimension $\bar{r_{n-1}}r_{n-2}\ldots r_i\ldots r_1r_0$ gives T_{n-1}) and clearly we say that all T_i share the same root node (r). □

Lemma 5.3.2. Spanning trees $(T_0, T_1, \ldots, T_i, \ldots, T_{n-1})$ generated by algorithm are mutually independent in the hypercube H_n.

Proof. Let T_i and T_j denotes two spanning trees in the hypercube topology. For any vertex $x : x_{n-1}x_{n-2}\ldots x_i\ldots x_j\ldots x_1x_0$, let $P_i(x)$ and $P_j(x)$ be the unique path from root node r to x in T_i and T_j, respectively. We are to show that $P_i(x)$ and $P_j(x)$ are internally disjoint (i.e., they do not have common vertices except for their endpoints). Let $v = v_{n-1}v_{n-2}\ldots v_i\ldots v_1v_0$ be any intermediate node between r and x in $P_i(x)$, and $w = w_{n-1}w_{n-2}\ldots w_j\ldots w_1w_0$ be any intermediate node between r and x in $P_j(x)$. In this way, v is an ancestor of x in T_i, and w is an ancestor of x in T_j. We are to show

x	$x_3x_2x_1x_0$	$H_{(r,x)}$	$succ(3)$	$2^{succ(3)}$	parent(T_3, x)
0	0000				
1	0001	$\langle 0\rangle$	–	–	1001 : 9
2	0010	$\langle 1\rangle$	–	–	1010 : 10
3	0011	$\langle 0, 1\rangle$	–	–	1011 : 11
4	0100	$\langle 2\rangle$	–	–	1100 : 12
5	0101	$\langle 0, 2\rangle$	–	–	1101 : 13
6	0110	$\langle 1, 2\rangle$	–	–	1110 : 14
7	0111	$\langle 0, 1, 2\rangle$	–	–	1111 : 15
8	1000	$\langle 3\rangle$	3	8	0000 : 0
9	1001	$\langle 3, 0\rangle$	0	1	1000 : 8
10	1010	$\langle 3, 1\rangle$	1	2	1000 : 8
11	1011	$\langle 3, 0, 1\rangle$	0	1	1010 : 10
12	1100	$\langle 3, 2\rangle$	2	4	1000 : 8
13	1101	$\langle 3, 0, 2\rangle$	0	1	1100 : 12
14	1110	$\langle 3, 1, 2\rangle$	1	2	1100 : 12
15	1111	$\langle 3, 0, 1, 2\rangle$	0	1	1110 : 14

Table 5.4: Parent of a node $x \in H_4$ in $T_3(r = 0), i = 3$

that $v \neq w$.

Notation $x : x_{n-1}x_{n-2} \dots x_i \dots x_j \dots x_1x_0 \rightarrow v : x_{n-1}x_{n-2} \dots \bar{x_i} \dots x_j \dots x_1x_0$ means x and v are adjacent at location i , and $x : x_{n-1}x_{n-2} \dots x_i \dots x_j \dots x_1x_0 \rightarrow w : x_{n-1}x_{n-2} \dots x_i \dots \bar{x_j} \dots x_1x_0$ means x and w are adjacent at location j. Next, to find all v and w in T_i and T_j with unique edges in parallel, there are two cases:

Case 1: For all leaf nodes x $(x_i = r_i)$ will connect to v and w by setting $v_i = \bar{x_i}$ and $w_j = \bar{x_j}$ in T_i and T_j, respectively. Since every leaf nodes x has unique binary address, so the connected edge to its parent is unique. Hence $v \neq w$.

Case 2: For all non-leaf nodes x $(x_i \neq r_i)$ will connect to v and w by setting $x \oplus 2^{succ(i)}$, where $succ(i)$ evaluated by hamming distance between x and r. Hence, the unique path connecting x to r in T_i is from the non-leaf node $(x : x_{n-1}x_{n-2} \dots x_i \dots x_j \dots x_1x_0)$ to $(y : x_{n-1}x_{n-2} \dots \bar{x_i} \dots x_j \dots x_1x_0) \rightarrow \dots \rightarrow (v : v_{n-1}v_{n-2} \dots v_i \dots v_1v_0) \rightarrow (r : v_{n-1}v_{n-2} \dots \bar{v_i} \dots v_1v_0)$, and the unique path connecting x to r in T_j is from the non-leaf node $(x : x_{n-1}x_{n-2} \dots x_i \dots x_j \dots x_1x_0)$ to $(y : x_{n-1}x_{n-2} \dots x_i \dots \bar{x_j} \dots x_1x_0) \rightarrow \dots \rightarrow (w : w_{n-1}w_{n-2} \dots w_j \dots w_1w_0) \rightarrow (r : w_{n-1}w_{n-2} \dots \bar{w_j} \dots w_1w_0)$. Since every non-leaf node x $(x_i \neq r_i)$ has unique binary address $(x : x_{n-1}x_{n-2} \dots x_i \dots x_1x_0)$ and

