Distributed Control Schemes for Fast Arbitration in Large Crossbar Networks

Joydeep Ghosh

Naveen Krishnamurthy

Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712-1084

Anujan Varma

Computer Engineering Department
University of California
Santa Cruz, CA 95064

February 25, 1993

Joydeep Ghosh was supported in part by an NSF Research Initiation Award MIP 9011-787, Texas Advanced Tech. Project (TATP) grant No. 14-9712, and by a faculty development award from TRW Foundation. Anujan Varma was supported by NSF Young Investigator Award MIP-9257103 and NSF Grant No. MIP-9111241.

Abstract

In a large nonblocking crossbar switch, the controller often becomes a bottleneck in terms of both performance and reliability. In this paper, we present a number of schemes for distributing the setup function among multiple controllers, thus improving both the performance and reliability of the switch. The controllers are symmetric and operate in parallel. The simplest of these schemes is the *symmetric triangular scheme*, where each request is serviceable only by one of the controllers, chosen based on the addresses of the ports involved in the request. This scheme has only a limited degree of fault-tolerance, because a controller-failure can be tolerated only by modifying the request-distribution, causing a temporary disruption of service. With N ports and K controllers, the number of buses needed to support nonblocking operation in this scheme is approximately $N\left(1-\frac{1}{2\sqrt{K}}\right)$.

In the *chessboard scheme*, every request is serviceable by one of two predetermined controllers — a primary controller and secondary controller. A request is routed to the secondary controller if the primary controller is not available. This scheme can tolerate any single failure among the controllers, but requires N buses for nonblocking operation. The *dynamic distribution scheme with global load-balance* is similar to the chessboard cheme, but routes each request dynamically to balance the load between the pair of controllers designated to handle the request. The number of buses for nonblocking operation in this case is shown to be $N\left(1-\frac{1}{2^{\lfloor \log_2 K \rfloor}}\right)$. This scheme requires sharing of the load information among the controllers. Such sharing is eliminated in the *distribution scheme with handovers*. In this scheme, each request is first routed to a pre-assigned primary controller, which may then hand over the request to any of the remaining controllers based on local information, if it is unable to satisfy the request. The numner of buses needed for nonblocking operation for this scheme is $N-K$. A comparison of the scheme by simulations is presented.

Contents

List of Figures

1 Introduction

Crossbar switches are used extensively as multiprocessor interconnection networks and for communication switching. Small crosspoint arrays are readily implemented on a single chip, while an array of crosspoint switching chips can be used to build larger switches. The design and implementation of several single- and multi-chip crossbar switches are described in recent literature. For example, Miracky, et al. [11] describe an implementation of a 72×72 crossbar switch on a single chip. Colbry, et al. [3] describe the design of a 64×128 crossbar with 40-bit data-paths for processor–memory interconnection using multiple chips. IBM has used a 64×64 CMOS crossbar chip, supporting a data rate of 40 Mbits/second per port, in their switching system for interconnection of serial I/O channels [7]. Several crossbar chip implementations for broadband packet-switching are also described in [1].

With current VLSI and packaging technologies, the circuit-complexity of $O(N^2)$ for an $N \times N$ crossbar switch is no longer the limiting constraint in a single-chip implementation. Rather, the primary limiting factors to the cost-effective single-chip realization of large crosspoint arrays now are (i) pin-out constraints and (ii) the allowable tolerance levels on the inductive noise generated by the line-drivers driving the output leads of the chip. The latter is known as the *simultaneous-switching* noise or the *Delta-I* noise [12, 4]. The simultaneous-switching noise is particularly severe in large crossbar switching chips because they have a large number of data output lines.

Both the pin-out constraint and the simultaneous-switching noise can be alleviated by constructing a large crossbar from one-sided crosspoint switching chips [5, 14, 9]. One-sided crossbar networks allow a pair of ports to be connected using one of many available internal switching paths. In simple terms, a one-sided crossbar switch with N ports consists of a number of buses, each of which can be used to connect any given pair of ports; thus, nonblocking operation is attained when the number of buses is at least $N/2$. In addition, by choosing the switching paths properly, it is possible to distribute the active line drivers uniformly over the chip matrix, thus reducing the Delta-I noise in each chip significantly in comparison to the worst case when all the drivers are active [15, 9].

The design and properties of large one-sided crossbars are presented in [9] for nonblocking operation, and in [15] for rearrangeable operation. In both cases, requests for connections and disconnections are satisfied by a central controller; the controller sets up the routing path for each connection by selecting a free bus from the available ones such that the active drivers are distributed among the individual chips as uniformly as possible. As the path-setup time often dominates the overall connection-setup time, it is important to perform the setup operation efficiently.

One approach to improving the service rate of the controller is pipelining; for example, a two-stage pipeline is employed in [6] for the path-selection process. A neural-network implementation of the controller has also been described, with some improvement in arbitration speed [10]. Still, a centralized controller becomes the bottleneck when the arrival rate for connection requests becomes comparable to the time required to set up a connection, and when several requests can occur simultaneously or in quick succession. The controller bottleneck is particularly a problem in real-time systems where connection-latencies must be kept low. Low latency is also critical if the crossbar is used for processor-memory interconnection in a multiprocessor. In such systems, it is desirable to maintain the time taken to access a remote memory comparable to that for accessing local memory, i. e., of the order of 100 ns. Such setup times are difficult to attain for large crossbars. For example, the crossbar switch described in [7] for interconnection of IBM's serial I/O channels has a minimum setup time of 800 ns.

A second problem with centralized control is that the controller becomes a central point of failure for the switch. Both the problems can be remedied by resorting to a distributed control scheme where connection requests are serviced by multiple controllers operating in parallel. In addition to increasing the rate at which connection-requests can be serviced, such a scheme also improves the reliability of the switch. A fault in one of the controllers can now be tolerated at the expense of some degradation in performance.

In this paper, we investigate the use of multiple controllers in large one-sided switching networks with the objective of improving both the performance and the reliability. In our schemes, each controller is responsible for a subset of the buses in the one-sided crossbar; a request for

connection between two ports is routed to one of the controllers based on the addresses of the ports involved in the connection. Our schemes differ primarily in the way the port-pairs are assigned to controllers. We restrict ourselves to nonblocking operation, where any request to connect two idle ports can be satisfied without disturbing existing connections. We evaluate the performance of a number of schemes for distributing control in both the fault-free case and with one faulty controller.

This paper is organized as follows: Section 2 introduces the architecture of one-sided crossbar networks to provide the necessary background. Section 3 presents a simple scheme for distributing requests among multiple controllers, called the *symmetric triangular scheme*. In this scheme, each request is serviceable only by one of the controllers, chosen based on the addresses of the ports involved in the request. In the event of a controller-failure the assignment of port-pairs to controllers must be modified, thus causing a temporary disruption of service. With this scheme, the number of buses needed to support nonblocking operation with K controllers is approximately $N\left(1-\frac{1}{2\sqrt{K}}\right)$.

Distributed control schemes with fault-tolerance are the subject of Sections 4, 5, and 6. These schemes allow the switch to continue operating in the event of the failure of one of the controllers. In the *chessboard scheme*, presented in Section 4, every request is serviceable by one of *two* predetermined controllers — a primary controller and secondary controller. A request is routed to the secondary controller if the primary controller is not available. Section 6 describes a similar scheme, but now each request is routed to one of two controllers so as to balance the load between them. The minimum number of buses for nonblocking operation in this case is shown to be $N\left(1-\frac{1}{2^{\lfloor \log_2 K \rfloor}}\right)$. This scheme requires the controllers to share the load information among themselves. Section 7 introduces a scheme which does not require such global information. In this case, each request is sent to a pre-determined primary controller, which may then hand over the request to any of the remainining controllers if it is unable to satisfy the request. Detailed simulation results evaluating the performance of the distributed control schemes are given in Section 8. Finally, some concluding remarks are given in Section 9.

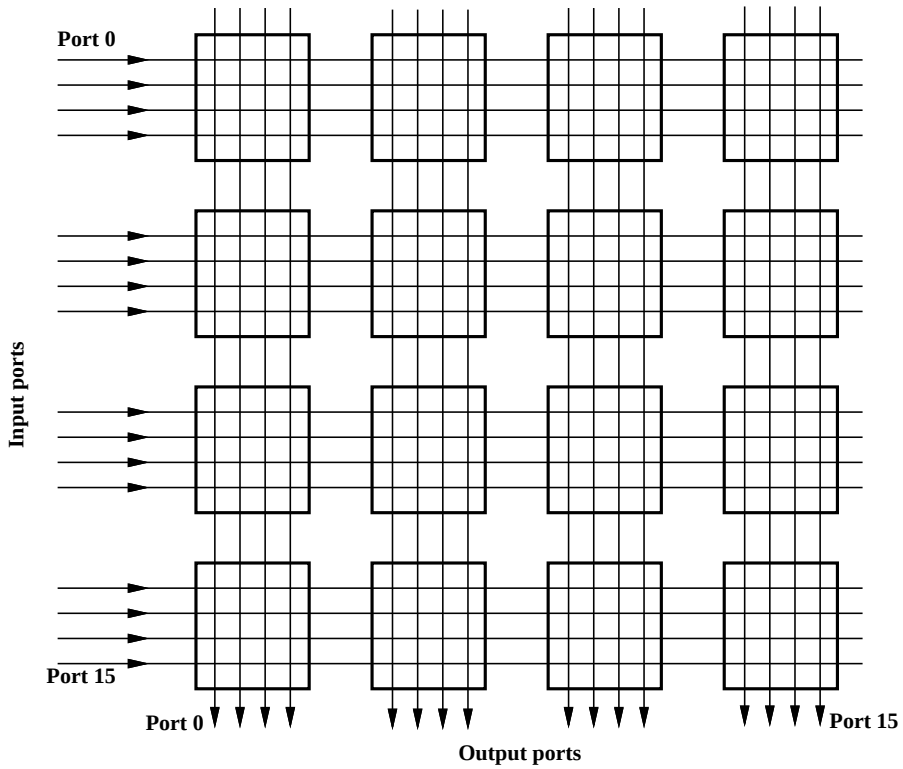


Figure 1: A conventional two-sided crossbar switch.

