ON PERFORMANCE EVALUATION OF X-TORUS TOPOLOGY

A DISSERTATION
Submitted in partial fulfillment of the
Requirements for the award of the degree of
BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE ENGINEERING

By
Nikhita Mishra (071204)
Neha Gupta (071211)
Swati Chawla (071290)

Department of Computer Science Engineering and Information Technology,
Jaypee University Of Information Technology,
Waknaghat,
Solan-173215,
Himachal Pradesh, INDIA

May 2011

Table of contents

CERTIFICATE

We, hereby certify that the work, which is being presented in the thesis, entitled "**ON PERFORMANCE EVALUATION OF X-TORUS TOPOLOGY**", is partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY In COMPUTER SCIENCE ENGINEERING and submitted in Computer Science and Engineering Department of Jaypee University Of Information Technology, Waknaghat, Solan is an authentic record of our own work, carried out under the supervision of Dr. Nitin. The matter presented in this thesis has not been submitted by us for the award of degree of any other degree of this or any other university.

[illegible]

(NIKHITA MISHRA)

neha

(NEHA GUPTA)

[illegible]

(SWATI CHAWLA)

This is to certify that the above statement made by the candidates is correct and true to the best of my knowledge.

Nitin

Supervisor:
Dr. Nitin
Professor
Department of Computer Science Engineering and Information Technology,
Jaypee University Of Information Technology,
Waknaghat,
Solan-173215,
Himachal Pradesh, INDIA

ACKNOWLEDGEMENT

We are very grateful to our guide, Dr. NITIN, whose guidance and insight have made working on this project a very enjoyable and fulfilling experience. His phenomenal breadth of knowledge and ability to discern key points on the project, and continuous encouragement continue to bolster our motivation and light our enthusiasm to further carry on the project with greater vigour and interest.

We would also like to thank Brig. Balbir Singh, Dr. Satish Chandra and Mr. Ravi Rastogi, for helping us in times of need. Without their co-operation, the project would have been a distant dream.

We would like to thank all the staff members of Computer Science Engineering Department of JUIT, Waknaghat for providing us all the facilities required for the completion of this thesis.

Summary

Interconnection networks enable fast data communication between components of a digital system. Today, interconnection networks are used in a variety of applications such as switch and router fabrics, processor-memory interconnect, I/O interconnect, and on-chip networks, to name a few.

The design of an interconnection network has three aspects—the topology, the routing algorithm used, and the flow control mechanism employed. The topology is chosen to exploit the characteristics of the available packaging technology to meet the requirements (bandwidth, latency, scalability, etc.) of the application, at a minimum cost. Once the topology of the network is fixed, so are the bounds on its performance. For instance, the topology determines the maximum throughput (in bits/s) and zero-load latency (in hops) of the network. The routing algorithm and flow control must then strive to achieve these performance bounds.

The function of a routing algorithm is to select a path to route a packet from its source to its destination. In this thesis, we demonstrate the significance of the routing algorithm used in the network towards achieving the performance bounds set by the topology. Central to this thesis, is the idea of load-balancing the network channels. A naive routing algorithm that does not distribute load evenly over all channels, stands to suffer from sub-optimal worst-case performance. However, unnecessary load-balancing is overkill. Spreading traffic over all channels when there is no uneven distribution of traffic, leads to sub-optimal best-case and average-case performance. This thesis explores routing algorithms that strive to achieve high worst-case efficiency without sacrificing performance in the average or best-case.

While performance metrics such as average latency and worst-case throughput are key parameters in evaluating a network, there are several other important measures such as amount of packet reordering, statistical guarantees on delay and network buffer occupancy, to name a few. In the last part of this thesis, we propose a method to analyze the performance of a class of load-balanced networks over these performance metrics.

List of figures

List of tables

CHAPTER 1

INTRODUCTION

1.1 A brief introduction to interconnection networks

An interconnection network is a programmable system that enables fast data communication between components of a digital system. The network is programmable in the sense that it enables different connections at different points in time. The network is a system because it is composed of many components: buffers, channels, switches, and controls that work together to deliver data.

Eight terminal nodes are connected to the network with bidirectional channels. When a source Terminal wants to communicate with a destination terminal, it sends data in the form of a *message* into the network and the network delivers the message. Using the same resources, the network can deliver the above message in one cycle, and a different message in the next cycle.

The interconnection network may be realized in several ways. One approach is to provision the system such that there is a possible point-to-point connection between every pair of terminals.

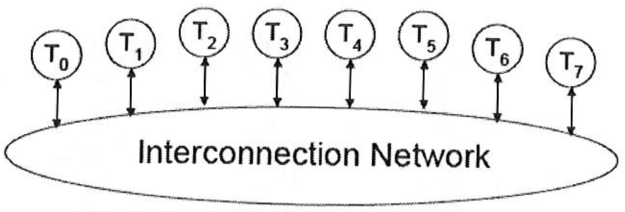


Figure 1.1: The functional view of an interconnection network. Terminals T_0 through T_7 are connected to the network with bidirectional channels.

As illustrated in Figure 1.2, one way of implementing such an arrangement is by connecting the terminals to a crossbar switch. At every cycle, the crossbar connects each source terminal with one distinct destination terminal.

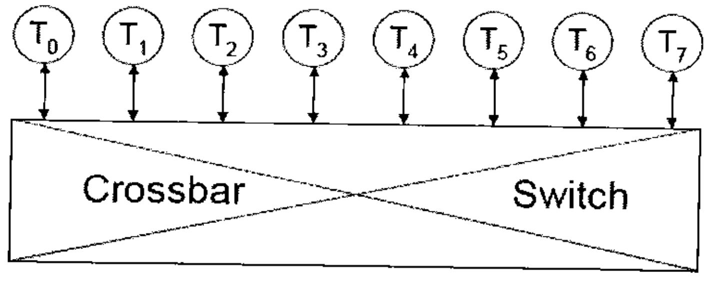


Figure 1.2: Realizing the 8-node network using a crossbar switch

An alternative way to implement the network may be to connect each terminal to a *router* node, and connect the router nodes in a ring using bidirectional channels (Figure 1.3). While this implementation connects the terminals with much less wiring than may be required in the crossbar realization, the drawback is that all terminals no longer have point-to-point connections. For this reason, this implementation is called a *multi-hop* network.

Irrespective of the way the interconnection network is realized, the network itself plays a vital role in determining the performance of the system as a whole. We next examine some digital systems wherein we might encounter interconnection networks and explain why the network's performance is a key component of the performance of the system as a whole.

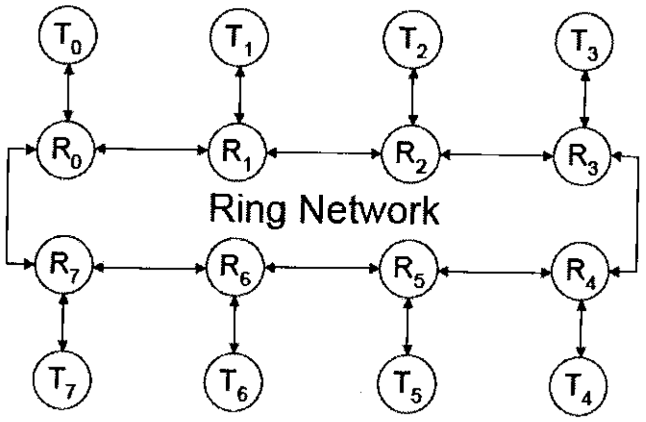


Figure 1.3: Connecting the 8 nodes in a ring

1.2 The importance of interconnection networks

Today, interconnection networks are used in almost all digital systems that have two or more components to connect. In a computer system, the "terminal nodes" from the previous section could be processors and memories, or I/O devices and controllers, communicating with each other. They could also be input and output ports in the case of communication switches and network routers. Interconnection networks may also connect sensors and actuators to processors in control systems, host and disk nodes in I/O networks and on-chip cores in chip multiprocessors.

The performance of most digital systems today is limited by their communication or interconnection, not by their logic or memory. Hence, it is imperative that the underlying interconnection network perform efficiently to improve the efficacy of the entire system. For instance, in a computer system, the interconnection network between processor and memory determines key performance factors such as the memory latency and memory bandwidth. The performance of the interconnection network in a communication switch largely determines the capacity (data rate and number of ports) of the switch.

The performance of an interconnection network can be measured using a rich set of metrics. The most common metrics are throughput and latency. Other important metrics include reliability, graceful degradation in the presence of faults, in-order delivery of data packets, and delay guarantees in communicating data. To meet the performance specifications of a particular application, the *topology*, *routing*, and *flow control* of the network must be implemented. The topology of the network refers to the arrangement of the shared set of nodes and channels. Once a topology has been chosen, the routing algorithm determines a path (sequence of nodes and channels) a message takes through the network to reach its destination.

Finally, flow control manages the allocation of resources to packets as they progress along their route. The topology sets limits on the performance of the network while the routing and flow-control strive to realize these performance limits.
In this thesis, we focus on the routing of packets through the network. In other words, given the topology, and assuming ideal flow control, we attempt to route packets through the network to give high performance on several metrics.

1.3 The need for load-balanced routing

The routing method employed by a network determines the path taken by a packet from a source terminal node to a destination terminal node. Networks with high *path diversity* offer many alternative paths between a source and destination. *Oblivious* routing algorithms choose between these paths based solely on the identity of the source and destination of the message while *adaptive* algorithms may base their decision on the state of the network. A good routing algorithm makes its route selection in a manner that exploits locality to provide low latency and high throughput on *benign* traffic. Many applications also require the interconnection network to provide high throughput on *adversarial* traffic patterns. In an Internet router, for example, there is no backpressure on input channels. Hence, the interconnection network used for the router fabric must handle any traffic pattern, even the worst-case, at the line rate, or else packets will be dropped. To meet their specifications, I/O networks must provide guaranteed throughput on all traffic patterns between host and disk nodes.

A routing algorithm must strike a balance between the conflicting goals of exploiting locality on benign traffic while load-balancing on adversarial traffic. To achieve high performance on benign traffic, *Minimal Routing* (MR)—that chooses a shortest path for each packet — is favored. MR, however, performs poorly on worst-case traffic due to load imbalance. With MR, an adversarial traffic pattern can load some links very heavily while leaving others idle. To improve performance under worst-case traffic, a routing algorithm must balance load by sending some fraction of traffic over non-minimal paths hence, destroying some of the locality.

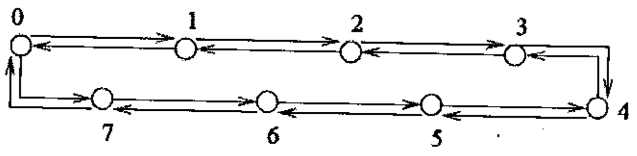


Figure 1.4: The 8 node ring with unidirectional channels

Through the most part of this thesis, we shall propose routing algorithms that strive to achieve good worst-case performance without sacrificing the locality inherent in benign traffic.

1.4 Problem Specifications

Since the dawn of networking, the need to make stable, fast, fault-tolerant systems has always been there. Researchers and scientists have been working on various issues to make a congestion-free traffic flow.

Its seen that one way for processors to communicate data is to use a shared memory and shared variables. However, this is unrealistic for large number of processors. A more realistic assumption is that each processor has its own private memory and data communication takes place using message passing via an interconnection network. To deal with the problem of congestion on Torus, X-Torus topologies, we first tested and simulated a set of oblivious and adaptive routing algorithms, and later developed two new adaptive algorithms for each topology, which shows better performance during high traffic flow, or congestion.

CHAPTER 2

SURVEY

The interconnection network plays a central role in determining the overall performance of a multicomputer system. If the network cannot provide adequate performance, for a particular application, nodes will frequently be forced to wait for data to arrive.

Some of the more important networks include :

- Fully connected, or all-to-all
- Mesh
- Rings
- Hypercube

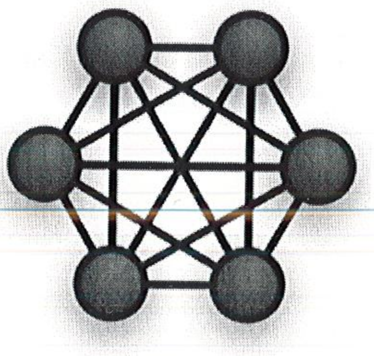
2.1 Networks

2.1.1 Fully connected or all-to-all

A fully connected network is a communication network in which each of the nodes is connected to each other. A fully connected network doesn't need to use switching nor broadcasting. However, its major disadvantage is that the number of connections grows quadratically with the number of nodes, per formula

$$c = \frac{n^2 - n}{2}.$$

and so is extremely impractical for large networks.



6 Nodes

15 Links

Figure 2.1

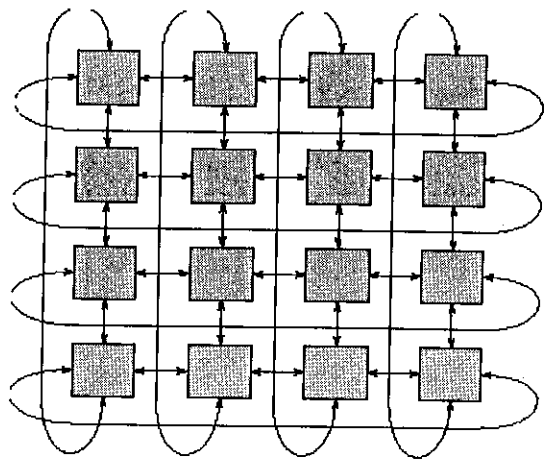
Each node has N-1 connections (N-1 nearest neighbours) giving a total of N(N-1) / 2 connections for the network.

2.1.2 Mesh (Torus)

Mesh networking (topology) is a type of networking where each node must not only capture and disseminate its own data, but also serve as a *relay* for other sensor nodes, that is, it must collaborate to propagate the data in the network.

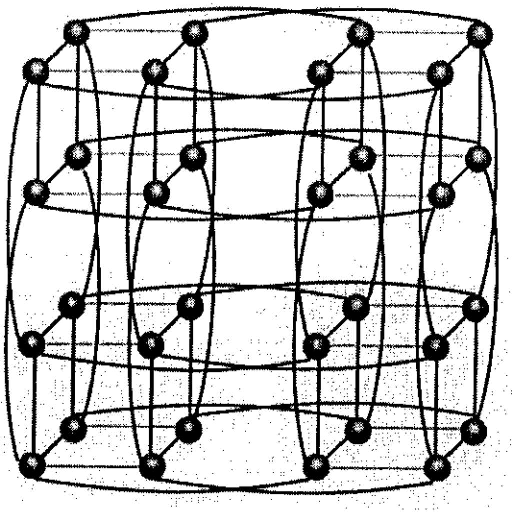
A mesh network can be designed using a *flooding* technique or a *routing* technique. When using a routing technique, the message propagates along a path, by *hopping* from node to node until the destination is reached. For insuring all its paths' availability, a routing network must allow for continuous connections and reconfiguration around broken or blocked paths, using *self-healing* algorithms.

a)



A 2-D Torus network

b)


K=3 w=4

With wraparound connections

c)

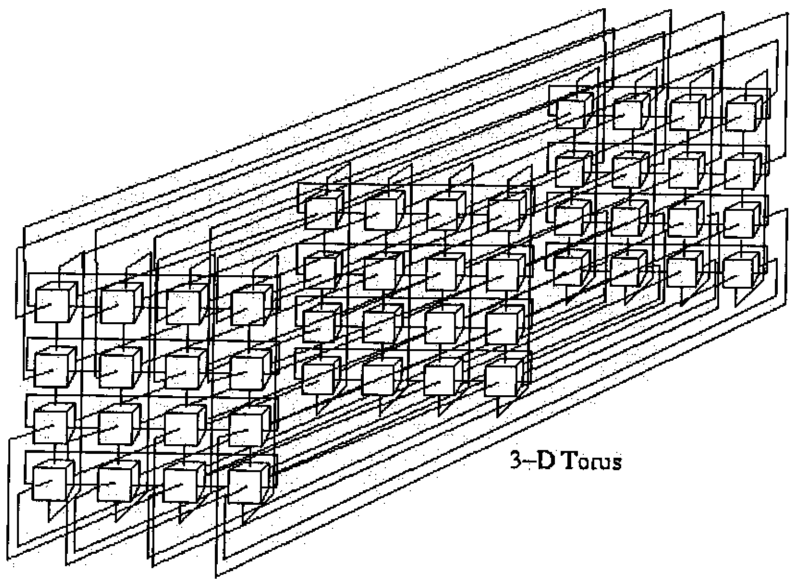


Figure 2.2 (a), (b), (c)

2.1.3 Rings

A ring network is a network topology in which each node connects to exactly two other nodes, forming a single continuous pathway for signals through each node - a ring. Data travels from node to node, with each node along the way handling every packet.

