Active Storage Networks: Topology, Routing and Application

Ajithkumar Thamarakuzhi, Ph.D.

University of Connecticut, 2011

High performance computing systems are often inhibited by the performance of their storage systems and their ability to deliver data. Active Storage Networks (ASN) provide an opportunity to optimize storage system and computational performance by offloading some computation to the network switch. An ASN is based around an intelligent network switch that allows data processing to occur on data as it flows through the storage area network from storage nodes to client nodes. A key design element for an ASN is the switching topology. In this thesis, we present an ASN switching topology named 2-Dilated flattened butterfly (2DFB) which is a nonblocking, low latency, low cost network compared to other nonblocking interconnecting networks. We have implemented this network topology using the NetFPGA as the basic building block of the switching network. The ASN 2DFB architecture has been used in a variety of applications including data sort, data search, data clustering, and min-max.

We have also developed an adaptive load balanced routing scheme (ALDFB) which exploits the topological properties of 2DFB network. ALDFB always gives priority to forwarding packets through the minimal path and therefore, for local and benign traffic the performance of this routing scheme is equal to that of the minimal routing. In adversarial traffic, ALDFB provides better load balance by one non minimal forwarding in each dimension. ALDFB provides high throughput on adversarial traffic patterns and

provides better latency on benign traffic patterns. We have compared the performance of ALDFB on a 2DFB network with non-minimal global adaptive routing (UGAL), Minimal Adaptive and Adaptive Clos routing algorithm for different traffic patterns. We observed that a 2DFB network with ALDFB routing provides high throughput with reduced latency compared to other routing schemes for all the traffic patterns.

Finally, we show how a 2DFB-based ASN can be used to improve parallel file system performance. We have done simulations of striping files and file writes using file locking protocols in a parallel file system. In an ASN we offload some operations from the end-terminals to the ASN switch. In the case of file striping, the splitting of files and the parity calculations are done on the run inside ASN switch. In the file locking case we offload the file locking protocol to the ASN switch. In both cases we observe a significant reduction in traffic through the network and this helps an ASN based parallel file system to offer significant performance improvement.

Active Storage Networks: Topology, Routing and Application

Ajithkumar Thamarakuzhi

B.Tech., Rajiv Gandhi Institute of Technology, 1997

M.Tech., Regional Engineering College, Calicut, 2002

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2011

APPROVAL PAGE

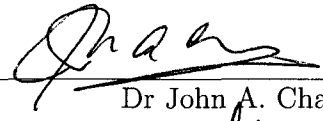
Doctor of Philosophy Dissertation

Active Storage Networks: Topology,

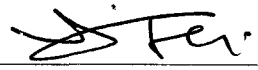
Routing and Application

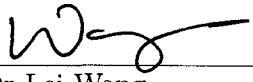
Presented by

Ajithkumar Thamarakuzhi, B.Tech., M.Tech.

Major Advisor 
Dr John A. Chandy

Associate Advisor 
Dr Mohammad H. Tehranipoor

Associate Advisor 
Dr Yunsi Fei

Associate Advisor 
Dr Lei Wang

Associate Advisor 
Dr Faquir Jain

University of Connecticut

2011

To my family

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Professor John A. Chandy for his continued support, understanding and patience during the past four years of my Ph.D study. Without his great support, this work would not have been possible.

Secondly, my special gratitude goes to my fellow lab mates Janardhan Singaraju and Sumit Narayan for the insightful discussions, lab assistance and their support. I also thank Jianwei Dai, Juan Carlos and Ranjith Ramanathan for all their support over the years and making my time at UConn a more enjoyable experience.

Last and most importantly, I would like to thank my parents, my dear wife and coworker Anu, and my wonderful daughter Saathvika for their constant support and encouragement in all my endeavors. I am forever grateful to their love and understanding that has made this thesis happen.

Ajithkumar Thamarakuzhi

University of Connecticut

August, 2011

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Overview

The trend in computer system design over the past few decades has seen microprocessor performance increase by leaps and bounds. However, storage systems have not seen corresponding increases in performance. Recent developments in object-based storage systems and other parallel I/O systems with separate data and control paths have demonstrated an ability to scale aggregate throughput very well for large data transfers. However, there are I/O patterns that do not exhibit strictly parallel characteristics. For example, HPC applications often use reduction operations that funnel multiple data streams from many storage nodes to a single compute node. In addition, many applications, particularly non-scientific applications, use small data transfers that cannot take advantage of existing parallel I/O systems. Present techniques in scalable file systems are approaching their limits because of the above issues. We suggest another approach called active storage networks - namely putting intelligence in the network along with smart storage devices to enhance storage network performance.

These active storage networks can potentially improve not only storage capabilities but also computational performance. In an active storage system, we are trying to provide intelligence to the switching network which connects computing nodes and storage devices. The goal is to do packet processing in the switch to improve storage to network data transfer efficiencies as well as improve computational efficiencies.

Because of the high aggregate throughputs required to build gigabit and multi-gigabit switches, these designs are typically done in silicon. In designing hardware systems, there are several choices for implementation, including coprocessors, FPGAs, and ASICs. We have decided to target the hardware component of this project to reconfigurable FPGAs. Two reasons drive this decision: cost and rapid customizability. ASIC designs are much too expensive from both a design and initial cost point of view. Secondly, FPGAs offer the flexibility to try various options, a point that is key to our research objectives. We have selected the NetFPGA as the basic block of a switching network [1]. The NetFPGA is an experimental board that consists of four Ethernet ports and two SATA ports. It allows us to experiment with new ways to process packets at line rate. The Stanford NetFPGA group has provided designs to use the board as a router or 4-port switch. In addition to that, we have developed an interface for the SATA port to design switching topology for the active storage system.

ASNs are similar in concept to that of active disks. In active disks, computation can be offloaded from the processor to the disk. Previous work has demonstrated the effectiveness of this approach particularly with functions such as storage management, data mining, and multimedia [42]. However, the drawback of active disks in a distributed storage setting is that the data is striped across several storage nodes and each processor at the storage node can only see data residing at that node. Thus, any intelligence at the storage node cannot operate on the entire set of data spread

across storage nodes. For example, when doing a query in a database for the k items closest to a particular key, each of the m storage nodes will return the k closest items in its portion of the data. The requesting client must then sort through mk items to determine the k closest items overall. The overall computation is O(n) + O(mk) where n is the number of data items per storage node.

In an ASN, the goal is to move intelligence to the network which has a better view of data than the individual storage node, thereby optimizing network performance. Processing ability on the network also eases some of the computational workload at the network client. Most of the applications that operate on large sets of data require transforming the data from one form to another. Examples include file compression, video editing and data encoding/decoding applications. Offloading data intensive parts of these applications to the network could ease client computing resources. It could also reduce network traffic as some of the data transfer operations that read and write data from client to the storage can be avoided. This further provides the impetus to embed intelligence in networks.

A critical choice in the design of an ASN switch is the topology of the switch. Some of the important design parameters are the number of interconnects, number of switching elements, overall latency, aggregate bandwidth, and whether it is non-blocking. The number of interconnects/links per switching element and the number of switching elements decide the total cost of the switch/topology network. Since we are using the NetFPGA as the switching element, cost will be primarily decided by the number of NetFPGA boards used. The nonblocking behavior is particularly important to insure that the switch can always deliver the maximum throughput, i.e. with a NxN switch with a per-port bandwidth equal to B, the aggregate throughput should be NB regardless of the connections between ports. We propose a cost efficient nonblocking

switching topology, 2DFB, which is derived from a flattened butterfly network [7]. The 2DFB is a high radix network and the diameter of of this topology is significantly less than other nonblocking topologies. A 2DFB network needs fewer hops for the routing of worst case traffic and therefore it provides excellent performance in terms of latency. We have compared the cost of 2DFB network with other popular nonblocking networks using our cost model. From this cost comparison we verified the reduced implementation cost of 2DFB network.

Another important factor which decides the performance of an ASN switch is the routing protocol. A routing algorithm can be considered as optimal if it provides low latency on local traffic and high throughput on adversarial traffic. Most algorithms must compromise one goal in order to achieve the other. Minimal routing, which always chooses the shortest path for each packet, provides minimum latency for local and benign traffic. However, it provides non acceptable throughput for adversarial traffic due to load imbalance. In order to improve the throughput in adversarial traffic, the routing algorithm should balance the load by sending some fraction of packets over non-minimal paths.

Researchers have been trying to address the issue of providing high worst-case performance while preserving locality. Valiant's randomized algorithm [2] gives good performance in worst case traffic but very poor performance for local traffic in terms of latency. Minimal adaptive routing [3] [4] suffers from global load imbalance. UGAL is an adaptive routing algorithm and it balances the load by doing a proper selection between minimal and non-minimal routing [5]. This selection is done based on the status of the packet queue. In UGAL if channel corresponding to the minimal path is busy, a random intermediate node is selected and packet is routed to and from the intermediate node minimally. Even though the random selection of intermediate node

helps to improve the load balancing it cannot fully avoid channel congestion in a 2DFB. We propose ALDFB, an adaptive load balanced routing scheme designed for 2DFB. It balances the load efficiently by allowing one non-minimal forwarding in each dimension in case of traffic congestion. It senses the traffic congestion from the packet queue. We observed the performance of ALDFB in 2DFB for both local (benign) and adversarial traffic patterns and observed that it outperforms other routing scheme in terms of latency and throughput.

An ASN can provide performance improvements in a vast variety of applications. Any application involving transformation or reduction data operations can be efficiently mapped in to ASN. The operations that we have selected to evaluate on an ASN implementation are file striping and file locking. Large scale data processing is heavily I/O dependent. Data must be retrieved from slow mechanical hard drives and then distributed across faster but still relatively slow (as compared to processors) networks. Parallel file systems provide a way to improve the I/O bandwidth. In this case, large files will be striped across multiple storage servers. In normal file striping, the whole file to be transferred is split and copied to a number of buffers in the client side. Then, the parity is calculated and the content of these buffers and the parity are written to the servers. In ASN based file striping the splitting of the file and parity calculations are performed on the fly inside the ASN switch. In parallel file systems maintaining atomicity is very important because the regions of data in a file are shared by multiple processes. Most of the solutions used for maintaining atomicity use some form of file locking. The usage of a scalable distributed lock manager (DLM) architecture [6] can be considered as an efficient way to maintain the atomicity. However, these locking protocols introduce additional traffic in the network and this can affect the overall performance. We propose offloading these file locking protocols from the lock servers

to the ASN switch. In both of the selected applications, we can reduce much of the traffic in the network, thereby improving the overall performance of the parallel file system.

1.2 Thesis Contributions

The contributions of this dissertation can be split in to three parts.

- Design and implementation of an ASN switching topology

We have designed a non-blocking version of a k-way bristled k-ary generalized hypercube which is named 2DFB. In addition, we have derived the equations to determine the dilation factor and number of channels required in the last dimension and presented a cost model and compared the cost of different non-blocking topologies with 2DFB. We have developed a static conflict free routing schedule for 2DFB. We have implemented an 8-port 2DFB network using the NetFPGA hardware platform

- Design and implementation of a new routing scheme (ALDFB) for an ASN

We have introduced a deadlock free, adaptive, load balanced routing algorithm called ALDFB for a 2DFB switching network. ALDFB is designed to exploit all positive topological properties of a 2DFB network. The algorithm takes full advantage of the reduced diameter and improved path diversity of 2DFB network. It provides better load balancing by allowing one non-minimal forwarding in each single dimension of 2DFB network. We have observed that ALDFB provides better throughput and reduced latency compared to other well-known routing schemes for all the traffic patterns that we used for the simulation.

- Implementation of two applications over ASN network

We have implemented two applications-file striping with parity and file locking-in a 2DFB based ASN network and compared its performance with existing parallel

file system counterparts. These implementations are done using the Omnet++ simulation platform. We observe that in both the applications ASN provides significant performance improvement.

1.3 Outline

This dissertation is organized as follows. In Chapter 2 we describe about the proposed ASN switching topology which is named as 2 dilated flattened butterfly (2DFB). In Chapter 3, we provide the details of implementing 2DFB using NetFPGA. The proposed deadlock free load balanced routing scheme for the 2DFB network (ALDFB) is explained in chapter 4. In Chapter 5 we analyze 2DFB based on-chip network. The selected ASN applications and its performance on an ASN network are given in Chapter 6. Finally Chapter 7 concludes this dissertation.

Chapter 2

2-Dilated Flattened Butterfly (2DFB)

2.1 Overview

High performance computing on distributed memory parallel processing systems such as clusters are very dependent on communications between processing nodes. As a result, the interconnection network that connects these nodes is a critical part of the performance of the system. For the past few decades, we have seen improving performance of processors and memory systems. In order to keep up with these gains, the network switch performance must also improve. The study of interconnection networks has a long history and a large number of network topologies have been studied by researchers. Among these networks, hypercube [8] and Clos [9] (or its derivatives) are the most popular networks.

The technological progress in modern ASICs has led to the availability of routers with high bandwidth in the range of Tb/s. This is achieved because of the increase in the signaling rate as well as the increase in the number of signals available to a router chip. The radix of a router is defined as the number of terminals connected to the

router. The use of high radix (and thus high bandwidth) routers reduces the hop count and leads to lower latency and lower cost. The Cray BlackWidow vector multiprocessor, which uses a radix-64 router, is an example of such a system [11]. It is estimated that by the end of 2011, the optimal radix will be approximately 256 [12]. To take full advantage of these high radix routers, a cost-efficient topology known as flattened butterfly [7] has been proposed. A flattened butterfly is derived from a butterfly network [13], and is generated by combining or flattening the routers in each row of a k-ary butterfly into a single router. Attractive features of this topology are its inherent path diversity and reduced number of links compared to other networks which have the same bisection bandwidth. The inherent path diversity of the flattened butterfly is utilized to achieve comparable throughput performance with the Clos network in adversarial traffic and the reduced number of links reduces the cost of the network.

Even though the flattened butterfly has good path diversity, the reduced number of links prevents it from being nonblocking if all the terminals offer full load to the network. A nonblocking network is a topology in which all nodes will achieve full bandwidth regardless of the traffic. Like the butterfly, the flattened butterfly is also a blocking network. This blocking behavior can cause an unacceptable switching delay or packet loss in some applications. The 2DFB can be considered as the nonblocking version of the flattened butterfly network. We observe that a 2DFB network outperforms other nonblocking topologies like folded-Clos and hypercube in terms of speed and implementation cost.

2.2 Background

The interconnection network and network topology play an important role in deciding the overall performance of any networking system. Some of the important design

parameters of a network topology are the number of interconnects, overall latency, aggregate bandwidth, and whether it is non-blocking. The latter is particularly important to insure that the switch can always deliver the maximum throughput, i.e. with a NxN switch with a per port bandwidth equal to B, the aggregate throughput should be NB regardless of the connections between ports. The simplest choice for building the switch is a shared bus architecture where each of the nodes are connected to a bus. Current generation PCI buses have a throughput of roughly 8 Gb/s which can barely handle a small 8x8 gigabit cluster. Thus, the bus solution is not scalable, and moreover, it is not non-blocking since only transaction can take place at a time on the bus. The simplest non-blocking interconnection network is the matrix or crossbar where all nodes are connected to all other nodes. However, the cost in terms of connections grows by n^2 making it impractical for large networks. Torus networks are another approach to interconnect nodes in a high performance network. Multistage switching networks such as the Benes network and their derivatives are attractive because of their limited switch points [10]. As you build larger and larger switches these switches become more complex, and a simple hierarchical set of smaller switches is no longer non-blocking. Instead of the hierarchical approach, most larger switches are built from smaller switches using variations of the fat-tree or Clos network [9]. In this section we describe three closely related topologies flattened butterfly [7], multiring [16] and bristled hypercube [29].

2.2.1 Flattened butterfly

A flattened butterfly is derived from a k-ary n-fly butterfly structure by flattening the routers in each row of the network into a single router [7]. A flattened butterfly

is composed of N/k routers of radix k'=$n(k-1)+1$ where N is the number of end-terminals in the network, k is the number of end-terminals connected to each router and the radix is the number of external ports associated with each router.

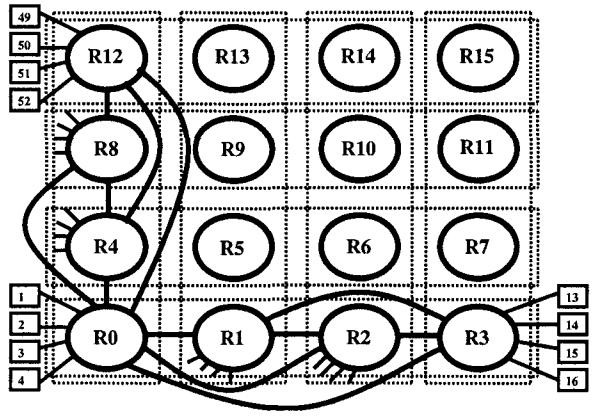


Figure 1: 4-ary 2-dimensional flattened butterfly structure

The routers are connected by channels in $n' = n-1$ dimensions, corresponding to the $n-1$ columns of inter-rank wiring in the butterfly. In each dimension d, from 1 to n', router i is connected to each router j given by

$$j = i + [m - (\lfloor \frac{i}{k^{d-1}} \rfloor \mod k)]k^{d-1} \quad (1)$$

for m from 0 to $k-1$, where the connection from i to itself is omitted. For example, a 4-ary 2-dimensional flattened butterfly for N=64 is shown in Figure 1. Each switching element is connected to k end-terminals (here k=4). All terminals are not shown in Figure 1. Only the interconnections between the routers in the first column and bottom row are shown. The interconnections between the routers in the rest of the columns are similar to the first column and the rest of the rows are similar to the bottom row.

We now consider a worst case traffic load for this network. Assume that all the end-terminals connected to the routers of first two columns are transmitting data to the end-terminals connected to the routers in the last two columns, and all the transmitting

terminals are transmitting data at their full bandwidth. The total data rate in one direction is $32b$ Gb/s where b Gb/s is the full bandwidth of each end-terminal. Assume that the bandwidth of all the channels are the same (b Gb/s). Nonblocking data flow is achieved when no channel is loaded with a data rate greater than b Gb/s. This type of overloading occurs if data from two input channels are directed to a single output channel. Since the bisection bandwidth of the flattened butterfly network in Figure 1 is $16b$ Gb/s, there is no routing schedule that can avoid overloading a single channel in worst case traffic. Therefore, it is clear that like the butterfly, the flattened butterfly is also a blocking structure.

2.2.2 Multiring

The multiring is a well-studied switching topology [17–21]. This network consists of $m \geq 2$ ring channels with different sequential connections of nodes. Each node which represents a switching element is connected to a single end-terminal and these nodes are numbered as 0,1,...,$(N-1)$ where N is the total number of nodes. The sequence of connections of nodes in a unidirectional ring along the direction of transfer is defined by the numbers $X_i^{(j)} \in [0, N-1], i = 0, 1, ...; 1 \leq j \leq m$, and $X_{i+1}^{(j)} = (X_i^{(j)} + S^{(j)})$ mod $N, 1 \leq S^{(j)} \leq N-1; 0 \leq X_0^{(j)} \leq S^{(j)} - 1$, where $S^{(j)}$ is called the step of the jth ring. A multiring structure of three rings and eight nodes is shown in Figure 2. The steps of these three rings are 1, 2 and 4 respectively.

A multiring with a set of rings $S_m = \{S^{(k)}\}_{k=1,,m}$ is said to be a simple p-ring if $S^{(k)} = ip^j$, where $k = i + (p-1)j, p \geq 2, 1 \leq \imath \leq p-1, 0 \leq j \leq \lfloor \log_p N \rfloor$. For example, the network shown in Figure 2 is a simple p-ring where $m = 3$ and $p = 2$. Packets can be routed from any source node to any destination node through these set of rings of

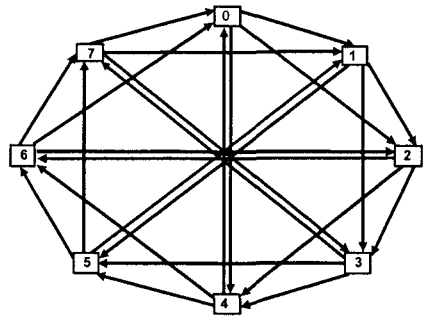


Figure 2: Multiring of three rings and eight nodes

different step size. The length d of any route can be decomposed in base p as

$$d = \sum_{i=0}^{m_0-1} d_i p^i, \tag{2}$$

where $0 \leq |d_i| \leq p-1, m_0 \leq \lceil \log_p N \rceil$

A multiring structure and flattened butterfly structure are closely related. If r is the dimension of a k-ary flattened butterfly, then there will be k^r nodes (switching elements) in the system and each node can be represented using a r-digit number, i.e. any node $x = x_{r-1}...x_i...x_0$ where $x_i \in [0, k-1]$ and $x = \sum_{i=0}^{r-1} x_i k^i$. In a flattened butterfly, any two nodes, whose numbers differ only in the ith digit, are joined by a duplex channel and it is known as the ith dimension channel. Every node contains $(k-1)$ channels in each dimension and each of them are assigned with formal length $jk^i (1 \leq j \leq k-1, 1 \leq i \leq r)$. Thus, like a multiring, a flattened butterfly also can be characterized by a set of channel lengths $S_m = \{S^{(k)}\}_{k=1,,m}$, where $S^{(1)} \leq S^{(2)} \leq ... \leq S^{(m)}$ and $S^{(j+(k-1)i)} = jk^i (1 \leq j \leq k-1, 0 \leq i \leq r), m = r(k-1)$ and the length of any route can be decomposed in base k as

$$d = \sum_{i=0}^{r-1} d_i k^i \tag{3}$$

An ith dimension channel of length $d_i k^i$ is said to join nodes with numbers containing the values y_i and x_i at the ith digit if $(x_i + d_i) \mod k = y_i$. The main difference between a multiring and a flattened butterfly is that in a multiring, each switching element is connected to a single end-terminal whereas in a k-ary flattened butterfly each switching element is connected to k end-terminals. Thus, like a hypercube, multiring also need low-radix routers for implementation and it have all the drawbacks (higher diameter, higher hop count etc.) of a low-radix network.

2.2.3 k-dilated k-way bristled hypercube

Hypercube networks have been used extensively in parallel computing systems. Researchers have already developed oblivious routing algorithms for conflict-free routing in a hypercube using minimal distance ($\log_2 N$) [30]. Therefore, a hypercube can be considered as a nonblocking switching network. Since only one end-terminal is connected to each switching element, a hypercube cannot take advantage of a high-radix network and the cost will become prohibitively large for larger network size. *Bristling* is the solution for this, where k end-terminals are connected to each switching element in a k-way bristled hypercube [29]. On the other hand, a k-way bristled hypercube loses its nonblocking nature, because its bisection bandwidth will be decreased by a factor k. A k-way bristled hypercube will show nonblocking behavior if we dilate each interconnection link by a factor k, which can be called a dilated bristled hypercube (DBHC). The bisection bandwidth of a DBHC will be equal to that of a hypercube. The structure of a 32-terminal DBHC is shown in Figure 3. DBHC has reduced hop count, improved latency and reduced implementation cost compared to a normal hypercube while maintaining the nonblocking property.