2 One-sided Crosspoint Networks

A conventional (two-sided) crossbar network consists of a set of input lines and a set of output lines, conceptually placed perpendicular to each other, with switches placed at each point where the lines cross. Large crossbars are implemented by partitioning the switching matrix into smaller rectangular blocks and assigning each block to a chip. For example, Figure 1 illustrates a two-sided crosspoint matrix with 16 inputs and outputs, constructed from sixteen 4×4 switching chips. Note that there is a unique path between every pair of input and output ports.

An alternate way of designing crossbar networks is by means of one-sided crosspoint switching chips. A one-sided crosspoint matrix consists of a set of port lines and a set of bus lines placed perpendicular to each other, with switching elements placed at the points of intersection. A connection between two port lines can be established through any of the internal (column) buses. In the full-duplex implementation, each port line is actually two wires, one for communication

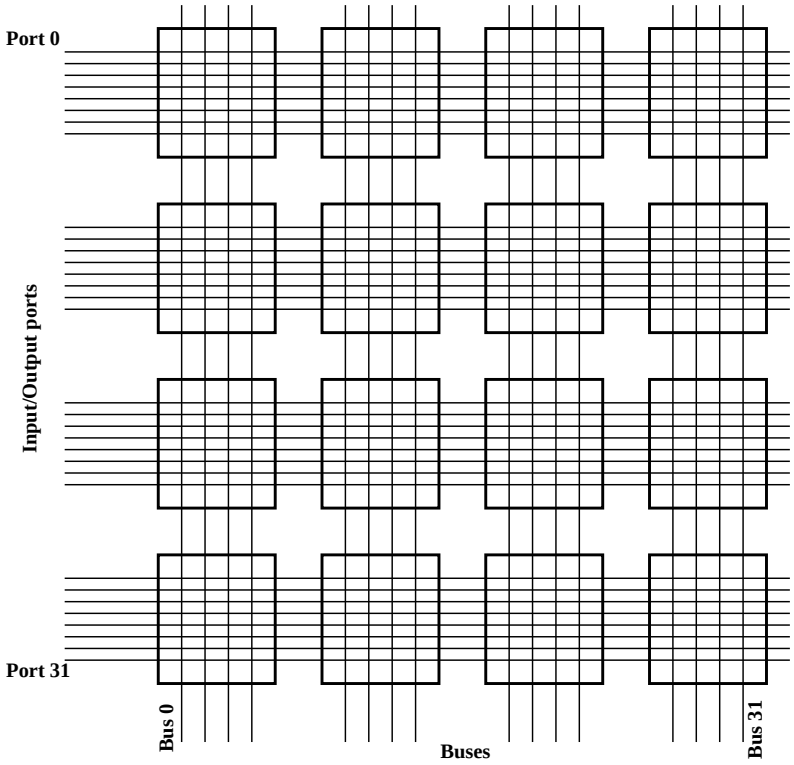


Figure 2: A one-sided crosspoint switch with 32 ports.

in each direction. Similarly, each bus line actually consists of two wires. The architecture and implementation details of one-sided crossbar networks are given in [9].

Figure 2 shows a one-sided network with $N = 32$ input/output ports and 16 internal buses. The switch matrix is constructed out of sixteen crosspoint chips, each of size 8×4. Note that, in a full-duplex implementation, each horizontal line in Figure 2 represents a full-duplex channel and each vertical bus a pair of bidirectional lines. An important property of the one-sided network is the presence of multiple paths for connecting any two ports — any of the available column buses can be used to connect a given pair of ports. One way to exploit this flexibility is to to allocate the buses so as to distribute the active line-drivers uniformly among the switching chips, thus minimizing the simultaneous-switching noise. Ghosh and Varma [9] describe such bus-allocation algorithms for one-sided multi-chip crossbar networks.

An input/output port in a one-sided crossbar can make two types of requests to the crossbar

controller:

1. A *connect request* to set up a communication path to another port, or

2. a *disconnect request* to remove an existing connection between two ports.

A connection between a source port and a destination port in the one-sided switch is established by locating an unused column-bus (internal bus) and then turning on the two crosspoints where the source and destination rows intersect with the selected bus. If these two crosspoints are within the same chip, no column drivers need be activated, and no column bus is used. This is referred to as an *internal connection*. Otherwise the signal needs to be routed off-chip. This requires the activation of a column driver in the source-chip as well the as the destination-chip to provide a full-duplex path. This is referred to as an *external connection*.

In this paper, we consider one-sided crosspoint matrices with N ports and M internal buses. The matrix is constructed from a rectangular array of one-sided switching chips with r rows and c columns, and the size of each chip is $n \times m$. Thus, $N = r \times n$ and $M = c \times m$. We label the N ports from 0 through $N - 1$.

When all the N ports are in use and all connections are external, the number of column buses needed to set up the connections is $N/2$. This is the lower-bound for non-blocking operation with a centralized controller. When control is distributed among multiple controllers, however, this lower-bound may no longer apply. In particular, if each controller is allowed to use only a subset of the M buses, and if each controller is allowed to service only a subset of the possible connections, more than $N/2$ buses will be needed to operate the switch in nonblocking mode.

In all the schemes for distributing control, we assume there are K identical controllers $C_0, C_1, \ldots, C_{K-1}$ in the system. A request originating at a switch port is routed to one of the controllers based on the addresses of the ports involved in the request and the distribution scheme used. In our schemes, a given request is serviceable only by a subset of the K controllers. It is necessary to keep the size of this subset small to limit the complexity of request-distribution from the ports to the controllers. In general, the distribution scheme can be static or dynamic: In a

static scheme, a request involving two given ports is always routed to the same subset of controllers. In a dynamic scheme, the servicing controller can be selected based on system conditions such as relative loading of the controllers. All the schemes described in this paper are static in nature, although some allow a choice of one out of two controllers depending on the load on the individual controllers.

In our schemes for distributing control, each of the K controllers can use only a subset of the buses for making a connection. That is, the set of buses is divided into disjoint subsets and each subset is assigned to a distinct controller. This allows the controllers to allocate buses independently. Thus, if the number of chip-columns c is a multiple of the number of controllers K, each controller can be assigned exclusively to a group of c/K columns. The main disadvantage of such a partitioning scheme, however, is that each partition must be designed to be individually nonblocking for the switch to operate in nonblocking mode. This generally increases the total number of buses needed for nonblocking operation above the single-controller case.

Mathematically, any of our distribution schemes can be specified by a *distribution function* f that maps port pairs to subsets of the K controllers. That is, for every pair of port addresses (a, b) with $0 \leq a, b < N$,

$$W_{a,b} = f(a, b)$$

specifies the subset of controllers that have been designated to handle the request. Because connect and disconnect requests involving a given pair of ports may originate at either of the ports, we enforce the condition that $f(a, b) = f(b, a)$, for all ports a and b.

There are two modes of servicing connection requests. In the *synchronous* mode or *batch mode*, all crosspoints are initially inactive; a set of connect requests is presented to the controller, and the controller tries to simultaneously set up as many connections as possible. After the desired communication ends, all connections are removed and a new set of requests is presented. The entire network is then reconfigured according to the new batch. Such batch requests are encountered in some multiprocessor systems where the processors operate synchronously and submit their requests to the network simultaneously [2].

Alternatively, the switch can be used in *incremental* or *on-line* mode, wherein connect or disconnect requests from the ports are made asynchronously and independently. This is a more realistic model for many situations such as packet-switching. We shall restrict ourselves to the incremental mode of operation in this paper.

3 The Symmetric Triangular Scheme

A simple method of distributing control is to divide the the set of all port-address pairs in the switch into K disjoint subsets and assign each subset to a unique controller. This affords only limited fault-tolerance as the failure of a controller results in a service interruption, but the scheme is simple to implement.

The *symmetric triangular scheme* is a simple method of assigning port-address pairs to servicing controllers. This is best illustrated by representing the set of port-address pairs in a triangular pattern. The N ports in the system can generate a total of $N(N-1)/2$ distinct pairs when no distinction is made of the order of appearance of the ports in a given pair. These $N(N-1)/2$ port-address pairs can be represented as a two-dimensional triangular matrix, as shown in Figure 3 for the case of $N = 16$.

With a uniform spatial distribution of requests, each pair of port-addresses has equal probability of appearing in a connect request. Therefore, on the average, an even distribution of the total load among the controllers is achieved if the number of port-address pairs assigned to each controller is approximately equal. Such a partition can be obtained by dividing the triangular area in Figure 3 horizontally into K regions of approximately equal area. This is the basic idea behind the symmetric triangular scheme.

Figure 4 illustrates the symmetric triangular scheme for the case of $N = 16$ ports and $K = 2$ controllers. This was obtained by partitioning the triangular matrix of 120 port-address pairs horizontally into two approximately equal subsets. The upper partition has 55 pairs and the lower partition has 65. Figure 5 shows a larger example with $N = 256$ ports and $K = 4$ controllers.