Because a ring topology provides only one pathway between any two nodes, ring networks may be disrupted by the failure of a single link. A node failure or cable break might isolate every node attached to the ring.

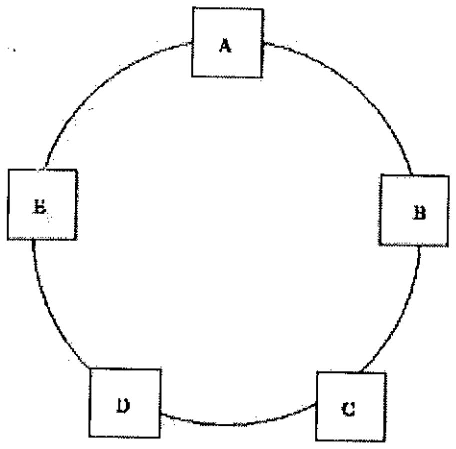
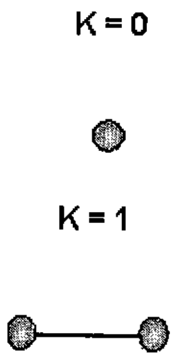


Figure 2.3

2.1.4 Hypercube Connection (Binary n-Cube)

Hypercube networks consist of N=2^k nodes arranged in a k dimensional hypercube.



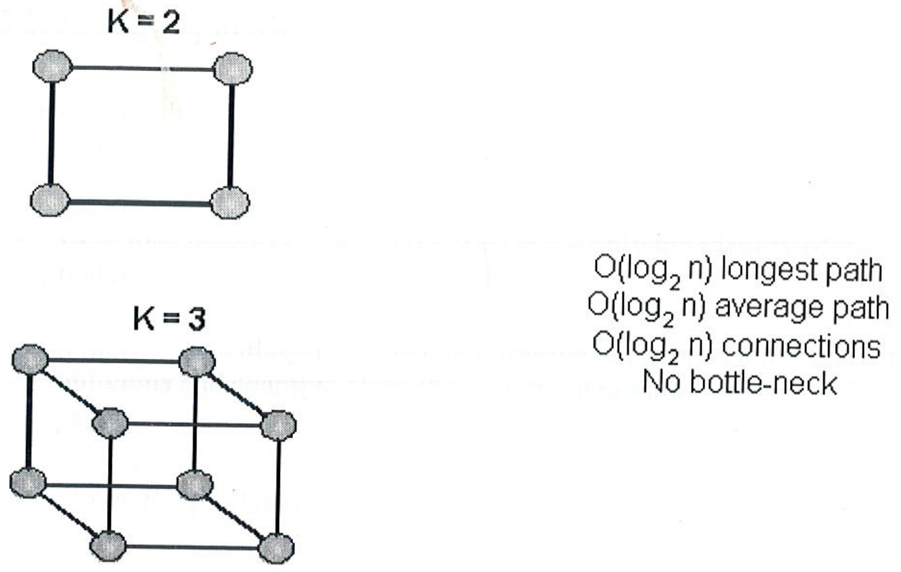

K = 2
K = 3
O($\log_2$ n) longest path
O($\log_2$ n) average path
O($\log_2$ n) connections
No bottle-neck

Figure 2.4

K dimensional hypercube is formed by combining 2K-1 dimensional hypercubes, and connecting the corresponding nodes. Hence, we see that hypercubes are recursive.

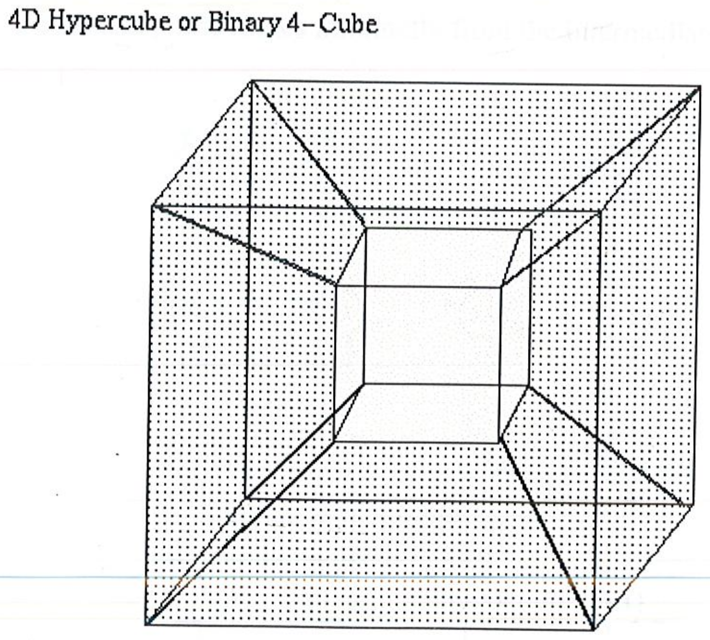


Figure 2.5

2.2 Routing Algorithms

A routing algorithm maps a source-destination pair to a path through the network from the source to the destination.

- *Oblivious* algorithms select the path using only the identity of the source and destination nodes.
- *Adaptive* algorithms may also base routing decisions on the state of the network. Both oblivious and adaptive algorithms may use randomization to select among alternative paths.
- *Minimal* algorithms route all packets along some shortest path from source to destination
- *Non-minimal* ones may route packets along longer paths.

2.2.1 Valiant's Randomized Routing

Valiant's randomized algorithm (VAL) is a two-phase randomized routing algorithm.

- In the first phase, packets are routed from the source to a randomly chosen intermediate node using minimal routing.
- The second phase routes minimally from the intermediate node to the destination.

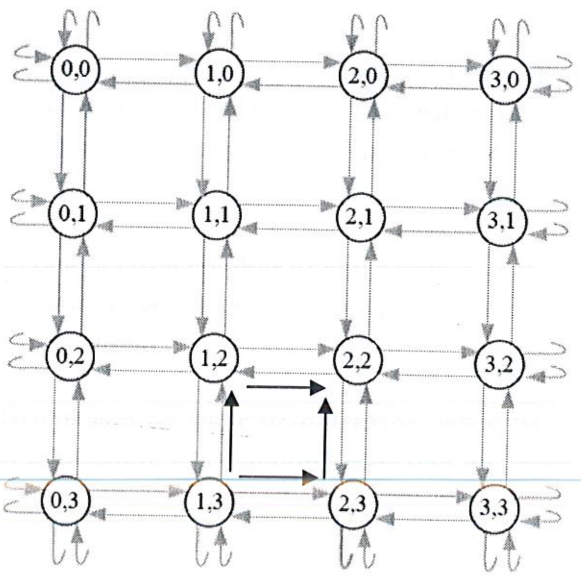


figure 2.6

2.2.2 Random Ordered Multiphase Minimal Routing

In VALIANT random algorithm, we found the intermediate node anywhere inside the k-ary 2-cube network. In this routing, we select an intermediate node inside the minimal quadrant only.

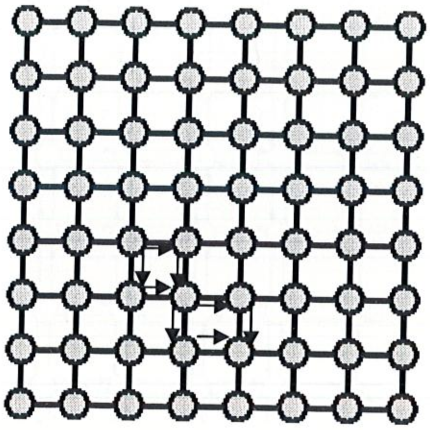


Figure 2.7

2.2.3 Randomized Local Balance (RLB)

In multiple dimensions RLB works, as in the one dimensional case, by balancing load across multiple paths while favoring shorter paths. Unlike the one dimensional case, however, where there are just two possible paths for each packet — one short and one long, there are many possible paths for a packet in a multi-dimensional network. RLB exploits this path diversity to balance load.

To extend RLB to multiple dimensions, we start by independently choosing a direction for each dimension just as we did for the one-dimensional case above. Choosing the directions selects the *quadrant* in which a packet will be routed in a manner that balances load among the quadrants.

To distribute traffic over a large number of paths within each quadrant, we route first from the source node, to a randomly selected intermediate node, within the selected quadrant, and then from source to the destination.

For each of these two phases we route in *dimension order*, traversing along one dimension before starting on the next dimension, but randomly selecting the order in which the dimensions are traversed.

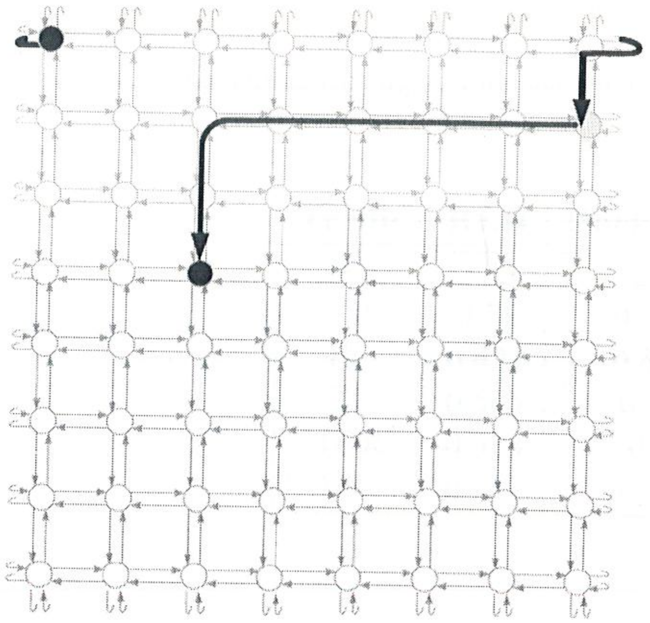


Figure 2.8- Routing using RLB

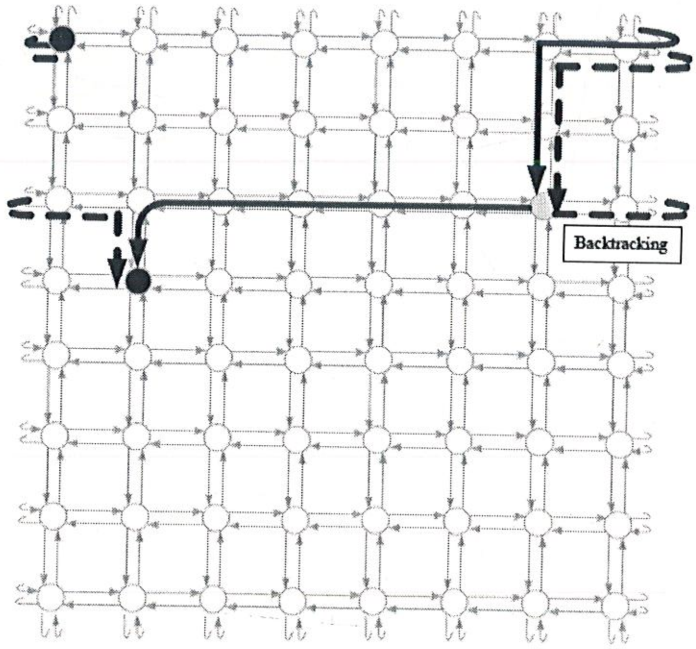


Figure 2.9- Routing using RLB with backtracking

A comparison of the routing technique, with and without backtracking is illustrated in Table 2.1.

Table 1

Traffic	RLB	Backtrack
NN	2.33	2.9
UR	0.76	0.846
BC	0.421	0.421
TP	0.565	0.50
TOR	0.533	0.4
WC	0.313	0.27

CHAPTER 3

TECHNICAL COMPOSITION

The following are the tools and packages used in the completion of the project.

3.1 NS-2 Simulator

3.1.1 Background

NS simulator is based on two languages: an object-oriented simulator, written in C++; and a OTcl (an object- oriented extension of Tcl) interpreter, used to execute user's command scripts.

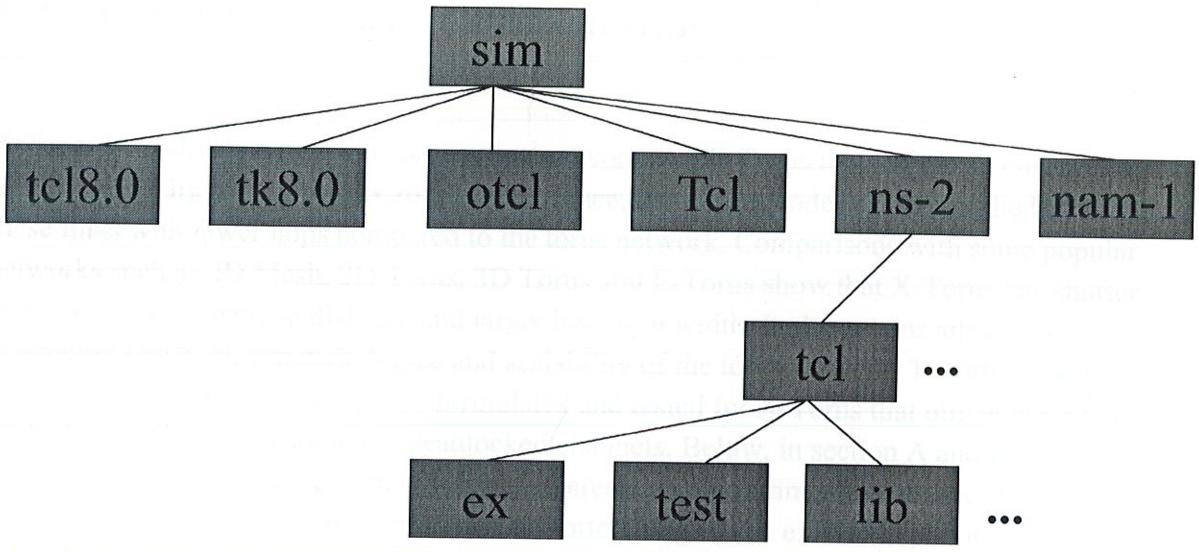
NS has a rich library of network and protocol objects. There are two class hierarchies: the compiled C++ hierarchy, and the interpreted OTcl one, with one-to-one correspondence between them.

The compiled C++ hierarchy allows us to achieve efficiency in the simulation and faster execution times. This is, in particular, useful for the detailed definition and operation of protocols. This allows one to reduce packets and event processing time.

Then is the OTcl script, provided by the user, we can define a particular network topology, the specific protocols, and applications that we wish to simulate and forms the output that we wish to obtain from the simulator. The OTcl can make use of the objects compiled in C++ trough, and OTcl linkage, that creates a matching of OTcl object for each of the C++ objects.

NS is a discrete event simulator, where the advance of time depends on the timing of events, which are maintained by the scheduler. An event is an object in the C++ hierarchy, with a unique id, a schedule time and the pointer to an object that handles the event. The scheduler keeps an ordered data structured with the events to be executed, and fires them one by one, invoking the handler of the event.