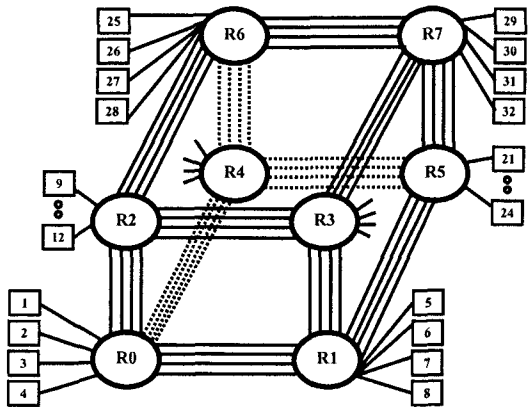


Figure 3: 4-dilated 4-way bristled hypercube

2.3 2-dilated flattened butterfly

A 2-dilated flattened butterfly (2DFB) is derived from a flattened butterfly by either duplicating all the interconnecting links between the switching elements in the flattened butterfly or replacing it with links of double bandwidth. Links between the end-terminals and switching elements remain the same. Figure 4 shows a 4-ary one-dimensional 2DFB structure where the total number of end-terminals $N = 16$. In the figure, a full duplex link is shown as two directed links with opposite direction.

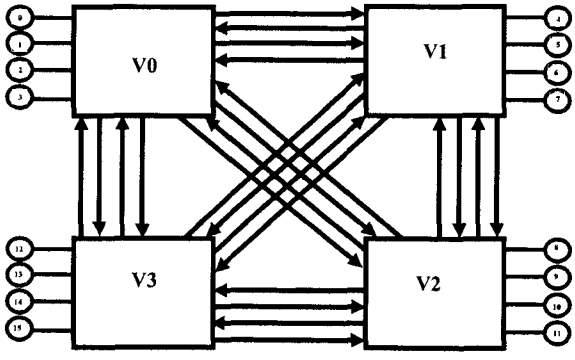


Figure 4: 4-ary 2-dilated flattened butterfly

There will be two separate channels for each channel length which can be represented as $S_A^{(i)}$ and $S_B^{(i)}$. The set of channel lengths can be represented as $S_m = \{S_A^{(1)}, S_B^{(1)}, S_A^{(2)}, S_B^{(2)}, ..., S_A^{(m)}, S_B^{(m)}\}$. In other words we can say that for a switching element there will be $2(k-1)$ links in each dimension if the network size (N) is a power of k. The total number of links associated with each switching element (radix) of a k-ary 2DFB is $k + (2(k-1)(\log_k N - 1))$. If we analyze a flattened butterfly (Figure 1) we can see that it is a k-way bristled generalized hypercube with a base k [8]. Thus, a 2DFB is simply a 2-dilated k-way bristled generalized hypercube with a base k. In a 2DFB, instead of physically duplicating each interconnecting channel, we can double the bandwidth of each channel which gives the same performance. So in our 2DFB hardware implementation we will be using interconnecting channels having double the bandwidth than that of the channels which are connected to the end-terminals.

2.3.1 Nonblocking property

A multiring shows nonblocking nature if it has $2(\sqrt{N} - 1)$ ring channels. In this case, there exists a conflict-free routing schedule which need only 2 hops to perform any routing permutation [17]. The total number of channel length elements in S_m of a k-ary 1-dimensional 2DFB structure with N end-terminals is $2(k-1)$ where $k = \sqrt{N}$ (see Figure 4). From this it is clear that 1-dimensional 2DFB is nonblocking for any value of k and any routing permutation can be performed by making use of a maximum of 2 hops. Higher dimensional 2DFB systems are constructed by combining one dimensional systems as done in a hypercube system. Nonblocking behavior of a hypercube network of any size and any dimension is shown in [30]. This nonblocking property is achieved because of the inherent bisection bandwidth of hypercube network. The bisection bandwidth of hypercube network is $N/2$, irrespective of its size or dimension. Let us

consider a k-ary 1-dimensional 2DFB with maximum number of end-terminals ($N = k^2$). The bisection bandwidth of this network is $2 \times (k/2) \times (k/2)$. That is the bisection bandwidth of a k-ary 1-dimensional 2DFB with maximum number of end-terminals is $N/2$. A 2-dimensional 2DFB system with a network size $N = k^3$ is implemented by interconnecting k number of 1-dimensional 2DFB system. That is the bisection bandwidth of a 2-dimensional 2DFB system is $k \times (k^2/2)$. So the bisection bandwidth of a 2-dimensional 2DFB system is $N/2$. For a k-ary d-dimensional (d=($\log_k N$)-1) 2DFB system with a network size N of power of k, the bisection bandwidth is $((k^2/2)(k^{d-1}))$ which is equal to $N/2$ where $N = k^{(d+1)}$ (same as that of a hypercube network). Therefore, a properly designed routing algorithm can route any permutation without conflict by making use of a maximum of $2d$ hops (2 hops in each dimension).

A flattened butterfly with a dilation factor of 2 can show nonblocking nature only when the network size is a power of k. Now, consider a 2DFB network with a network size which is not a power of k but a multiple of $k^{(\lfloor \log_k N \rfloor)}$. In this case, the number of switching elements in the last dimension will be less than k and the bisection bandwidth will be less than $N/2$. This network cannot be nonblocking. In order to make it nonblocking, we must increase the dilation factor to the links in the last dimension while maintaining the dilation factor of other links as 2. Let N_s be the number of subsystems with dimension $d-1$ in a N end-terminal d-dimensional 2DFB system where N is not a power of k.

$$N_s = N/(k^{\lfloor \log_k N \rfloor}) \tag{4}$$

Let N_{ld} be the number of links crossing the bisection of the last dimension,

$$N_{ld} = ((N_s/2)^2) \times k^{\lfloor \log_k N \rfloor - 1} \tag{5}$$

Let DF_d be the dilation factor of the links in the last dimension. Consider the bisection bandwidth along the last dimension (BW_d),

$$BW_d = N_{ld} \times DF_d \tag{6}$$

For nonblocking BW_d should be equal to $N/2$ and by using this condition we can find the dilation factor required to the links in the last dimension DF_d.

$$DF_d = (N/2)/(N_{ld}) \tag{7}$$

After substituting the value of N_{ld} in Equation 7,

$$DF_d = (2k^{(\lfloor \log_k N \rfloor + 1)})/N \tag{8}$$

The number of ports required for the connections in the last dimension,

$$Ports_d = (N_s - 1) \times \lceil DF_d \rceil \tag{9}$$

The number of ports required for the connections in all other dimensions remain the same $(2(k-1))$. For example consider a 4-ary 2DFB network with a network size of 32. Using Equation 8 the dilation factor is 4 and using Equation 9 the number of ports in the second dimension is 4. Thus, we can see that in this case two switching elements in the second dimension will be connected with four links.

2.3.2 Conflict-free static routing schedule

In this section we propose a procedure to construct a nonblocking static routing schedule for one dimensional and higher dimensional 2DFB networks. A static schedule can be represented by a matrix with N rows and n columns, where N is the network size and n is the number of cycles required to complete the routing. One cycle is considered as the total time required to pass one packet from one node to its neighbor

node. Each row of the matrix is a routing schedule T_d for a routing length d which can be represented as $(t_{d,1}, .., t_{d,j}, .., t_{d,n})$ in which the jth element defines the length covered by the step in the jth cycle, i.e., $t_{d,j} \in S_m$ and $t_{d,j} \in (d_0 k^0, ..., d_{r-1} k^{r-1})$ where r is the dimension of the network and the schedule satisfies the conditions $\sum_{j=1}^{n} t_{d,j} = d, t_{d,j} \in S_m \cup \{0\}$ for $1 \leq j \leq n$, and if $t_{d,j} \neq 0$ and $S_{d,j}$ is the corresponding channel length element, then $S_{d,j} \neq S_{d,i}$ for $1 \leq i \leq n$ and $i \neq j$.

In a k-ary 2DFB, since k end-terminals are connected to each switching node, there can be at most k routing requests for the same routing length. Therefore, there will be k routing schedules for each routing length d. The routing schedule is represented as $T_d^{(p)}$ where d is the routing length and p can vary between 1 to k. The static schedule matrix can be represented as

$$T(P,m,n) = \begin{pmatrix} T_0^{(1)} \\ T_0^{(2)} \\ .... \\ T_0^{(k)} \\ .... \\ T_{P-1}^{(1)} \\ T_{P-1}^{(2)} \\ .... \\ T_{P-1}^{(k)} \end{pmatrix} \tag{10}$$

where $P = N/k$, m is the total number of elements in S_m and n is the number of cycles required to complete the routing.

It is shown that for a multiring structure, if there exists a static schedule $T(P,m,n)$ where from the equality of the nonzero elements of any route schedules $t_{d_1,j}$ and $t_{d_2,j}$

in any cycle j and $t_{d_1,j} \neq 0$ and if the initial or final parts of these schedules are equal, then it follows that this schedule is conflictless [16]. If we construct a schedule with equality in the initial part, then that schedule is called the initial schedule $B(P, m, n)$ and if equality is only in the final part then it is called the final schedule $F(P, m, n)$. In an initial schedule if there exists two rows (schedules) with equal nonzero elements in the same cycle, then the initial part of both the rows will be same. Conflictless schedules for a 2DFB also should satisfy the above constraint. Along with this we have to make sure that in all k schedules for the same routing distance any channel length element should not appear more than once. For a 2DFB if there exists a schedule which satisfies the above two constraints it will be a conflictless schedule. With static scheduling of a multiring, the total routing distance is decomposed in base p and each step size is traversed in each cycle. From the equivalence of Equations 2 and 3, it is clear that with a 2DFB the routing can be also scheduled exactly the same way as with a multiring. Therefore, a conflict-free schedule for a multiring can also provide conflict freedom for a 2DFB. An example of a conflictless schedule for a 4-ary one dimensional 2DFB with $N = 16$ (Figure 4) is shown below.

$$T(4,6,2) = \begin{pmatrix} T_1^{(1)} \\ T_1^{(2)} \\ T_1^{(3)} \\ T_1^{(4)} \\ T_2^{(1)} \\ T_2^{(2)} \\ T_2^{(3)} \\ T_2^{(4)} \\ T_3^{(1)} \\ T_3^{(2)} \\ T_3^{(3)} \\ T_3^{(4)} \end{pmatrix} = \begin{pmatrix} S_A^{(1)} & 0 \\ S_B^{(1)} & 0 \\ S_A^{(2)} & S_A^{(3)} \\ S_B^{(3)} & S_B^{(2)} \\ S_A^{(2)} & 0 \\ S_B^{(2)} & 0 \\ S_B^{(1)} & S_A^{(1)} \\ S_A^{(3)} & S_B^{(3)} \\ S_A^{(3)} & 0 \\ S_B^{(3)} & 0 \\ S_A^{(1)} & S_A^{(2)} \\ S_B^{(2)} & S_B^{(1)} \end{pmatrix} \tag{11}$$

Consider that end-terminals 0, 1, 2 and 3 in Figure 4 are sending messages to end-terminals 4, 5, 6 and 7 respectively. In this case the routing distance is 1 for all the routing and schedules $T_1^{(1)}$, $T_1^{(2)}$, $T_1^{(3)}$ and $T_1^{(4)}$ are assigned to each end-terminal. If $T_1^{(1)}$ is assigned to end-terminal 0, then the step size selected is $S_A^{(1)}$ and all the packets are directed to the first channel between V_0 and V_1. Assume $T_1^{(3)}$ is assigned to end-terminal 2, then the step sizes selected are $S_A^{(2)}$ and $S_A^{(3)}$. In this case all the packets from end-terminal 2 are directed to the first channel between V_0 and V_2 and then from V_2 all are directed to the first channel between V_2 and V_1.

All the elements corresponding to the routing length 0 will be 0. Therefore, they are not shown in the matrix. In this manner, one can easily determine a conflictless schedule for one dimensional 2DFB network for any value of k. Since one node has k choices,

for the same length there should be a selecting mechanism associated to each node. For example, if a single node has two routing requests with length 1 and 3 respectively, then it cannot select schedules $T_1^{(1)}$ and $T_3^{(3)}$ because there will be a conflict in the channel $S_A^{(1)}$. So the selecting mechanism should select conflictless schedules which are always available. In this example, the selector can select $T_1^{(1)}$ and $T_3^{(1)}$.

Conflictless schedules for higher dimensional networks can be derived from the schedule of a one dimensional network with the help of the following theorem which is proved in [16].

$From the initial B(P_1, m_1, n_1) and final F(P_2, m_2, n_2) schedule one can construct a conflictless static s$
$m_2, n_1 + n_2)$.

This theorem claims the existence of a $n_1 + n_2$ cycle conflict free static schedule for a multiring with a network size of $P_1 \times P_2$ and $m_1 + m_2$ rings, if we know the n_1 cycle initial schedule of a multiring with a network size of P_1 and m_1 rings and n_2 cycle final schedule of another multiring with a network size of P_2 and m_2 rings. Even though this theorem is derived for a multiring structure, the same procedure can be used to construct conflict-free schedule for higher dimensional 2DFB network. Consider an example of constructing conflict-free schedules for a two-dimensional 4-ary 2DFB from the schedule of its one dimensional network. Any routing length in a 2-dimensional network can be decomposed to find the routing length in each dimension. Consider a routing length of 11, which can be decomposed into two dimensions using Equation 3 as $11 = 3 \times 4^0 + 2 \times 4^1$. This indicates that the routing can be split into two routing lengths, i.e. 3 in the first dimension and a routing length of 2 in the second dimension. The routing length in the first dimension is implemented using the initial schedule and the routing length in the second dimension is implemented using the final schedule of the one dimensional 2DFB. Thus, all the schedules corresponding

to 16 routing length of a 4-ary two dimensional 2DFB can be split into two parts such that one part should be the initial schedule and the other part should be the final schedule of the corresponding one dimensional schedule. The schedule of a 4-ary one dimensional network, which is given in Equation 11, can be considered as an initial schedule $B(4,6,2)$. Its final schedule is obtained by simply taking the mirror image of each row element which is as shown below.

$$F(4,6,2) = \begin{pmatrix} T_1^{(1)} \\ T_1^{(2)} \\ T_1^{(3)} \\ T_1^{(4)} \\ T_2^{(1)} \\ T_2^{(2)} \\ T_2^{(3)} \\ T_2^{(4)} \\ T_3^{(1)} \\ T_3^{(2)} \\ T_3^{(3)} \\ T_3^{(4)} \end{pmatrix} = \begin{pmatrix} 0 & S_A^{(1)} \\ 0 & S_B^{(1)} \\ S_A^{(3)} & S_A^{(2)} \\ S_B^{(2)} & S_B^{(3)} \\ 0 & S_A^{(2)} \\ 0 & S_B^{(2)} \\ S_A^{(1)} & S_B^{(1)} \\ S_B^{(3)} & S_A^{(3)} \\ 0 & S_A^{(3)} \\ 0 & S_B^{(3)} \\ S_A^{(2)} & S_A^{(1)} \\ S_B^{(1)} & S_B^{(2)} \end{pmatrix} \tag{12}$$

This final schedule is used to implement the routing length in the second dimension by replacing j in the $S_A^{(j)}$ and $S_B^{(j)}$ by $j+3$. Recall the relation $S^{(j+(k-1)i)} = jk^i (1 \leq j \leq k-1, 0 \leq i \leq r)$ and here the value of $k = 4$ and $i = 1$. So the four routing schedule for a routing length of 11 in a 4-ary two-dimensional network can be derived by making use of the $T_3^{(j)}$ and $T_2^{(j)}$ schedules of the 4-ary one dimensional 2DFB where

$1 \leq j \leq 4$. The conflict-free routing schedule of three routing length 11, 12 and 13 of a two dimensional 2DFB is shown below.

$$\begin{pmatrix} T_{11}^{(1)} \\ T_{11}^{(2)} \\ T_{11}^{(3)} \\ T_{11}^{(4)} \\ T_{12}^{(1)} \\ T_{12}^{(2)} \\ T_{12}^{(3)} \\ T_{12}^{(4)} \\ T_{13}^{(1)} \\ T_{13}^{(2)} \\ T_{13}^{(3)} \\ T_{13}^{(4)} \end{pmatrix} = \begin{pmatrix} S_A^{(3)} & 0 & 0 & S_A^{(5)} \\ S_B^{(3)} & 0 & 0 & S_B^{(5)} \\ S_A^{(1)} & S_A^{(2)} & S_A^{(4)} & S_B^{(4)} \\ S_B^{(2)} & S_B^{(1)} & S_B^{(6)} & S_A^{(6)} \\ 0 & 0 & 0 & S_A^{(6)} \\ 0 & 0 & 0 & S_B^{(6)} \\ 0 & 0 & S_A^{(5)} & S_A^{(4)} \\ 0 & 0 & S_B^{(4)} & S_B^{(5)} \\ S_A^{(1)} & 0 & 0 & S_A^{(6)} \\ S_B^{(1)} & 0 & 0 & S_B^{(6)} \\ S_A^{(2)} & S_A^{(3)} & S_A^{(5)} & S_A^{(4)} \\ S_B^{(3)} & S_B^{(2)} & S_B^{(4)} & S_B^{(5)} \\ S_A^{(2)} & 0 & 0 & S_A^{(6)} \\ S_B^{(2)} & 0 & 0 & S_B^{(6)} \\ S_B^{(1)} & S_A^{(1)} & S_A^{(5)} & S_A^{(4)} \\ S_A^{(3)} & S_B^{(3)} & S_B^{(4)} & S_B^{(5)} \end{pmatrix} \tag{13}$$

The above mentioned procedure can be used to construct the conflict-free routing schedule of any higher dimensional network if we know the initial and final schedule of the corresponding lower dimensional networks.

2.4 Comparison Results and Discussions

2.4.1 Network diameter

Network diameter is a measure of the shortest distance between the source and destination nodes. Since high priority traffic can be routed through this shortest path, the network diameter plays an important role in an interconnecting network. We have compared the diameter of a 32-ary 2DFB network with folded-Clos, DBHC, and hypercube. The radix of the router in a 2DFB is a function of the number of end-terminals connected to each switching element (k) and the network size (N). For the comparison, folded-Clos and DBHC are implemented using routers with the same radix as that required to implement a 2DFB network, for a fixed network size. In a hypercube, the value of k is always one, and therefore, the radix of the router will be always less than other topologies. The radix of the router and corresponding value of k for different topologies with different network size used for the comparison are shown in Table 1. In the 32-ary 2DFB network, since the network sizes 256 and 16384 are not powers of 32, Equation 9 is used to find the number of ports in the last dimension.

The diameter of a hypercube network is

$$Diameter_{hypercube} = \lceil \log_2 N \rceil \tag{14}$$

The diameter of a DBHC network is

$$Diameter_{DBHC} = \lceil \log_2 (N/k) \rceil \tag{15}$$

The diameter of a k-ary folded-Clos network is

$$Diameter_{folded-Clos} = 2(\lceil (\log_k N) \rceil - 1) \tag{16}$$

The diameter of a k-ary 2DFB network is

$$Diameter_{2DFB} = \lceil (\log_k N) \rceil - 1 \tag{17}$$

Table 1 Diameter comparison - where 'r' is radix of the router, 'k' is ary and 'd' is the diameter

Network topology	N=256			N=1 024			N=16 384			N=32 768		
	r	k	d	r	k	d	r	k	d	r	k	d
2DFB	88	32	1	94	32	1	154	32	2	156	32	2
folded-Clos	88	44	3	94	47	3	154	77	5	156	78	5
DBHC	88	17	4	94	11	7	154	12	11	156	12	12
Hypercube	9	1	8	11	1	10	15	1	14	16	1	15

Table 1 also compares the network diameter of 2DFB with other topologies for different sized networks This diameter comparison is depicted in Figure 5 As we can observe, 2DFB has the smallest network diameter compared to other network topologies

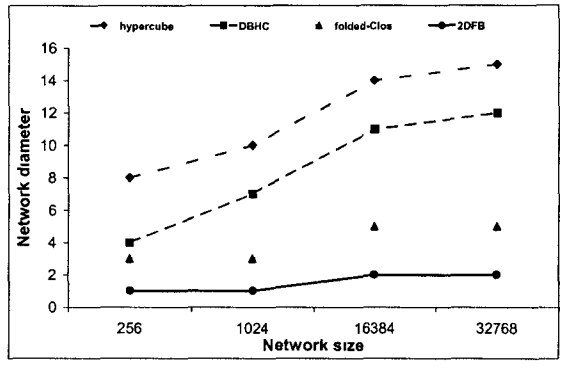


Figure 5 Network diameter

2.4.2 Switching element complexity of the network

The switching element complexity of a network is defined as the product of the total number of switching elements in the network and the number of cross points in each switching element Remember that the number of cross points in a switching element is

the square of the radix. We have compared the switching element complexity of a 32-ary 2DFB network with folded-Clos and DBHC. All these networks are implemented with switching elements of the same radix. This switching element complexity comparison is shown in Figure 6. Since all the topologies shown in Figure 6 are implemented using switching elements of same radix, switching element complexity of the network can also be considered as a measure of total number of switching elements required to implement the network.

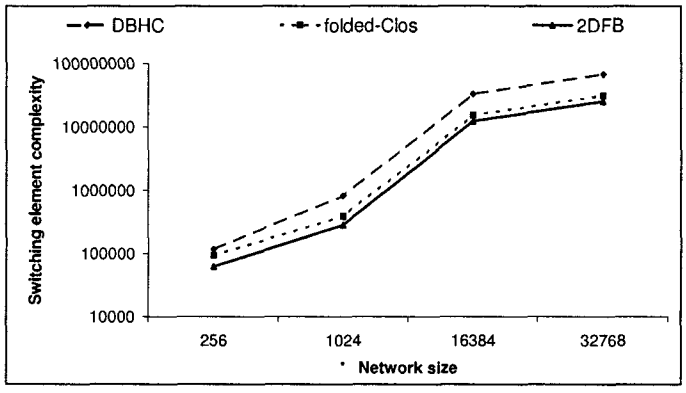


Figure 6: Switching element complexity of the network

The number of switching elements required in a DBHC network is

$$SE_{DBHC} = \lceil N/k \rceil \tag{18}$$

The number of switching elements required in a k-ary folded-Clos network is

$$SE_{folded-Clos} = (\lceil N/k \rceil)(\lceil \log_k N \rceil) \tag{19}$$

The number of switching elements required in a k-ary 2DFB network is

$$SE_{2DFB} = \lceil N/k \rceil \tag{20}$$

The value of k used for these topologies is shown in Table 1. As we can observe in Figure 6, 2DFB has the smallest network complexity compared to the other high-radix network topologies for the given range of network sizes.