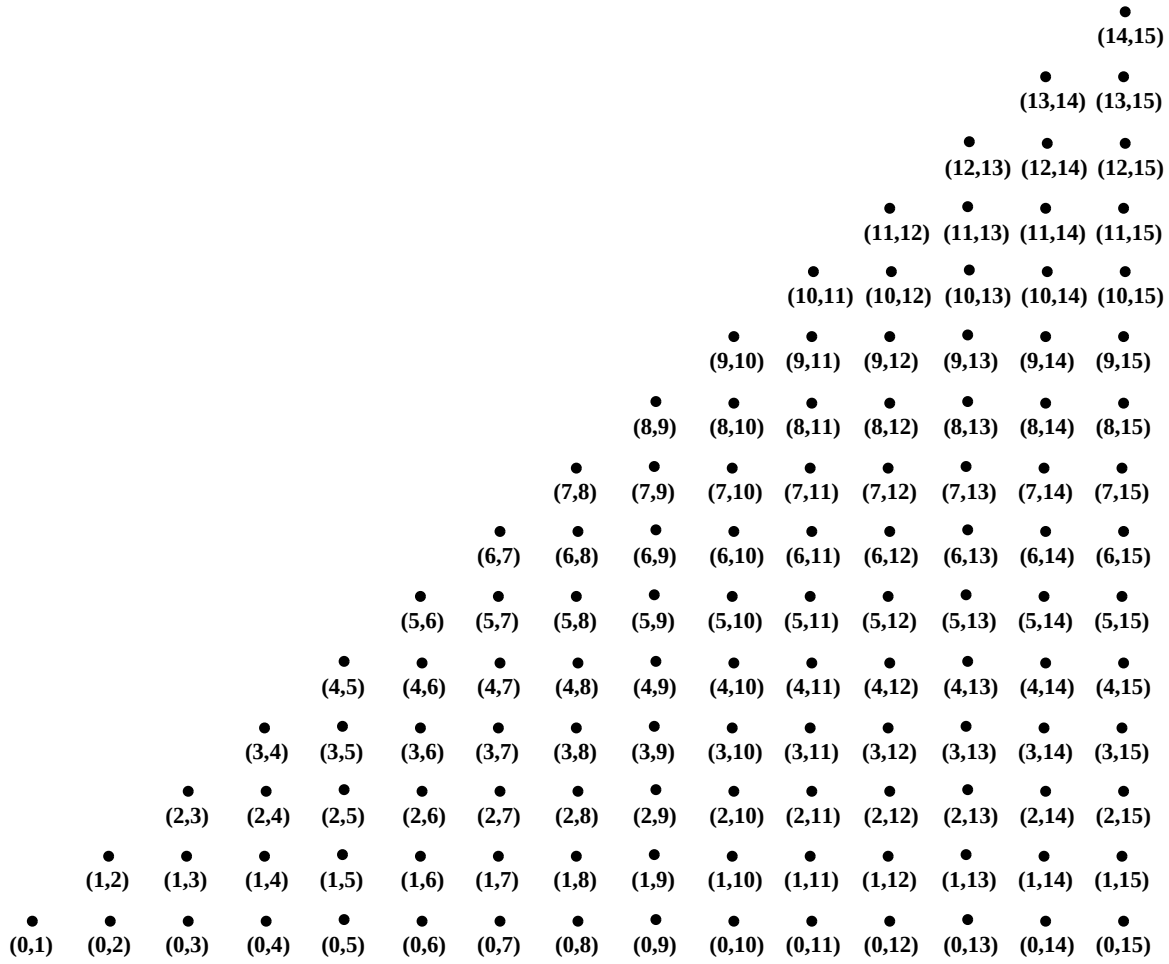


Figure 3: The port-address pairs in a 16-port switch arranged in a triangular pattern to illustrate the symmetric triangular scheme.

The scheme can be generalized to an arbitrary number of controllers K. The triangular matrix of port-address pairs is divided into horizontal bands such that the number of pairs within each band is close to $N(N-1)/2K$. Mathematically, a pair of part-addresses (a, b), with $a < b$, is assigned to a controller C_j if and only if $h_j < a \leq h_{j+1}$, where the h_j's are chosen as follows:

1. $h_0 = 0$;

2. h_j, for $1 \leq j < K$, are chosen such that the magnitude of

$$(h_j + 1)\left(\frac{N - h_j - 1}{2}\right) - \frac{N(N-1)}{2j}$$

is minimized among all integer values of h_j in the range $0 \leq h_j \leq N - 1$;

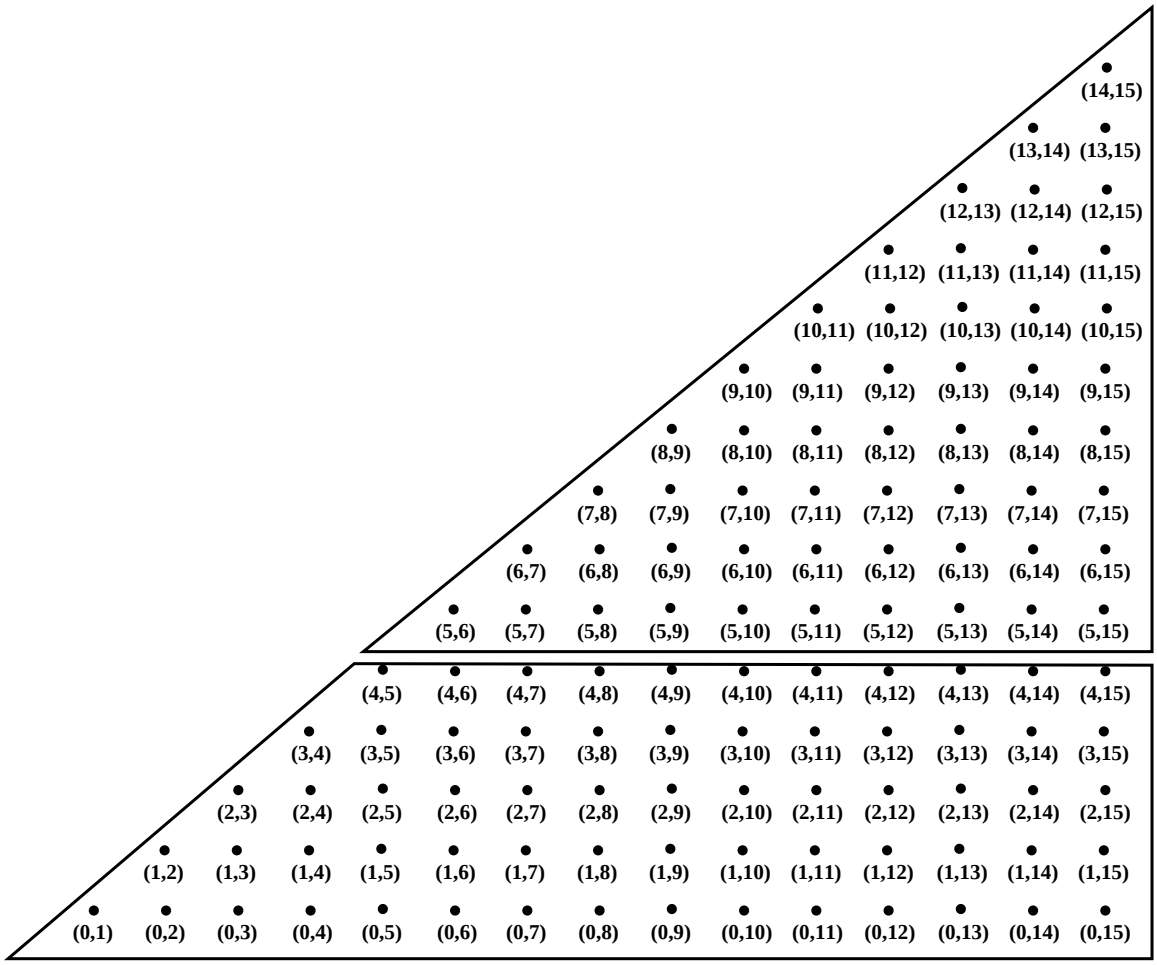


Figure 4: The port-address pairs in Figure 2 partitioned according to the symmetric triangular scheme with 2 controllers.

3. $h_K = N - 1$.

The value of h_j represents the height of the trapezoidal region covered by the set of j controllers $\{C_0, C_1, \ldots, C_{j-1}\}$. Choosing h_j's as above makes the area of this trapezoidal region closest to j/K times the total area of the triangular matrix.

It is easy to observe that blocking could result with the symmetric triangular scheme if the total number of buses is $N/2$. For example, in Figure 4, at least 5 buses are needed for each individual partition to operate without blocking — a total of 10 buses. An approximate expression for the total number of buses needed for nonblocking operation can be obtained using simple geometry. The uppermost partition has a triangular shape while the lower partitions are trapezoidal