x	$x_4x_3x_2x_1x_0$	$H_{(r,x)}$	$succ(4)$	$2^{succ(4)}$	parent(T_4, x)
0	00000	$\langle 3 \rangle$	–	–	10000 : 16
1	00001	$\langle 0, 3 \rangle$	–	–	10001 : 17
2	00010	$\langle 2, 3 \rangle$	–	–	10010 : 18
3	00011	$\langle 0, 1, 3 \rangle$	–	–	10011 : 19
4	00100	$\langle 2, 3 \rangle$	–	–	10100 : 20
5	00101	$\langle 0, 2, 3 \rangle$	–	–	10101 : 21
6	00110	$\langle 1, 2, 3 \rangle$	–	–	10110 : 22
7	00111	$\langle 0, 1, 2, 3 \rangle$	–	–	10111 : 23
8	01000				
9	01001	$\langle 0 \rangle$	–	–	11001 : 25
10	01010	$\langle 1 \rangle$	–	–	11010 : 26
11	01011	$\langle 0, 1 \rangle$	–	–	11011 : 27
12	01100	$\langle 2 \rangle$	–	–	11100 : 28
13	01101	$\langle 0, 2 \rangle$	–	–	11101 : 29
14	01110	$\langle 1, 2 \rangle$	–	–	11110 : 30
15	01111	$\langle 0, 1, 2 \rangle$	–	–	11111 : 31
16	10000	$\langle 4, 3 \rangle$	3	8	11000 : 24
17	10001	$\langle 4, 0, 3 \rangle$	0	1	10000 : 16
18	10010	$\langle 4, 1, 3 \rangle$	1	2	10000 : 16
19	10011	$\langle 4, 0, 1, 3 \rangle$	0	1	10010 : 18
20	10100	$\langle 4, 2, 3 \rangle$	2	4	10000 : 16
21	10101	$\langle 4, 0, 2, 3 \rangle$	0	1	10100 : 20
22	10110	$\langle 4, 1, 2, 3 \rangle$	1	2	10100 : 20
23	10111	$\langle 4, 0, 1, 2, 3 \rangle$	0	1	10110 : 22
24	11000	$\langle 4 \rangle$	4	16	01000 : 8
25	11001	$\langle 4, 0 \rangle$	0	1	11000 : 24
26	11010	$\langle 4, 1 \rangle$	1	2	11000 : 24
27	11011	$\langle 4, 0, 1 \rangle$	0	1	11010 : 26
28	11100	$\langle 4, 2 \rangle$	2	4	11000 : 24
29	11101	$\langle 4, 0, 2 \rangle$	0	1	11100 : 28
30	11110	$\langle 4, 1, 2 \rangle$	1	2	11100 : 28
31	11111	$\langle 4, 0, 1, 2 \rangle$	0	1	11110 : 30

Table 5.5: Parent of a node $x \in H_5$ in $T_4(r = 8), i = 4$

unique $succ(i)$ of the corresponding $h_{(x,r)}$, then the connected edge to its parent is also unique. Hence $v \neq w$.

We shown that there are unique paths connecting from nodes x to r in T_i and T_j. Hence, both case 1 and case 2 proves that T_i and T_j are mutually independent in H_n. □

Lemma 5.3.3. All n ISTs $(T_0, T_1, \ldots, T_i, \ldots, T_{n-1})$ on H_n of any node x utilize unique paths.

Proof. We have already shown that there are unique paths in T_i. From Lemma 3.2, when concatenating case 1 with case 2 will provide the complete unique path from x to r. Therefore, T_i utilize unique paths. □

For solving the problem of many-to-many disjoint paths in the hypercube H_n, the $h_{(x,r)}$ preprocessing is applied for finding unique paths in n ISTs with an arbitrary root $r = 0, 1, 2, \ldots$, or $2^n - 1$. We consider the node 1011 of H_4. Let $P_i(1011, 0000)$ denote the unique path from 1011 to 0000 in P_i for $0 \leq i \leq n-1$. $P_i(1011, 0000)$ for $0 \leq i \leq n-1$ is as follows:

1. $P_0(1011, 0000) : 1011 \rightarrow 1001 \rightarrow 0001 \rightarrow 0000$,
2. $P_1(1011, 0000) : 1011 \rightarrow 0011 \rightarrow 0010 \rightarrow 0000$,
3. $P_2(1011, 0000) : 1011 \rightarrow 1111 \rightarrow 0111 \rightarrow 0110 \rightarrow 0100 \rightarrow 0000$,
4. $P_3(1011, 0000) : 1011 \rightarrow 1010 \rightarrow 1000 \rightarrow 0000$.