2.4.3 Link complexity

Link complexity is defined as the ratio of the total number of links between the switching elements in the network and the network size. The cost of a switching network is mainly decided by the network complexity and the link complexity. The link complexity is a measure of the link cost. We have compared the link complexity of a 32-ary 2DFB network with folded-Clos and DBHC. All these networks are implemented with switching elements of the same radix. In this analysis we assume that all the links have the same bandwidth.

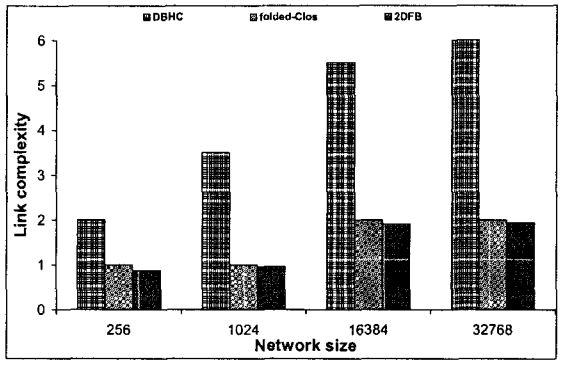


Figure 7: Link complexity

The total number of links in a folded-Clos is

$$Links_{folded\_Clos} = N(\lceil \log_k N \rceil - 1) \tag{21}$$

The total number of links in a 2DFB, if the network size is a power of k is

$$Links_{2DFB} = (N/k)(k-1)(\lceil \log_k N \rceil - 1) \tag{22}$$

If the network size is not a power of k and is a multiple of $k^{(\lfloor \log_k N \rfloor)}$ then the number of links in a 2DFB can be find out with the help of Equation 9. Total number of ports

in a DBHC is

$$Links_{DBHC} = (N/2)(\lceil \log_2(N/k) \rceil) \tag{23}$$

The value of k and radix used for these topologies for different network sizes are shown in Table 1. The link complexity comparison is shown in Figure 7. As we can observe in Figure 7, 2DFB has the smallest link complexity compared to the other high-radix network topologies.

2.4.4 Speed analysis

Message latency in a nonblocking network is proportional to the number of hops traveled during the routing process. Figure 8 represents the number of hops needed for routing the message for different high-radix network topologies with varying network sizes. The networks we are using for the comparison are 32-ary 2DFB, folded-Clos and DBHC, all implemented with routers of same radix as shown in Table 1. The number of hops required in a 2DFB is not the same as the network diameter for all source-destination pairs. For example, in Figure 4, if end-terminals 0, 1, 2 and 3 are sending messages to end-terminals 4, 5, 6 and 7 respectively with full bandwidth, then only messages from terminal 0 and 1 can be routed through the direct link between V_0 and V_1 and the messages from 2 and 3 should be routed through V_2 or V_3. In this case the number of hops required in the worst case is 2. For any one dimension 2DFB in which $k > 2$, the worst case number of hops for routing any message is 2. With a higher dimension 2DFB in which $k > 2$, in the worst case, 2 hops are required for routing the message in each dimension. The dimension of a k-ary 2DFB is $\lceil (\log_k N) \rceil - 1$. So the number of hops required (worst case) to complete any routing request in a k-ary 2DFB for $k > 2$ is $2\{\lceil (\log_k N) \rceil - 1\}$.

For $k = 2$, 2DFB becomes a 2-dilated 2-way bristled hypercube (DBHC). In this case the number of hops required is $\log_2(N/2)$. The number of hops required for the routing of any message in a DBHC is $\log_2(N/k)$. The number of hops required for a k-ary folded-Clos network is $2\{\lceil(\log_k N)\rceil - 1\}$. From the comparison we can notice that the number of hops required for a k-ary 2DFB in worst case is same as that of a k-ary folded-Clos network. Unlike folded-Clos, in 2DFB the number of required hops is not the same for all source-destination pairs. Most source-destination pairs need only one hop to traverse in one dimension. Therefore, the average number of hops in a 2DFB will be always less than that of the corresponding folded-Clos network. Thus, it is clear that a k-ary 2DFB provides better message latency than the corresponding folded-Clos network.

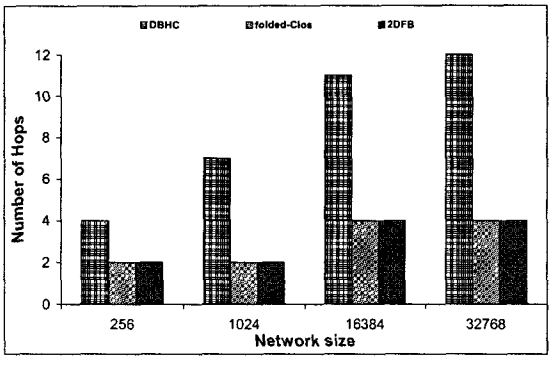


Figure 8: Speed comparison

DBHC, folded-Clos, and 2DFB networks are nonblocking, and proper routing algorithms can route all the messages without conflict by making use of the number of hops shown in Figure 8. We have already seen that a flattened butterfly is a blocking structure and a conflict-free routing schedule does not exist for this structure for heavy traffic conditions. So for heavy traffic conditions, where all the input nodes transmit

data with full bandwidth the latency of a flattened butterfly will be higher compared to all the networks shown in Figure 8.

2.4.5 Cost Analysis

A key determinant of the effectiveness of a network topology is the cost of the network relative to the performance it delivers. The cost of the network is determined by the network complexity and the link complexity of the network. As we mentioned in section 2.4.2 and section 2.4.3, 2DFB has the lowest network complexity and link complexity compared to the other nonblocking topologies. Now, we do a cost analysis, by considering some practical implementation aspects. Our cost model for a high performance Infiniband interconnection network is similar to the cost model described in [7], which is based on the cost of routers and links of different types and length as shown in Table 2.

Table 2: Cost breakdown of an interconnection network

Component		**Cost**
Router	Router Chip	\$90
	Development	\$300
Links	Backplane	\$1.95
	Electrical	$\$3.72 + \$0.81l$

The network components such as switches, processing nodes and communication links are packaged within a packaging hierarchy. Processing nodes will be in the lowest level of hierarchy. In the next level, modules are connected using a backplane. Modules and backplane are placed inside a cabinet. The whole network consists of several cabinet interconnected using communication links as per the network topology. The network cost is determined by the cost of the routers, backplane and cable links. The cost of a

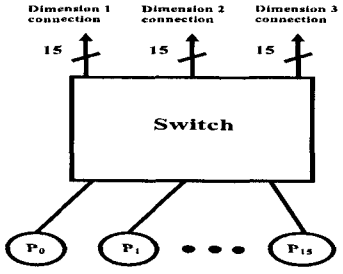
link is decided by its length and its position in the packaging hierarchy which is shown in Table 2. As we can see in the table, links within the backplane will be shorter and cheaper compared to other global links. Electrical cables are used for interconnecting cabinets and the cost of the cable is $\$3.72 + \$0.81l$ where l is the length of the cable expressed in meters [49]. The router cost can be split into the development (non-recurring) cost and the silicon (recurring) cost. The development cost depends on the number of router chips built. If we take a development cost of $\approx$ \$6M for 20k parts, the development cost per router is \$300. The recurring cost is the cost of silicon part and it is taken as $\approx$ \$90 per router using the MPR cost model [22].

Figure 9 shows a possible packaging of a 3-dimensional 16-ary 2DFB network. Each switching element is connected to 16 end-terminals and it has 15 channels dedicated to each dimension. These 45 channels are connected to other switching elements and the bandwidth of these channels should be twice the bandwidth of channels which are connected to end-terminals. A single switching element and channels connected to this switching element is shown in Figure 9(a). The 256 end-terminals which are connected in dimension 1 can be packaged in a subsystem as shown in Figure 9(b). This dimension 1 subsystem can be implemented using two cabinets each containing 128 end-terminals, and these cabinets are interconnected using short cables. Sixteen such dimension 1 subsystems can be grouped and its like elements are connected using dimension 2 channels forming a dimension 2 subsystem consisting of 4096 end-terminals. Up to 16 of these dimension 2 subsystems can be grouped and all the like elements can be inter connected using channels in dimension 3 leading to a network with up to 65 536 end-terminals. A possible packaging of this network is shown in Figure 9(c) where each box indicates a dimension 1 subsystem which can be implemented using two cabinets. All the interconnections in this dimension 1 subsystem can be implemented using short

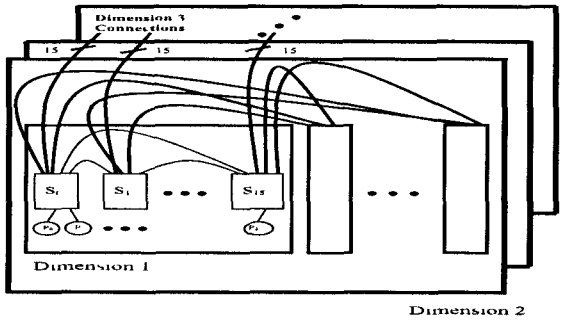
cables. Therefore, all the interconnections in dimension 1 can be implemented using a cheap backplane. Dimension 2 is mapped across columns and dimension 3 is mapped across rows and the connections are only shown for left lower subsystem. The packaging locality of the dimension 1 subsystem plays an important role in deciding the cost of the 2DFB network.

For large sized networks, the total cost of the network is dominated by the cost of the links. For example for $N \geq 4K$, the link cost accounts for 60% of the network cost of a hypercube and 80% of the network cost of a folded-Clos network [7]. The Infiniband 4x cables which support double data rate (DDR) and quad data rate (QDR) are commodity cables. Infiniband 4x-QDR cables can handle twice the bandwidth of Infiniband 4x-DDR cables. The cost of 4x-DDR and 4x-QDR cables are almost the same [31]. The cost of a 4x-QDR adapter is also comparable to that of a 4x-DDR adapter (less than 1.2 $\times$ cost of 4x-DDR) [32]. A 2DFB network is obtained from a flattened butterfly by dilating each interconnecting link between the switching elements by a factor of two. Assume that we have a flattened butterfly which uses 4x-DDR adapter and cables for all the interconnection. This can be converted to a 2DFB by replacing all the links between switching elements with 4x-QDR adapters and cables. Links between end-terminals and the switching elements are not changed. In this case, the total link cost of 2DFB will be less than 1.2 $\times$ link cost of flattened butterfly. Instead of doing this if we duplicate each 4x-DDR cables between the switching elements, the total link cost of 2DFB will become close to 2 $\times$ link cost of flattened butterfly. Thus, in an Infiniband 2DFB interconnecting system the link cost of the 2DFB can be reduced significantly (nearly 50%) by using cheaper double bandwidth links.

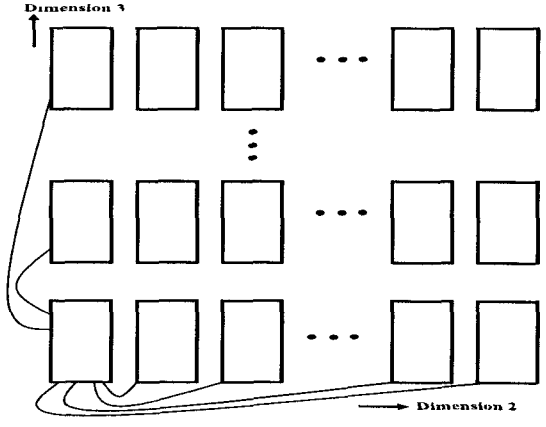
In a DBHC system each interconnecting link is dilated by a factor k. The total link cost of this network also can be decreased by making use of 4x-QDR. We are



(a) Switching element



(b) Network topology for 64K nodes



(c) Packaging block diagram

Figure 9: Topology and packaging of 3-dimensional 16-ary 2-dilated flattened butterfly

not considering higher bandwidth cables like 4x-FDR and 4x-EDR in our cost model because they are not commodity cables and are not comparatively cheaper. Folded-Clos also can be implemented using interconnecting links with double bandwidth. However, in this case one additional stage will be added to the network if we maintain same radix for the switching elements as that of 2DFB and if $k/2 < N/k$. Consider a folded-Clos network with network size N and L_T be the number of interconnecting links between the switching elements. Assume b is the bandwidth of each link. If we implement the same folded-Clos network using same radix switching element and interconnecting links between switches with $2b$ bandwidth (bandwidth of links between end-terminal and switch is b) then the total number of $2b$ bandwidth links will be $(L_T/2) + (N/2)$ if $k/2 < N/k$, otherwise it is $(L_T/2)$. In 2DFB and DBHC total number of $2b$ bandwidth links will be $(L_T/2)$. Total number of switching elements and $2b$ bandwidth links required for different topologies for different network sizes are shown in Table 3. We can see that a 2DFB needs fewer switching elements and $2b$ bandwidth links compared to other nonblocking topologies.

The 2DFB structure can be implemented efficiently in Ethernet based networks. To construct a 2DFB, a 1 Gb/s Ethernet twisted-pair link in the flattened butterfly could be replaced with a 2 Gb/s SATA link instead of using two separate 1 Gb/s Ethernet links. The cost of a 2 Gb/s SATA link is comparable to that of a 1 Gb/s Ethernet link.

Table 3 compares the total number of routers and interconnecting links required for the implementation of different nonblocking topologies for a variety of network sizes. The interconnecting links referred to here are the links with double the bandwidth of the end-terminal links. Router internal bandwidth is represented in terms of maximum bandwidth b of each end-terminal port. Routers with same bandwidth are used to implement all the nonblocking topologies with the same network size. The value of k

Table 3: Resource comparison:- where 'l' is the number of links with double bandwidth, 'n' is the number of routers and 'b' is the bandwidth of the router

Network topology	N=256			N=1 024			N=16 384			N=32 768		
	l	n	b	l	n	b	l	n	b	l	n	b
2DFB	112	8	88	496	32	94	15 616	512	154	31 744	1 024	156
Folded-Clos	128	12	88	512	44	94	16 384	639	154	32 768	1 263	156
DBHC	256	16	88	1 792	94	94	45 056	1 366	154	98 304	2 731	156

used for these topologies for different network size is shown in Table 1. From Table 3 it is clear that 2DFB needs fewer routers and interconnecting links compared to other nonblocking networks.

Table 4: Parameters and assumptions used for the cost comparison

parameter	value
nodes per cabinet	**128**
cabinet footprint	**0.57m x 1.44m**
density	**75 nodes/** m^2
number of pairs per port	**3**
cable overhead	**2m**

Figure 10 compares the cost per node of different networks for different network sizes based on our cost model. The parameters and assumptions used for the cost comparison is shown in Table 4 [11].

Here we compare the cost of DBHC, folded-Clos, 2DFB and flattened butterfly topologies for different network sizes. The value of k and router bandwidth used for the implementation of each network are shown in Table 1 and Table 3. We assume that 128 nodes are packed in a cabinet. The router cost for the different networks are appropriately adjusted, based on the internal bandwidth required in each network. We know that router internal bandwidth of a flattened butterfly is nearly half of that of

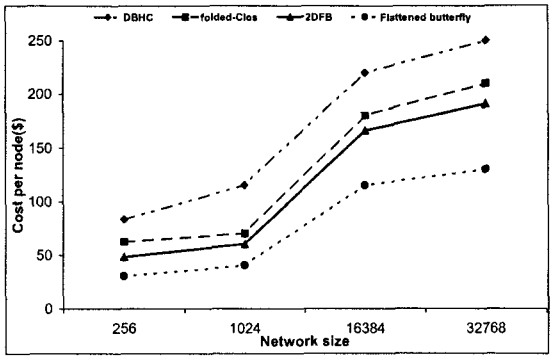


Figure 10: Cost comparison

corresponding 2DFB. In this model, we have also assumed that the cost per length of a $2b$ link is $1 + \gamma$ times that of a link with bandwidth b and the value of γ can vary between 0 and 1. For example, consider a high-performance Infiniband switching network with 4x Infiniband DDR and QDR forming the links in the 2DFB switching network. The cost of 4x Infiniband DDR cable and adapter are comparable to that of QDR cable and adapter. If we consider the ratio of the cost of these two channel links, it will be less than 1.2, in other words, $\gamma < 0.2$ [31]. A gigabit Ethernet based network $2b$ link can be implemented using a single 2-Gb/s SATA link, and the cost of this link is comparable with the b Ethernet link. In this case, γ is very close to 0, since the cost of a SATA cable and an Ethernet cable are comparable. However, as a worst case approximation we have taken 0.3 as the value of γ.

From Figure 10, we can see that the cost of a 2DFB is less than that of corresponding folded-Clos and DBHC networks. Obviously, the cost of 2DFB will be more than that of flattened butterfly which need less router and link bandwidth. In the previous analysis we have seen that the performance of 2DFB network is better than corresponding folded-Clos and DBHC networks. From this analysis we argue that a 2DFB provides

better performance and lower cost compared to corresponding folded-Clos and DBHC networks.

2.5 Simulation results

We have modeled 2DFB, DBHC, folded-Clos and flattened butterfly networks for different network sizes using the OMNeT++ simulation library [27]. All the nonblocking topologies are implemented using interconnecting links of 2 Gb/s bandwidth. All the end-terminals are sending packets with a maximum bandwidth of 1 Gb/s. We have used a packet size of 121 bytes. Higher size packets are also following the same trend. The default OMNeT switch model was modified in order to include the 2 Gb/s channel.

We assume that the data transmission through the network is permutation type - i.e. a unique source and destination are assigned to any data element and the elements are permuted up on transmission. We have selected three traffic patterns to consider the best case and worst case scenario of 2DFB topology which are named as below.

1) *Benign* : In a 2DFB structure each switching element is connected to $k-1$ switching elements using direct links in each dimension. In *benign* traffic pattern all the traffic can be routed through these directed links, that is in this pattern the number of hops required for the routing of any packet will be equal to the diameter of the 2DFB network. In this pattern each pair of end-terminals connected to a switching element will be sending traffic to different directly connected switching elements. 2DFB provides minimum latency for *benign* traffic pattern.

2) *Adversarial* : In this traffic pattern all the end-terminals connected to a switching element S_i will be sending traffic to end-terminals which are connected to another single switching element S_{i+j}. If this pattern is used in a 2DFB only two end-terminals which are connected to a switching element can send traffic through the direct link.

All other $k-2$ end-terminals should send traffic through indirect links. 2DFB provides worst case latency for *adversarial* traffic pattern.

3) *Random* : In this pattern destination terminals are selected randomly. Latency provided by 2DFB for this pattern will be between that of *benign* and *adversarial* patterns.

We have compared the throughput and latency of different network topologies for different traffic patterns mentioned before. We have used static routing schedule.

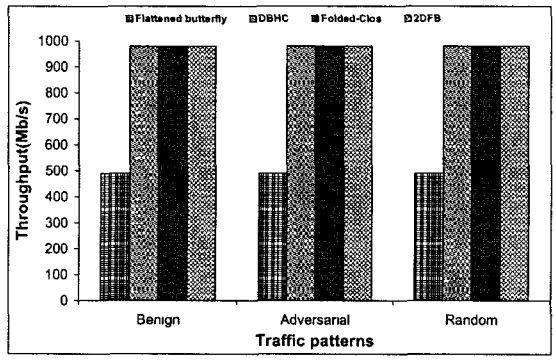


Figure 11: Throughput comparison of 64-terminal networks

The average throughput for the different network topologies for a network size of 64 is shown in Figure 11. The static routing scheme is used for this analysis. An 8-ary 1-dimensional 2DFB is compared to DBHC, folded-Clos and flattened butterfly networks. All the networks except flattened butterfly are implemented using routers of same internal bandwidth. These nonblocking topologies are also implemented using double bandwidth interconnecting links. All the end-terminals are transmitting data with maximum bandwidth (1 Gb/s). We can see that, like DBHC and folded-Clos networks, 2DFB also provide 100% throughput for all the traffic patterns. We can also notice the blocking behavior of the flattened butterfly networks. For all the traffic

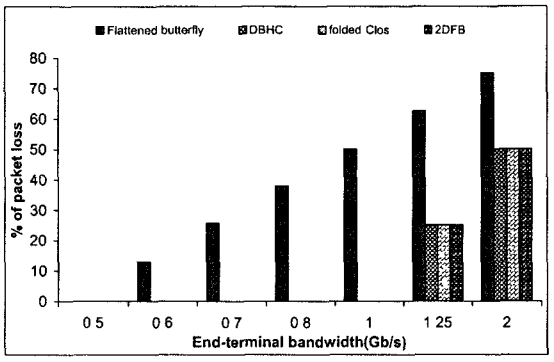


Figure 12: Packet loss of 64-terminal networks for different input load

patterns, the flattened butterfly can provide only 50% of the maximum throughput if all the end-terminals are transmitting data with full bandwidth.

Another measure of the effectiveness of a switch topology is the amount of packet loss as input data rates are increased. Figure 12 shows the packet loss for different 64-terminal networks for the *adversarial* traffic pattern by varying the input data rate of each end-terminal. We can see that flattened butterfly network start dropping packets when the end-terminal data rate goes above 500M. However, DBHC, folded-Clos and 2DFB networks have zero packet loss up to a data rate of 1 Gb/s due to their nonblocking nature.

We have measured the average end-to-end packet delay of different 64-terminal networks for maximum input load condition and it is shown in the Figure 13. For a nonblocking network, end-to-end packet delay will be proportional to the average number of hops needed for the routing process. Packet delay is also decided by the switching delay of each switching element. This switching delay is proportional to the radix (bandwidth) of each switching element. In our comparison we have implemented all the nonblocking networks using switching elements of equal radix. Therefore, the

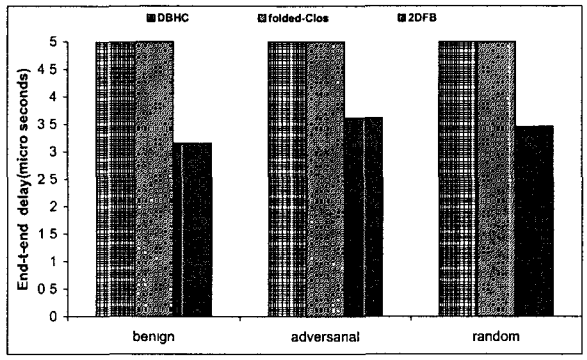


Figure 13: Latency comparison for a 64-terminal network

effect of switching delay will be same for all the nonblocking networks described here. We have used *benign*, *adversarial* and *random* traffic patterns for this latency comparison. 2DFB exhibits maximum delay for *adversarial* traffic pattern compared to other traffic patterns. In this pattern the average number of hops needed for routing the packets in a 8-ary 64-terminal 2DFB is close to 2. Average number of hops required in a 8-ary 64-terminal 2DFB for *benign* pattern is 1. The average number of hops needed for a *random* traffic pattern will be in between that of *benign* and *adversary* traffic patterns.