3.1.2 NS-2:Directory Structure



Architecture view of NS-Figure 3.1

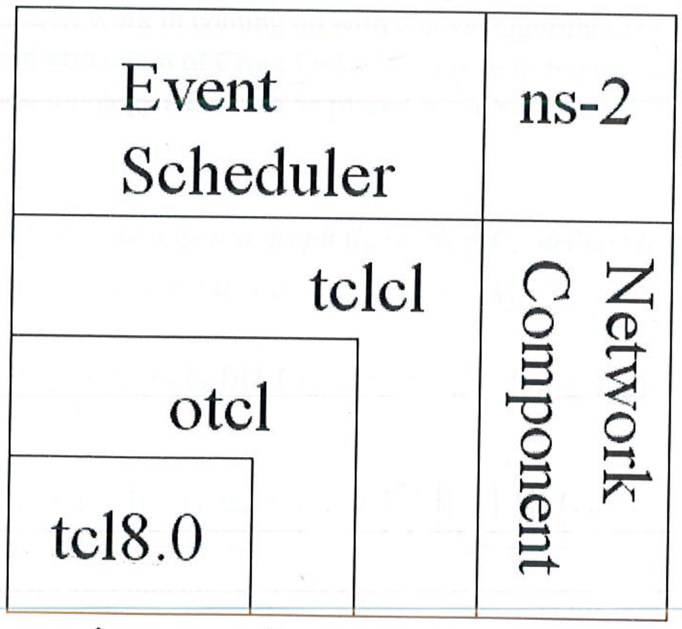


Figure 3.2

CHAPTER 4

X-Torus Topology and its Generalization

X-Torus is relatively a new interconnection network. An X-Torus topology is an enhancement of torus network by adding some Cross Links. Hence, the distant nodes can be reached by using these links with fewer hops compared to the torus network. Comparisons with some popular networks such as 2D Mesh, 2D Torus. 3D Torus and E-Torus show that X-Torus has shorter diameter, shorter average distance and larger bisection width. It also retains advantages such as symmetric structure, constant degree and scalability of the torus network. Therefore, new algorithms and formulae need to be formulated and tested for X-Torus that utilize these Cross Links in avoiding congestion and deadlocked channels. Below, in section A and B, formulae 1-4 and 8-11 respectively, have been developed to create an algorithm which utilizes Cross Links efficiently to overcome congestion issues, a shortcoming which existed in previous Shortest path algorithm.

The addition of Cross Links makes it a potentially efficient topology since it is a modification of Torus by addition of Cross Links, which in many ways improves the overall performance. Hence we dedicate our research work in coming up with a novel algorithm for X-Torus that balances the load by optimal utilization of Cross Links. As shown in Figure (4.2), X-Torus topology is a two-dimension topology and it can be placed in an X-Y frame where each node is labeled as (a, b).

Definition 4.1 A $k_x x\ k_y$ X-Torus topology is a graph $G_x = N_x\ x\ C_x$, defined in [2] *as follows:*

$$N_x = \{(a, b) |\ 0 \leq a \leq k_x, 0 \leq b \leq k_y\}$$

$$C_x = \left\{ \langle (u_a, u_b), (v_a, v_b) \rangle \middle| \left((u_a = v_a \cap u_a = [v_b \pm 1]_n) \right.\right.$$

$$\left.\left. \cup \left(u_a = \left[v_a + \left\lfloor \frac{k_x}{2} \right\rfloor \right]_{k_x} \cap u_b = \left[v_b + \left\lfloor \frac{k_y}{2} \right\rfloor \right]_{k_y} \right) \right) \cap \left((u_a, u_b), (v_a, v_b) \epsilon\ N_x \right) \right\}$$

Where $k_x \geq 2, k_y \geq 2$ and (u_a, u_b) and (v_a, v_b) are the coordinates of the nodes u and v respectively. In this paper, we only consider the case of $K_x = K_y = k$.

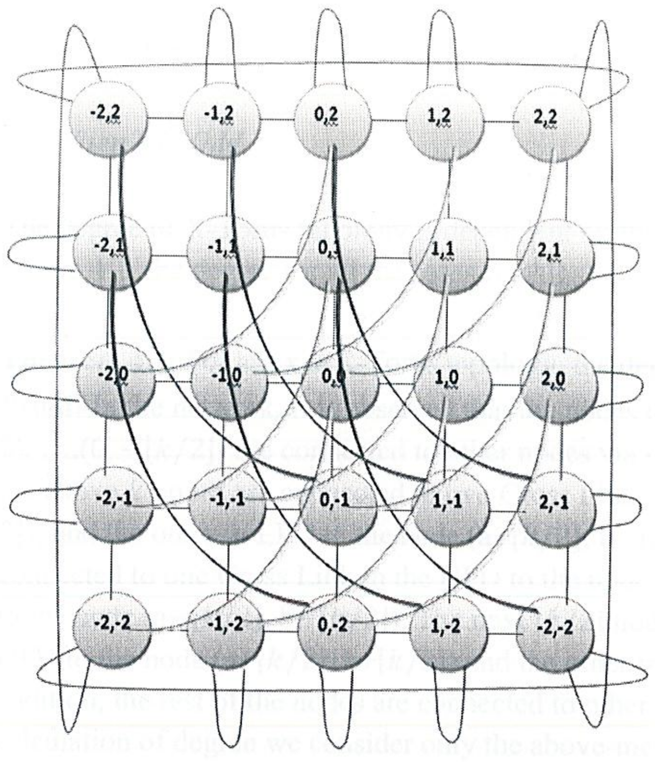


Fig. 4.1. Illustration of Odd X-Torus Topology (5x5).

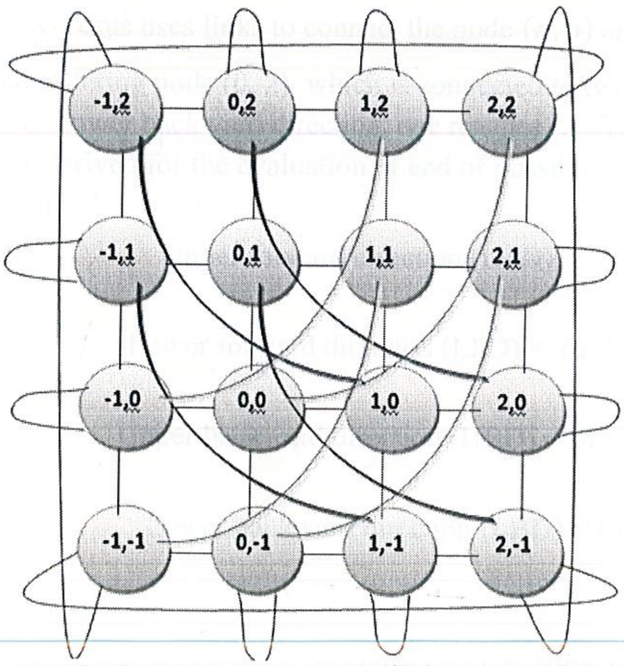


Fig. 4.2. Illustration of Even X-Torus Topology (4x4).

A. *When k is Odd*

The degree of X-Torus topology is dependent on the parity of k. If k is Odd, such as shown in Figure 4.1, the degree for all the cases remains 6.

Theorem 4.1: For a k x k X-Torus topology, the degree is 6 (where k is odd).
Proof: In the network, it is observed that the nodes (0, $\lfloor k/2 \rfloor$), (0, $\lfloor k/2 \rfloor - 1$), (0, $\lfloor k/2 \rfloor - 2$),...0, 0), ...,(0, $-\lfloor k/2 \rfloor$) are connected to other nodes via two Cross Links. The first $\lfloor k/2 \rfloor$ nodes out of the above k nodes are connected to two Cross Links; one in LBD to the node (a$-\lfloor k/2 \rfloor$, b$-\lfloor k/2 \rfloor$); and the other in LFD to the node (a+$\lfloor k/2 \rfloor$, b$-\lfloor k/2 \rfloor - 1$). The central node (0, 0) is connected to one Cross Link in the UFD to the node (a+$\lfloor k/2 \rfloor$, b+$\lfloor k/2 \rfloor$) and the second in LBD to the node (a$-\lfloor k/2 \rfloor$, b$-\lfloor k/2 \rfloor$). The next $\lfloor k/2 \rfloor$ nodes are connected to two Cross Links; one in UFD to the node (a+$\lfloor k/2 \rfloor$, b+$\lfloor k/2 \rfloor$) and the other in UBD to node (a$-\lfloor k/2 \rfloor$, b+$\lfloor k/2 \rfloor$+1). In addition, the rest of the nodes are connected to other nodes via one Cross Link. Thus for the calculation of degree we consider only the above-mentioned nodes. Moreover, every node has four neighbors, so the degree will be 4 + 2 = 6.

X-Torus uses links to connect the node (a, b) and the node $\left(\left[a + \left\lfloor\frac{k}{2}\right\rfloor_k, b + \left\lfloor\frac{k}{2}\right\rfloor_k\right]\right)$. For e.g. considering node (0, 2), which is connected to two Cross Links, using Lower forward direction and Lower backward direction, one reaches (2, -1) and (-2, 0) respectively. The formulae (1-4) are derived for the evaluation of end of possible Cross Links, where source is (a, b) and k is Odd:

$$\text{Upper forward direction (UFD)} = (a + (\lfloor k/2 \rfloor), b + (\lfloor k/2 \rfloor)) \tag{1}$$

$$\text{Lower forward direction (LFD)} = (a + (\lfloor k/2 \rfloor), b - (\lfloor k/2 \rfloor) - 1) \tag{2}$$

$$\text{Upper backward direction (UBD)} = (a - (\lfloor k/2 \rfloor), b + (\lfloor k/2 \rfloor) + 1) \tag{3}$$

$$\text{Lower backward direction (LBD)} = (a - (\lfloor k/2 \rfloor), b - (\lfloor k/2 \rfloor)) \tag{4}$$

Using the above set of formulae (1-4) X-Torus topologies can be drawn for higher dimensions of k, where k is odd. From the symmetry observed in the pattern of Cross Links we obtain that there are ($\lfloor k/2 \rfloor$+1).($\lfloor k/2 \rfloor$+1) Cross Links in the UFD and ($\lfloor k/2 \rfloor$+1).($\lfloor k/2 \rfloor$) Cross Links in the UBD. Now since the Cross Links between any two nodes are bidirectional, the UFD Cross Link is same as the LBD Cross Link. Similarly, the UBD Cross Link is same as the LFD Cross Link.

Let $M_d(k)$ denote the number of Cross Links in a $k\ x\ k$ X-Torus topology in the direction 'd', where d can be UFD, LFD, UBD or LBD.

$$M_d(k)\begin{cases}(\lfloor k/2\rfloor).(\lfloor k/2\rfloor) \mid d \in \{UFD, LBD\} \\ \\ (\lfloor k/2\rfloor).(\lfloor k/2\rfloor) \mid d \in \{UBD, LFD\}\end{cases} \tag{5}$$

Hence the general formula for the total number of Cross Links in a $k\ x\ k$ X-Torus topology (odd):

$$M(k) = (\lfloor k/2\rfloor+1).(\lfloor k/2\rfloor+1) + (\lfloor k/2\rfloor+1).(\lfloor k/2\rfloor) \tag{6}$$

Since each Cross Link is connected to two nodes, $M(k)$ is multiplied by 2; and from the result k (the number of nodes that are connected to two Cross Links) is subtracted. From (6), Total nodes in the network, $N= 2*[(\lfloor k/2\rfloor+1).(\lfloor k/2\rfloor+1)+(\lfloor k/2\rfloor+1).(\lfloor k/2\rfloor)]-k$ and It is observed that $N \approx k$. This implies that there is no node in the network, which is not connected to a Cross Link, hence the above formulas (1-4) are general for all odd X-Torus topologies.

B. When k is Even

If k is Even, shown in Figure (4.2), the degree for all the cases remains 5.

Theorem 4.2: For a k x k X-Torus topology, the degree is 5 (k is even).
Proof: In this network, the Cross Link which connects the node (a, b) and the node ($[a+\lfloor k/2\rfloor]_k$, $[b+\lfloor k/2\rfloor]_k$) and the Cross Link which connects the node ($[a+\lfloor k/2\rfloor]_k$, $[b+\lfloor k/2\rfloor]_k$) and ($[[a+\lfloor k/2\rfloor]_k+\lfloor k/2\rfloor]_k]$, $[[b+\lfloor k/2\rfloor]_k+\lfloor k/2\rfloor]_k]$) are the same. Thus every node is connected only to a single Cross Link. Also every node has four neighbors, so the degree will be 4 + 1 = 5.
The formulas (8-11) are derived for the evaluation of end of possible Cross Links, where source is (a, b) and when k is Even:

$$\text{Upper forward direction (UFD)} = a + (\lfloor k/2\rfloor), b + (\lfloor k/2\rfloor)) \tag{8}$$
$$\text{Lower forward direction (LFD)} = a + (\lfloor k/2\rfloor), b - (\lfloor k/2\rfloor)) \tag{9}$$
$$\text{Upper backward direction (UBD)} = a - (\lfloor k/2\rfloor), b + (\lfloor k/2\rfloor)) \tag{10}$$
$$\text{Lower backward direction (LBD)} = a - (\lfloor k/2\rfloor), b - (\lfloor k/2\rfloor)) \tag{11}$$

Using the above set of formulae (8-11) X-Torus topologies can be drawn for higher dimensions of k, where k is even. From the symmetry observed in the pattern of Cross Links we obtain that there are $(\lfloor k/2 \rfloor).(\lfloor k/2 \rfloor)$ Cross Links in the UFD and $(\lfloor k/2 \rfloor).(\lfloor k/2 \rfloor)$ Cross Links in the UBD.

$$M_d(k) \begin{cases} (\lfloor k/2 \rfloor).(\lfloor k/2 \rfloor) \mid d \in \{UFD, LBD\} \\ \\ (\lfloor k/2 \rfloor).(\lfloor k/2 \rfloor) \mid d \in \{UBD, LFD\} \end{cases} \tag{12}$$

Hence the general formula for the total number of Cross Links in a k x k X-Torus topology (even):

$$M(k) = (\lfloor k/2 \rfloor).(\lfloor k/2 \rfloor) + (\lfloor k/2 \rfloor).(\lfloor k/2 \rfloor) \tag{13}$$

Since each Cross Link is connected to two nodes, $M(k)$ is multiplied by 2. From (13), Total nodes in the network, $N = 2*[(\lfloor k/2 \rfloor).(\lfloor k/2 \rfloor) + (\lfloor k/2 \rfloor).(\lfloor k/2 \rfloor)]$ and It is observed that $N = k^2$. This implies that there is no node in the network, which is not connected to a Cross Link, hence the above formulae (8-11) are general for all even X-Torus topologies.

CHAPTER 5
MAPPED ILLUSTRATION OF THE DISTANCE NODES FROM THE SOURCE

A topology is evaluated in terms of a number of parameters. The following diagram is an improvement to existing illustration [2] of the distance of nodes from the source, as not only it shows placement of nodes through hop metric, but through co-ordinate co-relation of nodes, it adds to intuitiveness. Therefore, this mapping illustration will help us to understand the symmetry of X-Torus topology and to calculate the number of hops between source and destination directly.

A. When k is Odd

In Figure (4), we use nodes of different colors to show their respective distances from the source node (0,0). Like all the nodes that are two hops away from the source (as represented in red color) helps us in calculating the number of hops between two nodes and calculate their diameter, as in this case the green-grass nodes are the farthest are $(k/2) + 1$ hops away from the source, therefore, $(k/2) + 1$ becomes the diameter of this network. Since the distance in network is measured by number of hops traversed, this illustration helps in evaluating distance of various nodes from source and hence in the development of algorithm, where minimal paths are replaced by alternate paths during congestion.

Comparing with Figure (2), we can see that number of nodes, one hop distance from source are 6; 4 through vertical/horizontal paths, namely (0, 1), (0,-1), (1, 0), (-1, 0) while 2 through Cross Links, namely (2,2) and (-2,-2), the nodes are represented in Figure (4) as purple colored circles.