This shows that all n ISTs on H_n of the same root r utilize unique paths.

Theorem 5.3.4. Optimal data broadcasting via ISTs is done in $O(n)$ time, where $n = log_2 N$.

Proof. Parallel generation of the $h_{(r,x)}$ tables and corresponding $succ(i)$ for all x with respect to root r requires $N - 1$ processing elements when $(x \neq r)$. $h_{(r,x)}$ require n iterations for the different bits between x and r, so the time complexity is $O(n)$. All $N - 1$ processing elements requires $n - 1$ iterations to compute the corresponding $h_{(r,x)}$ tables in $O(n)$ time. Therefore, data broadcasting is done in $O(n)$ time, where $n = log_2\ N$ via ISTs. □

Broadcasting via ISTs on H_n can construct an optimal broadcast tree. To solve the single node broadcast problem, it is sufficient to transmit the data along the optimal broadcast tree produced by algorithm. It requires $2^n - 1$ link transmissions. This is optimal, since each of the other $2^n - 1$ nodes must receive the information from the source node. The number of time steps equals the diameter of H_n, which is also optimal and the single node broadcast using the optimal broadcast tree on H_n requires n transmission steps.

5.4 Experimental Results

By simulation results, we have analysed the performance of our efficient parallel algorithm to generate the n ISTs on H_n. The evaluation of the algorithm is based on response time and reliable broadcasting. We have evaluated the response time by counting the computation steps, and evaluated the reliable broadcasting by calculating the percentage of fault tolerance. At last, a number of simulation results have been taken on H_n in various system sizes ($N = 16$, 32, 64, 128, and 256).

5.4.1 Response Time of the Parallel Construction of n ISTs

In simulation, the response time of the parallel construction of n ISTs using N PEs was studies in terms of the number of computation steps. The response time mainly contains the following three functions:

1. Computation of $H_{(r,x)}$ in $t_1 \leq n$ steps (suppose required t_1 time)
2. Computation of $succ(n)$ in $t_2 \leq n$ steps (suppose required t_2 time)
3. Construction of n ISTs in $t_3 = n$ steps (suppose required t_3 time)

Table 5.6 depict the parallel response time (say t_R), where $t_R = t_1 + t_2 + t_3 = 3n$ steps, executed by all PEs (i.e., $x = 0, 1, 2, \ldots, N-1$ except root r). In experiments, our response time of ISTs have been improved in comparison of the HDLS-ISTs on H_n for all system sizes ($N = 2^n$). Our efficient algorithm taking less computing steps because the computation time is dependent on $H_{(r,x)}$ table and $succ(n)$ table.

Figure 5.3, showed that our IST linearly requires increasing time when system sizes increases, whereas the HDLS-IST need more time.

Response time: HDLS-based ISTs vs Our algorithm based ISTs		
N	HDLS-ISTs	Our ISTs
16	40	15
32	61	19
64	87	23
128	118	27
256	154	31

Table 5.6: Comparison in terms of no. of computation steps on H_8

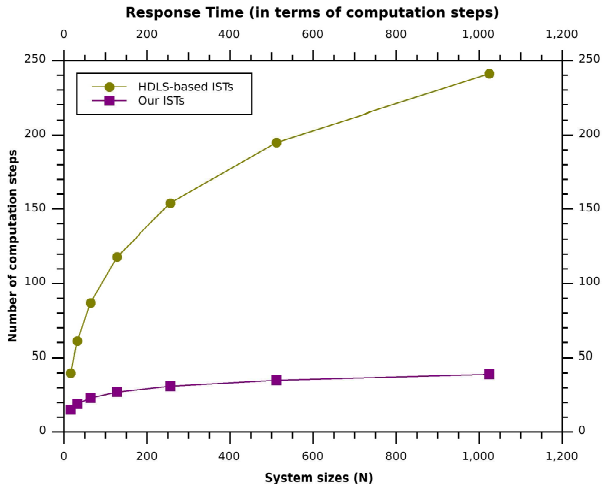

Figure 5.3: Response time of the parallel construction of n ISTs

5.4.2 Reliable Broadcasting

We have measured fault tolerance for successful broadcasting on a number of dynamic link faults. During n ISTs broadcasting on H_4, there were several possibilities of link faults. For example in Figure 5.2, if link $(1, 9)$ in T_0 was faulty, then nodes $8, 9, 10, 11, 12, 13, 14, 15$ would not received the messages. If links $(2, 6), (10, 8), (3, 7), (11, 9)$ in T_1 were faulty, then nodes $4, 5, 6, 7, 8, 9$ would not received the messages. If links $(12, 8), (5, 1), (13, 9)(6, 2), (7, 3)$ in T_2 were faulty, then nodes $1, 2, 3, 8, 9$ would not received the messages, and if links $(9, 1), (10, 2), (12, 4), (14, 6)$ in T_3 were faulty, then nodes $1, 2, 4, 6$ would not received the messages. So, we can say that the broadcasting was successful because all sixteen nodes $(0 \leq x \leq 15)$ received the onwards message (i.e., nodes $1, 2, 3, 4, 5, 6, 7$ in T_0, nodes $1, 2, 3, 10, 11, 12, 13, 14, 15$ in T_1, nodes