A folded-Clos implementation of a 64-terminal network with the same radix router as that of 2DFB and double bandwidth links needs three stages of switching elements ($N/k > k/2$), and therefore, the number of hops needed for routing any of the traffic pattern is 4. The average number of hops for a DBHC for the given traffic patterns are close to $\log_2(N/k)$. The average number of hops for a 64-terminal DBHC network that is implemented using the same radix router as that of 2DFB is 4. We know that the flattened butterfly is a blocking structure, and therefore, in heavy load conditions, the end-to-end packet delay will be determined by the queuing delay. This queuing delay

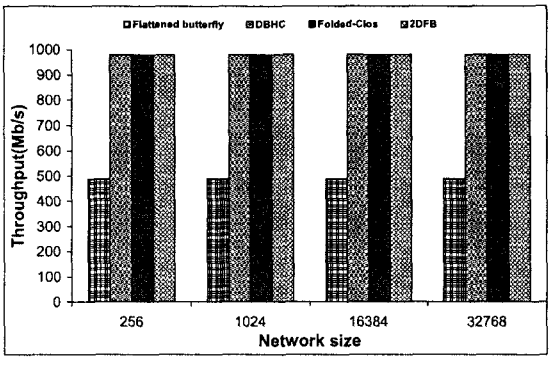


Figure 14: Throughput comparison for different network size

is significantly higher compared to the end-to-end packet delay of other nonblocking networks. From this latency comparison we can observe that a 64-terminal 8-ary 2DFB provides minimum average end-to-end packet delay compared to other networks for all the traffic patterns.

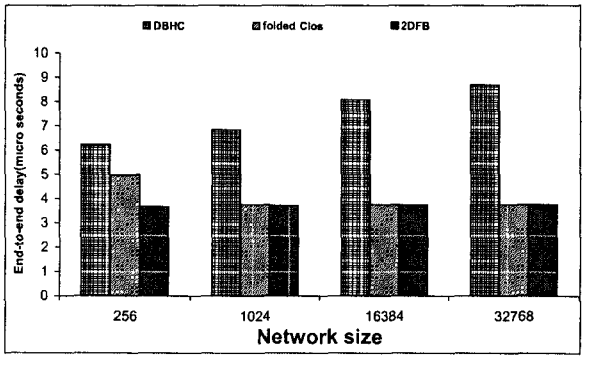


Figure 15: Latency comparison for *adversarial* traffic

We have observed the throughput and end-to-end packet delay of 2DFB network for varying network size from 256 to 32 768 and compared it with that of DBHC,

folded-Clos and flattened butterfly. We have implemented 1-dimensional 2DFB with static routing schedule, where the value of k selected is $\lceil\sqrt{N}\rceil$. The higher dimension implementation also follows the same behavior. We have implemented DBHC, and folded-Clos using switching elements of same bandwidth as that of 2DFB. All these topologies are implemented using interconnecting links of 2 Gb/s bandwidth. All the end-terminals are transmitting data with 1 Gb/s bandwidth. The throughput comparison is shown in Figure 14. We have used *adversarial* traffic pattern for this comparison. As we can see in the figure 14, like other nonblocking networks 2DFB also provides 100% throughput for the all given network sizes. Obviously flattened butterfly provides a maximum of 50% throughput for all the network sizes.

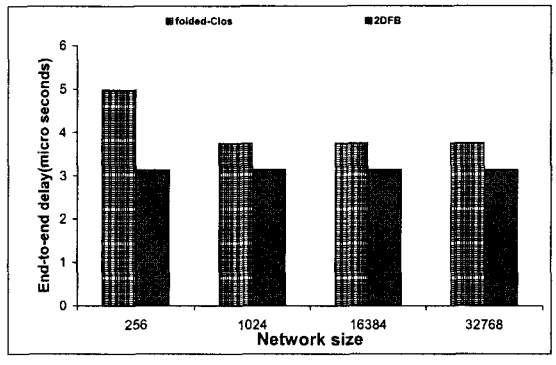


Figure 16: Latency comparison for *benign* traffic

The end-to-end packet delay comparison for the *adversarial* traffic pattern is shown in Figure 15. The average number of hops of the given 2DFB network for any network size is close to 2, because all the networks are implemented in one dimension. A folded-Clos network with a network size of 256 needs three stages ($N/k > k/2$) of switching elements, and therefore, the average number of hops for this network is 4. All the folded-Clos networks with network size higher than 256, which are shown in the figure,

need only two stages ($N/k \leq k/2$) of switching elements and the average number of hop count of these networks is 2. Thus, for larger networks end-to-end delay of 2DFB will be very close to that of folded-Clos network for *adversarial* traffic pattern. End-to-end delay of a DBHC will be larger compared to corresponding folded-Clos and 2DFB for *adversarial* traffic pattern and the increment in the delay is proportional to the network size.

Figure 16 shows the comparison of end-to-end packet delay of 2DFB for different network sizes to that of folded-Clos networks for *benign* traffic pattern. For *benign* traffic, a 1-dimensional 2DFB needs only one hop to complete any routing irrespective of the network size whereas folded-Clos networks with higher network size need two hops. Therefore, the end-to-end packet delay of 2DFB will be less than that of folded-Clos network for any network size for *benign* traffic pattern.

The end-to-end packet delay of DBHC will be larger compared to the other two nonblocking topologies and is not shown in Figure 16. In normal practice, the traffic patterns will be random, and in this case the delay of 2DFB will be in between that of *adversarial* and *benign* traffic patterns. So, the end-to-end packet delay of 2DFB will be always less than that of folded-Clos for any network size.

All these simulation results reveals the reduced latency of a 2DFB over folded-Clos and DBHC networks for different traffic patterns and network sizes. Thus, we can consider 2DFB as a nonblocking high speed network with reduced cost.

Chapter 3

Hardware Implementation of 2DFB

3.1 Overview

A critical component of an ASN is the network switch since the switch implements the data processing on data as it is aggregated and distributed from multiple sources. Typically, custom silicon is used to build gigabit and multigigabit switches and these switches offer the best performance. FPGAs on the other hand provide an intermediate design point by offering maximum flexibility in the network processing while achieving high performance. For this reason we use a switch built using an FPGA.

The selection of switching topology is also extremely important in an ASN. The topology has an impact on the flexibility of computations that can be performed on an ASN. Aspects like the type and nature of the interconnects used, latency, nonblocking, etc. are decided by the switching topology and thus have an important impact on the overall performance and scalability of the network. We have selected 2DFB as the switching topology for ASN [14, 35]. The ASN switch is built using a NetFPGA board designed by Stanford University and Digilent Inc. to help build prototypes of hardware-accelerated networking systems [1].

3.2 Hardware implementation

We implemented a 2-dimensional 2-ary 2DFB network with a network size of 8 terminals using the NetFPGA as the switching element. The NetFPGA board is a PCI card, which contains a Virtex 2Pro (XC2VP50) FPGA, specifically designed for network applications by a research group at Stanford University [1]. It has four 1 Gigabit/second Ethernet (GigE) interfaces and two SATA ports which make it suitable to build a switching network. The NetFPGA research group also provides the source code for the board so that it can be used as a hardware accelerated OpenFlow switch [25]. We have extended this work by developing two interfacing modules SATA TX_Q and SATA RX_Q which are shown in Figure 17 and we have achieved serial data transmission through the SATA link up to 2Gb/s.

The structure of the OpenFlow switch hardware is shown in Figure 17. It is a five stage pipeline structure where each module communicates using a simple packet based synchronous FIFO push interface which makes it easy to add additional modules to the structure for the purpose of packet processing. The user data path is 64 bits wide and it is driven by a 125 MHz clock. Therefore, the switch can handle a maximum throughput of 8 Gb/s - i.e. the hardware can process packets at line rate.

The RX queues accept data from the Ethernet MAC core or Aurora core and convert it to the format required for the user data path. This module also generates an 8-bit control word and the module header. This 64-bit header is prepended to the beginning of each packet and the 8-bit control word is generated for each 64 bit word. The module header contains the length of the packet in bytes, the source port as a binary-encoded number and the packet length in 64-bit words. This module header is named the IOQ module header. The Input Arbiter selects an RX queue in which packets are available

and pushes a packet into the Output Port Lookup module without modifying the module header. Based on the desired destination port, the Output Port Lookup module decides which output port the packet should be transmitted to, and this information is added as a one-hot-encoded number to the IOQ module header. The Output Queues module looks at the IOQ module header and decides the output queue according to the output port information and it stores the packet into the selected output queue. The output queues are in SRAM and are not in the FPGA. The packet length from the IOQ module header is used to store the packet efficiently. The stored packets in the output queue are removed and pushed into the corresponding destination TX queue. The TX queue module changes the data format according to the requirements of the core (MAC or Aurora) and pushes it to the respective core inputs. The TX queue module also removes the IOQ module header from each packet before transmission and it generates all the control input signals for the core.

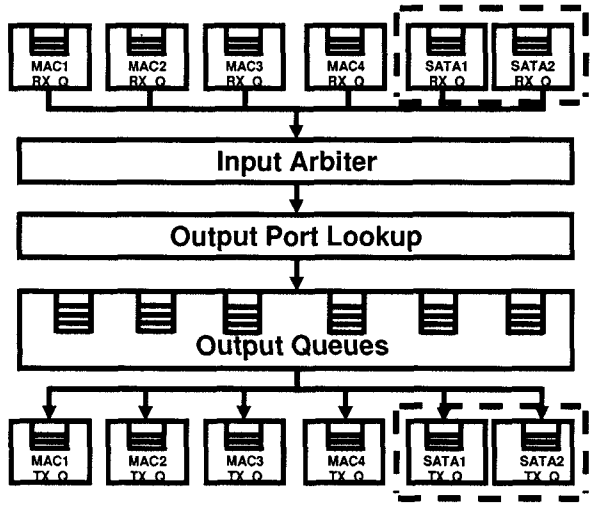


Figure 17: NetFPGA switch architecture

In the 2DFB with a network size of 8, we are making use of the SATA ports of the NetFPGA card for the interconnection between switching elements. The NetFPGA

card supports two SATA ports. We are making use of the Aurora protocol for serial communication through SATA [23]. Aurora is a LogiCORE IP designed by Xilinx to enable easy implementation of the FPGA RocketIO transceivers while providing a light-weight user interface on top of which designers can build a serial link. It is a scalable, lightweight, link-layer protocol for high-speed serial communication. It also supports full duplex operation and flow control.

SATA with Aurora can give the maximum throughput of 3.125 Gb/s if we drive the RocketIO transceivers using a differential clock network [24]. Unfortunately, the current implementation of the NetFPGA does not provide a differential clock pair, but instead provides a single ended clock generator of 125MHz. Using the Aurora Protocol we can transmit two bytes in a single clock pulse. Thus, the maximum throughput provided by Aurora in the present system is 2Gb/s which is still twice that of gigabit Ethernet. We have used LogiCORE Aurora v3.0 for our implementation. The Aurora core can be customized to suit a wide variety of requirements using the CORE Generator software provided by Xilinx.

3.2.1 Customizing Aurora core

One of the important parameters that we should select while generating the core is $LaneWidth$ which decides the number of bytes transmitted during the period of one clock cycle. Aurora supports $LaneWidth$ of 2 and 4. If we set the $LaneWidth$ to 4, the user clock (USER_CLK) should be half of the reference clock (REFCLK), whereas with a $LaneWidth$ of 2, the frequency of USER_CLK and REFCLK should be the same. So, the effective data rate will be same for both cases. Moreover, the clock generation for a $LaneWidth$ of 4 is complicated than for a $LaneWidth$ of 2. So we have selected the default $LaneWidth$ value of 2. There are two types of data path interfaces used for the

core which are *Framing* and *Streaming*. The *Streaming* interface is a simple word-based interface with a data valid signal to stream data through the Aurora channel whereas the *Framing* interface is a Local Link interface that allows encapsulation of data frames of any length. Since in our application we have to transmit packets through the SATA link, we selected *Framing* interface. The Aurora core supports two data flow modes, *Simplex* and *Duplex*. We have selected *Duplex* mode because we need to transmit data in both directions. The Aurora core also supports several clock inputs to drive the RocketIO transceivers. BREFCLK and BREFCLK2 are low-jitter differential clock networks that can support line rates up to 3.125 Gb/s. However, the NetFPGA platform does not support differential clock networks, and as a result we have selected the REFCLK, which is a reference clock input for low rate serial connections. The Aurora REFCLK frequency was set to 125MHz to match the NetFPGA clock - thus giving an Aurora/SATA line rate of 2 Gb/s (125 MHz * 2 byte lane * 8 bits). The Aurora core also supports $UserFlowControl$ and $NativeFlowControl$, but we are not making use of either of these flow controls.

3.2.2 Clock interface for Aurora

Aurora cores require a low jitter reference clock for generating and recovering high-speed serial clocks in the RocketIO transceivers. We can use either the differential clock pair BREFCLK or the single ended clock REFCLK as the reference clock. Since the differential clock pair is not available in the NetFPGA platform, we have used *gtx_clk*, which is the common TX clock of 125MHz, which we can see in the top level module of the source code, as the source for the REFCLK. Two-byte lane Aurora cores also use a single clock to synchronize all signals between the core and the user application called USER_CLK. All logic that connects to the core must be driven by USER_CLK, which

in turn must be the output of a global clock buffer (BUFG). Figure 18 shows how the reference clock and USER_CLK from *gtx_clk* are generated.

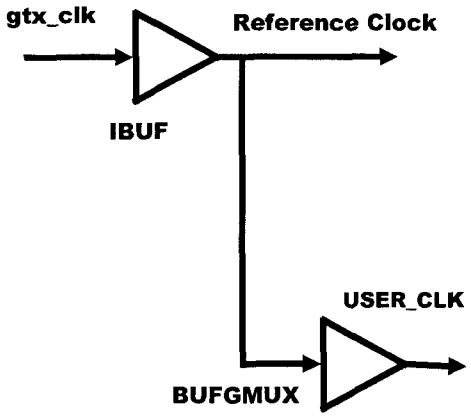


Figure 18: Clocking for 2-Byte Aurora core

3.2.3 Aurora core and Interfacing modules

The top-level interfacing unit of the Aurora core, which can be directly connected to the other modules of the project, is known as the Aurora Local Link Interface. The Aurora Local Link Interface, with Local link-compliant ports for TX and RX data, is shown in Figure 19 [24].

3.2.3.1 Transmitting

Table 8 lists port descriptions for Local Link TX data ports. The timing diagram for a simple data transfer at the Aurora transmitting side is shown in Figure 20 [24].

All the signals required to feed the Aurora Local Link transmitter are provided by the module SATA TX_Q, which is shown in Figure 21. The 64-bit data and the 8-bit control, which are coming from the output queue, are used to generate the 72-bit Arranged Data. This 72-bit Arranged Data is stored in a FIFO in the SATA TX_Q

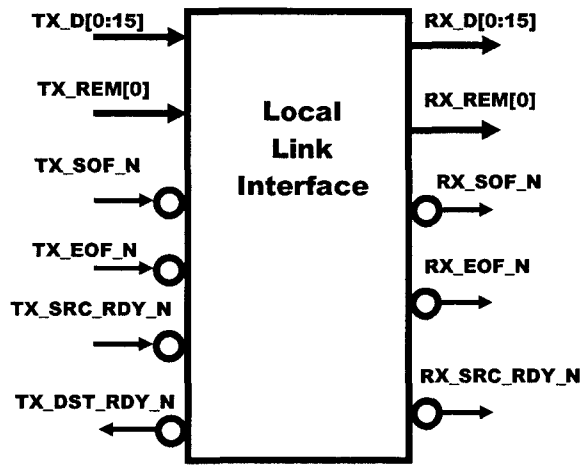


Figure 19: Aurora Core Framing Interface [24]

module. The data and control bits are arranged in such a way that, the FIFO should output 2 bytes of data and the corresponding two control bits in a single FIFO output read cycle. The writing data width of the FIFO is 72 and the reading data width is 18. The *in_wr* signal, which is asserted whenever the incoming data are valid, is used as the write enable signal for the FIFO. The SATA TX_Q module also generates a signal *in_rdy* which is asserted whenever this module is ready to accept data from Output Queue module. All the other control signals required for the FIFO and for the Aurora Local Link Interface transmitter are generated by a State Machine in the SATA TX_Q module. A state machine controls the reading process of the FIFO by generating FIFO read enable signal. Sixteen bits of data are extracted from the 18-bit read data and it is given to the TX_D port of Aurora transmitter. TX_DST_RDY_N is an incoming signal which is asserted by Aurora core and the Aurora core will accept the data only when this signal is asserted. So the state machine should assert FIFO read enable only after checking the TX_DST_RDY_N signal for successful transmission of data. State Machine generates TX_SOF_N, TX_EOF_N and TX_SRC_RDY_N signals by making use of 8-bit control and in_wr signals from Output Queue module. Since

Table 5: LocalLink User I/O Ports (TX)

Name	Description
TX_D[0:15]	Outgoing data.
TX_DST_RDY_N	Asserted (Low) during clock edges when signals from the source will be accepted. Deasserted (High) on clock edges when signals from the source will be ignored.
TX_EOF_N	Signals the end of the frame.
TX_REM[0]	Specifies the number of valid bytes in the last data beat; valid only while TX_EOF_N is asserted.
TX_SOF_N	Signals the start of the outgoing channel frame (active-Low).
TX_SRC_RDY_N	Asserted (Low) when LocalLink signals from the source are valid. Deasserted (High) when LocalLink control signals or data from the source should be ignored.

Aurora transmits two bytes per clock, it is essential to indicate the last byte of the frame. This information is passed through the signal TX_REM[0].

The Aurora core supports a feature called clock compensation that allows up to ± 100 ppm difference in the reference clock frequencies used on each side of an Aurora channel. This feature is used in systems where a separate reference clock source is used for each device connected by the channel. Since each NetFPGA platform uses its own reference clock we have to make use of this clock compensation feature. A clock compensation module is generated with the Aurora core. To perform Aurora-compliant clock compensation, DO_CC must be asserted for several cycles during every clock compensation period. The duration of the DO_CC assertion and the length of time between assertions is determined based on the width of the RocketIO data interface.

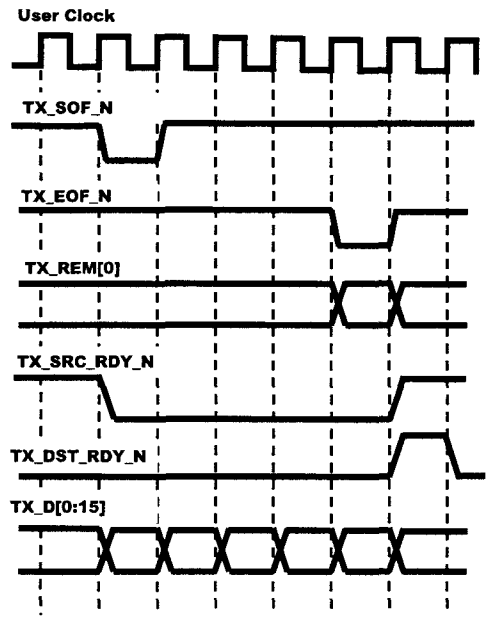


Figure 20: Data transfer in Aurora transmitter [24]

For a lane width of 2, DO_CC is asserted for 6 USER_CLK cycles after each 5000 USER_CLK cycles.

3.2.3.2 Receiving

Table 6 lists port descriptions for Local Link RX data ports. All the signals coming from the Aurora Local Link receiver, which are shown in Figure 22 [24], are given to the module SATA RX_Q, which is shown in Figure 23.

The 16-bit data, which is coming from port RX_D[0:15] and corresponding two control bits generated in the SATA RX_Q are written in a FIFO. The writing data width of the FIFO is 18 and the reading data width is 72. We know that the user data path of the NetFPGA switch is 64 bits wide and so in each read cycle, the FIFO should output 64-bits of data as well as the corresponding 8-bit control signal which are given to the Input Arbiter as shown in Figure 23. All the operations in the SATA

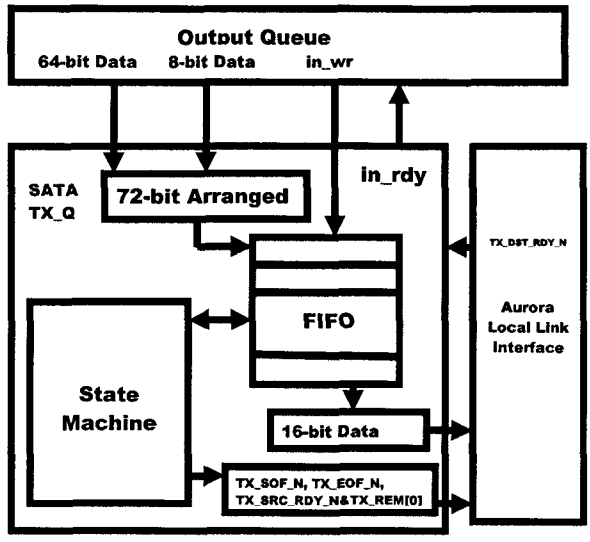


Figure 21: SATA TX_Q module

RX_Q module are controlled and synchronized by state machines. Both the FIFO read enable and FIFO write enable signals are generated by state machines. The FIFO read enable is asserted only when *out_rdy* signal is asserted which is an incoming signal coming from Input Arbiter. *out_rdy* signal is asserted only when Input Arbiter is ready to accept data. RX_SRC_RDY_N is an incoming signal coming from Aurora core which indicates whether the received data is valid or not. This signal is similar to the Ethernet MAC core's *emacclientrxdvld* signal that is asserted at the beginning of each frame and is de-asserted only after the last byte of the frame. *emacclientrxdvld* is not de-asserted during the transmission of a frame. However, in the Aurora core, RX_SRC_RDY_N can be de-asserted during the transmission of a frame because Aurora will stop the transmission for a few clock cycles whenever DO_CC is asserted for clock compensation. Therefore, the signals RX_SOF_N and RX_EOF_N are the only way to identify the starting and ending of a frame. State machine should assert FIFO write enable only when RX_SRC_RDY_N is asserted. Since the Aurora receiver delivers 2 bytes for each clock cycle, State machine should make use of RX_REM[0] to find out

Table 6: LocalLink User I/O Ports (RX)

Name	Description
RX_D[0:15]	Incoming data.
RX_EOF_N	Signals the end of the incoming frame (active-Low).
RX_REM[0]	Specifies the number of valid bytes in the last data beat; valid only when RX_EOF_N is asserted.
RX_SOF_N	Signals the start of the incoming frame (active-Low).
RX_SRC_RDY_N	Asserted (Low) when data and control signals from an Aurora core are valid.

the last byte of the frame. The SATA RX_Q module also generates a header which carries information like word length, byte length and the number of input port. This 64-bit header is written at the beginning of each frame. SATA RX_Q also generates 8-bit control signal which carries useful information like starting of a frame, ending of a frame and the position of the header, for each 64-bit data which is passed to the Input Arbiter.