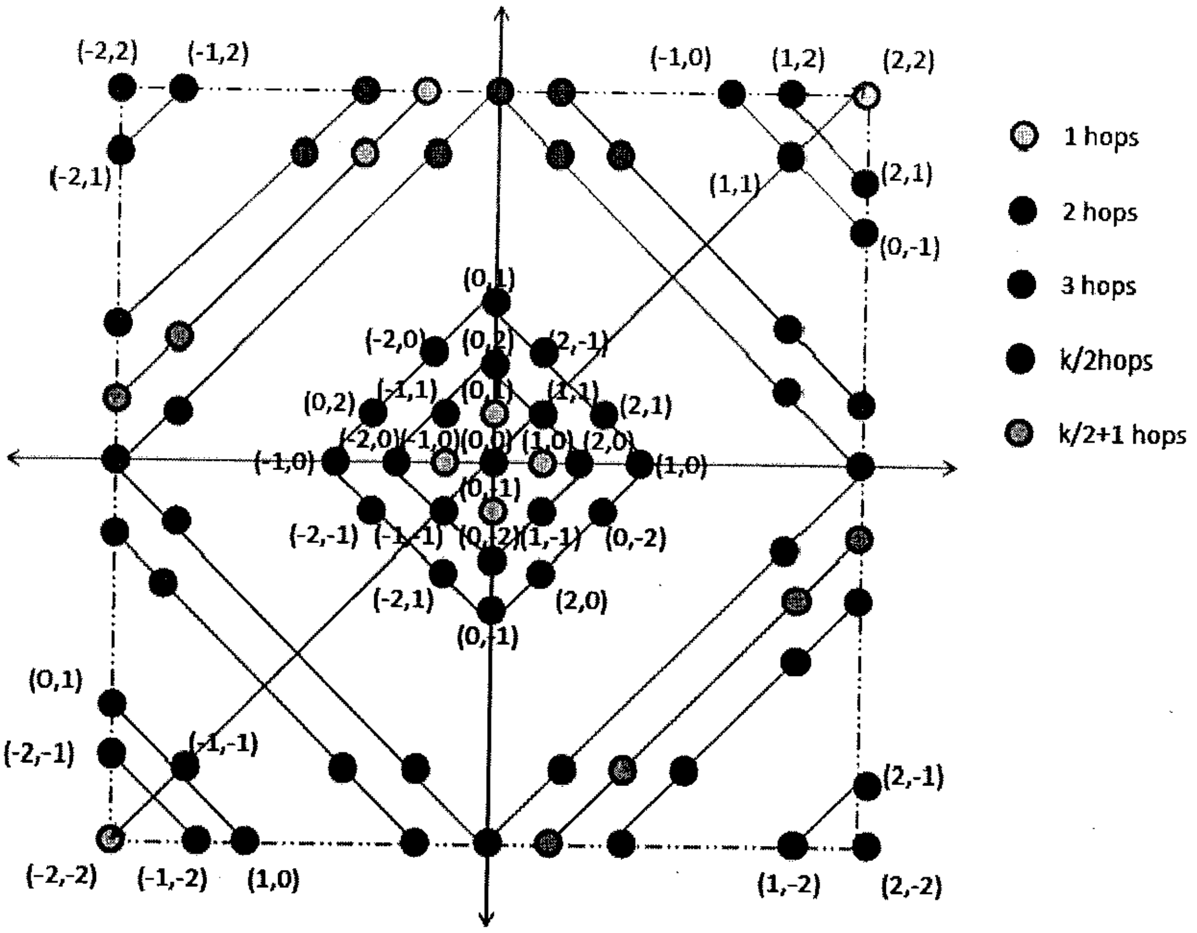


Fig. 5.1. Mapped Illustration for Odd X-Torus.

B. When k is Even

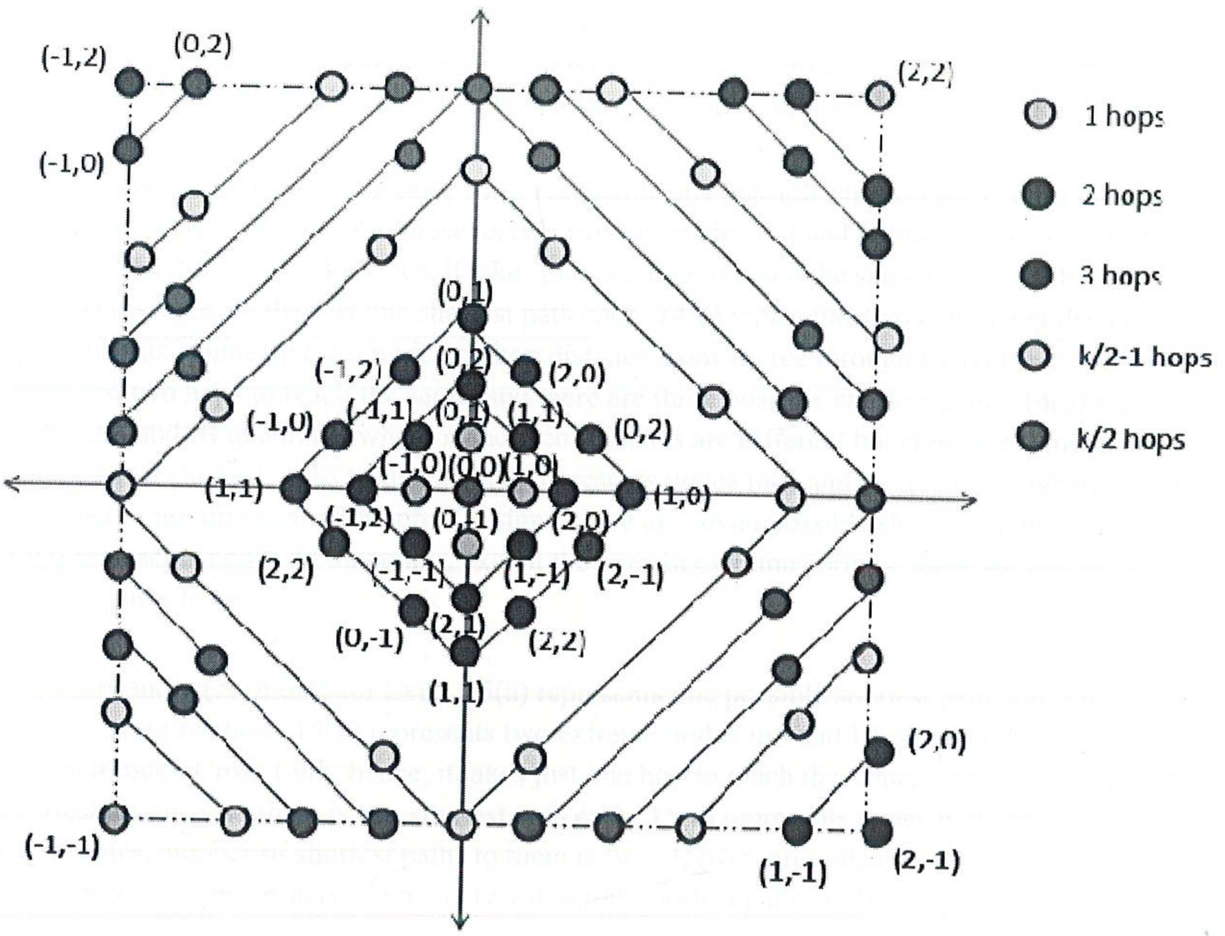


Fig. 5.2. Mapped Illustration for Even X-Torus.

Similarly, when k Even as in Figure (5), blue nodes are the farthest at a distance of $k/2$ hops, therefore, $k/2$ becomes the diameter of this network. Comparing with Figure (3), we can see that number of nodes, one hop distance from source are 6; 4 through vertical/horizontal paths, namely (0, 1), (0,-1), (1, 0), (-1, 0) while 2 through Cross Links, namely (2,2) and (-1,-1), the nodes are represented in Figure (5) as purple colored circles. Calculating k/2 here gives 4/2=2, hence, we note that reaching any destination from source will take maximum of 2 hops through a combination of horizontal/vertical and/or Cross Links.

5.1 Analysis of Number of Possible Shortest Paths

Considering node as (a, b) and $k \ x \ k$ topology for X-Torus. S is the maximum number of shortest paths that can be obtained using shortest path algorithm to reach (a, b). When we analyze our network for the number of shortest possible paths, we derived (14) and (15). As an illustration, if the packet has to reach the node (-2, 2) in Odd topology, we have three possible shortest paths using all possible links as seen in 14(c), each of 2 hops, via (0, -1) or (-2, -2) or (2, 2). The difference in the formulae between Odd and even topology reflects the difference in the arrangement of Cross Links. Please note that in all cases, (0, 0) has been considered as source node.

Understanding formulae for Odd, 14(a) represents one possible shortest path, since destination is same as source here. 14(b) represents two extreme nodes in I and III quadrants, connected to source through Cross Link; hence, it takes just one hop to reach the same, since one hop is shortest distance, so there is one shortest path each. 14(c) represents two extreme nodes in II and IV quadrants, connected to a node one hop distance from source through Cross Link; hence, it takes just two hops to reach the same, and there are three possible shortest paths. 14(d) represents nodes in I and III quadrants where x and y coordinates are different but signs are same, there are two possible shortest paths to them. 14(e) represents nodes in II and IV quadrants where x and y coordinates are different with opposite signs, there are seven possible shortest paths to them. 14(f) represents nodes on diagonals, except the ones in extreme corners; there are two possible shortest paths to them.

Understanding formulae for Even, 15(a) represents one possible shortest path, since destination is same as source here. 15(b) represents two extreme nodes in I and III quadrants, connected to source through Cross Link; hence, it takes just one hop to reach the same, since one hop is shortest distance, so there is one shortest path each. 15(c) represents nodes with different x and y coordinates, number of shortest paths to them is two. 15(d) represents nodes on diagonals, except the ones in extreme corners; there are two possible shortest paths to them.

$$for\ Odd =$$

$$\begin{cases} S = 1, if\ a\ or\ b = 0 \\ S = 1, if\ a = b = \left\lfloor \frac{k}{2} \right\rfloor, where\ a\ and\ b\ have\ same\ sign \\ S = 3, if\ a = b = \left\lfloor \frac{k}{2} \right\rfloor, where\ a\ and\ b\ have\ different\ sign \\ S = 2, if\ a \neq b\ and\ a = \left\lfloor \frac{k}{2} \right\rfloor or\ b = \left\lfloor \frac{k}{2} \right\rfloor, where\ a\ and\ b\ have\ same\ sign \\ S = 7, if\ a \neq b\ and\ a = \left\lfloor \frac{k}{2} \right\rfloor or\ b = \left\lfloor \frac{k}{2} \right\rfloor, where\ a\ and\ b\ have\ different\ sign \\ S = 2, if\ a = b \neq \left\lfloor \frac{k}{2} \right\rfloor \end{cases}$$

Equations 14(a) – 14(f)

$$
for\ Even = \begin{cases} S = 1, if\ a\ or\ b = 0 \\ S = 1, if\ a = b = \frac{k}{2} \\ S = 2, if\ a \neq b\ and\ a = \frac{k}{2}\ or\ b = \frac{k}{2} \\ S = 2, if\ a = b \neq \frac{k}{2} \end{cases}
$$

EQUATIONS 15(A) – 15(D)

A symmetry in observed in the topology for the above conditions/equations and hence we find out the maximum number of possible paths a packet will take to traverse to a particular node which helps in reducing traffic at a single path(which might be shortest) and avoiding congestion. The analysis further helps in designing and simulation of ALBR Algorithm.

CHAPTER 6

ADAPTIVE LOAD BALANCED ROUTING (ALBR) ALGORITHM

Since X-Torus is part of Torus family of topologies, hence all algorithms running over the same will work on X-Torus as well. The Cross Links further assist in creating many short routes from source to destination using shortest path algorithm [2], however, with shortest paths the problem of congestion occurs, as all packets from source to destination tend to use shortest/minimal path. To avoid this situation, we propose an Adaptive Load Balanced Routing Algorithm, where congestion or network load imbalance is sensed using Channel queues while at the same time relying on the network's implicit backpressure to collect approximate global information from further parts of the network, as done in Adaptive Channel Queue Routing Algorithm (CQR) [13]. The key point to note here is that, ALBR is not an improvement over CQR; rather it is a modified implementation of the same by using Cross Links with minimal look-ups through backtracking on X-Torus topology, which will be soon explained. Moreover, to avoid congestion and improve fault tolerance, it is very important that packets route themselves to non-minimal/alternate paths when all minimal paths are congested, this will help improving performance by reducing the time required between source and destination than otherwise. Time Complexity of ALBR is $O(N^2)$. This is due to checking congestion (through a buffer) and reach function which chooses random sequences of X and Y traversals.

Definition 1.4: Adaptive Load Balanced Routing Algorithm is an implementation of adaptive routing behavior that comes into action after congestion is sensed in minimal paths from source to destination. The algorithm chooses alternate paths by backtracking from destination using Cross Link(s) connected to it therefore reducing number of lookups, and reaching the other end of it by random sequence of x,y traversals, further from where using destination Cross Link, packet reaches destination without loss of sequence.

In the process of simulating network performance of ALBR algorithm, we found its potential for improved performance under both light and heavy load. Hence, helping the network during Congestion by routing the paths adaptively. The nature and structure of Cross Links play very important role in determining the performance. We notice, that the structure of Cross Links differs from even and odd versions of X-Torus and that the pattern of routing paths differs from position to position of destination in the Network. On this basis, we study the uses of Algorithm, considering two cases with positions of destination for each, odd and even versions of X-Torus topology.

Case 6.1: When Destination is not on Axis:

When destination is not on any of the axis, the number of Cross Links connected to each node is 1. Without loss of generality, we choose the node (0,0) as in Figure (3) or node 9 as in Figure (6) as source. Figure (6), illustrates the algorithm on Even topology, where node 9 is source and 6 as destination, during initial phase the packets choose minimal path such as:

9-5-6

9-10-6

Later, when congestion is sensed using CheckCongestion(x,y) function, node 12 is chosen by backtracking from destination using Cross Link, which is further reached from source using a sequence of random x, y traversal using Reach(Sx,Sy,Δx,Δy) as:

9-8-12

9-13-12

Hence, the paths during congestion from source to destination are:

9-8-12-6

9-13-12-6

Either of them, chosen one at a time randomly to balance load better using LinksE(x,y). The backtracking aspect saves time to search the Cross Link connected to destination by minimizing number of look-ups, therefore, the node on the other end of the destination's Cross Link can be reached using random selection of x, y traversals from source, hence balancing load during congestions.

Similar is the case for Odd topology as Figure (7), where source is 12 and destination as 1. Under minimal conditions paths traversed are:

12-4-0-1

12-20-0-1

While during congestion, making use of Cross Links, paths traversed are:

12-13-18-1

12-17-18-1

Due to random nature of Algorithm, only one of these paths will be used at a particular moment of time.