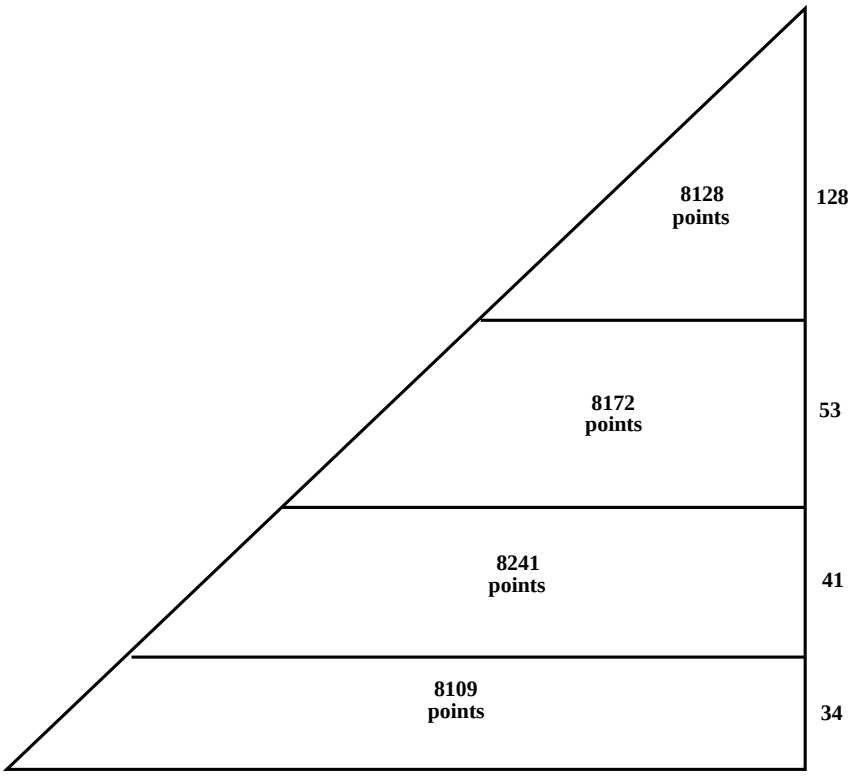


Figure 5: Port-address pairs in a 256-port switch partitioned according to the symmetric triangular scheme with 4 controllers.

in shape. The height of the uppermost triangular partition is given by $h_t = (h_K - h_{K-1})$, while the total height of the $K-1$ trapezoidal regions below it is given by h_{K-1}. Because the number of address pairs within the upper triangular partition is approximately $1/K$ times the total number of pairs,

$$\frac{h_t(h_t-1)}{2} \approx \frac{N(N-1)}{2K},$$
$$\text{or,} \quad h_t \approx \frac{N}{\sqrt{K}}.$$

For the uppermost triangular region, $h_t/2$ buses are sufficient for nonblocking operation, while for the remaining trapezoidal region with height h_{K-1}, the number of buses needed is h_{K-1}. Thus, the total number of buses for nonblocking operation is approximately given by

$$\begin{aligned} \frac{h_t}{2} + h_{K-1} &\approx \frac{N}{2\sqrt{K}} + N\left(1 - \frac{1}{\sqrt{K}}\right) \\ &= N\left(1 - \frac{1}{2\sqrt{K}}\right). \end{aligned} \tag{1}$$

Thus, with $K = 4$, controllers, the number of buses needed is approximately 0.75N, a 50 percent increase over the single-controller case. For example, the top partition in Figure 5 has a height of 128 and therefore requires 64 buses for nonblocking operation. The lower partitions require 53, 41, and 34 buses, respectively. The total number of buses needed is, therefore, $64+53+41+34 = 192$, which matches with $0.75N$.

The chief advantage of the symmetrical triangular scheme is its relatively low overhead in the number of buses needed for nonblocking operation, when compared with the other schemes in the paper. Its primary disadvantage is the limited fault-tolerance. The failure of one of the controller causes a disruption of service for requests involving the port-pairs assigned to the failed controller. Recovery from such a failure is possible by re-distributing the port-address pairs assigned to the faulty-controller among the surviving ones. This causes at least a temporary disruption of service. In addition, the number of buses in each partition may not be adequate to support nonblocking operation. In the next section, we describe a distribution scheme with more flexibility in assigning requests to controllers, thus providing a higher degree of fault-tolerance.

4 The Chessboard Scheme

The symmetric triangular scheme in the previous section divided connection requests into disjoint sets based on the port-address pair involved; each pair of port-addresses was assigned to a single controller. Consequently, the failure of a single controller results in a service disruption for requests assigned to the failed controller, till the corresponding port-address pairs can be re-assigned among the operational controllers. In real-time applications such unavailability is not acceptable. Thus, to achieve tolerance against controller faults, it is necessary that every connection request (a, b) is satisfiable by at least two controllers, so that no disruption of service results from the failure of a single controller.

An obvious improvement over the symmetric triangular scheme is to assign two controllers for every pair of port-addresses, a *primary* controller and a *secondary* controller. During normal operation, every request is serviced by the corresponding primary controller; in the event of a

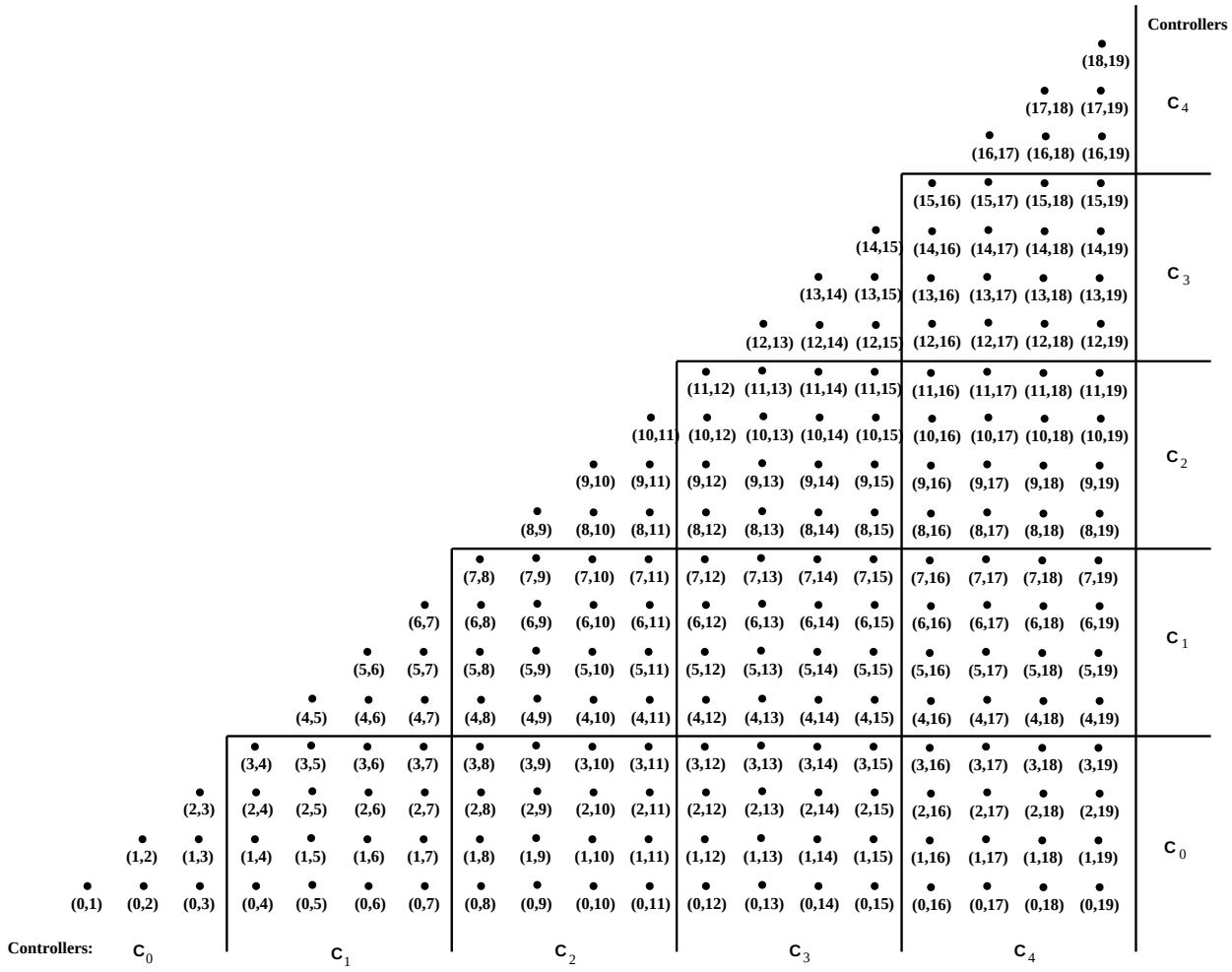


Figure 6: Grouping of port-pairs in the chessboard scheme.

controller failure, each request normally serviced by the failed controller is directed to the corresponding secondary controller. In this section, we describe such an assignment, nicknamed the *chessboard scheme*.

The chessboard scheme assigns two controllers — $P(a,b)$ and $S(a,b)$ — for every pair of port-addresses (a,b) with $a < b$. $P(a,b)$ is the primary controller assigned to handle the request (a,b), while $S(a,b)$ is the secondary controller assigned to the request. Ideally, $P(a,b)$ and $S(a,b)$ should be defined such that, in the event of a controller failure, the additional load is distributed uniformly among the surviving controllers. The chessboard scheme attempts to achieve this objective by partitioning the triangular matrix of port-address pairs along the rows and the columns into