$4, 5, 6, 7, 10, 11, 12, 13, 14, 15$ in T_2, and nodes $3, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15$ in T_3).

For different sizes of a system for evaluation ($N = 16, 32, 64, 128, 256$), Table 5.7 depicts dynamic link faults for reliable broadcasting along n ISTs. The common spanning tree for successful broadcasting cannot allow any faulty link.

N	No. of links in n ISTs	Faulty links	Fault tolerance (%)
16	60	14	23.3
32	155	28	18.0
64	378	59	15.6
128	889	124	14.0
256	2040	238	11.7

Table 5.7: Reliable broadcasting in faulty links

In simulation, we have taken randomly faulty links $(1, 2, 3, 4, \ldots)$ in all possible ISTs. In performance perspective, we have checked the successful/unsuccessful broadcasting by setting random link faults. Broadcasting was verified. If all nodes receives the message then it is called successful broadcasting and if all nodes does not receives the message then it is called unsuccessful broadcasting. We repeated the process until all nodes receives the message.

For different sizes of a system, Figure 5.4 represented the ratio between faulty links over total links. This case conceded fault tolerance $12\% - 23\%$ in reliable broadcasting. As we can see from the Table 5.7 that for smaller N the fault tolerance was nearly 23%, which is more than that (12%) for the larger N. For the hypercube topology, we found successful broadcasting in a system under 14 faulty links over 60 links (or 23.3%). For the large $N(N = 256, n = 8)$, the successful broadcasting was done under 238 faulty links over 2040 links (or 11.7%). So, for every node x in every IST $(T_0, T_1, T_2, \ldots, T_{n-1})$, there exists n-degree fault tolerance. As we can see in Figure 5.2, for example the node is $x = 9$. For this, there are four links to reach at node 9, which is the link $(1, 9)$ in T_0, the link $(11, 9)$ in T_1, the link $(13, 9)$ in T_2, and the link $(8, 9)$ in T_3.

5.5 Conclusion

In this chapter the basic idea of hypercubes and their contribution to the improvement of spanning tree protocols in broadcasting was discussed. This chapter proposed a

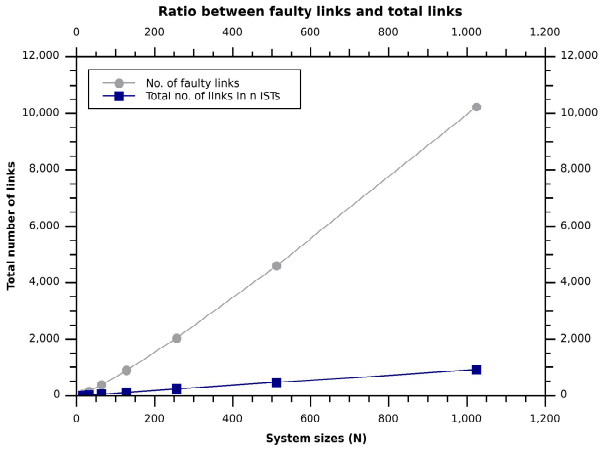

Figure 5.4: Ratio b/w faulty links vs total links in reliable broadcasting.

fault-tolerant broadcasting protocol by means of multiple independent spanning trees (ISTs) in a hypercube network H_n. Fault tolerance can be achieved by sending n copies of the message along n independent spanning trees rooted at the same root r. The proposed algorithm is applied to solve any node broadcast problem for hypercubes. All independent spanning trees constructed by the algorithm presented in this chapter broadcast the data in $O(n)$ time. Since $N = 2^n$ which can rewrite in this way $n = log_2 N$. Thus we can say that this algorithm runs in Logarithmic Time.

With the help of Table 5.8, one can analysed that our proposed algorithm is fast and more reliable. In our approach, one can broadcast the message at any node to remaining nodes at the same time.

Authors	Year	Approach	Advantages
S. M. Tang	2004	Sequential	Fast computing
J. S. Yang	2007	Parallel HDLS matrix	More reliable
J. Werapun	2012	Hamming distance	More reliable
Our	2014	Parallel	Fast and reliable for broadcasting

Table 5.8: Comparison in terms of approach and advantages