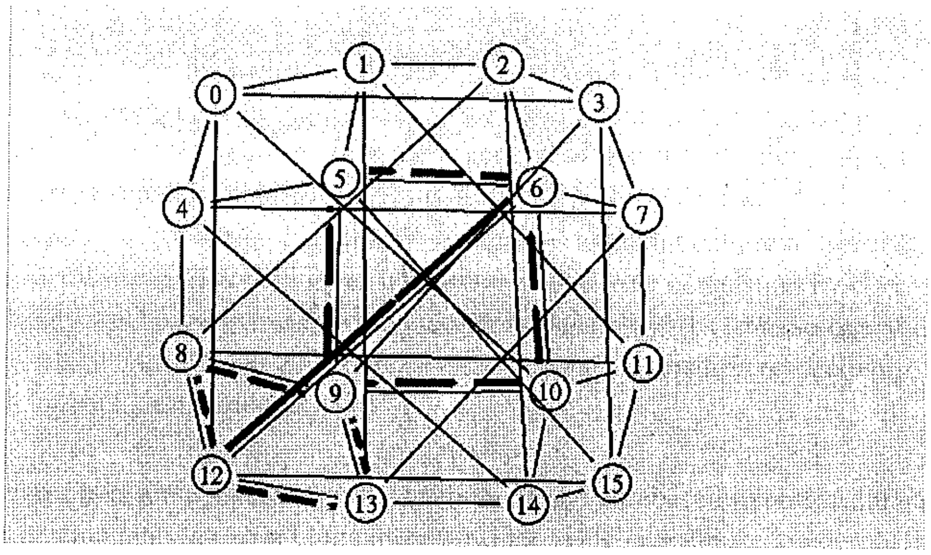
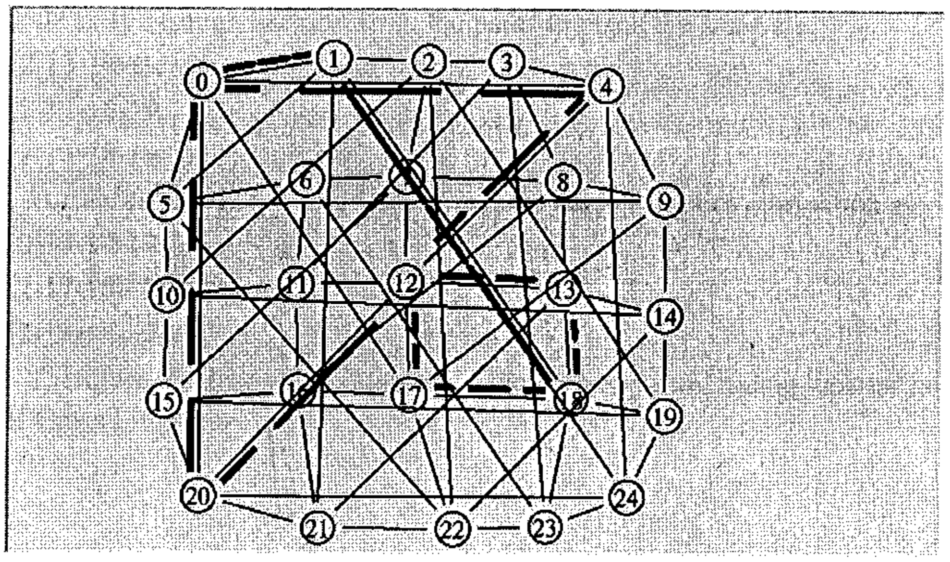


Fig. 6.1 Case 6.1: Simulation of all Possible Paths during Adaptive Load Balanced Routing Algorithm in Even X-Torus on NS-2.



Case 6.2: When Destination is on Axis.

When destination in on axis (either X or Y), the routing pattern changes due to nature and structure of Cross Links.

In Even version of X-Torus topology, every node on X or Y axis have single Cross Link connected. As in Figure (8), where source and destination are 9 and 1 respectively, we see that using minimal path, the traffic flows as:

9-5-1

While during congestion, making use of Cross Link and backtracking node 11 is chosen, which is further reached through a path sequence of 9-8-11 and finally completing the path as 9-8-11-1. Since the intermediate node or the node at the other end of destination through Cross Link is on the same axis as source, this eliminates the possibility of Y direction traversal.

In Odd version of X-Torus topology, the Y axis posses a unique feature of hosting 2 Cross Links from every node on it, which gives it an extra edge during congestion by choosing either of them randomly and hence increasing the number of possible paths from source to destination. In Figure (9), where node 12 and 2 are chosen as source and destination respectively, the minimal path to reach destination is:

12-7-2

Since 2 Cross Links are connected to the destination, the number of intermediate nodes connected to destination via Cross Links increases to 2, which are chosen randomly through ALBR's LinksO(x,y) function's condition for Y axis. The intermediate nodes are further reached through a minimal random sequence of X and Y traversals. Considering intermediate node as 10, we have only 1 set of other path traversing through 12-11-10-2, while considering intermediate node as 19 through backtracking, the paths to reach the same are:

12-13-14-19

12-17-18-19

12-13-18-19

Further from where using Cross Link, the packet reaches destination as:

12-13-14-19-2

12-17-18-19-2

12-13-18-19-2

Therefore, dealing with congestion efficiently, by minimizing lookups and making optimal use of Cross Links.

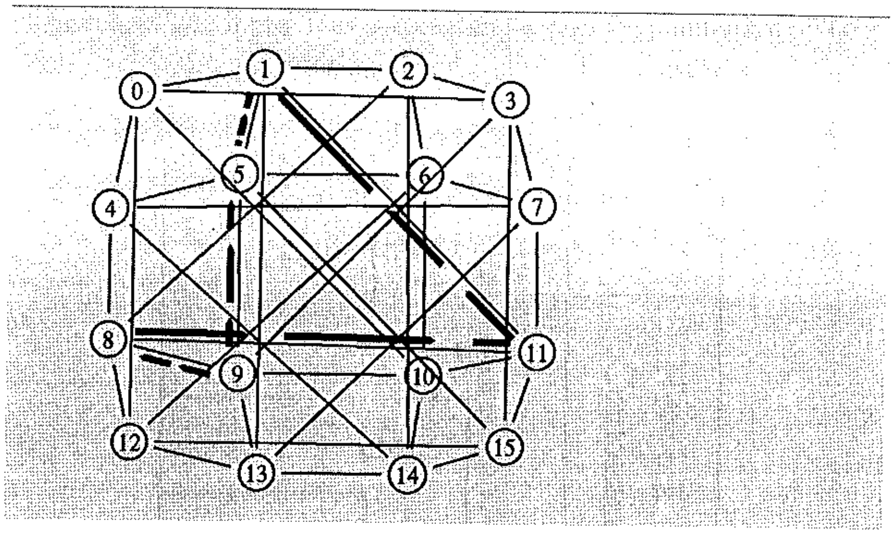


Fig. 6.3 Case 6.2: Simulation of all Possible Paths during Adaptive Load Balanced Routing Algorithm in Even X-Torus on NS-2.

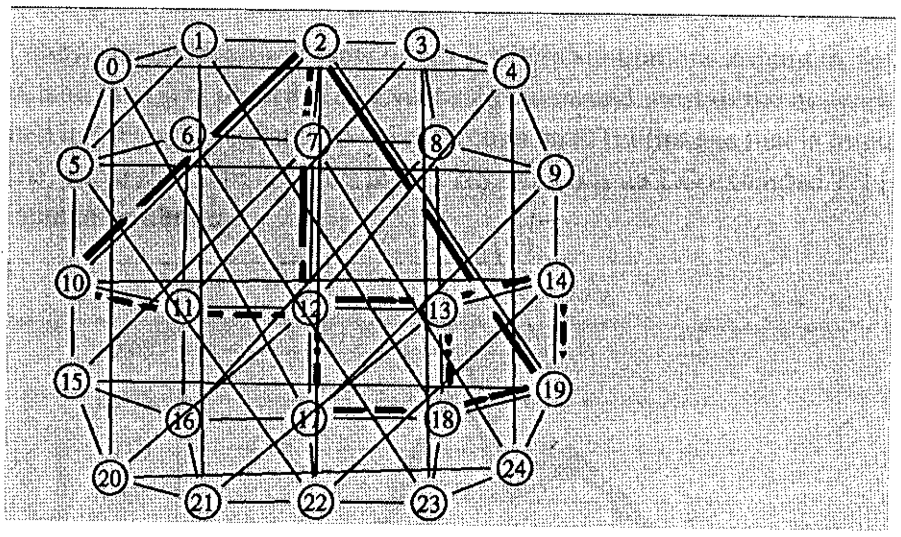


Fig. 6.4 Case 6.2: Simulation of all Possible Paths during Adaptive Load Balanced Routing Algorithm in Odd X-Torus on NS-2.

6.1 Testbed

For the purpose of implementation and simulation of the algorithm we have used Network Simulator's 2nd version (popularly known as NS-2), NS-2 is a discrete Event simulator targeted at networking research, where it provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

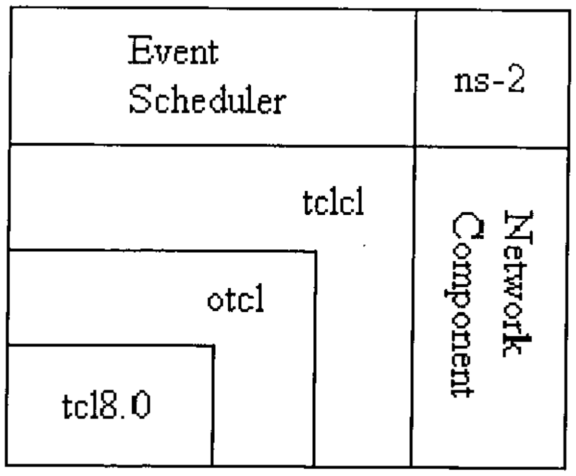


Fig. 6.5 . Architectural view of NS-2.

Figure (10), shows an architectural view of NS-2 where simulations are run in Tcl using the simulator objects in the OTcl library. The Event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using Tclcl. The whole structure together makes NS, which is an OO extended Tcl interpreter with network simulator libraries.

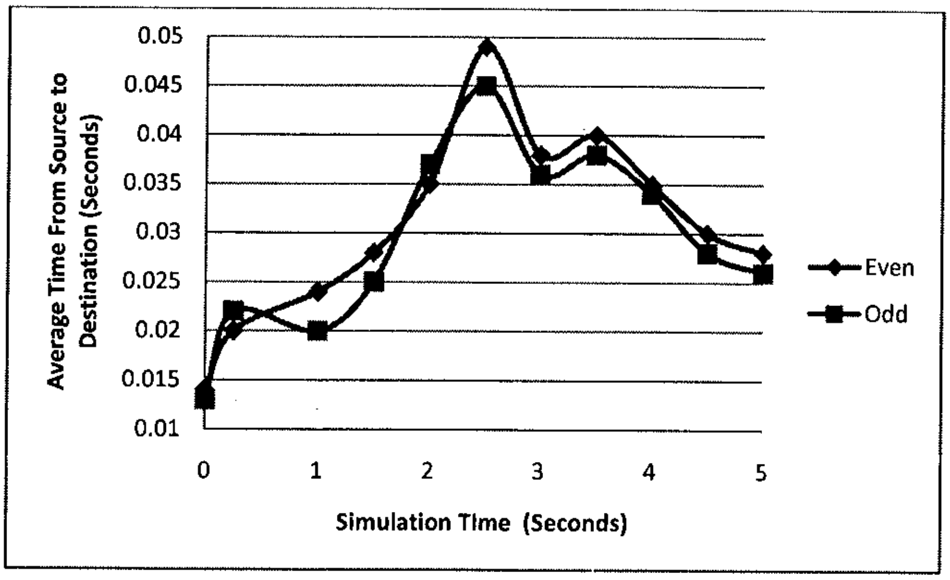


Fig. 6.6 Analysis of Average Time taken by Packets to Flow from Source to Destination in Even and Odd X-Torus Topologies.

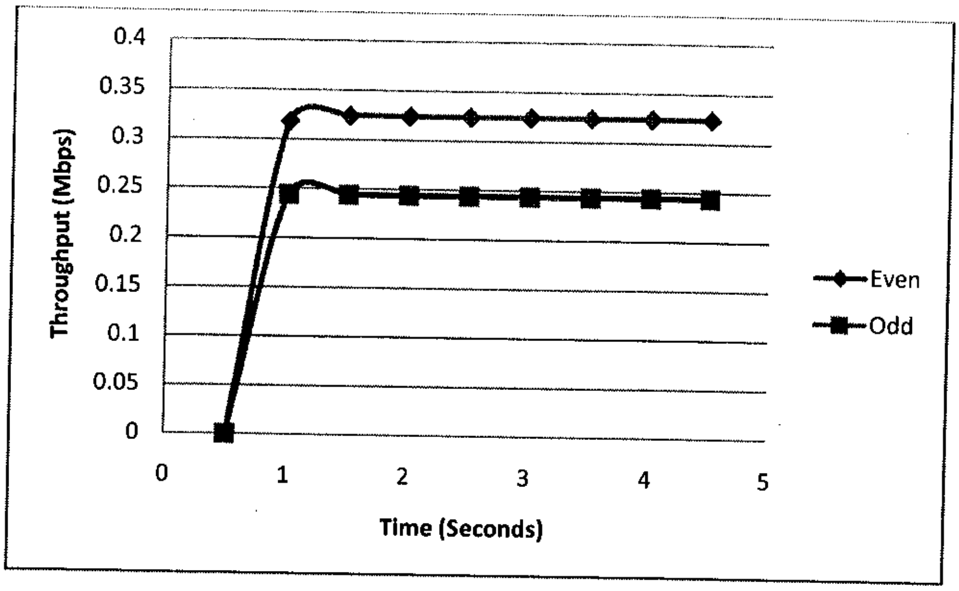


Fig. 6.7 Analysis of Throughput at Destination Nodes for Even and Odd X-Torus Topology when Granularity = 0.5.

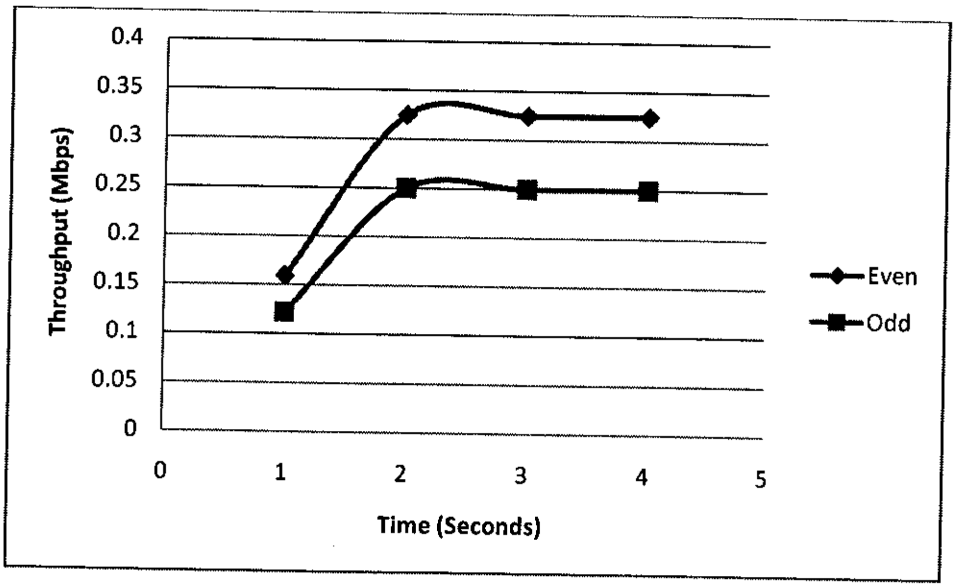


Fig. 6.8. Analysis of Throughput at Destination Nodes for Even and Odd X-Torus Topology when Granularity = 1.

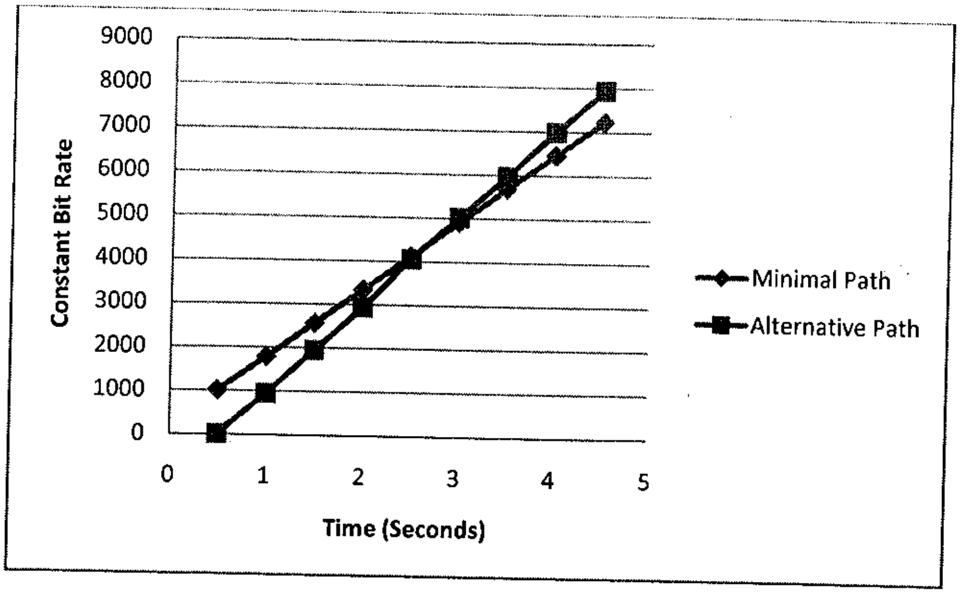


Fig. 6.9. Analysis of Constant Bit Rate on Minimal and alternate path during Congestion for Even X-Torus.