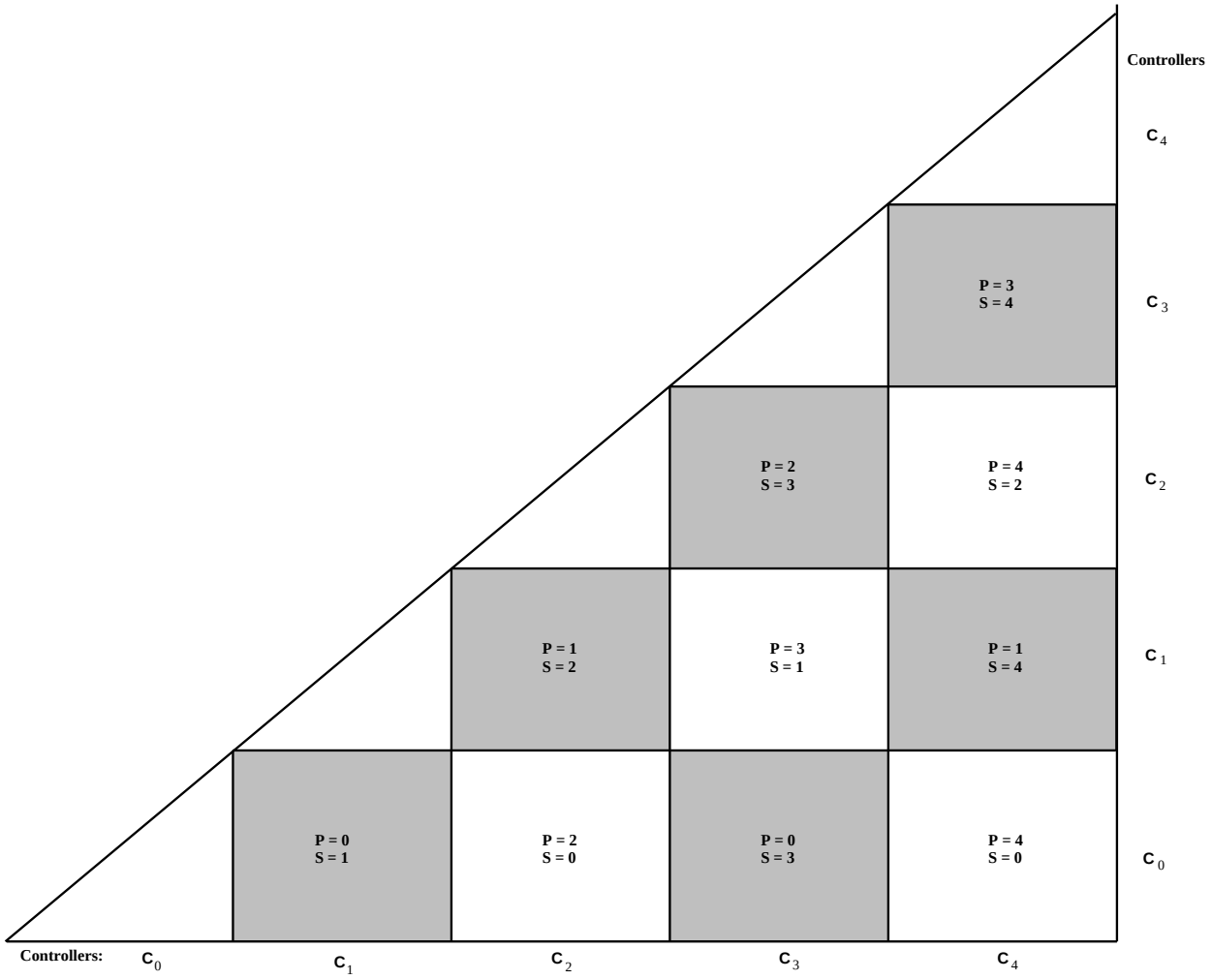


Figure 7: Assignment of controllers to port-pairs in the chessboard scheme for $K = 5$ controllers.

square blocks of equal area. This is illustrated in Figure 6 with respect to a system with $N = 20$ ports and $K = 5$ controllers. The controllers can be imagined as occupying the rows and columns of the blocks; thus, the requests falling within a square are serviceable by the controllers assigned to the corresponding row and the corresponding column. Among these controllers, the primary and the secondary controllers are determined such that, on the average, the controllers are equally loaded during normal operation.

One way to assign the primary and secondary controllers to the squares is to use an alternating pattern as in a chessboard. This is shown in Figure 7. For the shaded blocks in the figure,

the controller along the row serves as the primary controller while the controller along the column serves as secondary; the roles are reversed for the unshaded squares. In this chessboard scheme, it is easy to observe that every block is assigned to a *unique pair* of controllers. Consequently, when a controller fails, the requests assigned to it (as primary) are distributed uniformly among the $K-1$ operational controllers.

We can now define the chessboard scheme mathematically. We assume that the number of controllers K is equal to the number of chip-rows r. This allows connections between port-pairs within each of the triangular areas of Figure 7 to be realized as internal connections. Consequently, these connections need not be considered while estimating the number of buses needed for non-blocking operation. In addition, choosing $K = r$ allows many faults within one of the rows of chips to be treated as equivalent to the failure of the corresponding controller. The scheme can easily be generalized to the case of K being an integer multiple of r.

In the following, the notation $\lfloor x \rfloor$ denotes the largest integer less than or equal to x. Mathematically, the chessboard scheme is associated with two request-distribution functions, $P(a,b)$ and $S(a,b)$. Let w denote the width of a square in Figure 7, that is $w = N/K$. Then, for every pair of ports (a,b) within one of the square blocks, the primary and secondary controllers are determined by:

$$P(a,b) = \begin{cases} \lfloor \frac{a}{w} \rfloor, & \text{if } \lfloor \frac{a}{w} \rfloor + \lfloor \frac{b}{w} \rfloor \text{ is odd;} \\ \lfloor \frac{b}{w} \rfloor, & \text{otherwise.} \end{cases} \tag{2}$$

$$S(a,b) = \begin{cases} \lfloor \frac{b}{w} \rfloor, & \text{if } \lfloor \frac{a}{w} \rfloor + \lfloor \frac{b}{w} \rfloor \text{ is odd;} \\ \lfloor \frac{a}{w} \rfloor, & \text{otherwise.} \end{cases} \tag{3}$$

The above assignment applies to only ports within the square blocks, that is for $N > b \geq w(1 + \lfloor a/w \rfloor)$. To be complete, controllers for the internal connections within the triangular areas in Figure 7 need also to be specified. One such assignment is:

$$P(a,b) = \left\lfloor \frac{a}{w} \right\rfloor$$

$$\text{and} \qquad S(a,b) = \left(\left\lfloor \frac{a}{w} \right\rfloor + 1 \right) \bmod K.$$

In the example of Figure 7, each controller is the primary controller for exactly two square

blocks and secondary controller for another two. Note that there are a total of $K(K-1)/2$ square blocks. In general, if K is odd, the chess-board scheme results in every controller being primary and secondary controller for $(K-1)/2$ blocks each. If K is even, however, the assignment given by equations (2) and (3) does not result in such an equal distribution. A uniform distribution can still be achieved in the case of even K if the chessboard pattern is chosen at the level of the individual labels instead of at the level of the square blocks. That is, for ports within the range $N > b \geq w(1 + \lfloor a/w \rfloor)$, the controller assignment is given by:

$$P(a,b) = \begin{cases} \left\lfloor \frac{a}{w} \right\rfloor, & \text{if } (a+b) \text{ is odd;} \\ \left\lfloor \frac{b}{w} \right\rfloor, & \text{otherwise.} \end{cases} \tag{4}$$

$$S(a,b) = \begin{cases} \left\lfloor \frac{b}{w} \right\rfloor, & \text{if } (a+b) \text{ is odd;} \\ \left\lfloor \frac{a}{w} \right\rfloor, & \text{otherwise.} \end{cases} \tag{5}$$

How many buses are required for non-blocking operation? We observe that $w = N/K$ buses are sufficient for each controller during normal operation, since any request directed to the controller C_i must involve at least one port from the set $\{wi, wi+1, wi+2, \ldots, w(i+1)-1\}$ as the source or the destination. Thus, a total of $wK = N$ column-buses are sufficient for nonblocking operation. This is twice the number of buses as compared to the single-controller case.

The chessboard scheme affords improved availability over the symmetrical triangular scheme. The failure of a controller can be detected in many ways. One approach is to design the controllers to be self-testing, that is, each controller tests itself concurrently with its normal operation [13]. Alternately, the failure of a controller can be detected by a port from failure of the handshake protocol employed for servicing requests. When a controller is found to be faulty, each request normally destined to it is handed over to the designated secondary controller based on the port-addresses specified in the request. Note that given any pair of controllers C_i, C_j, there is a unique block in the chessboard pattern of Figure 7 that uses the two controllers. Thus, if a controller fails, the affected requests are evenly among the surviving controllers. In addition, the switch remains nonblocking because any request directed to the controller C_i must still involve at least one port from the set $\{wi, wi+1, wi+2, \ldots, w(i+1)-1\}$.

The properties of the chessboard scheme may be formalized using the theory of block designs

and finite projective planes [18, 17]. The theory of block designs deals with the problem of arranging a set of distinct objects S into subsets such that every subset contains exactly k objects; every object appears exactly α times in all the subsets taken together; and pairs, triplets, and similar groups of objects occur a specified number of times. Each subset of such a configuration is called a *block*. An important class of block design is a *regular design* which is defined as a collection of k-sets from S such that every member of S belongs to exactly α of the k-sets. If v is the size of the set S in a regular design and b is the number of blocks, it can be shown by a simple counting argument that $bk = v\alpha$. If we consider S as the set of all squares in the chessboard pattern of Figure 7, and interpret the subset of squares assigned to a given controller as primary or secondary as forming a block, then, for odd values of K, the chessboard scheme is a regular design with $b = K(K-1)/2$, $k = 2$, $v = K$, and $\alpha = K - 1$. This formulation further validates the observation that each group falls under the domain of exactly two controllers. Other ideas from combinatorial design theory may be fruitfully explored to design alternate partitioning strategies.

The chessboard scheme still allows only a single controller to service a request involving a certain pair of port-addresses during normal operation. A further improvement over this scheme is to select the servicing controller for each request dynamically from the subset of two controllers assigned to it. This allows dynamic balancing of the total load among the controllers. In addition, the scheme effectively doubles the number of buses that can be used to connect a given pair of ports, potentially reducing the total number of buses needed for nonblocking operation over the chessboard scheme. We study such a scheme in the next section.

5 Dynamic Distribution Scheme with Global Load-Balance

The chessboard scheme achieves tolerance to controller faults at the cost of 100% overhead in terms of the number of column buses required. This overhead can be reduced if some flexibility is allowed in the selection of the servicing controller for a given request. Thus, we may associate *two* servicing controllers with each request involving a given pair of port-addresses; one of the two may be dynamically selected as the servicing controller for each request based on the load on individual

controllers. We describe such a scheme in this section.