3.2.4 Implementation of 2DFB

A 2-dimensional 2-ary 2DFB network with a network size of 8 terminals is shown in Figure 24 where all the inter-switch links have double the bandwidth compared to the end-terminal links.

Each switching element is implemented using a NetFPGA. So in order to implement the network shown in the Figure 24 we need four NetFPGA boards. Each end-terminal is connected to the NetFPGA board using 1 Gb/s Ethernet link, and SATA links are used for the interconnection between the NetFPGA boards which will support a data rate of 2Gb/s. So we are making use of two Ethernet ports and two SATA ports of

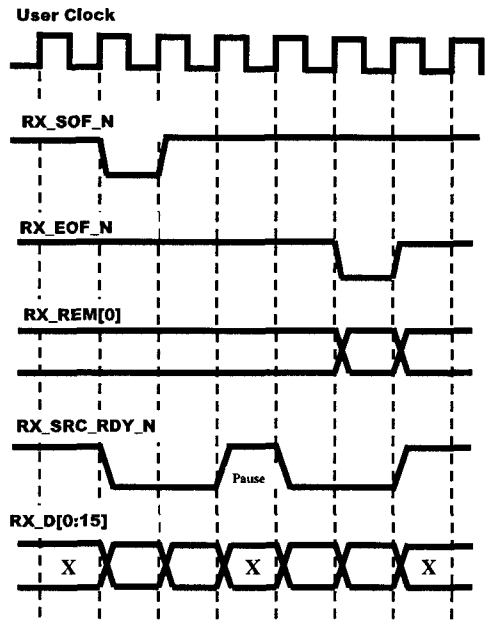


Figure 22: Data transfer in Aurora receiver [24]

each NetFPGA board. We have derived a conflict free routing schedule which is used for the implementation of the 2-dimensional 2-ary 2DFB network.

3.3 Results

We have observed the throughput performance of the network using IPERF [26], which is a network testing tool that can create TCP and UDP data streams and measure the throughput of a network that is carrying them. We have used UDP data stream for the throughput measurement because in UDP mode we can specify the input data rate and we will get the exact statistics of packet loss. We have also implemented three other network topologies of the same network size for the throughput comparison. The other three topologies are butterfly, flattened butterfly and Benes [10] which is a 2-ary Clos. For these three topologies Ethernet links are used for all the interconnections. Four boards are needed for the implementation of butterfly and flattened butterfly networks,

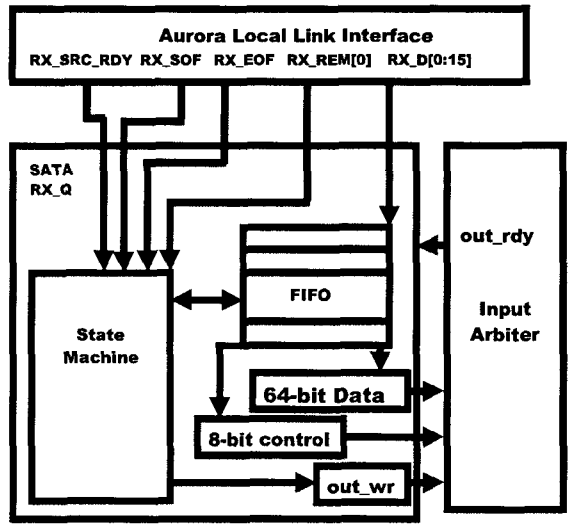


Figure 23: SATA RX_Q module

whereas a Benes network requires six NetFPGA boards. Throughput measurements are done for three different traffic patterns as described in Table 7. We have selected these patterns to consider the best case and worst case scenario of different topologies. All the nodes are named as in Figure 24.

Table 7: 8-terminal network traffic pattern

Straight	(1,5),(2,6),(3,7),(4,8)
Random	(1,4),(2,3),(5,7),(6,8)
Cross	(1,8),(2,7),(3,6),(4,5)

The throughput comparison of the four network topologies is shown in the Figure 25. The source terminal is transmitting UDP data stream with a rate of 1 Gb/s. The average throughput for all the three traffic patterns is taken for the comparison. We can see that more than 50% of the packets are lost in both flattened butterfly and butterfly network for all the given traffic patterns. We observe that like a Benes network, a 2DFB also shows nonblocking behavior. We can also see a small amount of

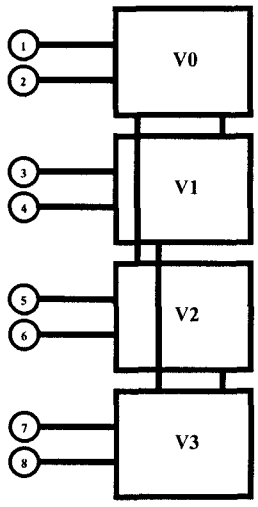


Figure 24: 2-ary 2-dilated flattened butterfly structure

packet loss in a 2DFB for the *cross* traffic pattern which is the worst case traffic for this network. In this traffic pattern, some switching elements need to handle comparatively higher traffic and we observe that in this case, the NetFPGA switch is slightly deviating from its idle switching performance.

We have also observed the percentage of packet loss for different load conditions for the above mentioned network topologies. We have selected *cross* traffic pattern for this measurement because *cross* is the worst case traffic pattern for all the given networks. We have observed the percentage of packet loss by varying all end-terminal data rate from 0.25Gb/s to 1Gb/s. This percentage of packet loss is shown in Figure 26. We can see that both butterfly and flattened butterfly networks starts dropping packets when the end-terminal data rate go above 500 Mb/s and we can also see that for both Benes and 2DFB, the packet drop is close to 0% up to a data rate of 1 Gb/s. An 8-terminal

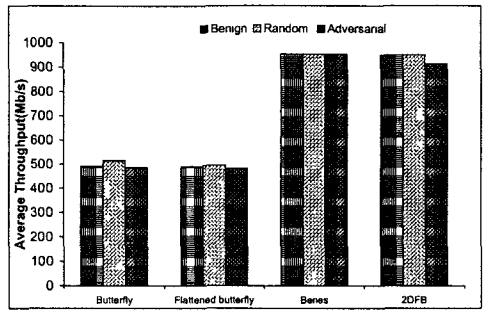


Figure 25: Throughput comparison

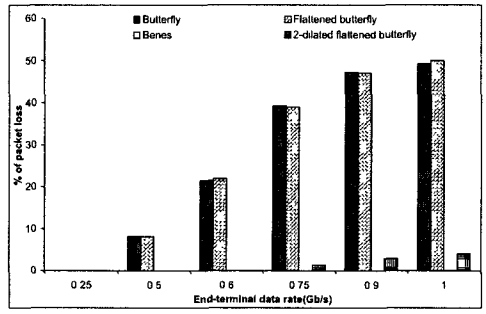


Figure 26: Packet loss for different load condition

Benes implementation requires six NetFPGA boards whereas a 2DFB needs only four NetFPGA boards. As a result, the maximum data rate handled by each board in a 2DFB topology is greater than that of Benes for the worst case traffic pattern *cross* and this causes a small packet loss in a 2DFB network when the end-terminal data rate is closer to 1Gb/s.

The above hardware implementation results reveal that 2DFB implementation using NetFPGA is able to provide maximum throughput. It also requires reduced implementation cost because the number of NetFPGA boards required in this case is lesser

compared to other topologies. We also make use of the whole internal bandwidth of the NetFPGA switch (8Gb/s) by making use of the two SATA interconnection.

Chapter 4

ALDFB: An Adaptive Load Balanced Deadlock free Routing Scheme

4.1 Overview

In this chapter, we introduce an adaptive load balanced deadlock free routing scheme called ALDFB that is designed to exploit all the positive topological properties of a 2DFB network. ALDFB achieves load balance by allowing one non minimal forwarding in each dimension in case of network congestion. This algorithm provides high throughput on adversarial traffic patterns and better latency on benign traffic patterns.

A routing algorithm can be considered as optimal if it provides low latency on local traffic and high throughput on adversarial traffic. Most algorithms must compromise one goal in order to achieve the other. Minimal routing, which always chooses the shortest path for each packet, provides minimum latency for local and benign traffic.

However, it provides non acceptable latency for adversarial traffic due to load imbalance. In order to improve the throughput in adversarial traffic, the routing algorithm should balance the load by sending some fraction of packets over non-minimal paths.

Researchers have been trying to address the issue of providing high worst-case performance while preserving locality. Valiant's randomized algorithm [2] gives good performance in worst case traffic but very poor performance for local traffic in terms of latency. Minimal adaptive routing [3] [4] suffers from global load imbalance. UGAL is a load balanced adaptive routing algorithm designed for a torus network [5]. It provides better load balance with improved performance for local traffic. UGAL will take a wise decision in the selection of minimal or non-minimal route according to the status of the channel. In the non-minimal routing phase, UGAL will select a random intermediate router and the packets will be routed to and from the intermediate router minimally. This random selection of intermediate router helps UGAL to balance the load. Adaptive Clos [33] is an adaptive routing algorithm designed for Clos network which provides optimum performance for a high-radix Clos network. UGAL and adaptive Clos are considered as the most efficient routing schemes for interconnecting networks in a high performance computing system.

The adaptive routing algorithm, ALDFB, that we propose is designed for a 2DFB network. It is designed in order to make use of all the positive topological properties of a 2DFB network. For benign or local traffic it selects the minimal path and makes use of the reduced diameter of a 2DFB network. In adversarial traffic, ALDFB make use of the improved path diversity of a 2DFB network. In case of traffic congestion ALDFB balances the load efficiently by allowing one non-minimal forwarding in each dimension. ALDFB guarantee deadlock and live lock freedom. In order to achieve deadlock freedom we have made use of the concept of virtual channels and dimension

ordered routing. We observed the performance of ALDFB over a 2DFB network and compared its performance with UGAL, minimal adaptive and adaptive Clos routing schemes. We show that ALDFB over a 2DFB network outperforms other routing schemes in terms of throughput and latency. We have also compared the performance of ALDFB over a 2DFB network with other well-known networks like Mesh, folded-Clos and dilated bristled hypercube.

4.2 Routing Algorithm: ALDFB

The proposed routing algorithm ALDFB is designed to explore the topological properties of a 2DFB network. A 2DFB network is similar to a k-ary generalized hypercube (GHC) [8] except that in a 2DFB k end-terminals are connected to each switching element. A 2DFB can be considered as a k-way bristled 2-dilated GHC. If r is the dimension of a k-ary flattened butterfly, then there will be k^r nodes (switching elements) in the system and each node can be represented using a r-digit number, i.e. any node $x = x_{r-1}...x_i...x_0$ where $x_i \in [0, k-1]$. In a 2DFB network any two nodes, whose numbers differ only in the i^{th} digit, are joined by a duplex channel and is known as the i^{th} dimension channel. Thus by comparing the r bit number associated to the current switching element and the destination switching element, one can find out the set of dimensions in which forwarding of the packet is required. Every node contains $(k-1)$ channels in each dimension.

ALDFB has two phases of operation, the minimal forwarding phase and the non-minimal forwarding phase. The minimal forwarding phase of ALDFB is shown in Algorithm 1. In the minimal phase, the algorithm considers the set of dimensions in which forwarding is required and it adaptively selects the dimension if the direct link in the selected dimension is ready to use. This adaptive selection is possible in the

same cycle if we make use of a priority encoder. The input of the priority encoder is a P bit register where P is the total number of ports of a router. Each bit of this input register indicates the status of corresponding output queue. For example if the number of packets in the queue is less than a threshold value the bit will be '1', and otherwise it is '0'. Output of the priority encoder will be the address of the port selected according to the priority. If we have more than one empty queue we can give priority to the port in the higher dimension to preserve the dimension order routing nature. In the minimal routing phase only the status of the direct ports are given as the input of the priority encoder. If no usable direct link is available in the selected dimension, then the algorithm will turn in to non-minimal phase of operation.

In the non-minimal phase, ALDFB will consider all the ports in the selected dimension and adaptively check the availability of any of these non-minimal links. This non-minimal forwarding phase is shown in Algorithm 2. Again, a priority encoder is used for the adaptive selection of output port. The status of all the selected ports is given as input to the priority encoder and it will output the address of selected port. The packet will be forwarded to this non-minimal link. We constrain this non-minimal forwarding by adding a one bit flag in the header of each packet and we call this flag as the priority flag. ALDFB allows only one non-minimal forwarding in each dimension. If the switching element sees that the priority flag is set for the received packet, then that packet will be sent to a minimal direct link even though all minimal output queues have packets more than the threshold level. In the next cycle some portion of the traffic coming from the other switches will be adaptively rerouted to any non-minimal link which will reduce the traffic congestion. Thus, by the combined use of minimal and non-minimal phase of operation, ALDFB will balance the load efficiently and it will reach the steady state within a few iterations.

Algorithm 1: Minimal routing phase of ALDFB.

```
1  begin
2      if s_d == d_d then
3          LinkSel = 1, Link_out = Local Link
4      else
5          Set Pi = 0, Find Dr (Dimensions selected for routing)
6          Find Pr (Direct Links corresponds to Dr)
7          Add queue status of Pr to Pi
8          if Ps == true then
9              Link_out = Po, h_1=0, LinkSel=1
10             send(frame, Link_out)
11         else
12             if h_1 == 1 then
13                 Link_out = higher dim Link from Pr, h_1=0, LinkSel=1
14                 send(frame, Link_out)
15             else
16                 Go to non-minimal routing phase
17             end
18         end
19     end
20 end
```

ALDFB always gives priority to the minimal forwarding and therefore for local traffic and benign traffic, the performance of this routing scheme will be equal to the minimal routing. With the worst case traffic ALDFB will use at most two links per dimension. In the worst case also a fraction of traffic is routed through direct links which will help to reduce the latency.

Algorithm 2: non-minimal routing phase of ALDFB.

```
1  begin
2      if LinkSel != 1 then
3          Set Pi = 0
4          Find Dr (Dimensions selected for routing)
5          Add queue status of all ports in Dr to Pi
6          Link_out = Po
7          h_1=1
8          LinkSel=1
9          send(frame, Link_out)
10     end
11 end
```

We have numbered each link based on the number of the router at the starting point and the number of the router at the ending point. We have numbered each router as shown in Figure 1. For a 2DFB network of dimension r we have used $2 \times r$ digits for numbering each link. Out of this $2 \times r$ digit number the first r digits represents the number of the router from which the link is starting and last r digits represents the number of the router to which the link is connected. For example a 64 terminal 2

dimensional 2DFB network need four digits for numbering each link. In this case each router will have six incoming links and six outgoing links (all links connected to the end terminals are not considered here). The numbering of all links which are connected to the router R_{12} in the Figure 1 is shown in Figure 27. Each time after selecting the output link ALDFB compare the output link number with the corresponding input link number. If the output link number is greater than the input link number corresponding forward virtual channel is selected for routing the packets, else the packets are routed through corresponding reverse virtual channel.

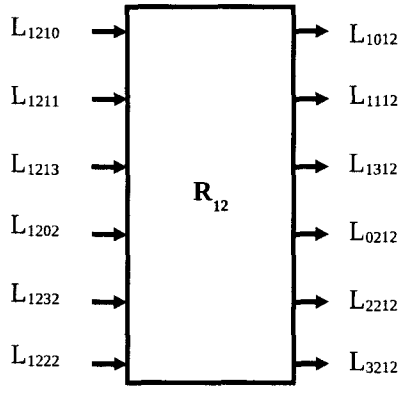


Figure 27: Links associated to router R_{12}

4.2.1 Terminologies used in the algorithm

The minimal routing phase of ALDFB is shown in the Algorithm 1 and the non-minimal phase is shown in the Algorithm 2. A one bit flag is added to the header of each packet to indicate the switching priority and it is represented as h_1. An output link is selected by considering the number of packets already in queue in the corresponding output queue. The link is selected if the number of packets in the output queue is less than the threshold value T_h. The preferred dimensions to which the packets can be forwarded are decided by comparing the r digit representation of the current switching

element and the destination switching element, where r is the dimension of the network. r digit representation of current switching element and destination switching element is represented as $s_d[r]$ and $d_d[r]$ respectively. A priority encoder is used to select an output link adaptively according to the queue status corresponding to each link. The input of the priority encoder is a P bit register where P is the total number of outgoing links associated to each router and this input is represented as Pi. Output of the priority encoder is the address of the selected link and is represented as Po. There is a flag to indicate the validity of priority encoder output and it is represented as Ps.

4.2.2 Deadlock and Livelock

A deterministic minimal dimension ordered routing is free from deadlock [52]. We have made use of the concept of dimension ordered routing to ensure deadlock freedom. As mentioned in Section 4.2 we have numbered each link and the packets which are routed in the increasing channel order are forwarded through forward virtual channels and the packets which are routed in the decreasing channel order are forwarded through reverse virtual links.

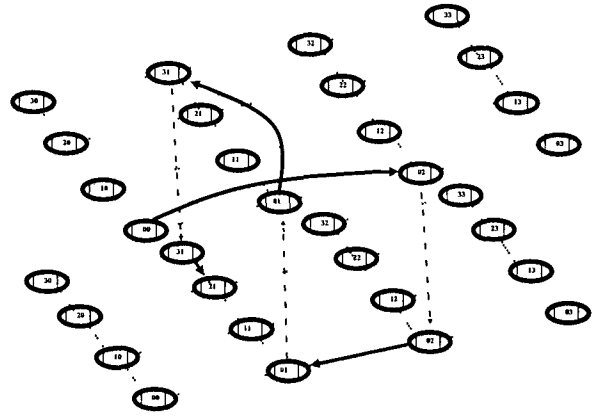


Figure 28: Packet flow through virtual links

Dally, in his work [50] showed that virtual channels increase network throughput and reduce the dependence of throughput on the depth of the network. He also had introduced deadlock-free adaptive routing algorithms by making use of virtual channels [51]. Virtual channels can be used to eliminate cycles in the resource dependency graphs. Figure 28 helps to visualize the concept. The upper plane in Figure 28 consists of all forward virtual channels and the bottom plane consists of all reverse virtual channels. An example of packet flow is shown here. The Packet is starting from an end-terminal which is connected to router R_{00} and it is going to an end-terminal which is connected to the router R_{21}. In the router R_{00} ALDFB fail to get any direct path in the minimal routing phase and it find a non-minimal path in the non-minimal routing phase and the packets are forwarded to the link L_{0200} through the forward virtual path. Before this non-minimal forwarding the priority bit in the header is set to one and therefore the router R_{02} has to assign a minimal path to forward the packet. The minimal link assigned by the router R_{02} is L_{0102} which has a lower link number compared to the input link and therefore the packets are forwarded through the reverse virtual channel. The transition from forward virtual channel to reverse virtual channel is shown as the dashed vertical arrow towards down. Notice that now the distance between the source router and the destination router is decreased by one dimension. Because of the use of the priority header in ALDFB after each non-minimal forwarding there will be a mandatory minimal forwarding which will further decrease the distance between source and destination by one dimension. Before the packet is forwarded from R_{02} the priority header is changed back to zero and so when the packet reaches in the router R_{01} ALDFB running in router R_{01} can go to the non-minimal phase if the minimal links are not available. This process repeats and the packet reaches at the destination router after four forwarding stages. From this example it is clear that in

each virtual plane packets are forwarded according to the link order and whenever there is a pair of transition from one virtual plane to the other virtual plane the distance between the source and destination will be decreased by one dimension. Thus the channel dependence graph will be free from cyclic dependency and the routing will assure deadlock freedom. Restricting the non-minimal forwarding using priority header also makes ALDFB free from live lock.

Algorithm 1 and Algorithm 2 are modified by restricting the packet flow through virtual channels in order. This provides deadlock freedom and the modified minimal phase is shown in Algorithm 3. The modified non minimal phase is shown in Algorithm 4. We are making use of two virtual channels per physical link. Half of the outgoing virtual channels are dedicated to the packets flow in the increasing channel order which are named as forward virtual channels ($ForwardVC$) and half of the outgoing virtual channels are dedicated to the packets flow in the decreasing channel order which are named as reverse virtual channels ($ReverseVC$). After selecting the outgoing link, the routing scheme selects appropriate output virtual channel to which the packet is to be forwarded based on the incoming link number and outgoing link number.

Algorithm 3: Minimal routing phase of ALDFB.

```
1  begin
2      if s_d == d_d then
3          LinkSel = 1, Link_out = Local Link
4      else
5          Set Pi = 0, Find Dr (Dimensions selected for routing)
6          Find Pr (Direct Links corresponds to Dr)
7          Add queue status of Pr to Pi
8          if Ps == true then
9              Link_out = Po, h_1=0, LinkSel=1
10             Go to Assign VC_out
11         else
12             if h_1 == 1 then
13                 Link_out = higher dim Link from Pr, h_1=0, LinkSel=1
14                 Go to Assign VC_out
15             else
16                 Go to non-minimal routing phase
17             end
18         end
19         Assign VC_out: if Link_in < Link_out then
20             VC_out = ForwardVC_{Link_out}, send(frame, VC_out)
21         else
22             VC_out = ReverseVC_{Link_out}, send(frame, VC_out)
23         end
24     end
25 end
```

Algorithm 4: non-minimal routing phase of ALDFB.

```
1  begin
2      if LinkSel != 1 then
3          Set Pi = 0
4          Find Dr (Dimensions selected for routing)
5          Add queue status of all ports in Dr to Pi
6          Link_out = Po
7          h_l=1
8          LinkSel=1
9          if Link_in < Link_out then
10             VC_out = ForwardVC_Link_out
11             send (frame, VC_out)
12         else
13             VC_out = ReverseVC_Link_out
14             send (frame, VC_out)
15         end
16     end
17 end
```

4.2.3 Algorithms used for comparison

We have selected minimal adaptive routing, non-minimal global adaptive routing (UGAL) and adaptive Clos for the performance comparison with our proposed routing

scheme ALDFB. The minimal adaptive algorithm will always route packets in the shortest path. UGAL will take a wise decision in the selection of minimal or non-minimal route according to the status of the output queue. In the non-minimal routing phase UGAL will select a random intermediate router and the packets will be routed to and from the intermediate router minimally. This random selection of intermediate router helps UGAL to balance the load. The adaptive Clos routing algorithm uses forward and backward phases. In the forward phase any of the links in the forward path is adaptively selected by checking the status of each forward link. In the reverse phase, routing is deterministic as there exists only a single path to the destination. The adaptive Clos routing scheme is implemented over a folded-Clos network.