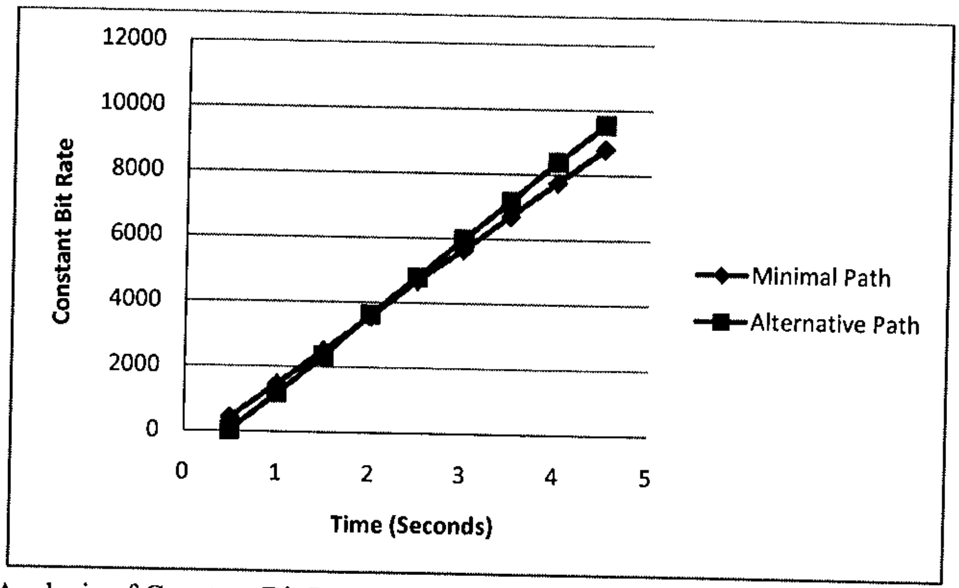


Fig. 6.10. Analysis of Constant Bit Rate on Minimal and alternate path during Congestion for Odd X-Torus.

TABLE 2

ANALYSIS OF AVERAGE TIME TAKEN BY PACKETS TO FLOW FROM SOURCE TO DESTINATION IN EVEN AND ODD X-TORUS TOPOLOGIES

Simulation Time (Seconds)	Average Time (Seconds) Taken From Source to reach Destination using different X-Torus Topology	
	Even X-Torus Topology	Odd X-Torus Topology
0	0.014	0.013
0.25	0.02	0.022
1.0	0.024	0.02
1.5	0.028	0.025
2.0	0.035	0.037
2.5	0.049	0.045
3.0	0.038	0.036
3.5	0.04	0.038
4.0	0.035	0.034
4.5	0.03	0.028
5.0	0.028	0.026

TABLE 3

ANALYSIS OF THROUGHPUT AT DESTINATION NODES FOR EVEN AND ODD X-TORUS TOPOLOGY WHEN GRANULARITY = 0.5

Simulation Time	Throughput for different X-Torus Topology	
(Seconds)	Even X-Torus Topology	Odd X-Torus Topology
0.504	0	0
1.002	0.318	0.244
1.502	0.324	0.244
2.002	0.324	0.244
2.502	0.324	0.244
3.002	0.324	0.244
3.502	0.324	0.244
4.002	0.324	0.244
4.502	0.324	0.244

TABLE 4

ANALYSIS OF THROUGHPUT AT DESTINATION NODES FOR EVEN AND ODD X-TORUS TOPOLOGY WHEN GRANULARITY = 1

Simulation Time (Seconds)	Throughput for different X-Torus Topology	
	Even X-Torus Topology	Odd X-Torus Topology
1.002	0.159	0.122
2.002	0.3245	0.2495
3.002	0.3245	0.2495
4.002	0.3245	0.2495

TABLE 5

ANALYSIS OF CONSTANT BIT RATE ON MINIMAL AND ALTERNATE PATH DURING CONGESTION FOR EVEN AND ODD X-TORUS TOPOLOGY

Simulation Time (Seconds)	Throughput for different X-Torus Topology			
	Even X-Torus Topology		Odd X-Torus Topology	
	Minimal Path	Alternate Path	Minimal Path	Alternate Path
0.5	1014	9.2	421	32
1.0	1791	946	1468	1172
1.5	2569	1946	2515	2348
2.0	3346	2939.2	3562	3623
2.5	4124	4056	4679	4784
3.0	4901	5019	5656	5984
3.5	5679	5966	6703	7160
4.0	6456	6986	7750	8384
4.5	7234	7939	8797	9536

6.2 DUAL ADAPTIVE ROUTING (DAR) ALGORITHM

The routing method employed by a network determines the path taken by a packet from a source terminal node to a destination terminal node. A route or path is an ordered set of channels $P =$

$c_1, c_2, \dots\dots\dots c_k$ where the output node of channel c_i equals the input node of channel c_{i+1}, the source is the input to channel c_1, and the destination is the output of channel c_k. In some networks, there is only a single route from each source to each destination node, whereas in others, such as the torus network in Figure (1), there are many possible paths. When there are many possible paths, a good routing algorithm balances the load uniformly across channels regardless of the offered traffic pattern. Continuing the roadmap analogy, while the topology provides the roadmap, the roads and the intersections, the routing method steers the car, making the decision on which way to turn at each intersection. Just as in routing cars on a road, it is important to distribute the traffic – to balance the load across different roads rather than having one road become congested while parallel roads are empty [10].

DAR algorithm exploits the fact that X-Torus topology behaves as a simple Torus network before the Cross Link is used for traversal. It thus uses a blend of two popular Torus routing algorithms, Oblivious Routing Algorithm [11] and Adaptive Routing Algorithm [10]. Before congestion is sensed in the network, Shortest or Minimal Path Algorithm is used. As soon as the network senses congestion, DAR comes into play. We initiate the process by looking for the Connecting node, which is the node connected to the Destination node directly via Cross Link. Now the major task is to reach from the Source node to the Connecting node, for which we use the above stated blend of the two algorithms. Finally, we traverse from the connecting node to the destination node making use of the Cross Link.

The major advantage of this algorithm is that the oblivious selection of quadrant balances the load globally, whereas the adaptive routing within the selected quadrant performs the local balancing.

C. Pseudo-code

The algorithm involves the following steps:-

- **Backtracking:** Backtrack from the Destination node denoted by $D(x1, y1)$ where x and y are the co-ordinates of the node using the Cross Link connected to it to reach the Connecting node denoted by $C(x2, y2)$. This step is done to minimize the number of lookups thereby reducing the time to search for $C(x2, y2)$. $C(x2, y2) =$ BACKTRACK $D(x1, y1)$. The backtracking step is similar to the one used in ALBR, which can be referred from Section 6(A) Function:(4)LinksO (for Odd) and Function:(5)LinksE (for Even).
- **Quadrant Selection:** Select a minimal quadrant obliviously having the Source node $S(x, y)$ and the Connecting node $C(x2, y2)$ as the corner nodes.
- **Initial Phase Routing:** Random selection of a node from within the selected quadrant called the Intermediate node denoted by $I(x3, y3)$ where $x2 \geq x3 \geq x$ and $y2 \geq y3 \geq y$.
- **Intermediate Phase Routing:** Use Adaptive routing to route packets from $S(x, y)$ to $I(x3, y3)$ and then again from $I(x3, y3)$ to $C(x2, y2)$.
- **Final Phase Routing:** After reaching $C(x2, y2)$ use the Cross Link connected to it to reach the destination node$D(x1, y1)$.

6.2.1 *Working of DAR*

In the Figure 16(a), considering the node (2, 1) as the source node and the node (-1, 2) as the destination node, the working of Dual Adaptive Routing is explained as follows: On backtracking from the destination node using the Cross Link connected to it, we reach the node (1, -1). Here the node (1, -1) is called the connecting node since it is connected to the destination node via a Cross Link. Now we apply minimal oblivious routing algorithm to route packets from the source node i.e. (2, 1) to the connecting node i.e. (1, -1) and once we reach the connecting node we route packets directly through the Cross Link to the destination node. Minimal quadrant is chosen such that it contains the source and connecting nodes as the corner nodes. Figure 16(b) shows the minimal quadrant chosen in this case. Thereafter, an intermediate node is selected from within the quadrant. Packets are routed from the source to the intermediate node and then from the intermediate node to the connecting node using adaptive routing. In this example, there are six possible intermediate nodes as shown in the Figure 6.11.3

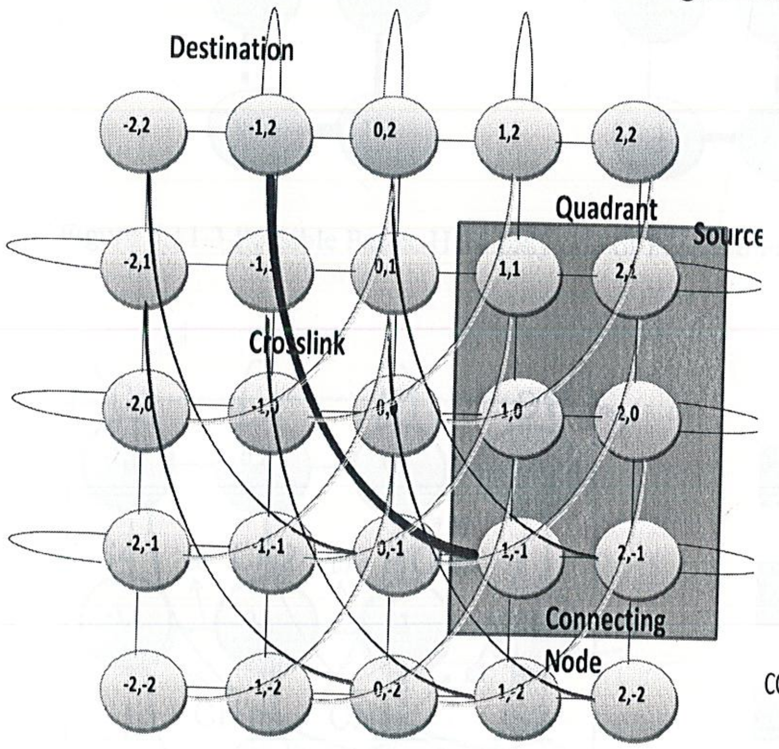


Figure 6.11.1 Illustration of DAR on 5 x 5 X-Torus Topology.

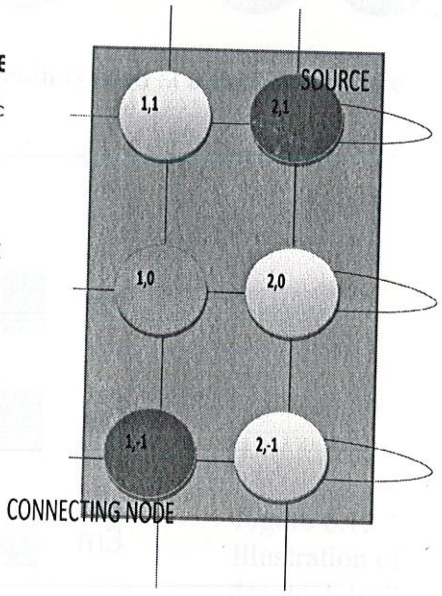


Figure 6.11.2 Minimal Quadrant.

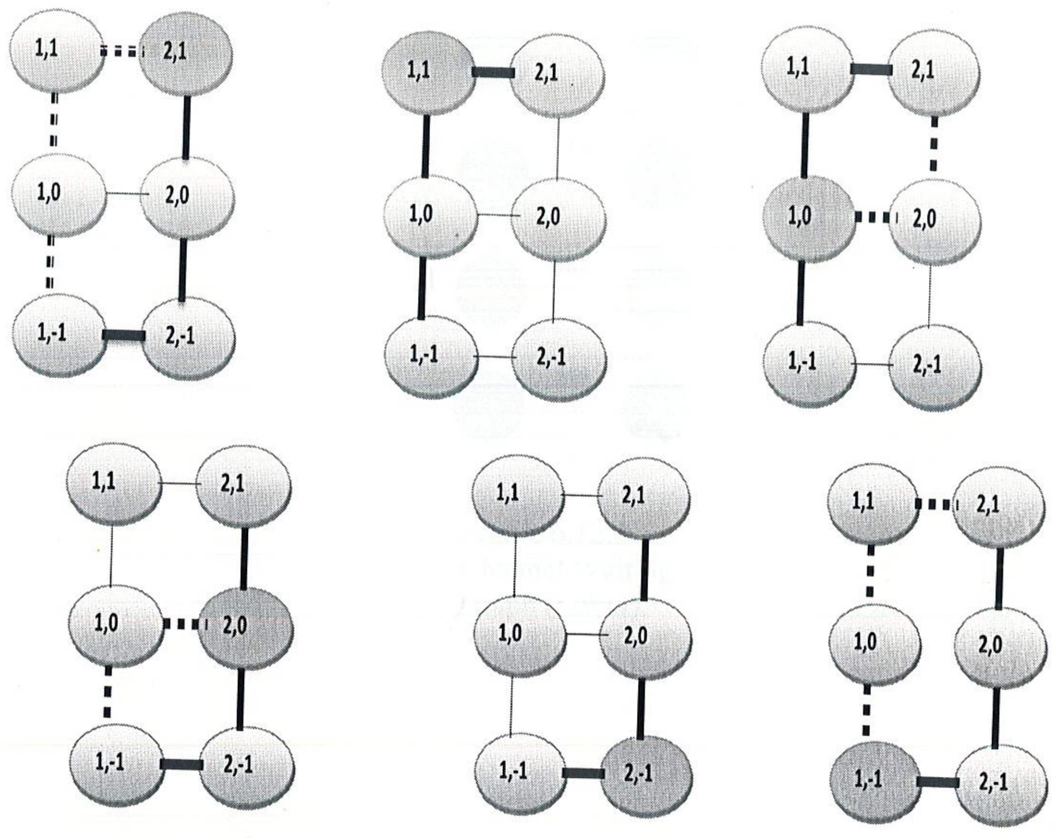


Figure 6.11.3 Possible Paths. Here only one of Possible Path is used at a Particular Time.

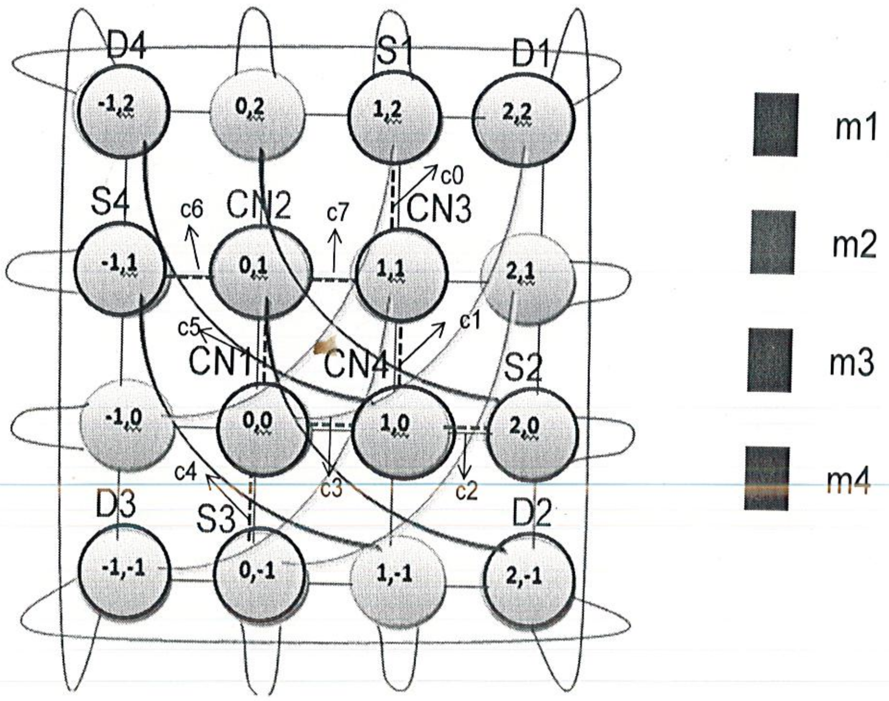


Figure 6.12.1 Illustration of deadlock in X

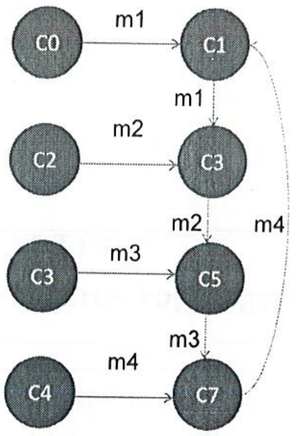


Figure 6.12.2
Channel Waiting
Graph (CWG).