One way to specify the pair of servicing controllers for a given request is to associate one controller with each of the ports involved in the request. That is, each port a is now associated with a *home controller* $h(a)$; a request (a, b) may be serviced by one of the home controllers $h(a)$ and $h(b)$. Thus, the distribution function f for this scheme is given by

$$f(a, b) = \{h(a), h(b)\}. \tag{6}$$

If all the column buses in the partition controlled by one of the controllers are in use, the request can be routed to the other controller. This allows some sharing of the buses among the partitions, thus reducing the total number of buses needed for nonblocking operation.

The distribution function in equation (6) provide no redundancy when $h(a) = h(b)$. This special case can be handled by assigning the home-controllers such that $h(a) = h(b)$ only if the ports a and b fall within the same row of chips; the connection (a, b) can now be implemented as an internal connection and can be handled exactly as in the chessboard scheme.

How does this dynamic distribution scheme compare with the chessboard scheme in terms of the number of buses needed for nonblocking operation? The dynamic distribution allows better balancing of load among the controller, as the servicing controller for each request can now be selected as the one with lower load among the pair of controllers assigned to it. If each request is allowed to select the servicing controller from the pair of controllers assigned to it, we show that $N(1 - 1/K)$ buses are both necessary and sufficient for nonblocking operation when K is a power of 2. This is an improvement of N/K over the chessboard scheme.

5.1 Number of Buses for Nonblocking Operation

In this section, we derive an expression for the number of buses needed for nonblocking operation of the switch with the dynamic distribution scheme. We start with the following assumptions:

1. Every port a is associated with a unique home controller $h(a)$. The home controllers are uniformly assigned among the K controllers so that each controller is home to N/K ports.

2. For each request (a, b), the servicing controller selected is $C_{h(a)}$ if the number of connections implemented by $C_{h(a)}$ is less than the number implemented by $C_{h(b)}$; otherwise, $C_{h(b)}$ is selected as the servicing controller.

3. If the port-addresses are such that $h(a) = h(b)$, the connection (a, b) can be implemented as an internal connection and no external bus needs to be allocated.

We first show that $N(1 - 1/K)$ buses are necessary for nonblocking operation with this scheme.

THEOREM 1: In a one-sided crossbar switch with N ports and K controllers, at least $N(1 - \frac{1}{2^{\lfloor log_2 K \rfloor}})$ buses are required for nonblocking operation if the dynamic distribution scheme is employed.

The theorem is proved by giving a sequence of connect and disconnect requests that would force blocking unless $(N/K)(1 - \frac{1}{2^{\lfloor log_2 K \rfloor}})$ buses are available per controller. Before presenting the proof, we show an example where this lower bound on the number of buses is actually reached.

EXAMPLE 1: Consider an example configuration with $N = 64$ ports and $K = 8$ controllers. Let the home controllers be assigned such that

$$h(a) = \left\lfloor \frac{h}{N/K} \right\rfloor ,$$

for each port a. An initial set of $N/2 = 32$ connections is formed as

$$\{(16x + a, 16x + a + 8) \mid 0 \leq x < 4; 0 \leq a < 8\}.$$

These 32 requests can be distributed among the 8 controllers uniformly as shown in Table 1.

C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
0, 8	4, 12	16, 24	20, 28	32, 40	36, 44	48, 56	52, 60
1, 9	5, 13	17, 25	21, 29	33, 41	37, 45	49, 57	53, 61
2, 10	6, 14	18, 26	22, 30	34, 42	38, 46	50, 58	54, 62
3, 11	7, 15	19, 27	23, 31	35, 43	39, 47	51, 59	55, 63

Table 1: Initial state of the controllers in Example 1.

Now assume that the connections assigned to odd-numbered controllers in Table 1 are removed and the ports are reconnected to form the following connections:

$$\{(4,20),(5,21),(6,22),(7,23),$$
$$(12,28),(13,29),(14,30),(15,31),$$
$$(36,52),(37,53),(38,54),(39,55),$$
$$(44,60),(45,61),(46,62),(47,63)\}.$$

Note that half of the new connections formed can only be serviced by even-numbered controllers. Thus, each of the even-numbered controllers must now service two connections. Distributing the new connections among the servicing controllers uniformly results in the state in Table 2.

C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
0, 8	12, 28	16, 24	14, 30	32, 40	44, 60	48, 56	46, 62
1, 9	13, 29	17, 25	15, 31	33, 41	45, 61	49, 57	47, 63
2, 10		18, 26		34, 42		50, 58	
3, 11		19, 27		35, 43		51, 59	
4, 20		6, 22		36, 52		38, 54	
5, 21		7, 23		37, 53		39, 55	

Table 2: State of the controllers after the first sequence of connection reassignments.

Each of the even-numbered controllers is now implementing 6 connections. The maximum number of connections per controller can be increased further by performing a second set of connection reassignments, this time involving only the last two connections realized by controllers C_2 and C_6 in Table 2. Assume that these four connections are removed and the ports involved are paired in new connections as follows:

$$\{(6,38),(7,39),(22,54),(23,55)\}.$$

Each of the connections $(6,38)$ and $(7,39)$ can be serviced only by the pair of controllers C_0 and C_4. Assigning one of these new connections to each of the controllers C_0 and C_4 results in the state shown in Table 3.

Thus, the controllers C_0 and C_4 need 7 buses each if blocking is to be avoided. Because a similar sequence of connections and disconnections can be used to increase the load on any of the

C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
$0, 8$	$12, 28$	$16, 24$	$14, 30$	$32, 40$	$44, 60$	$48, 56$	$46, 62$
$1, 9$	$13, 29$	$17, 25$	$15, 31$	$33, 41$	$45, 61$	$49, 57$	$47, 63$
$2, 10$		$18, 26$		$34, 42$		$50, 58$	
$3, 11$		$19, 27$		$35, 43$		$51, 59$	
$4, 20$		$22, 54$		$36, 52$		$23, 55$	
$5, 21$				$37, 53$			
$6, 38$				$7, 39$			

Table 3: State of the controllers after the second sequence of connection reassignments.

K controllers, each of the controllers needs 7 buses. Thus, the total number of buses needed is $7 \times 8 = 56$, which matches with the lower bound of Theorem 1.

The situation in Table 3 cannot be made more extreme, since any new connections created by by freeing ports from existing connections can always be satisfied without exceeding 7 connections per controller. We shall later show that the lower bound of Theorem 1 is also an upper bound when K is a power of 2.

We can now generalize the sequence of connections and disconnections in Example 1 to formulate a proof of Theorem 1.

PROOF OF THEOREM 1: For simplicity, we assume that N/K is an integer and that K is a power of 2. The proof can easily be extended for arbitrary values of K. We assume that the home controllers are assigned such that

$$h(a) = \left\lfloor \frac{h}{K} \right\rfloor ,$$

for each port a. The proof works by constructing a sequence of connections and disconnections that forces at least one controller in the system to implement $(N/K)(1 - \frac{1}{2^{\lfloor log_2 K \rfloor}})$ connections.

Step 1: Initially, all the N ports are assumed to be idle. A set of $N/2$ non-conflicting connection requests are made as follows:

$$\left\{ \left(\frac{2N}{K}x + a, \frac{(2N+1)}{K}x + a \right) \mid 0 \leq x < K/2; 0 \leq a < \frac{N}{K} \right\} .$$

Note that all the requests are of the form (a, b), where $h(a)$ is even and $h(b) = h(a) + 1$. The most uniform use of the controllers results when each of the controllers is handling $N/2K$ requests. Without loss of generality, we can assume that the subset of connections assigned to controller i is

the set

$$\left\{\left(\frac{2N}{K}i + a, \frac{(2N+1)}{K}i + a\right) \mid 0 \leq a < \frac{N}{2K}\right\}, \qquad \text{for even } i;$$
$$\left\{\left(\frac{2N}{K}i + \frac{N}{2K} + a, \frac{(2N+1)}{K}i + \frac{N}{2K} + a\right) \mid 0 \leq a < \frac{N}{2K}\right\}, \qquad \text{for odd } i.$$

Step 1 is followed by a series of re-distribution steps. In each step, a subset of the connections made in the previous step are removed and new connection requests generated so as to increase the load on some of the controllers. The final goal is to maximize the load on controller 0.

Step 2: In this step, the connections between pairs of ports that are being handled by the odd-numbered controllers are disconnected. The freed ports are reconnected to generate $N/8$ requests of the form

$$\{(a,b) |\ h(a) \bmod 4 = 0 \text{ and } h(b) \bmod 4 = 2\};$$

and $N/8$ requests of the form

$$\{(a,b) |\ h(a) \text{ is odd and } h(b) \text{ is odd, } h(a) \neq h(b)\}.$$

The first set of $N/8$ connections must now be realized by the even-numbered controllers, thus increasing the load on each of them by $N/4K$.

Step 3: In this step, connections connections are removed from controllers C_i with $i \bmod 4 = 2$. The set of connections removed are the $N/4K$ connections assigned to each of them in Step 1. The $N/8$ ports so freed up are reconnected to generate $N/32$ connections of the form

$$\{(a,b) |\ h(a) \bmod 8 = 2 \text{ and } h(b) \bmod 8 = 6\};$$

and $N/32$ requests of the form

$$\{(a,b) |\ h(a) \bmod 8 = 0 \text{ and } h(b) \bmod 8 = 4\};$$

The first set of $N/32$ connections must be realized by the subset of controllers $C_i, i \bmod 4 = 0$. When the new connections are allocated uniformly to this subset of $K/4$ controllers, the load in each of them increases by $(\frac{N}{32})/(\frac{K}{4}) = N/8K$.