4.2.4 UGAL Vs. ALDFB

If we compare minimal adaptive and UGAL routing schemes over a 2DFB network we can see that UGAL outperforms minimal adaptive because it provides better throughput and reduced latency for both benign and adversarial traffic patterns whereas the minimal adaptive algorithm shows poor performance for adversarial traffic patterns. Therefore, it will be worthwhile to do a comparison between the behavior of UGAL and ALDFB for different traffic conditions over a 2DFB network. Both routing schemes always give priority for links corresponding to the minimal distance (direct links). So benign or reduced traffic both the routing scheme route packets through the direct links and therefore the performance will be same. For heavy and adversarial traffic the behavior of both the routing scheme will be different. In a k-ary 2DFB network since k end-terminals are connected to a router all the packets cannot route through direct links in heavy and adversarial traffic conditions. In this traffic condition in order to balance the load UGAL will select a random intermediate node and the

packets are transmitted to and from the intermediate node through minimal path. In this traffic condition ALDFB consider all the non-minimal channels in all dimensions in which traversal of the packet is required and select an available channel. After selecting the non-minimal channel, ALDFB sets the priority header and sends the packet through the non-minimal link. In the case of UGAL after selecting the intermediate node packets are not allowed to take any non-minimal link. Even though this random selection of intermediate node helps for load balancing, there can be many corner cases in which a single channel is overloaded and this will affect the overall performance of the network. One corner case situation in which UGAL provides bad performance due to overloading a channel is shown in Figure 29. Since bandwidth of each link in 2DFB is twice than that of end-terminal injection bandwidth, packets from two end-terminals with full bandwidth can be passed through a link without much delay. A channel in 2DFB will be overloaded if three end-terminals are trying to send packets through one link with full bandwidth. In the Figure 29, one terminal connected to R_{01}, R_{11} and R_{31} are selected R_{22} as the intermediate node. Assume that all the three links between these three starting routers and R_{21} ($L_{2101}, L_{2111}, L_{2131}$) are having reduced traffic and all the packets are routed to R_{21}. According to UGAL the router R_{21} do not have any other option and have to forward all the packets towards R_{22} and this will overload the link L_{2221}.

A similar worst case situation in ALDFB is depicted in Figure 30. Assume that one end-terminal connected to R_{23} is sending packets to an end-terminal connected to R_{12} through R_{13} with full bandwidth. Then, another end-terminal connected to R_{11} starts sending packets to R_{12} through R_{13} with full bandwidth. Now the link L_{1213} is fully loaded. In this situation assume one more end-terminal connected to R_{10} sending packets to R_{12} through R_{13}. Since this is a non-minimal forwarding, the packets that

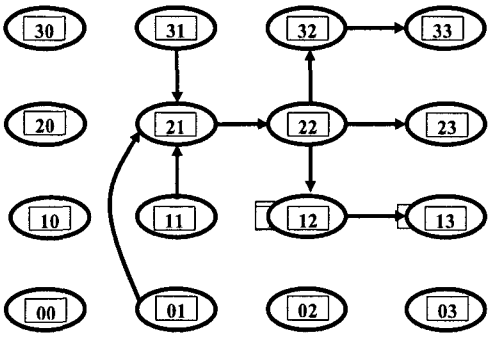


Figure 29: A traffic flow in UGAL which will cause overloaded channels

reach at R_{13} do not have any other option and are forwarded to the link L_{1213} and make this link overloaded. However, at the router R_{13} we can notice that the priority header of all packets which are coming from R_{23} is not set. Therefore, during the next forwarding cycle the packets coming from R_{23} will be forwarded to another available non-minimal link and this further reduces the traffic through the link L_{1213}. Thus, ALDFB is more adaptive in nature and it effectively balances the load.

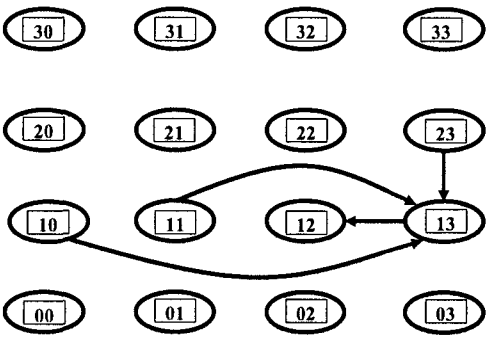


Figure 30: A worst case traffic flow for ALDFB

4.3 Simulation results

We have modeled 2DFB, folded-Clos, DBHC and Mesh networks for a network size of 64 using the OMNeT++ simulation library [27]. All simulated topologies have the same bisection bandwidth, and all end-terminals send packets with a maximum

bandwidth of 1 Gb/s. We have used a packet size of 121 bytes to simulate the worst case, but larger size packets show similar behavior. 2DFB, folded-Clos and DBHC networks are implemented using interconnecting links of 2 Gb/s bandwidth. The low-radix mesh is implemented using interconnecting links of 4 Gb/s bandwidth. The default OMNeT switch model was modified in order to include these higher bandwidth channels. We have compared the throughput and latency of these network topologies for different routing schemes and traffic patterns. We assume that the data transmission through the network is permutation type - i.e. a unique source and destination are assigned to any data element and the elements are permuted according to the traffic patterns. We have selected *Benign*, *Adversarial* and *Random* traffic patterns as explained in chapter2 to consider the best case and worst case scenario of 2DFB topology.

We have compared the average throughput of a 4-ary 2-dimensional 2DFB network with a network size (N) of 64 for three different routing algorithms, Minimal, UGAL and ALDFB. The throughput comparison is done for three different traffic patterns as mentioned before and is depicted in Figure 39.

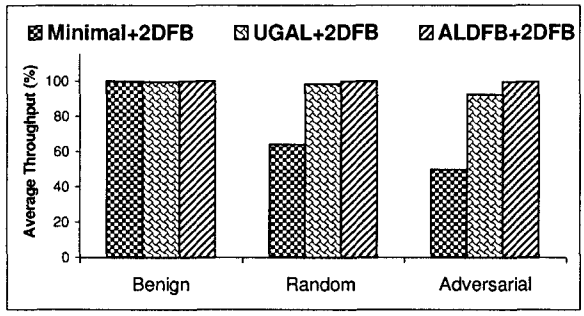


Figure 31: Throughput comparison for different routing schemes

All the end-terminals are injecting packets at a rate of 1 Gb/s. ALDFB provides a nonblocking traffic flow and the throughput is very close to 100% for all the traffic patterns. UGAL also provides throughput close to 100% for benign and random

traffic patterns, but considerable reduction (8%) for adversarial traffic patterns. For adversarial traffic patterns UGAL fail to provide efficient load balancing due to the random selection of intermediate router whereas ALDFB exploits the path diversity in the 2DFB network and balances the load efficiently. Minimal adaptive routing is not offering any load balancing and so all the direct links in 2DFB will be overloaded and it provides poor throughput for random and adversarial traffic patterns.

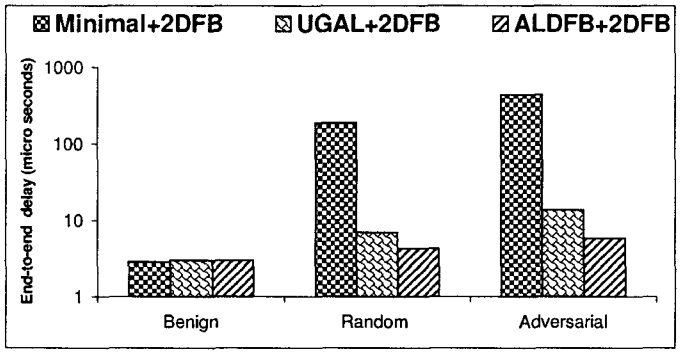


Figure 32: End-to-end packet delay comparison for different routing schemes

The average end-to-end packet delay comparison for the above mentioned routing schemes over 64-terminal 2-dimensional 2DFB network is shown in Figure 32. Please note that the delays are plotted at log-scale. All the end-terminals are injecting packets with a rate of 1 Gb/s. For benign traffic the delay is same for all routing schemes because all packets are routed through the direct link without overloading it. For the adversarial traffic pattern, the minimal adaptive algorithm shows poor delay performance. Even though UGAL provides better load balancing compared to minimal adaptive it is not able to avoid overloading few interconnecting links in adversarial traffic patterns. We can see that ALDFB provides minimum delay compared to other two routing schemes.

The end-to-end packet delay comparison for reduced load condition where each end-terminals are injecting packets at a rate of 500 Mb/s is shown in Figure 33. In this

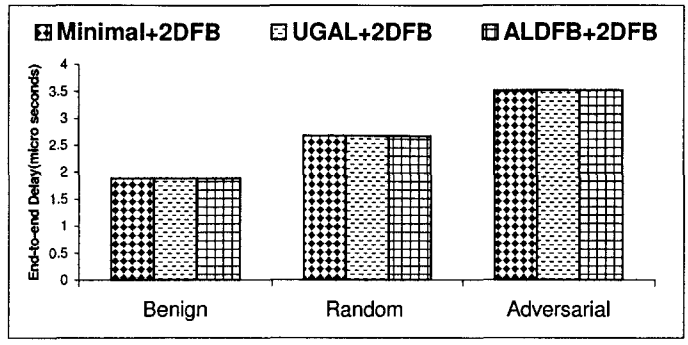


Figure 33: Delay comparison for different routing schemes for reduced load

reduced load condition all three routing schemes forward packets through the minimal links without overloading it and thus the delay performance will be same.

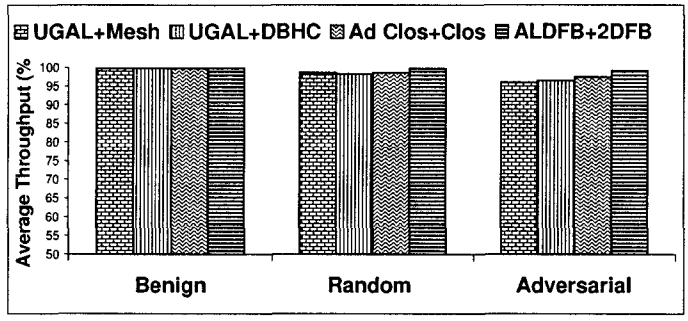


Figure 34: Throughput comparison for different network topologies

We have also compared the average throughput of four different network topologies with the same bisection bandwidth and with the network size of 64. The topologies that we have selected are 4-ary folded-Clos, dilated 4-way bristled hypercube (DBHC), Mesh and 4-ary 2DFB. Mesh and DBHC use UGAL, folded-Clos uses adaptive Clos and 2DFB uses ALDFB for routing packets. This throughput comparison is shown in Figure 40. We can see that with an adversarial traffic pattern, 2DFB with ALDFB provides slightly better throughput performance compared to the other three topologies.

While the improvement in throughput is small, the effect of the ALDFB algorithm is much more pronounced on the packet latencies. The average end-to-end packet

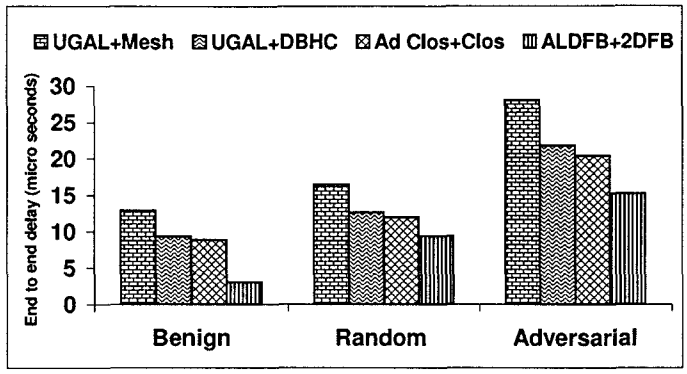


Figure 35: End-to-end packet delay comparison for different network topologies

delays for the above mentioned network topologies with an end-terminal injection ratio of 1 Gb/s are shown in Figure 35. This delay comparison reveals the reduced delay performance of a 2DFB network with ALDFB routing scheme. A low-radix mesh provides the worst delay performance because of its large diameter and the number of hops needed for routing. We can see that ALDFB makes use of the reduced diameter and larger path diversity of 2DFB network and provides minimum delay performance for all traffic patterns.

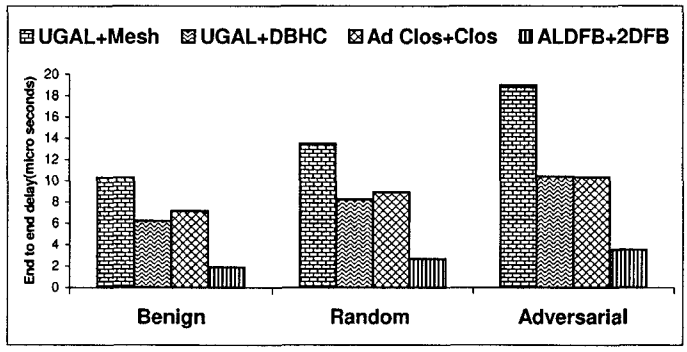


Figure 36: Delay comparison for different network topologies for reduced load

The average end-to-end packet delay comparison for an end-terminal injection ratio of 500 Mb/s for different network topologies is shown in Figure 36. As can be seen, even

with a reduced load, a 2DFB network with ALDFB provides minimum delay because it uses the minimum number of hops for routing compared to other networks.

From the above throughput and latency comparison, it is clear that 2DFB with ALDFB provides maximum throughput with reduced latency. For benign and random traffic the performance of UGAL and ALDFB are almost the same as in a 2DFB network. However, at full load and with adversarial traffic, ALDFB significantly outperforms UGAL in terms of throughput and latency. We can also see the improved latency performance of ALDFB on 2DFB compared to other networks for all the simulated traffic patterns. Thus, we consider ALDFB over 2DFB as an optimal candidate for a high performance interconnection system with reduced cost.

Chapter 5

2DFB based On-Chip Networks

5.1 Overview

The on-chip interconnection network (OCIN) and routing plays an important role in the performance of a chip multiprocessor (CMP). As the number of cores in CMP increases, the OCIN also should scale efficiently to make use of the increasing processing capacity. An ideal OCIN should provide maximum throughput and minimum latency. We propose using a 2-dilated flattened butterfly (2DFB) instead of a mesh as a high-radix OCIN because of its nonblocking property and reduced diameter. We also propose using ALDFB, an adaptive load balanced routing scheme for 2DFB based OCIN networks. We evaluate the performance of the 2DFB OCIN with ALDFB routing using synthetic traffic patterns and compare it with a mesh network having the same bisection bandwidth, which uses adaptive minimal routing scheme. We also compare the performance of ALDFB with two other most popular adaptive routing schemes. We observe that 2DFB with ALDFB provides significant improvement in terms of throughput and latency.

The advancement in VLSI technology has led to a doubling of available processing cores on a single chip every third year. As the processing power of a chip has increased and data intensive applications have emerged, designing efficient on-chip interconnection networks (OCIN) and routing schemes has attracted increasing attention [36,37]. Most multiprocessor chips currently available such as the RAW processor [38], the TRIPS processor [39] and the 64-node chip multiprocessor from Tilera [40] are making use of low radix 2-D mesh architecture. Low radix 2-D mesh networks are attractive because of their simplicity and short wire requirement for interconnection. However, the drawback is its large diameter and average number of hops used for the routing process. As a result, 2-D mesh based OCINs suffer from increased latency and power dissipation.

Recent work has shown that a high-radix OCIN, based on a flattened butterfly, can outperform the corresponding low-radix mesh network in terms of latency and power [41]. This improvement is achieved due to its reduced diameter and average number of hops used for the communication. The 2-dilated flattened butterfly (2DFB) is a nonblocking version of a flattened butterfly network. In this chapter, we analyze the OCIN, based on 2DFB, and compare it with a mesh based OCIN. We also propose using ALDFB routing scheme for 2DFB based OCIN. ALDFB balances the load efficiently by allowing one non-minimal forwarding in each dimension in case of traffic congestion. It senses the traffic congestion from the packet queue. We observed the performance of ALDFB in 2DFB based OCIN for both local (benign) and adversarial traffic patterns and observed that it outperforms other routing scheme in terms of latency and throughput. We have also compared the performance of a mesh based OCIN with 2DFB based OCIN of same bisection bandwidth and observed that 2DFB based OCIN provides significant reduction in latency for reduced load condition.

5.2 OCIN Networks

In this section we describe OCINs using mesh and 2DFB interconnecting topologies. We also compare the resources needed and different parameters of these OCINs.

5.2.1 Mesh based OCIN

The mesh topology is the simplest topology and is well suited for tiled architecture implementations. The length of each interconnecting link is equal to the length of a tile (T_s). A 64 core mesh based CMP is depicted in Figure 37. We assume

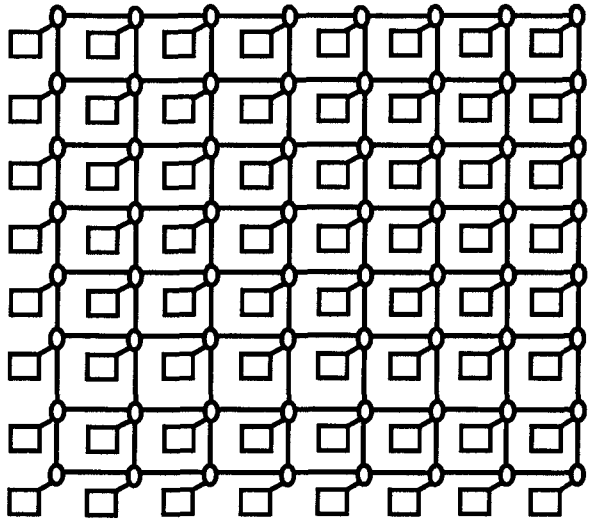


Figure 37: 64-core CMP using mesh topology

that the width of each channel in the mesh network is 64 bits. Therefore, the total number of wires passing through the bisection is 8×64. For simplicity we express the bisection bandwidth as the total number of wires that pass through the bisection. Each switching element can be implemented by making use of a 5X5 crossbar switch.

5.2.2 2DFB based OCIN

A 64-core 2DFB based OCIN is depicted in Figure 38. Here four processing cores are connected to each switching element. A 10X10 crossbar switch is needed to implement each switching element. The width of each interconnecting link is selected as 32 bits so that the total number of wires (16×32) crossing the bisection (bisection bandwidth) is same as that of the mesh system shown in Figure 37. As shown in Figure 38, a 2DFB implementation requires interconnecting links of different length. The minimum channel length required for a k-ary 2DFB based OCIN is $\sqrt{k}\times$ tile span (T_s). T_s is the width of a single processor core/tile. We assume that the cores are square - i.e. the width and height are the same. In a 4-ary 64-core system, the minimum channel length is $2T_s$ and the length of other channels are $4T_s$ and $6T_s$. Since horizontal and vertical channels cross each other, a silicon implementation would require two metal layers. Horizontal interconnection links and the links connected to the cores can be implemented in one metal layer and all vertical interconnection links can be implemented in another metal layer without conflict.

5.2.3 Mesh Vs 2DFB

A comparison of some critical parameters of above mentioned mesh and 2DFB based 64-core CMP implementation is shown in Table 8. The bisection bandwidth is represented as the number of wire segments that cross the bisection and it is the same for both networks. Wiring density is defined as the maximum number of tile-to-tile wires routable across a tile it is also same for both networks. The degree of each crossbar switch in a 64-core 2DFB is twice than that of mesh, but the diameter and maximum hop count of 2DFB is much less than that of a mesh network and this helps to improve the performance of 2DFB in terms of latency and power.

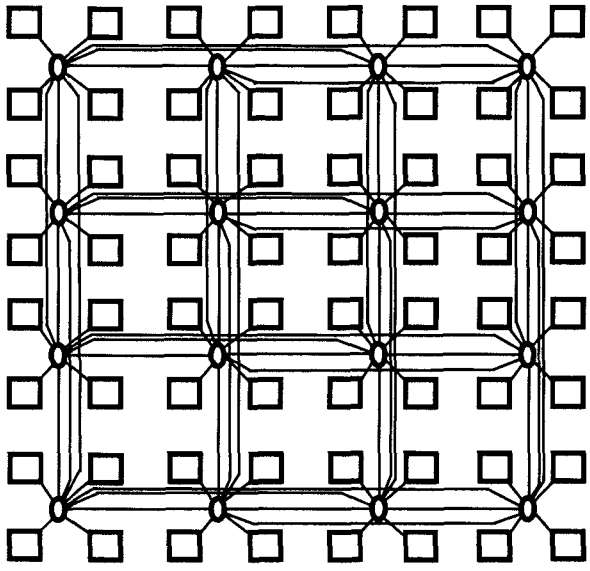


Figure 38: 64-core CMP using 2DFB topology

Table 8: Parameters comparison for 64-core CMP networks

Parameter	**Mesh**	**2DFB**
Bisection bandwidth	512	512
Wiring density	64	64
Maximum hop count	14	6
Diameter	14	2
Degree of each crossbar switch	5	10

A resource comparison of mesh and 2DFB based 64-core CMP implementations is shown in Table 9. Wire segments are separated into four groups having different lengths and these lengths are represented in terms of tile span (T_s). Even though 2DFB uses wire segments of larger length, because of the reduced number of wires, the total wire length used is 28.5% less than that of a mesh network.

A 5X5 router is used in the 64-core mesh network and each router needs 5 input and 5 output FIFOs. In a 64-core CMP there are 64 such routers and therefore the total number of FIFOs needed in a 64-core mesh network is 640. The size of each router

Table 9: Resource comparison

Resource	**Mesh**	**2DFB**
No. of wire segments:L,2L,4L,6L	7168,0,0,0	0,768,512,256
Total wire length (in T_s)	7168	5120
Total No. of cross point transistors	102400	51200
Total No. of FIFOs	640	320

in the 64-core 2DFB network is 10X10 and each router needs 10 input and 10 output FIFOs. There are 16 such routers in the 64-core 2DFB based CMP and therefore the total number of FIFOs needed in a 64-core 2DFB based CMP is 320. Thus the total number of FIFOs used in a 64-core 2DFB based CMP is 50% less than that required for a 64-core mesh based CMP. The channel width of a 2DFB based OCIN is 32, so the total number of cross point transistors used in a single router of 2DFB network is 32×100 and the total number of cross point transistors used in the 2DFB based CMP is $16 \times 32 \times 100$. The channel width of a mesh based OCIN is 64 and hence the total number of cross point transistors used in a single router of mesh network is 64×25 and the total number of cross point transistors used in the mesh based CMP is $64 \times 64 \times 25$. Thus, the total number of cross point transistors required for the 64-core 2DFB implementation is 50% less than that of the corresponding mesh network.