CHAPTER 7
Deadlock in X-Torus Topology

Network interface is used to transmit the data between nodes. Packets travel multiple switches in a switch based network whereas they travel multiple intermediate nodes in a direct network. Due to various factors [10], the delivery of the packet might be hindered. This is true even if there are no faulty connections between source and destination nodes.

At each node, fragments of bits are stored in a buffer. If the frequency of incoming packets is greater than outgoing, there might be chances that the buffer gets saturated while more traffic is coming. A deadlock occurs in such situations where the packets may not reach to their destination due to limited buffer capacity. In this case, the packet is blocked forever, as the resources needed by it are never free.

Another case of packet blocking is livelock. In this scenario, packets actively move across nodes but are not able to reach their destination. This is because the channels involved are providing resources to other packets. Finally, if the blocking continues over a long period under high traffic, the packet might get dropped, a process known as starvation. As deadlock causes packets to wait indefinitely, this might also lead to livelock situation where the packet moves around its destination. Hence, these three events-deadlock, livelock and starvation needs to be removed for efficient propagation of data in interconnection networks [16]-[20].

7.1 Deadlock Avoidance, Recovery and Detection Mechanism

Routing algorithms are designed in a manner, which tends to avoid deadlock. This is done by utilizing both physical and virtual channel resources. Avoidance based routing algorithms tend to cause restrictions in the network while recovery based algorithms need to avoid potential deadlocks while recovering from the deadlock.

The Channel Waiting Graph (CWG) model captures the relationship between channels in the same way as the channel dependency graph. However, this model does not distinguish between routing and selection functions. Thus, it considers the dynamic evolution of the network because the selection function takes into account channel status. However, the most important result is a necessary and sufficient condition for deadlock-free routing. This condition assumes that a packet can wait on any channel supplied by the routing algorithm. It uses the concept of true cycles. A cycle is a true cycle if it is reachable, starting from an empty network. The theorem states that a routing algorithm is deadlock-free if and only if there exists a restricted CWG that is wait-connected and has no true cycles. This condition is valid for incoherent routing functions and for routing functions defined on $C \times N$. However, it proposes a dynamic condition for deadlock avoidance, thus requiring the analysis of all the packet injection sequences to determine whether a cycle is reachable (true cycle).

Deadlock recovery techniques, as the name suggests, are a way to recover from already occurred deadlocks. They provide a way to recover the resources required by a waiting process. However, this is only feasible in an environment where deadlock occurrence is rare. It requires an understanding of all the properties causing the deadlocks are formed in the interconnection network. Consequently, it releases the resource causing the deadlock. Deadlock recovery methods differ in ways in which they recognize deadlock. It involves multiple packets and requires transfer of information between nodes. Typically, there are two types of detection mechanisms, centralized and distributed. While the former involves exchange of data between nodes, the latter involves distribution of information locally. This is more preferred as deadlock packets might occupy channels in centralized.

As the packet is discovered, the resource management process starts whose purpose is to free the allocated resources. This can be achieved by either taking the resources from the current process or by liberating the deadlocked process of its resources. A progressive recovery technique de-allocates resources from a process and gives it to the process, which is causing the deadlock. Regressive process on the other hand takes resources from the deadlocked process by killing them. If a packet is detected at the source node, a kill signal is sent across the nodes to free up the resources. This is the solution proposed in compression less routing. After a random delay, the packet is injected again into the network. This re-injection requires a packet buffer associated with each injection port. Note that a packet that is not really deadlocked may resume advancement and even start delivering flits at the destination after the source node presumes it is deadlocked. Thus, this situation also requires a packet buffer associated with each delivery port to store fragments of packets that should be killed if a kill signal reaches the destination node. If the entire packet is consumed without receiving a kill signal, it is delivered. Obviously, this use of packet buffers associated with ports restricts packet size. The texts explain here is taken from [10], [11], [16]-[20].

7.2 Deadlock Detection Algorithm for X-Torus

- List all the channels $c_1, c_2, \dots \dots \dots c_{n-1}$ that are being held by packets between any two $S(x1, y1)$ and $D(x2, y2)$ where $x1$ and $x2$ are the x-coordinates and $y1$ and $y2$ are the y-coordinates of the nodes S and D respectively. Suppose a channel c_j is acquired by a message M_i released between nodes $S(x1, y1)$ and $D(x2, y2)$, then we say that M_i holds c_j
- $i = 0 \; to \; n-1$
- $j = 0 \; to \; n-1 \neq i$
- $k = 0 \; to \; n-1$

- flag: variable used to detect the presence of any cycles in the resource wait-for graphs
- l = set of all values of i or j already taken into consideration
- Initial Values: $i = 0, j \neq i, k = 0, l = \varphi$

```
Step1: Create a new class $C_k$ and set flag := $c_{i-\{1\}}$
Step2: Search for a message $M_i$ holding a channel $c_j$
    $M_i$ := holds($c_j$)
Step3: Find out whether $M_i$ is waiting for another channel $c_j$ to proceed or not
    Assign $i$ to $j$ ($j := i$)
        If yes Then check whether $c_j$ is equal to flag.

      If $c_j$ == flag is True
          This means that we have reached the channel with which we started the new
          sub-class and hence a cycle is formed indicating a deadlock. Store the
          value of '$k$' as this class will be included in the recovery phase.
        End class $C_k$. Increment $k$ and go to step 1.

        If $c_j$ == flag is False Then go to step 2.
    If $M_i$ is not waiting for another channel $c_j$ then this means that no cycles
    are formed and thus no deadlock. Discard the value of $k$ as this class
    would not be included in the recovery phase.
        End class $C_k$. Increment $k$ and go to step 1.
```

7.3 A progressive Deadlock Recovery Approach

Deadlocks are quite rare in a network. In case of Adaptive Routing Algorithm, it has been found that deadlock recovery routing algorithms exhibit superior performance characters over their deadlock avoidance counterparts. Deadlock occurs in an interconnection network when a group of agents, usually packets, is unable to make progress because they are waiting on one another to release resources, usually buffers or channels. If a sequence of waiting agents forms a cycle, the network is deadlocked. Deadlock is catastrophic to a network. For deadlock free routing a necessary and sufficient condition is the absence of cycles in channel dependency graph. There are two phases in any deadlock recovery algorithm: detection and recovery [10], [16]-[20].

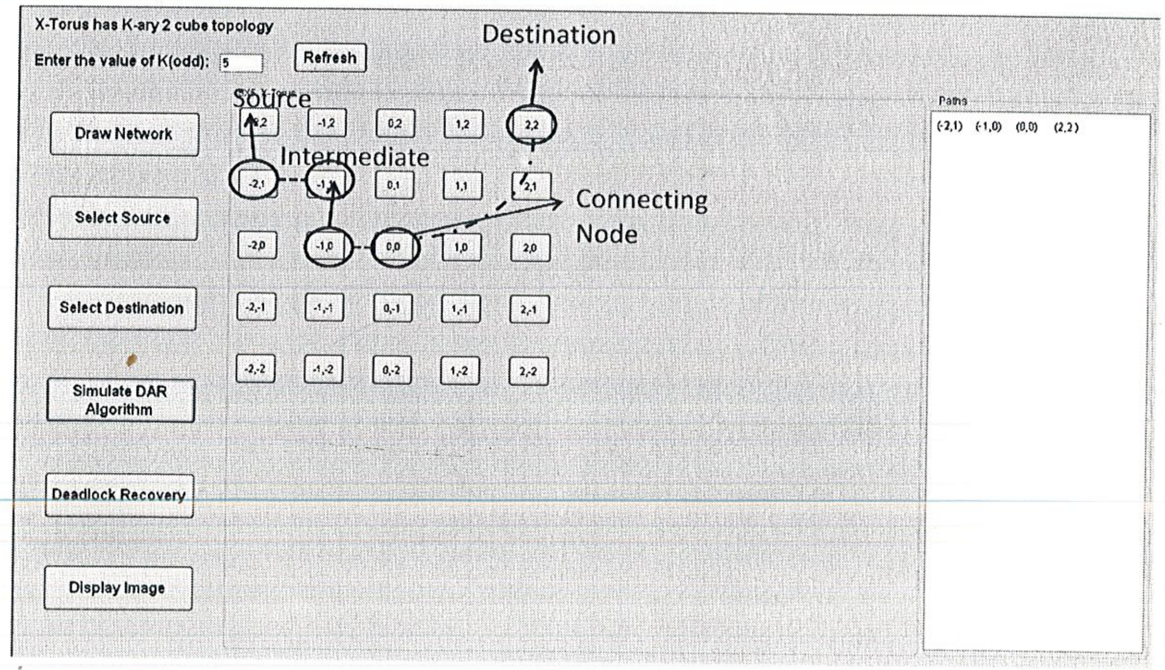
The method of detection we present is very simple and inexpensive that detects the presence of all the deadlocks. In addition, a progressive approach to remove deadlock is used which does not require any exclusive buffer to store a deadlocked packet and solves the problem taking advantage of the high path diversity of X-Torus. It works by diverting the deadlocked packets from the current node to its one of the free neighbors and then transmission is resumed back from the free neighbor to the current node.

7.4 Deadlock Recovery Algorithm for X-Torus

1. For every class , , in which deadlock was detected in phase 1 repeat the following steps:
2. **Step1:** For every channel in the class held by packets in between the nodes and check whether any of the neighboring nodes of are free except the one that is deadlocked until one such node is found. Let this node be denoted by .
3. **Step2:** Divert the deadlocked packets from to for some time. Once all other deadlocked packets move towards their destination then packets are traversed back from to).

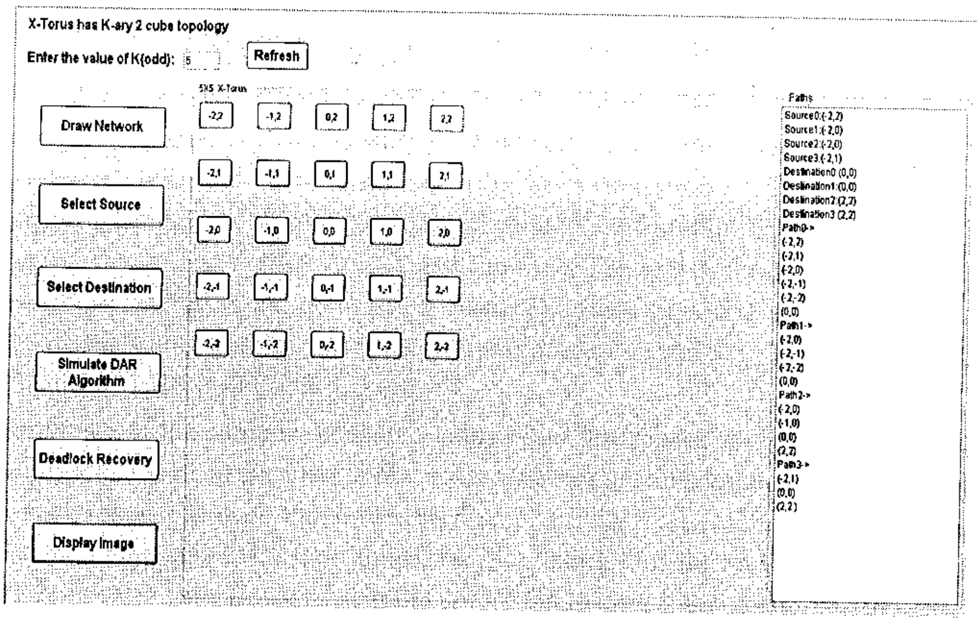
7.5 X-Torus Topology Interface

Finally, to understand the working of DAR we have designed a X-Torus Topology Interface (build using Visual C#.NET Technology) that implements the DAR algorithm incase of multiple source and destination nodes. The interface is designed for both even and odd X-Torus topologies and calculates the connecting node by examining the Cross Link that is connected to the destination node and also finds the intermediate node and hence displays all the nodes in the path from source to destination node and draws the route of packets accordingly. It also displays an error message to the user incase a deadlock occurs when the user selected multiple source and multiple destinations. A snapshot of the interface are shown using Figure 7.1



(-2,1) (-1,0) (0,0) (2,2)

Figure 7.1 Single Source & Single Destination.



X-Torus has K-ary 2 cube topology

Enter the value of K(odd): 5

```
Source0:(-2,2)
Source1:(-2,0)
Source2:(-2,0)
Source3:(-2,1)
Destination0 (0,0)
Destination1:(0,0)
Destination2:(2,2)
Destination3 (2,2)
Path0->
(-2,2)
(-2,1)
(-2,0)
(-2,-1)
(-2,-2)
(0,0)
Path1->
(-2,0)
(-2,-1)
(-2,-2)
(0,0)
Path2->
(-2,0)
(-1,0)
(0,0)
(2,2)
Path3->
(-2,1)
(0,0)
(2,2)
```

Figure 7.2 Multiple Sources & Multiple Destinations.

CONCLUSION AND FUTURE WORK

We have proposed a new Adaptive Load Balanced Routing Algorithm for Odd and Even forms of X-Torus topologies, where after sensing the congestion through Channel Queues, packets are diverted to alternate paths, chosen by backtracking from destination using Cross Link connected to it, and reaching the other end of it by minimal sequence of random x, y traversal. Superseding the work done earlier on X-Torus topology [2], which focused only on shortest route algorithm .In current work, congestion/deadlock issues caused due to it has been tackled. To come across the same, several formulas from equation 1-4 and 8-11 were developed, which eventually lead to ALBR and DAR algorithms.

The performance of the same is measured using NS-2 and Gawk, a 60% drop in average time for Even while 59.37% drop for Odd is observed after congestion, by the end of 5 seconds of simulation, with respect to peak value observed respectively. Hence, showing almost similar behavior between Odd and Even topologies. Further, using Perl for text analysis of tracefile, stable throughput after congestion is also observed. On testing the algorithm using CBR, we observe the rise of the same in alternate path during congestion after around mid-time. Time Complexity of ALBR is $O(N^2)$. Therefore, showing the algorithm's capability to perform optimal in all kind of loads on network.

We have also proposed Dual Adaptive Routing (DAR) Algorithm, which is a blend of Oblivious Routing and Adaptive Routing. The major advantage of this algorithm is that the oblivious selection of quadrant balances the load globally, whereas the adaptive routing within the selected quadrant performs the local balancing. It efficiently makes use of Cross Links and hence works explicitly for X-Torus topology thus providing more efficient routing. The proposed progressive deadlock recovery mechanism takes advantage of the high path diversity of X-Torus. The low cost and simple mechanism for deadlock detection as against many conservative mechanisms is very effective.

A lot of issues concerning X–Torus needs further research. Some of them are implementation of similar Adaptive Load Balanced Routing Algorithm on non–symetric forms of topology, which will further require new set of formulations of Cross Links and more concentration on fault–tolerant aspects of the same. Other issues are: broadcasting, multicast routing and fault tolerant routing, Broadcasting and multicasting are fundamental collective communication operations. What's more, as the number of nodes in an X-Torus topology increases, the chance of failure also increases. Hence, it is essential to design a fault-tolerant algorithm that can route packets in the presence of faulty components. Further, since this is the first improvement over existing algorithm for X-Torus topology and the earlier was not simulated [2], in the current version we used Constant Bit Rate and simulation lasted for 5 seconds. However, we plan to use Gaussian distribution to approximate the behavior of VBR traffic for longer duration in future work.