This process of redistributing connections can be continued, each step involving half the connections created in the previous step. In the last step, $(\frac{N}{K})/(\frac{K}{2})$ connections are removed from each of the two controllers $C_{K/4}$ and $C_{3K/4}$, forcing half of the new connections to be shared by the controllers C_0 and $C_{K/2}$. This increases the load on each of these controllers by N/K^2. The total number of connections implemented by each of the controllers C_0 and $C_{K/2}$ at this point is given by

$$\frac{N}{2K}\sum_{j=0}^{2^k-1}\frac{1}{2^j} = \frac{N}{K}\left(1-\frac{1}{K}\right),$$

where $k = \log_2 K$.

Since the choice of the two controllers to maximize the load in the above sequence is arbitrary, every controller in the system must be designed to handle $(N/K)(1-1/K)$ connections if blocking is to be avoided. Thus, a total of $N(1-1/K)$ buses are necessary for nonblocking operation of the switch.

If K is not a power of two, then we can simply consider connections involving the first $2^{\lfloor \log_2 K \rfloor}$ controllers, and use the same sequence of connection/disconnection requests as before. Therefore, the minimum number of buses required is $N\left(1-\frac{N}{2^{\lfloor \log_2 K \rfloor}}\right)$.

This concludes the proof of Theorem 1.

When K is a power of 2, we can show that the lower-bound of Theorem 1 is also an upper-bound on the number of buses needed for nonblocking operation. That is, no blocking results if each of the K controllers is provided with $(N/K)(1-1/K)$ buses.

THEOREM 2: In a one-sided crossbar switch with N ports and $K = 2^k$ controllers employing the dynamic distribution scheme of this section, no blocking can occur if each controller is provided with $\frac{N}{K}(1-\frac{1}{K})$ buses.

PROOF: We assume that the home controllers are assigned such that

$$h(a) = \lfloor \frac{a}{N/K} \rfloor,$$

for each port a. Let S_i denote the subset of ports whose home controller is C_i. That is,

$$S_i = \{j |\ \frac{N}{K}i \le j < \frac{N}{K}(i+1)\}.$$

These subsets form a uniform partition of the N ports in the system.

A request (a, b) is routed to controller $C_{h(a)}$ if the load on $C_{h(a)}$, that is the number of buses currently used by controller $C_{h(a)}$, is less than the load on Controller $C_{h(b)}$; otherwise, $C_{h(b)}$ is selected as the servicing controller. We do not consider the case of $h(a) = h(b)$, as these connections can be implemented as internal connections.

The proof is by induction on the number of controllers K.

Base step: For $K = 2$, $N/4$ buses per controllers are sufficient, as the load is equally distributed among them.

Inductive step: Given that the result is true for the case of $K/2$ controllers, we now show that it is true for K controllers.

Consider a one-sided crossbar switch with N ports and K controllers. Let the number of buses available to each controller be $B = \frac{N}{K}(1-\frac{1}{K})$. Assume, if possible, that a connection (a, b) was blocked because the home-controllers of both the ports p and q were realizing B connections each. Let C_i be the home-controller for port a and C_j that of port b. Then, the combined load on C_i and C_j at the time was $2B$. Among these $2B$ connections, let α be the number of connections between partitions S_i and S_j, and β the number from S_i and S_j to other partitions. The total number of distinct ports involved in the connections carried by C_i and C_j, and belonging to partitions S_i and S_j, is $2\alpha + \beta$. This cannot exceed the total number of ports in partitions S_i and S_j taken together. In addition, the ports in the new connection a, b also belong to the two partitions, but were not counted in $(2\alpha + \beta)$. Thus, we must have

$$2\alpha + \beta \leq 2\frac{N}{K} - 2. \tag{7}$$

Also, the total number of connections realized by the two controllers is $\alpha + \beta$. Equating this to the number of available buses,

$$\alpha + \beta = 2B = \frac{2N}{K}\left(1 - \frac{1}{K}\right). \tag{8}$$

From equations (7) and (8),

$$\alpha \leq \frac{2N}{K^2} - 2; \tag{9}$$

$$\beta \geq \frac{2N}{K}\left(1-\frac{2}{K}\right)+2.$$
$$\geq \frac{N}{K/2}\left(1-\frac{1}{K/2}\right)+2. \quad (10)$$

Now assume that the same sequence of connections was applied to a one-sided crossbar with N ports and $K/2$ controllers, where a single controller $C_{i'}$ acted as the home controller for the ports in $S_i \cup S_j$. Then, at the time the new request (a,b) was made, the controller $C_{i'}$ must be realizing β connections from $S_i \cup S_j$ to other partitions. According to induction hypothesis, the maximum load per controller ever reached in the $K/2$-controller switch is $\frac{N}{K/2}\left(1-\frac{1}{K/2}\right)$. Thus,

$$\beta \leq \frac{2N}{K}\left(1-\frac{2}{K}\right). \quad (11)$$

This contradicts with equations (10). Hence, no blocking can result in the original switch with K controllers if the number of buses per controller is $\frac{N}{K}\left(1-\frac{1}{K}\right)$.
This concludes the proof of Theorem 2.

Thus, it is possible to meet the lower bound of Theorem 1 if each controller is aware of the load on all the other controllers. For example, consider the following distributed control mechanism: Let the primary and secondary controllers for request (a,b) be predetermined by the chessboard assignment of Section 4. Each controller maintains a local table to keep a record of the number of buses used by every other controller. A connection request is first sent to the corresponding primary controller. This controller checks its load table and forwards the request to the secondary controller if that controller is less loaded than itself. Otherwise the primary controller uses one of its available busses to satisfy the request. A forwarded request is always satisfied by the receiving (secondary) controller. Also, a controller broadcasts its identity along with a one-bit tag whenever a column bus that it controls is assigned (tag = 1) or relinquished (tag= 0), so that the other controllers can update their local load tables. Note that since the secondary controller is predetermined for a given request, no local record needs to be maintained for the destination of a forwarded request; a subsequent request for disconnection can always be forwarded to the same controller which handled the connection.

A potential problem with the above scheme is that the load tables maintained by the individual controllers can become inconsistent for short periods of time, resulting in incorrect decisions on forwarding of requests, Thus, the lower bound of Theorem 1 for nonblocking operation can no longer be maintained. The situation is simlar to the cache-coherency problem in multiprocessors, and can be remedied by implementing the table updates as atomic bus transactions.

The fault tolerance and graceful-degradation properties of the scheme outlined above are inherited from the chessboard scheme. However, in the event a controller-failure, nonblocking operation cannot be guaranteed with $N/K(1-1/K)$ buses per controller. A similar situation is encountered if the tag-broadcast bus becomes faulty, forcing requests to be always handled by the primary controller. In either case, the system continues to operate, albeit in a degraded mode.

6 Distribution Scheme with Handovers

The dynamic distribution scheme of the previous section requires load-information to be shared among the controllers to achieve the lower bound of $N(1-1/K)$ buses for nonblocking operation. Is it possible to have a fault-tolerant, distributed control scheme that only relies on local information, and yet requires fewer than N buses for nonblocking operation? This section answers the question in the affirmative by presenting such a scheme. This distribution scheme uses $N/K-1$ buses per controller as compared to $N/K(1-1/K)$ for the previous scheme. Each connect request is first forwarded to a primary controller; the primary controller is predetermined from the address of the ports involved in the request. If the primary controller has no bus available to realize the connection, the request is "handed over" to a secondary controller. The secondary controller is selected dynamically based on local information maintained by the primary controller.

The primary controllers in this scheme can be assigned similar to the scheme of Section 5. Thus, for each port a,

$$h(a) = \left\lfloor \frac{h}{N/K} \right\rfloor ,$$

is the home controller. The primary controller for a request (a,b) is $C_{h(a)}$ if $h(a) < h(b)$, and $C_{h(b)}$ otherwise. As in the previous schemes, we do not consider requests with $h(a) = h(b)$, as these

connections can be realized as internal connections.

Each controller maintains a local lookup table to record each connection currently realized by the controller. An entry in the table maintained by Controller C_i consists of three fields: the first two are the port-addresses involved in a connection currently carried by C_i, and the last field identifies an alternate controller to process this request. At the time of setting up a connection, the controller field for that entry is set as follows:

- If C_i is the home-controller for the connection (a, b) currently being set up, that is if $i = h(a)$, then the controller field is set as $h(b)$.
- If C_i is not the home-controller for the connection, then the connection was forwarded to it by another controller C_j. In this case, the controller field is set to the address of the forwarding controller j.

A request for connection (a, b) is first sent to the primary controller assigned to that request. Let the primary controller be C_i, where $i = h(a)$. If the primary controller is busy, the request is forwarded to a secondary controller. The secondary controller C_j is chosen from the local lookup table of C_i as follows:

Case 1: If $h(b)$ appears in the lookup table, implying that C_i is currently carrying a connection forwarded by controller $C_{h(b)}$, then $j = h(b)$.

Case 2: If $h(b)$ does not appear in the lookup table, then there is some controller whose address appears two or more times in the table. Such a controller is chosen to forward the request.

By a simple counting argument, it can easily be shown that one of the above cases must always be true if the number of buses per controller is $N/K - 1$, and if $N \geq K^2$. In addition, the selection of the servicing controller C_j as above guarantees that C_j has at least one free bus available to service the request. Thus, the scheme limits the number of handovers to one per request under fault-free conditions.

Note that an associative search through the lookup table is needed to determine the servicing controller if all the buses controlled by the primary controller are in use. In addition, each controller

must also maintain the identity of the servicing controller for each forwarded request so that a subsequent request for disconnection can be sent to the same controller.