The disadvantage of 2DFB is that it needs longer wire segments and a higher degree crossbar switch. Longer wires may need larger driving power or need a series of driving buffers or repeaters to reduce the wire delay. In the case of mesh, even though the wire length is short the packet has to pass through a large number of hops to reach the destination and this increases the power requirement. This additional power requirement will be comparable to the additional power needed to drive longer wires in 2DFB. A larger degree crossbar switch also needs a complex control unit, but the

total number of switching elements in a 2DFB is much less than that of a mesh and therefore the total area requirement for the control units in 2DFB will be comparable or less than that of a mesh network. From this resource comparison it is clear that a 2DFB based CMP implementation outperforms a mesh based CMP in terms of area and cost.

5.3 Simulation results

We have modeled 64-core 2DFB and mesh based OCINs for the same bisection bandwidth using the OMNeT++ simulation library [27]. We consider 125 MHz as the router clock frequency. With this clock frequency, the channel bandwidth of the mesh will be 8 Gb/s and the 2DFB will be 4 Gb/s. We also consider 2 Gb/s as the maximum injection ratio of the end terminals. Beyond 2 Gb/s both the mesh and 2DFB networks show blocking behavior. Our motive is to study the performance of mesh and 2DFB as a nonblocking switching network. We have selected a packet size of 32 bytes and we consider a store and forward type routing protocol. Output queue size considered is 64 bytes so that we can store only two packets in the buffer. In the case of traffic congestion packets will be dropped inside the switch. We have compared the throughput and latency of these network topologies for different traffic patterns. We have also compared the performance of ALDFB with Minimal adaptive and UGAL routing schemes on mesh and 2DFB networks. We assume that the data transmission through the network is permutation type - i.e. a unique source and destination are assigned to any data element and the elements are permuted upon transmission. We have selected *Benign*, *Adversarial* and *Random* traffic patterns to consider the best and worst case scenario of 2DFB topology.

5.3.1 Throughput comparison

We have compared the average throughput of a 64-core, mesh and 2DFB based OCIN with the same bisection bandwidth. Adaptive Minimal, UGAL and the proposed ALDFB routing schemes are used for the throughput comparison.

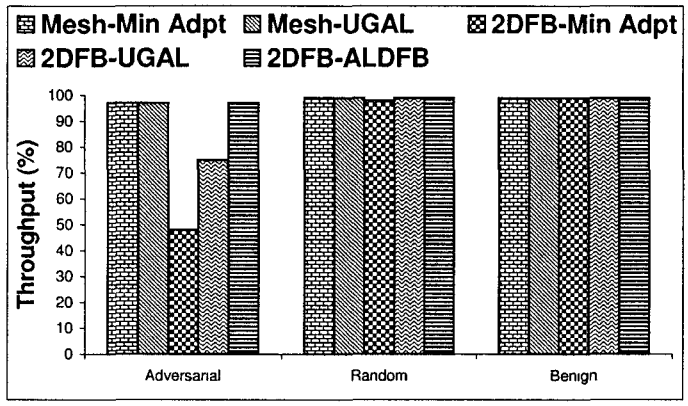


Figure 39: Throughput comparison for an injection rate of 2Gb/s

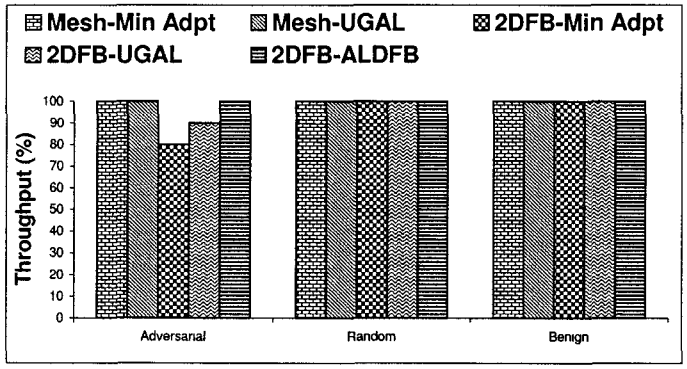


Figure 40: Throughput comparison for an injection rate of 1Gb/s

The throughput comparison is done for three different traffic patterns as mentioned before. Figure 39 shows the throughput comparison for an end-terminal injection rate of 2Gb/s which is considered as the maximum injection rate of the network. In a 2DFB network, for adversarial traffic pattern ALDFB effectively balances the load and is able to achieve a throughput very close to 100%. We notice the performance degradation of

Minimal adaptive and UGAL routing schemes in 2DFB network for adversarial traffic pattern. Minimal adaptive routing will always route packets through direct links (link corresponds to the shortest path) and will overload the direct links. This will cause a large amount of packet loss for adversarial traffic pattern. UGAL routing will try to improve the throughput by forwarding packets to a random intermediate node. Forwarding to and from the intermediate nodes are done minimally. From our simulation we observe that this technique also fails to avoid overloading few interconnecting links in an adversarial traffic pattern. ALDFB in adversarial traffic pattern shows 29.3% improvement in throughput over UGAL and 102% improvement in throughput over Minimal adaptive in the 2DFB based OCIN. ALDFB provides the same throughput as the low radix mesh network. Even though the throughput performance of the low radix mesh is good it shows poor performance in terms of latency. We can see that ALDFB helps the high radix 2DFB network to provide good throughput performance while keeping low latency. Random traffic patterns are distributed in nature and so the chances of channel congestion are comparatively less and there is no chance of channel congestion in benign traffic pattern.

The throughput comparison for an end-terminal injection ratio of 1Gb/s (half load condition) is shown in Figure 40. We can notice that Adaptive minimal and UGAL routing schemes fail to provide 100% throughput even in the half load condition for adversarial traffic conditions in a 2DFB network. At the same time ALDFB provides 100% throughput.

5.3.2 End-to-end packet delay comparison

We have compared the average end-to-end packet delay of mesh and 2DFB based 64-core OCIN for different routing algorithms. Average end-to-end packet delay comparison for an end-terminal injection rate of 2 Gb/s is shown in Figure 41.

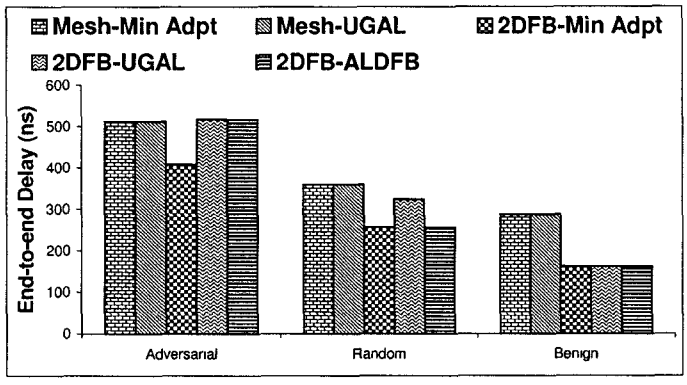


Figure 41: Packet delay comparison for an injection rate of 2 Gb/s

The low latency of the 2DFB network for random and benign traffic patterns is because of the reduced number of hop counts in the high radix 2DFB network. With an adversarial traffic pattern, almost all the routers in 2DFB are fully loaded. However, because of the increased number of routers, all the low radix routers in the mesh are not fully loaded. Moreover the channel bandwidth of mesh is twice than that of 2DFB. Because of these reasons in adversarial traffic pattern even though the hop count of mesh is larger, the overall latency will be comparable to that of a 2DFB network. For random traffic patterns, a 2DFB based OCIN with ALDFB routing provides 28.8% reduction in latency compared to a mesh based OCIN and for benign traffic pattern the reduction in latency is 43.5%. We can also notice that ALDFB shows 20.98% reduction in latency compared to UGAL for random patterns over 2DFB network.

In Figure 42 we compare the average end-to-end packet delay of mesh and 2DFB based 64-core OCIN for and end-terminal injection rate of 1Gb/s (half load condition).

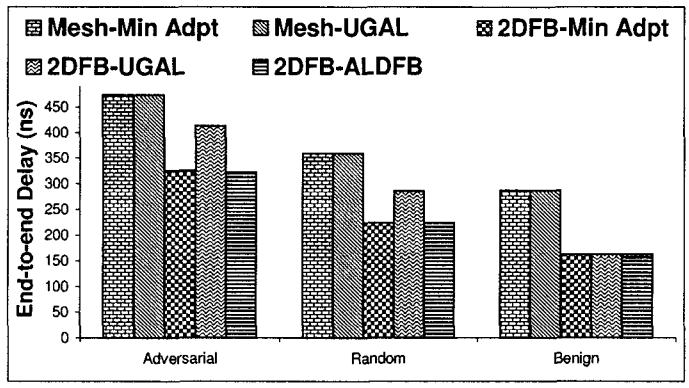


Figure 42: Packet delay comparison for an injection rate of 1Gb/s

Here we notice the reduced latency of 2DFB network over mesh network even for adversarial traffic pattern. In this reduced load condition more percentage of traffic are forwarded through the direct links. Thus, in a 2DFB network, the reduced hop count leads to reduction in latency even for adversarial traffic pattern. We observe that 2DFB with ALDFB shows 32.06% reduction in latency compared to the mesh network for adversarial traffic pattern. The reduction in latency of a 2DFB network with ALDFB over a mesh network for random and benign traffic patterns are 37.6% and 43.5% respectively. We also observe that ALDFB shows 22.03% reduction in latency for adversarial traffic pattern and 21.6% reduction in latency for random pattern compared to UGAL over 2DFB network.

These throughput and latency comparison reveal the advantage of the 2DFB network with the proposed routing scheme ALDFB. The ALDFB routing scheme succeeds in providing maximum throughput even for adversarial traffic conditions over 2DFB networks. It also exploits the reduced diameter of 2DFB network and provides significant reduction in latency for random and benign traffic patterns and in reduced load conditions. From the simulation results we observe that ALDFB outperforms UGAL and minimal adaptive routing schemes in terms of throughput and latency.

Chapter 6

ASN Applications

6.1 Overview

Several commercial and academic parallel file systems have been developed by allowing dedicated data and control paths thus demonstrating aggregate throughput scalability for very large systems. However, these systems do not scale well when I/O requests are too small to stripe across multiple nodes or when applications do many metadata operations such as file creations and deletions, fstats, and directory reads. Parallel I/O systems also do not perform well when a single client wants to read data from many storage nodes and perform a reduction operation like min-max or a search. In such cases, with high performance storage nodes, we can easily saturate the network connection to the client. Consider that if a storage node can deliver 1 Gb/s of data, 10 nodes can potentially deliver 10 Gb/s of data, thereby overwhelming a single 1 Gb/s network connection to a client. Thus, the client does not see the benefit of the parallel I/O.

We counter the problem by attacking the problem where it exists - i.e. the network. By optimizing the flow of data in the parallel I/O system, the overall performance of

the system can be enhanced. This is the principle behind an active storage network (ASN), a network with embedded intelligence. In this chapter, we demonstrate the power of an ASN by performing data processing in an intelligent switching system. File system operations that we have selected to demonstrate the power of ASN are file striping with parity and file locking. We show the performance improvements made by offloading processing from the computation node to the network.

6.2 File striping with parity

The first file system operation that we have selected to run over ASN system is file striping with parity in a parallel file system. In recent years network components, CPU and memory have made great improvement in their performance. However storage systems have not seen corresponding increases in performance. Parallel file systems have demonstrated an ability to scale aggregate throughput very well for large data transfers. In parallel file systems large size files are striped across number of storage systems and improve the aggregate I/O bandwidth. There are many parallel storage file systems which have been developed and are in common use [45–47].

Performance reliability is another critical factor in I/O intensive applications such as transaction processing and supercomputing. The reliability of the storage subsystem is important even in less crucial environments such as simple file server usage. High performance parallel file systems such as Lustre [45] and PVFS [46] have ignored storage node failures. As cluster sizes increase, these failures will become more common and more important. In order to improve the reliability, parallel file systems can make use of parity based redundancy [48]. However, the use of redundancy can reduce performance by 25% which has led most high performance computing parallel file system to disable parity. We will use an ASN to lower the performance penalty. There are different levels

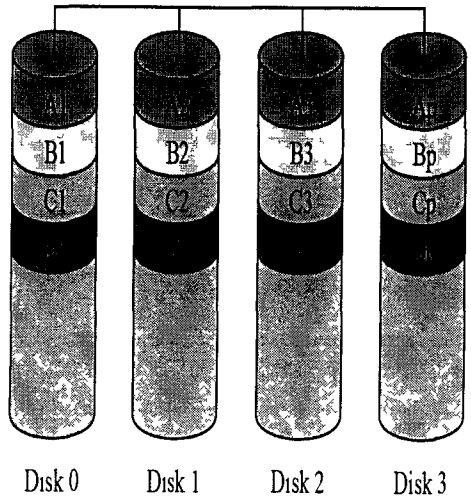


Figure 43: RAID 4-block-level striping with dedicated parity

of Redundant Array of Inexpensive Disk systems (RAID) [43]. We have selected RAID 4 (block-level striping with dedicated parity) for our implementation and is shown in Figure 43.

We have modeled a 4-ary 2DFB network with 16 end-terminals using the Omnet++ simulation tool [27]. File striping using RAID4 is implemented over this network which is shown in Figure 44. Eight terminals are considered as clients who send write requests after a random time interval. The time interval between each write request is generated using a Poisson distribution. Each client stripes a file across four servers which are selected according to the data provided by the metadata server.

In PVFS the whole file is split in to n parts and each part is buffered separately in the client side. Narayan proposed a parity mechanism for PVFS [48] whereby the parity is calculated for each block and is also buffered separately - i.e. the data to be written to each server is copied to corresponding buffers and after that, data from these buffers are written to corresponding servers independently. The amount of time taken for these splitting and copying operations is proportional to the size of the file.

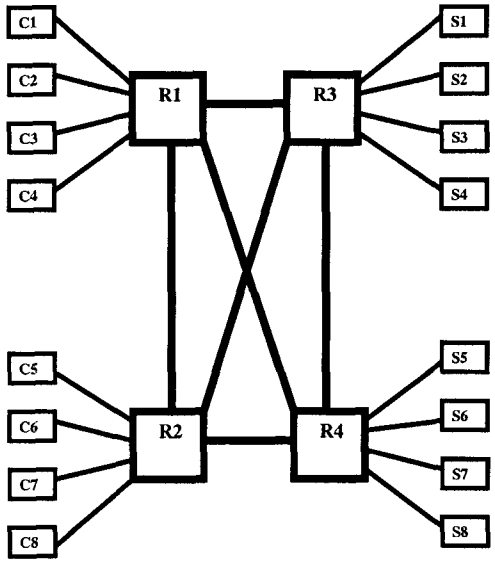


Figure 44: 4-ary, 16 terminal 2DFB network

We have implemented a simulation of this normal RAID4 file striping and compared it with the RAID4 file striping done in an ASN network. In an ASN network, the splitting of a file for different servers is done on the fly in the ASN switches. The parity calculation is also done in the switch which has a global view of all the servers used for striping. Since the splitting of the file is done on the fly in the ASN switch, the initial delay for splitting and copying large files in the client side can be avoided. The second advantage of an ASN is that the traffic provided by the clients to the network is significantly decreased because the client is not providing any parity data to the network. Parity is calculated in the ASN switch, which will be normally the last switch which is directly connected to the servers. This reduction in the traffic also helps ASN network to provide significant improvement in performance in terms of delay.

Consider a RAID4 file striping example in the ASN network shown in Figure 44. Assume that a file $F1$ in client $C1$ is to be striped between three servers $S1$, $S2$, $S3$ and the parity is written in server $S4$. The client $C1$ sends the file $F1$ as such without doing

any splitting and without any parity. Splitting of the file and the parity calculation is done in the router $R3$. In this case we can notice that compared to the normal file striping case the traffic offered to each switching element in the ASN network will be 25% less because the parity is introduced only at the output of the last switch which is directly connected to the servers used for striping.

6.2.1 Simulation results

We have implemented RAID4 file striping over normal 2DFB network and 2DFB ASN network using Omnet++ simulation tools. We have selected files of two different sizes (192KB and 768KB) for striping across four servers (three servers for file and one for parity). All the eight clients shown in Figure 44 are generating write requests after random time interval according to a Poisson distribution. The average of this random time interval is plotted in X-axis and the average of the end-to-end delay of each write operation is plotted in Y-axis.

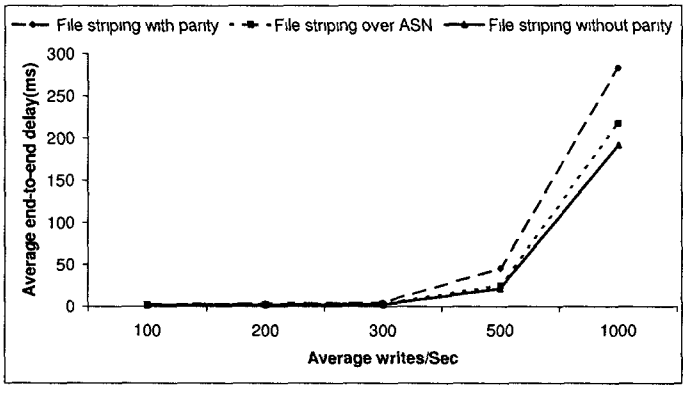


Figure 45: File stripe (file size-192kB)

Figure 45 shows the delay comparison of the file stripe for a file size of 192 KB. The stripe size used is 64 KB. An end-to-end delay comparison for lower write rates for the same file size is depicted in Figure 46

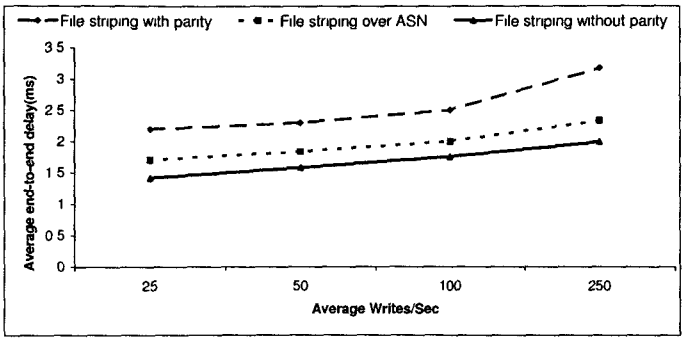


Figure 46: File stripe for lower write rates (file size-192kB)

Figure 47 shows the delay comparison of the file stripe for a file size of 768 KB. The stripe size used is 64 KB. An end-to-end delay comparison for lower write rates for the same file size is depicted in Figure 48

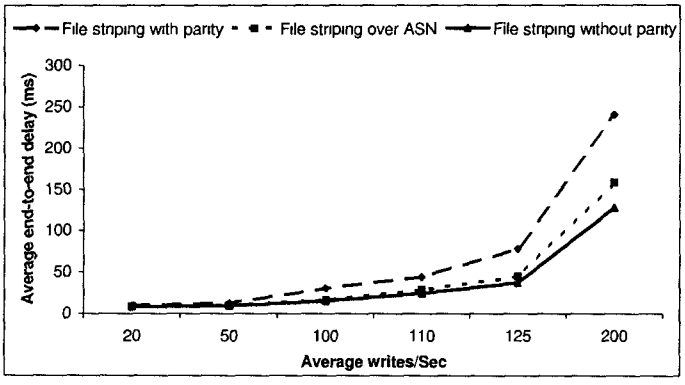


Figure 47: File stripe (file size-768kB)

It can be seen that using an ASN provides up to 44% decrease in latency compared to a non-ASN implementation, Moreover, the increase in latency compared to no parity is only 13% making it more acceptable to use redundancy in a high performance file system.

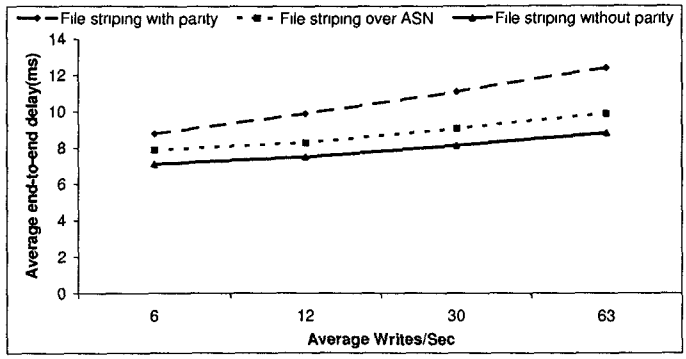


Figure 48: File stripe for lower write rates (file size-768kB)

6.3 File locking

In parallel file systems large size files are striped across number of storage systems and therefore high-level I/O operations should maintain atomicity for the correctness of the data. For example, assume one process P_1 is writing a contiguous file which spans two I/O servers I_1 and I_2. Assume process P_1 writes to I_1 first and then I_2. Assume another process P_2 reads I_2 first and then I_1. In this case P_2 may see only part of P_1's write, violating atomicity.

File locking is the most common technique used for maintaining atomicity. Parallel virtual file systems (PVFS) today do not have an optimum file locking technique. Three different locking methods that can be applied to a parallel file system are given below.

POSIX Locking

Read and write operation in POSIX use a pointer to a contiguous region of memory, the current file pointer location, and a count, to access data. The locking interface of POSIX will acquire all locks to the necessary file regions before doing any I/O operation. Even though POSIX lock ensures atomicity it has several drawbacks. The number of lock requests required should be at least equal to the number of noncontiguous file

regions which will increase the traffic significantly. The situation becomes worst due to splitting the locks on lock server boundaries.

List Locking

In list locking, a list based description is used to specify multiple noncontiguous regions in both memory and file [53]. This list based description provides a way to take advantage of the high-level I/O information available from data types. If the application performs a unstructured data access, lists of offset and lengths are used to describe the access patterns to the lock servers. In such cases if the size of the noncontiguous regions is small, the length of the list locking description can be comparable to that of the actual data.

Datatype Locking

In datatype locking, structured data access are concisely described with a derived datatype. The datatype access pattern can consists of a tree of datatypes as opposed to the offset and length pairs used in list locking. Datatype locking reduces network traffic and the number of lock requests when moving the access pattern description across a network. Datatype locking breaks down into list locking if the access pattern has no regularity.

Two phase Locking

When multiple locks are acquired and released in a parallel file system, the main problem that can appear is deadlock. In deadlock, execution of all transactions is stalled because of the mutual blocking between transactions. Deadlock can be avoided

by making use of two phase locking protocol for locking the files [44]. As the name indicate two phase locking consists of two phases of operations, expanding phase and shrinking phase. In expanding phase the locks are acquired and no locks are released. In shrinking phase locks are released and no locks are acquired. All the files are acquired and released in order and the order can be imposed based on the file offset. This serialization in acquiring and releasing locks avoid the possibility of deadlock.