SOURCE CODE

A. The ALBR Algorithm

1) Main Function

```
Begin
 Switch(k)
  Case I: k is Odd
  {
  //Condition I under Odd
  If (x=Δ and y=0)
     Send the message to the local node and EXIT;
   //Condition II under Odd
  If (|Δx|+|Δy|<=k-1/2)
   {
   IsCongestion = CheckCongestion(x,y);
   If(IsCongestion = True)
    {
    LinksO(x,y);
    //Calling Cross Links function for Odd topology
    }
   Else
      D_Forward = Select (Δx,Δy);
  }
  //Condition III under Odd
  Else if (Δx>0 and Δy>0)
      {
      IsCongestion=
      CheckCongestion(x,y);
      If(IsCongestion = True)
       {
       LinksO(x,y);
       //Calling Cross Links function for Odd topology
       }
      Else
         D_Forward = UFD;
     }
  //Condition IV under Odd
  Else if (Δx<0 and Δy<0)
      {
       IsCongestion = CheckCongestion(x,y);
       If(IsCongestion = True)
        {
        LinksO(x,y);
        //Calling Cross Links function for Odd topology
        }
       Else
          D_Forward = LBD;
      }
 //Condition V under Odd
 Else
   {
    IsCongestion = CheckCongestion(x,y);
    If (IsCongestion = True)
     {
     LinksO(x,y);
     //Calling Cross Links function for Odd topology
     }
    Else
       D_Forward = LBD or UFD;
   }
}//End of Case I
Case II: k is Even
{
//Condition I for Even
If (x=Δy=0)
   Send the message to the local node and EXIT;
//Condition II for Even
If (|Δx|+|Δy|<=k/2)
 {
  IsCongestion = CheckCongestion(x,y);
  If (IsCongestion = True)
   {
   LinksE(x,y);
```

```
          //Calling Cross Links function for Even topology
          }
         Else
            D_Forward = Select(Δx,Δy);
       }
     //Condition III for Even
     Else
       {
        IsCongestion = CheckCongestion(x,y);
        If (IsCongestion = True)
         {
          LinksE(x,y);
          //Calling Cross Links function for Even topology
         }
         Else
            D_Forward = UFD;
       }
    }//End of Case II
EXIT
```

2) *Select Function*

```
//Choose one direction between d1 and d2;
Select(d1,d2)
{
r = random(0,1);
//choose 0 or 1 with equal probability for r;
If(r=0)
 {
 If(d1>0) return X+;
   Else return X-;
 }
 Else
   {
   If(d2>0) return Y+;
     Else return Y-;
   }
}//End of Select function
```

3) *CheckCongestion Function*

```
// Function to check congestion
CheckCongestion (x,y)
{
 //Using Channel Queues, network load imbalance is sensed while at the same time relying the
  network's implicit backpressure to collect approximate global information from further parts of
  the network, as done in Adaptive Channel Queue Routing Algorithm
}
```

4) *LinksO Function (for Odd)*

```
//Function to use Cross Links during Congestion in Odd topology
LinksO(x,y)
{
 If (x>0 and y>0) //I Quadrant
  {
  //Finding other end of link by Backtracking
  Ix = x-k/2;
  Iy = y-k/2;
  Δx1 = Ix-Sx1; //Sx1,Sx1 is source, in this case 0,0
  Δy1 = Iy-Sy1;
  If (Δx1==0 and Δy1==0)
     D_Forward = UFD;
   Else
     {
     //Reach from source to other end of Cross Link, connected to Destination
       Reach(Sx1, Sy1, Δx1, Δy1);

     //Reach from the other end of Destination's Cross Link to Destination
       D_Forward = UFD;
     }
}
If (x<0 and y>0) //II Quadrant
 {
      //Finding other end of link by Backtracking
      IIx = x+k/2;
      IIy = y-k/2-1;
      Δx2 = IIx-Sx2; //Sx2,Sx2 is source, in this case 0, 0
      Δy2 = IIy-Sy2;

      //Reach from source to other end of Cross Link, connected to Destination
       Reach(Sx2, Sy2, Δx2, Δy2);

      //Reach from the other end of Destination's Cross Link to Destination
       D_Forward = UBD;
     }
    If (x<0 and y<0) //III Quadrant
    {
     //Finding other end of link by Backtracking
     IIIx = x+k/2;
     IIIy = y+k/2;
     Δx3 = IIIx-Sx3; //Sx3,Sx3 is source, in this case 0,0
     Δy3 = IIIIy-Sy3;
     If (Δx3==0 and Δy3==0)
       D_Forward = LBD;
      Else
        {
         //Reach from source to other end of Cross Link, connected to Destination
          Reach(Sx3, Sy3, Δx3, Δy3);

         //Reach from the other end of Destination's Cross Link to Destination
          D_Forward = LBD;
        }
    }
    If (x>0 and y<0) //IV Quadrant
    {
    //Finding other end of link by Backtracking
     IVx = x-k/2;
     IVy = y+k/2+1;
     Δx4 = IVx-Sx4; //Sx4,Sx4 is source, in this case 0,0
     Δy4 = IVy-Sy4;
```

```
//Reach from source to other end of Cross Link, connected to Destination
  Reach(Sx4, Sy4, Δx4, Δy4);

//Reach from the other end of Destination's Cross Link to Destination
  D_Forward=LFD;
}
If (x=0) //Y Axis
{
r=random(1,0);
//choose 0 or 1 with equal probability for r;
//For using Cross Links as in I Quadrant
If(r=0)
  {
//Finding other end of link by Backtracking
    Vx = x-k/2;
    Vy = y-k/2;
    Δx5 = Vx-Sx5; //Sx5,Sx5 is source, in this case 0,0
    Δy5 = Vy-Sy5;

  //Reach from source to other end of Cross Link, connected to Destination
    Reach(Sx5, Sy5, Δx5, Δy5);

  //Reach from the other end of Destination's Cross Link to Destination
    D_Forward = UFD;
  }
  //For using Cross Links as in II Quadrant
  Else if(r=1)
  {
  //Finding other end of link by Backtracking
  VIx = x+k/2;
  VIy = y-k/2-1;
  Δx6 = VIx-Sx6; //Sx6,Sx6 is source, in this case 0,0
  Δy6 = VIy-Sy6;

  //Reach from source to other end of Cross Link, connected to Destination
   Reach(Sx6, Sy6, Δx6, Δy6);
```

```
        //Reach from the other end of Destination's Cross Link to Destination
          D_Forward = UBD;
        }
    }
If (y=0) //X Axis
    {
    //For using Cross Links as in I Quadrant
    If(x>0)
      {
      //Finding other end of link by Backtracking
        VIIx = x-k/2;
        VIIy = y-k/2;
        Δx7 = VIIx-Sx7; //Sx7,Sx7 is source, in this case 0,0
        Δy7 = VIIy-Sy7;

      //Reach from source to other end of Cross Link, connected to Destination
        Reach(Sx7, Sy7, Δx7, Δy7);

      //Reach from the other end of Destination's Cross Link to Destination
        D_Forward = UFD;
      }
   //For using Cross Links as in III Quadrant
   Else if(x<0)
     {
      //Finding other end of link by Backtracking
        VIIIx = x+k/2;
        VIIIy = y+k/2;
        Δx8 = VIIIx-Sx8; //Sx8,Sx8 is source, in this case 0,0
        Δy8 = IIIIy-Sy8;

      //Reach from source to other end of Cross Link, connected to Destination
        Reach(Sx8, Sy8, Δx8, Δy8);

      //Reach from the other end of Destination's Cross Link to Destination
        D_Forward = LBD;
     }
    }

} //End of LinksO
```

5) LinksE Function (for Even)

```
//Function to use Cross Links during Congestion in Even Topology
LinksE(x,y)
{
 If (x>0 and y>0) //I Quadrant
  {
  //Finding other end of link by Backtracking
  Ix = x-k/2;
  Iy = y-k/2;
  Δx1 = Ix-Sx1; //Sx1,Sx1 is source, in this case 0,0
  Δy1 = Iy-Sy1;
  If (Δx1 == 0 and Δy1 == 0)
    D_Forward = UFD;
  Else
     {
     //Reach from source to other end of Cross Link, connected to Destination
       Reach(Sx1, Sy1, Δx1, Δy1);

     //Reach from the other end of Destination's Cross Link to Destination
       D_Forward = UFD;
     }
  }
If (x<=0 and y>0) //II Quadrant
 {
 //Finding other end of link by Backtracking
 IIx = x+k/2;
 IIy = y-k/2;
 Δx2 = IIx-Sx2; //Sx2,Sx2 is source, in this case 0,0
 Δy2 = IIy-Sy2;

 //Reach from source to other end of Cross Link, connected to Destination
   Reach(Sx2, Sy2, Δx2, Δy2);
  //Reach from the other end of Destination's Cross Link to Destination
    D_Forward=UBD;
  }
 If (x<=0 and y<=0) //III Quadrant
  {
  //Finding other end of link by Backtracking
  IIIx = x+k/2;
  IIIy = y+k/2;
  Δx3 = IIIx-Sx3; //Sx3,Sx3 is source, in this case 0,0
  Δy3 = IIIIy-Sy3;

  //Reach from source to other end of Cross Link, connected to Destination
    Reach(Sx3, Sy3, Δx3, Δy3);

  //Reach from the other end of Destination's Cross link to Destination
    D_Forward = LBD;
  }
 If (x>0 and y<=0) //IV Quadrant
  {
  //Finding other end of link by Backtracking
  IVx = x-k/2;
  IVy = y+k/2;
  Δx4 = IVx-Sx4; //Sx4,Sx4 is source, in this case 0,0
  Δy4 = IVy-Sy4;

  //Reach from source to other end of Cross Link, connected to Destination
    Reach(Sx4, Sy4, Δx4, Δy4);

  //Reach from the other end of Destination's Cross Link to Destination
    D_Forward = LFD;
  }
} //End of LinksE
```

6) Reach Function

//Function to Reach from source to the other end of Cross Link, connected to Destination through a randomized sequence of X and Y traversals.

```
Reach (Sx, Sy, Δx, Δy)
{
 While (Δx == Sx and Δy == Sy)
    {
    r = random(0,1);
    If (r=0)
     {
      If(Δx>0)
         Sx+;
        Else
           Sx-;
     }
    Else
      {
       If(Δy>0)
         Sy+;
        Else
           Sy-;
      }
    }
}
```

REFERENCES

[1] H.J. Siegel and C.B. Stunkel, Inside Parallel Computers: Trends in Interconnection Networks, IEEE Computational Science and Engineering, Vol. 3, No. 3, pp. 69-71, 1996. Invited.

[2] H. Gu, Q. Xie, K. Wang, J. Zhang and Y. Li, X-Torus: A Variation of Torus Topology with Lower Diameter and Larger Bisection Width, ICCSA 2006, LNCS (3984), pp. 149-157, 2006.

[3] Ed. Anderson, J. Brooks, C. Grassl and S. Scott, Performance of the Cray T3E Multiprocessor, In Supercomputing (97), San Jose, California, pp. 1-17, 1997.

[4] C.L. Seitz, The Architecture and Programming of the Ameteck Series 2010 Multicomputer, In Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications (1), pp. 33-36, 1988.

[5] L. Lillevik, The Touchstone 30 Gigaflop DELTA Prototype, Proceedings of the 6th Distributed Memory Computing Conference, pp. 671-677, 1991.

[6] W.J. Dally, Scalable Switching Fabrics for Internet Routers, White paper, Avici Systems Incorporation, 2001.

[7] L.N. Bhuyan (guest editor). Special issue of Interconnection networks. IEEE Computer, Volume 20, Issue 6, DOI: 10.1109/MC.1987.1663580, June 1987.

[8] H.J. Siegel, Interconnection Network for Large Scale Parallel Processing: Theory and Case Studies, McGraw Hill, ISBN 0-07-057561-4, 1990.

[9] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, Tata McGraw-Hill, India, ISBN 0-07-053070-X, 2000.

[10] J. Duato, S. Yalamanchili and L.M. Ni, Interconnection Networks: An Engineering Approach, Morgan Kaufmann, ISBN 1-55860-852-4, 2003.

[11] W. Dally, B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann, San Francisco, ISBN 978-0-12-200751-4, 2004.

[12] H.R. Arabnia and M.A. Oliver, Arbitrary Rotation of Raster Images with SIMD Machine Architectures, International Journal of Eurographics Association (Computer Graphics Forum), Vol. 6, No. 1, pp. 3-12, 1987.

[13] S.M. Bhandarkar, H.R. Arabnia and J.W. Smith, A Reconfigurable Architecture For Image Processing And Computer Vision, International Journal of Pattern Recognition And Artificial Intelligence (special issue on VLSI Algorithms and Architectures for Computer Vision, Image Processing, Pattern Recognition And AI), Vol. 9, No. 2, pp. 201-229, 1995.

[14] S.M. Bhandarkar and H.R. Arabnia, The Hough Transform on a Reconfigurable Multi-Ring Network, Journal of Parallel and Distributed Computing, Vol. 24, No. 1, pp. 107-114, 1995.

[15] M.A. Wani and H.R. Arabnia, Parallel Edge-Region-Based Segmentation Algorithm Targeted at Reconfigurable Multi-Ring Network, The Journal of Supercomputing, Vol. 25, No. 1, pp. 43-63, 2003.

[16] J. Duato. "A New Theory of Deadlock-free Adaptive Routing in Wormhole Networks". IEEE Transactions on Parallel and Distributed Systems, 4(12), pp. 1320-1331,1993.

[17] J. Duato, A Necessary and Sufficient Condition for Dead lock-free Adaptive Routing in Wormhole Networks, IEEE Transactions on Parallel and Distributed Systems, 6(10), pp. 1055-1067,1995.

[18] William J. Dally and Charles L. Seitz, Deadlock-Free Message Routing in Multiprocessor Interconnection Networks, IEEE Transactions on Computers, Vol. C-36, No. 5, pp. 547-553, 1987.

[19] W.J. Dally and H. Aoki, Deadlock-Free Adaptive Routing in Multi computer Networks Using Virtual Channels, IEEE Transactions on Parallel Distributed Systems, Vol. 4, No. 4, pp. 466-475, 1993.

[20] J. Duato, Deadlock-Free Adaptive Routing Algorithms for the 3DTorus: Limitations and Solutions, In Proceedings of Parallel Architectures and Languages Europe 93, 1993.

[21] Nitin and A. Subramanian, Efficient Algorithms to Solve Dynamic MINs Stability Problems using Stable Matching with Complete TIES, Journal of Discrete Algorithms, Vol. 6, No. 3, pp. 353-380, 2008.

[22] A. Singh, W.J. Dally, A.K. Gupta, B. Towels, Adaptive Channel Queue Routing on k-ary n-cubes, Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, 2004.

[23] Nitin and D.S. Chauhan, Stochastic Communication for Application Specific Networks-on-Chip, Journal of Supercomputing, DOI: 10.1007/s11227-010-0472-5, pp. 1-32, 2010.

[24] Nitin, S. Garhwal and N. Srivastava, Designing a Fault-tolerant Fully-chained Combining Switches Multi-stage Interconnection Network with Disjoint Paths, The Journal of Supercomputing, DOI 10.1007/s11227-009-0336-z, pp. 1-32, 2009.

[25] N. Rakesh and Nitin, Analysis of Multi-Sort Algorithm on Multi-Mesh of Trees (MMT) Architecture, Journal of Supercomputing, DOI: 10.1007/s11227-010-0404-4, pp. 1-38, 2010.

[26] Nitin and D.S. Chauhan, Comparative Analysis of Traffic Patterns on *k-ary n-tree* using Adaptive Algorithms based on Burton Normal Form, Journal of Supercomputing, DOI: 10.1007/s11227-010-0454-7, pp. 1-20, 2010.

[27] Nitin, V.K. Sehgal and P.K. Bansal, On MTTF analysis of a Fault-tolerant Hybrid MINs, WSEAS Transactions on Computer Research, ISSN 1991-8755, Vol. 2, No. 2, pp. 130-138, 2007.

[28] Nitin, Component Level Reliability analysis of Fault-tolerant Hybrid MINs, WSEAS Transactions on Computers, ISSN 1109-2750, Vol. 5, No. 9, pp. 1851-1859, 2006.