Hybrid Locking

A scalable distributed lock manager (DLM) architecture proposed in [6] uses hybrid lock protocols to maintain the atomicity. Hybrid lock protocol makes use of Two phase locking along with an optimistic approach to get all the locks in parallel. At first the hybrid protocol tries to get all the locks from the servers by sending parallel requests to all servers. Then it releases the locks that are out-of-order. For example assume that client C1 in the Figure 44 wants to acquire locks on offset-length pairs (1, 9), (2, 9), (3, 9), and (4, 9). Assume that file (1,9) is in server S1, file (2, 9) is in server S2, file (3,9) is in server S3 and file (4,9) is in server S4. At first client C1 optimistically tries to acquire all locks by sending parallel requests to all these servers. Client C1 waits for the responses from all the lock servers and then revokes locks which are out-of-order. If client C1 has received locks from servers S1, S3 and S4, then it has to release the locks from the servers S3 and S4. After releasing these locks client C1 will make use of two phase file locking to attain locks from servers S3 and S4. In Hybrid locking one can combine optimistic and two phase locking in number of ways. One-try Lock protocol is one way to implement Hybrid locking and in this protocol, a client first tries the optimistic lock protocol to acquire all locks in parallel. If it fails to acquire

all the locks in parallel, it releases the out -of-order locks and then reverts to the two-phase lock protocol of acquiring locks in-order. Most of the I/O access patterns are not overlapping and therefore one-try lock protocol can speed up the application.

File locking in ASN network

In all the file locking protocols mentioned above, the client generates a lock request and it is transmitted to the file locking servers and the response is transmitted back to the client. The number of requests and responses are proportional to the number of noncontiguous file segments and the number of conflicting file segments. These requests and responses add additional traffic load to the system and affect the overall performance. Ching shows that there is a 30% performance degradation when using locking [6]. An ASN can be considered as a better solution to decrease the additional traffic load introduced by this lock requests and responses. The ASN reduces the traffic by offloading the file locking protocols to the ASN switch. In an ASN, all the lock requests are processed by ASN switching elements and it will send the responses back to the clients, thus eliminates the traffic between switch and lock servers.

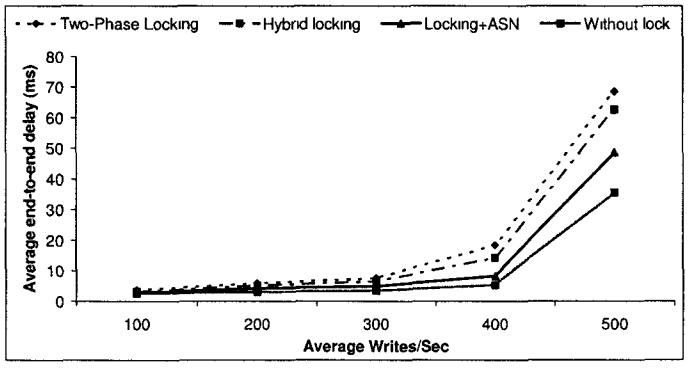


Figure 49: File stripe (file size-768kB)

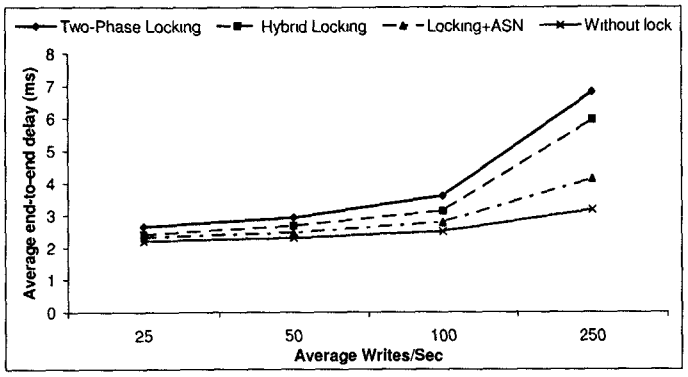


Figure 50: File stripe for lower write rates (file size-768kB)

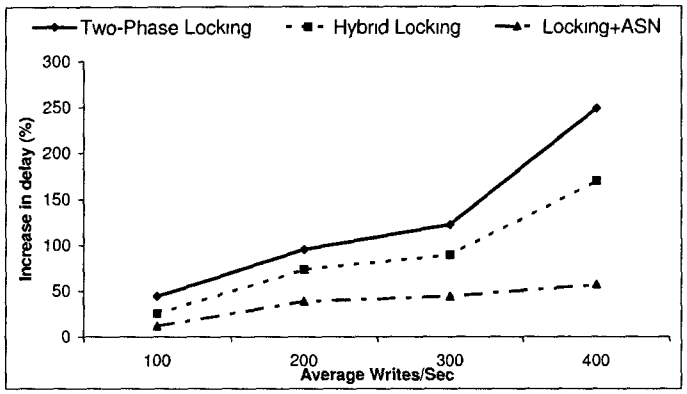


Figure 51: File stripe for lower write rates (file size-768kB)

As an example consider client C1 in Figure 44 is attempting to write a file which is spanned across servers S1, S2, S3 and S4. At first client C1 collect all the metadata information from the metadata server. The metadata server provides the information regarding the servers, offset values and the address of the ASN switch which handles all the locks for that file. The ASN switch selected to handle all the locks of a file should be the nearest switch which has a global view of all the servers in which the file has to be striped. In this example, an ASN can provide optimum performance if we select R3 as the switch which handle all the locks of the file. After collecting the metadata information client C1 will send the lock request to the switch R3. After getting the lock request the switch will check the status of the locks corresponding to each servers

and provides the lock if it is available. Otherwise the lock request is loaded to the wait-request-queue and the lock is issued in order according to the availability. All the lock information of the requested file will be there in the local memory of the active switch. So the switch can effectively handle the issue of the locks in the order of file offset and there will not be any chance of deadlock. An ASN switch can be implemented using NetFPGA card as discussed in Chapter 3. A NetFPGA card has an SRAM capacity of 4.5 MB and DRAM capacity of 64 MB. This memory capacity is sufficient for storing the lock information of the files even in larger networks. Since one active switch is handling all the lock information of a particular file the client C1 need to send only one lock request to the active switch R3. Moreover, the active switch need not send requests to any server because all the lock information is available in the switch and the switch is controlling the issue of locks. Thus compared to any other file locking system, the ASN based file locking offers less traffic to the network and provides an overall performance improvement.

6.3.1 Simulation results

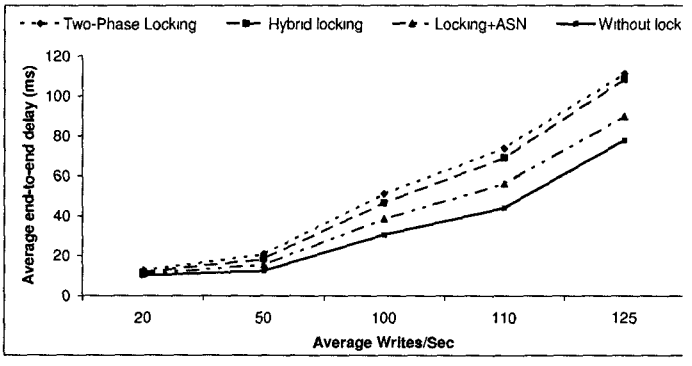


Figure 52: File stripe (file size-768kB)

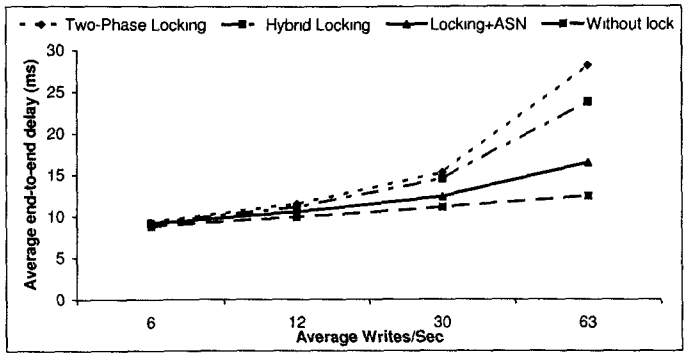


Figure 53: File stripe for lower write rates (file size-768kB)

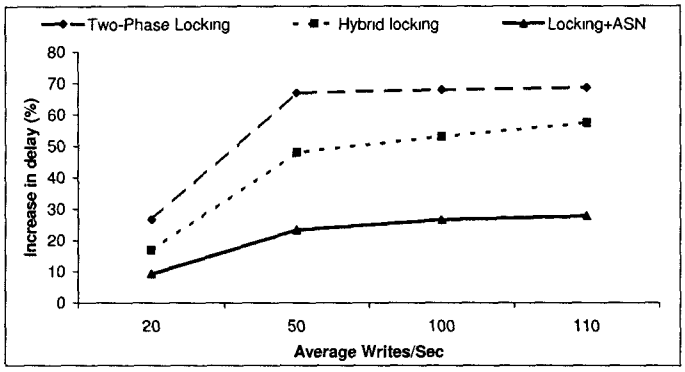


Figure 54: File stripe for lower write rates (file size-768kB)

We have implemented two phase locking and Hybrid locking for parallel file systems using the Omnet++ simulation tool and compared the performances with the ASN based file locking system. Figure 49 shows the delay comparison of different file locking protocols for a file size of 256 KB. The stripe size used is 64 KB. End-to-end delay comparison for lower write rates for the same file size is depicted in Figure 50. For comparison we have also plotted the file striping without using any locking protocol. We can notice that ASN based locking protocol provides significant reduction in End-to-end delay for the parallel write operation. Figure 51 shows the percentage increase in delay provided by two phase locking, hybrid locking and ASN based locking with respect to the file striping without using any locking protocol for a file size of 256 KB.

For an average file writes/second of 400, the network will be close to saturation and in this case the end-to-end delay of file locking in ASN is 54% lesser than two phase locking and 41% lesser than hybrid locking. For a low average file writes/Second of 100, the end-to-end delay of file locking in ASN is 22% lesser than two phase locking and 10% lesser than hybrid locking.

Figure 52 shows the delay comparison of different file locking protocols for a file size of 1 MB. The stripe size used is 64 KB. End-to-end delay comparison for lower write rates for the same file size is depicted in Figure 53. Figure 54 shows the percentage increase in delay provided by two phase locking, Hybrid locking and ASN based locking with respect to the file without using any locking protocol for a file size of 1 MB. For an average file writes/Second of 110, the network will be close to saturation and in this case the end-to-end delay of file locking in ASN is 24% lesser than two phase locking and 18% lesser than hybrid locking. For a low average file writes/Second of 20, the end-to-end delay of file locking in ASN is 13% lesser than two phase locking and 6% lesser than hybrid locking. We can notice that the difference in delay between ASN based File locking and other locking protocols increases as the network approaches saturation. We can also notice that the ASN based file locking provides considerable reduction in delay even for reduced load conditions.

Chapter 7

Conclusion and Future Work

In this dissertation, we presented a study of different aspects of an active storage network (ASN). We have designed and implemented an innovative 2-dilated flattened butterfly (2DFB) topology for an ASN that can provide a nonblocking behavior in the network. We have developed an adaptive load balancing deadlock free routing protocol for the 2DFB network. We have also presented two applications that can take advantage of ASN. In this chapter, we conclude our work and discuss about the opportunities for further investigation.

7.1 Conclusions

An active storage network consists of intelligent switches which are capable of doing data processing and data computation in line speed. ASNs can improve data intensive computations by offloading data transformation and reduction operations to the network switch. Offloading appropriate processing from end machines to the active switches also reduces the net traffic flow and decreases the end-to-end delay. One of

the critical components which enhance the performance of ASN is the switching topology. We have developed and implemented a nonblocking switching topology 2DFB for ASN. We have proposed a procedure to develop a conflict-free static routing schedule for 2DFB networks. We have compared the performance of a k-ary 2DFB with other nonblocking topologies like folded-Clos and DBHC and we have seen that 2DFB outperforms other topologies in terms of speed. Then we compared the cost of the k-ary 2DFB with other nonblocking topologies and we have verified that the cost of a k-ary 2DFB is lower than other nonblocking topologies. We have also implemented an 8-terminal hardware switch using NetFPGA boards and verified the nonblocking nature of the 2DFB network.

In any switching topology there should be a routing schedule which route packets efficiently from the source node to destination node. Recent studies reveal the advantages of adaptive routing schemes over static routing schemes. We have introduced a deadlock free adaptive load balanced routing algorithm called ALDFB for a 2DFB switching network. ALDFB is designed to exploit all positive topological properties of a 2DFB network. The algorithm takes full advantage of the reduced diameter and improved path diversity of 2DFB network. It provides better load balancing by allowing one non-minimal forwarding in each single dimension of 2DFB network. This algorithm also provides good performance for local and benign traffic by providing priority to the selection of direct links. We have compared the performance of ALDFB with the UGAL and minimal adaptive algorithms running over the same 2DFB network. We have also compared the performance of ALDFB over 2DFB with three other well-known topologies with the best available routing schemes. We have observed that ALDFB provides better throughput and reduced latency for all the traffic patterns.

Finally we demonstrated the power of ASN network by implementing two applications over Omnet++ simulation platform. We have selected file striping with parity in a parallel file system as the first application. Unlike the traditional way to splitting the file and calculating parity in the client side, in ASN these operations are done on the run in active switches. By properly selecting the switch to perform these actions the network traffic can be reduced significantly. ASN will also have the advantage of having less software processing time because the data processing is done in the intelligent hardware switch. The second application that we have implemented over ASN is file striping using file locking protocol. In ASN all the locking operations are done in the active switch which has a global view of all the servers used to stripe the file. This reduces the number of lock requests and responses between client and servers and helps ASN to provide significant performance improvement. We have compared the performance of ASN based file locking with conventional two phase locking and hybrid locking protocols. Both the applications implemented in ASN provide a significant reduction in the end-to-end delay compared to other systems.

7.2 Future Work

We have implemented a 2DFB network using NetFPGA boards for a network size of 8. We have used static routing schedules for the performance comparison. In future, we are interested to observe the performance of ALDFB routing schedule in the hardware system. Similarly we also interested to implement the two ASN applications that we have implemented in the simulation platform. Future research will also include the design of NetFPGA switch specifically for ASN. The challenging hardware design problem will be handling the data processing in the case of packet drop. There should be some form of handshaking protocol between client and the switch to ensure the

retransmit of the dropped packets. Also there should be some mechanism to reproduce the correct order of the processed data. Researchers can also experiment with various applications that can be effectively implemented over ASN.

Bibliography

[1] NetFPGA, *http://www.netfpga.org*

[2] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," *Proc. of the ACM Symposium on the Theory of Computing*, pp. 263-277, 1981.

[3] L. Gravano, G. Pifarre, P. Berman and J. Sanz, "Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, pp. 1233-1252, 1994.

[4] D. Linder and J. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *ACM Trans. on Computer Systems*, vol. 40, pp. 2-12, 1991.

[5] A. Singh, "Load-Balanced Routing in Interconnection Networks," *PhD thesis*, Stanford University, 2005.

[6] Avery Ching, Wei keng Liao, Alok Choudhary, Robert Ross and Lee Ward, "Noncontiguous locking techniques for parallel file systems," *Proc. of the 2007 ACM/IEEE conference on Supercomputing*, pp. 2-12, 2007.

[7] J. Kim, W. J. Dally and D. Abts, "Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks," *Proc. of the International Symposium on Computer Architecture (ISCA)*, pp. 126-137, Jun 2007.

[8] L. N. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Computers*, vol. 33, pp. 323-333, 1984.

[9] C. Clos, "A Study of Non-Blocking Switching Networks," *The Bell System Technical Journal*, vol. 32, pp. 406-424, 1953.

[10] V. Benes, Optimal rearrangeable multistage connecting networks, *The Bell System Technical Journal*, vol. 43, pp. 16411656, July 1964.

[11] S. Scott, D. Abts, J. Kim and W. J. Dally, "The BlackWidow High-radix Clos Network," *Proc. of the International Symposium on Computer Architecture(ISCA)*, Boston, MA, Jun, 2006.

[12] J. Kim, W. J. Dally, S. Scott and D. Abts, "Cost-efficient dragonfly topology for large-scale system," *IEEE Micro Top Picks*, vol. 29, pp. 3340, 2009.

[13] W. J. Dally and B. Towles, "Principles and Practices of Interconnection Networks," *MorganKaufmann*, 2004.

[14] A. Thamarakuzhi and J. A. Chandy, "2-Dilated Flattened Butterfly: A nonblocking switching network," *International Conference on High Performance Switching and Routing (HPSR 2010)*, Jun 2010.

[15] A. Thamarakuzhi and J. A. Chandy, "2-Dilated flattened butterfly: A nonblocking switching topology for high-radix networks," *Elsevier Computer Communications*, May 2011.

[16] A. V. Allenov and V. S. Podlazov, "Throughput of the Set of Ring Channels II. Ring Switches," *Avtom. Telemekh*, pp. 162-172, 1996.

[17] V. S. Podlazov, "Conflict-free Static Decentralized Routing for Ring Switches and Hypercubes," *Avtom. Telemekh*, pp. 79-89, 1999.

[18] V. S. Podlazov, "Generalized Crossed Rings - Multirings with a Decreased Degree of Node," *Avtom. Telemekh*, vol. 68, pp. 160-170, 2007.

[19] V. S. Podlazov, "Nonblocking Conditions for Multiring Commutators and Generalized Hypercubes in Arbitrary Commutations. I. Internodal Commutation. Multirings," *Avtom. Telemekh*, pp. 118-126, 2001.

[20] V. S. Podlazov, "Nonblocking Conditions for Multiring Commutators and Generalized Hypercubes for Arbitrary Commutations. II. Generalized Hypercubes. Intranode Commutation," *Avtom. Telemekh*, pp. 114-124, 2001.

[21] V. S. Podlazov, "Nonlockability in Multirings and Hypercubes at Serial Transmission of Data Blocks," *Avtom. Telemekh*, vol. 63, pp. 120-130, 2002.

[22] The Insider's Guide to Microprocessor Hardware, *http://www.mdronline.com/*

[23] AuroraWebsite, *http://xilinx.com/products/design_resources/conn_central/grouping/aurora.htm*

[24] User Guide, *http://xilinx.com/support/documentation/ip_documentation/aurora_8b10b_ug353.pdf*

[25] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller and Nick McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pp. 1-9, 2008.

[26] OpenSS7 IPERF Utility Installation and Reference Manual, *http://www.openss7.org/iperf_manual*

[27] Varga, András, "The OMNeT++ Discrete Event Simulation System," *Proceedings of the European Simulation Multiconference (ESM'2001)*, 2002.

[28] ELPEUS, *http://www.elpeus.com/*

[29] Jose F. Martnez, Josep Torrellas and Jose Duato, "Improving the Performance of Bristled CC-NUMA Systems Using Virtual Channels and Adaptivity," *International Conference on Supercomputing (ICS)*, Jun 1999.

[30] Zhiyong Liu and David W. Cheung, "Oblivious routing for LC permutations on hypercubes," *ELSEVIER Parallel Computing*, vol. 25, pp. 445-460, 1999.

[31] TMCSCSI, *http://www.tmcscsi.com/QSFP_cables.shtml*

[32] SUN, *https://shop.sun.com/store/productsa/27a63daa-0dc8-11de-8c47-080020a9ed93*

[33] J. Kim, W. J. Dally and D. Abts, "Adaptive Routing in High-radix Clos Network," *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC06)*, 2006.

[34] A. Thamarakuzhi and J. A. Chandy, "Adaptive Load Balanced Routing for 2-Dilated Flattened Butterfly Switching Network," *International Conference on Networking (ICN 2011)*, January 2011.

[35] A. Thamarakuzhi and J. A. Chandy, "Design and Implementation of a Nonblocking 2-Dilated Flattened Butterfly Switching Network," *IEEE Latin-American Conference on Communications 2010*, September 2010.

[36] T. Bjerregaard and S.Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv*, vol. 38, 2006.

[37] D. Seo, W.T. Lim, N. Rafique and M. Thottethodi, "Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks," *In Proc. of the International Symposium on Computer Architecture (ISCA)*, pp. 432-443, 2005.

[38] M. B. Taylor, W. Lee, S. Amarasinghe and A. Agarwal, "Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures," *International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 341-353, 2003.

[39] P. Gratz, C. Kim, R. McDonald, S. Keckler and D. Burger, "Implementation and Evaluation of On-Chip Network Architectures," *International Conference on Computer Design (ICCD)*, 2006.

[40] A. Agarwal, L. Bao, J. Brown, B. Edwards, M. Mattina, C.C. Miao, C. Ramey and D. Wentzlaff, "Tile Processor: Embedded Multicore for Networking and Multimedia," *Hot Chips*, 2007.

[41] J. Kim, J. Balfour and W. Dally, "Flattened Butterfly Topology for On-chip Networks," *International Symposium on Microarchitecture*, pp. 172-182, 2007.

[42] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagle, "Active disks for large-scale data processing," *IEEE Computer*, vol. 34, pp. 68-74, 2001.

[43] D. A. Patterson, G. A. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *In Proc. of 1988 ACM SIGMOD International Conference on Management of Data*, pp. 109-116, 1988.

[44] P. A. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems," *Addison-Wesley*, 1987.

[45] P. J. Braam and R. Zahir, " Lustre technical project summary.," *Technical report, Cluster File Systems*, Inc., Mountain View, CA, July 2001.

[46] P. H. Carns, W. B. L. III, R. B. Ross, and R. Thakur, " PVFS: A parallel file system for linux clusters," *In Proceedings of the Annual Linux Showcase and Conference*, pages 317327, Oct. 2000.

[47] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, " Ceph: A scalable, high-performance distributed file system," *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA, page 22, Nov. 2006.

[48] S. Narayan and J. A. Chandy, " Parity Redundancy in a Clustered Storage System," *Fourth International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI 2007)*, 2007.

[49] Gore, *http://www.gore.com/electronics*

[50] W.J. Dally, " Virtual Channel Flow control," *IEEE Trans. on Parallel and Distributed systems*, vol.3, no. 2, pp 194-205, Mar 1992.

[51] W.J. Dally and H.Aoki, " Deadlock-Free adaptive routing in multicomputer networks using virtual Channels," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4. pp 466-475. Apr. 1993.

[52] W.J. Dally and C.Seitz, " Deadlock free message routing in multiprocessor interconnection networks," *IEEE transactions on Computers*, pp 547553, 1987.

[53] A. Ching, A. Choudhary, W. K. Liao, R. Ross, and W. Gropp, " Noncontiguous I/O through PVFS," *Proceedings of the IEEE International Conference on Cluster Computing*, September 2002.