As an example to illustrate this scheme, consider a switch with 32 ports and 4 controllers. Each controller in this switch is the home controller for 8 ports. Assume that each controller is provided with $N/K - 1 = 7$ buses. Assume that the local lookup table in Controller C_0 at some instant of time is as shown in Table 4. All the entries in the table are requests for which C_0 is the primary controller. The last column in the table is the controller field discussed earlier.

Connection		Contents of
a	b	controller field
0	8	1
1	10	1
2	9	1
3	15	1
4	13	1
5	19	2
6	21	2

Table 4: Contents of lookup table in controller C_0.

All the buses controlled by C_0 are now in use and any subsequent request arriving at C_0 must be forwarded to a secondary controller. Assume that the the next request is $(7, 22)$. In this case C_2, the home controller of port 22, appears in the lookup table of C_0. Hence, C_2 is the forwarding controller. If, on the other hand, the new request were $(7, 27)$ instead of $(7, 22)$, then the home controller of port 27 does not appear in the lookup table. Therefore, the table is scanned to look for two entries that are identical in the controller field. In this case, both 1 and 2 appear twice in the last column of Table 4, so the secondary controller can be chosen as either C_1 or C_2.

The principal advantage of this scheme is arbitration speed, as there is no need to access a global bus or to have access to global load information. This is achieved at the expense of a larger overhead in the number of buses as compared to the dynamic distribution scheme with global load-balance.

7 Simulation Studies

To study the performance of the different distributed control schemes proposed above, we simulated a one-sided crosspoint network with 64 ports and 4 controllers using the different schemes. The performance of these control schemes was measured under different load conditions. In all the simulations, the load on the switch was specified in terms of the the average number of ports involved in connections at any time as a fraction of the total number of ports N. Thus, an average load of 70% indicates that, on the average, 70% of the ports were connected and 30% were idle. The connections were generated randomly, with the ports for each connection selected uniformly from the set of idle ports at that time. The basic performance measure used to evaluate the schemes is the *blocking probability* computed by counting the number of requests blocked as a percentage of the total requests presented to the switch.

Note that $N/2 = 32$ buses are needed for nonblocking operation of the example switch configuration if a single controller is used. The number of buses required to achieve nonblocking operation with K controllers depends on the distributed control mechanism employed. Thus, for each control scheme, we started with 32 buses and observed the percentage of requests that are blocked; and then measured the decrease in this percentage as more buses are added.

For the control schemes of Sections 5 and 6, which have the capability of tolerating controller failures, we have also measured the blocking probability when one of the controllers is faulty. Observe that only three-fourth of the buses are accessible when one of the controllers is faulty; hence, the switch must have at least $\lceil 32 \cdot 4/3 \rceil = 43$ buses to guarantee nonblocking operation even when no limit is placed on the number of handovers allowed to satisfy a request.

Another interesting performance measure for distribution schemes that allow handovers is the percentage of requests that were satisfied, but required one handover. A large percentage of handovers is undesirable as it increases the average service time significantly.

Figur e 8 shows the results for the symmetric triangular scheme of Section 3. With 32 buses, up to 30% of requests are not granted when the average load is 90%. The blocking decreases significantly as the number of buses is increased, and only about 3% of the requests is blocked

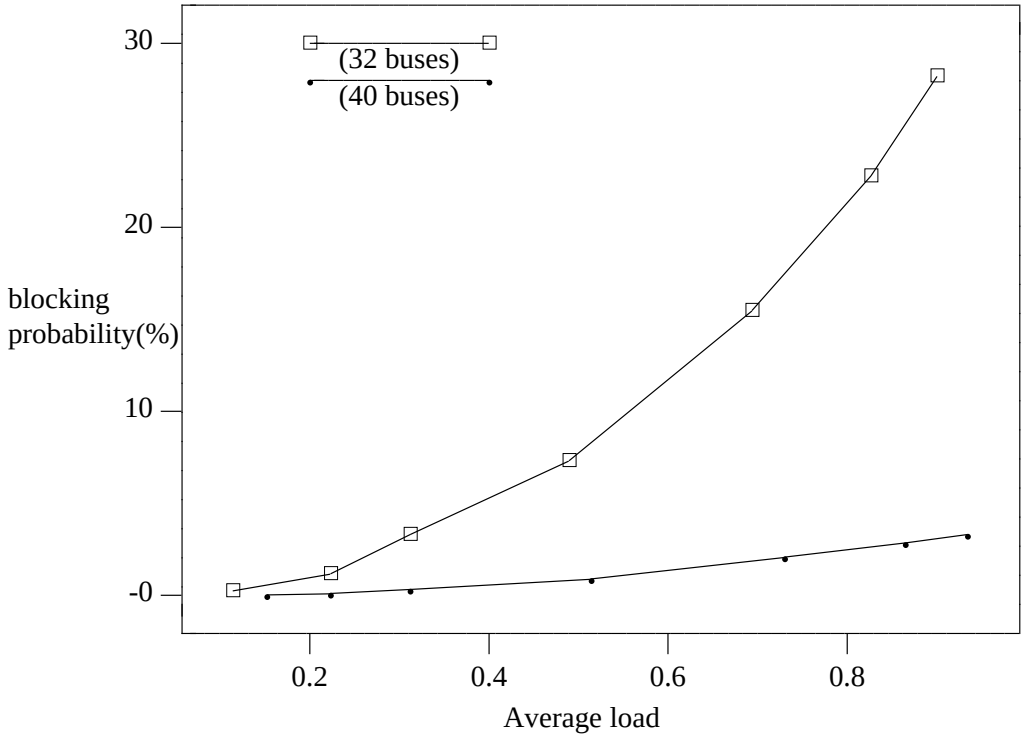


Figure 8: Simulation results for the symmetric triangular scheme for a 64-port one-sided crossbar with four controllers.

when 40 buses are available. Note that 48 buses are needed to guarantee nonblocking from theory, but in practice the number is reduced significantly if a small amount of blocking is acceptable.

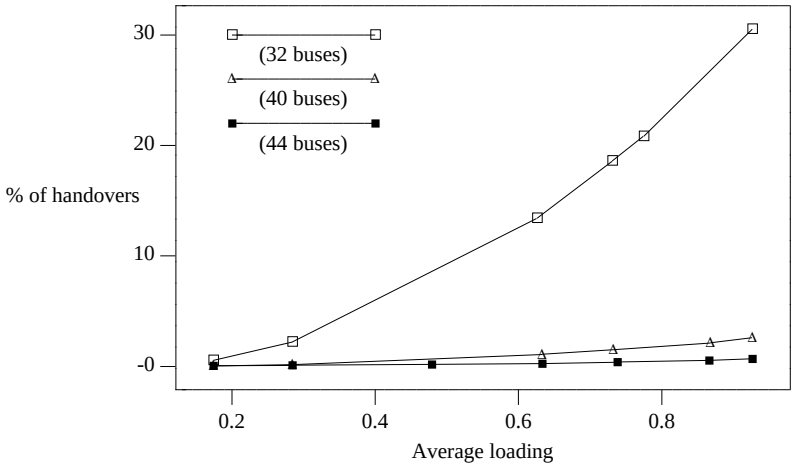
Figure 9(a) and 9(b) show the results from simulating the chessboard control scheme of Section 4. The following algorithm was used to process requests in the fault-free case:

1. Each request is sent to the primary controller determined statically by the chessboard assignment given by equations (4) and (5).

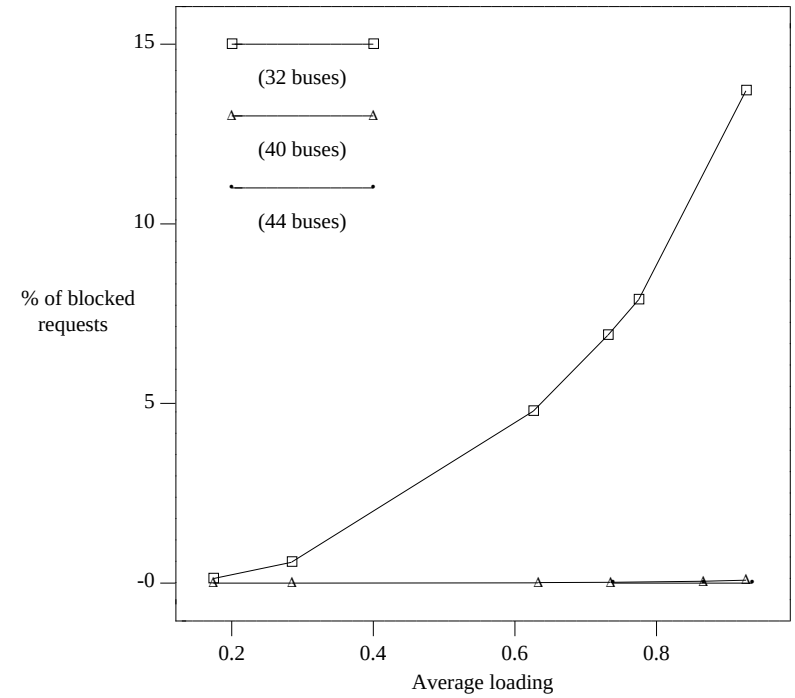
2. If all the buses available to the primary controller are in use, the request is handed over to the corresponding secondary controller.

3. A request is blocked if both the primary and secondary controllers are unsuccessful in allocating a bus.

If the primary controller assigned to a request is faulty, it is routed to the corresponding secondary



(a) Percentage of handovers vs average loading of the network



(b) Percentage of blocked requests vs average loading of the network

Figure 9: Simulation results for the chessboard scheme applied to a 64-port one-sided crossbar with four controllers.

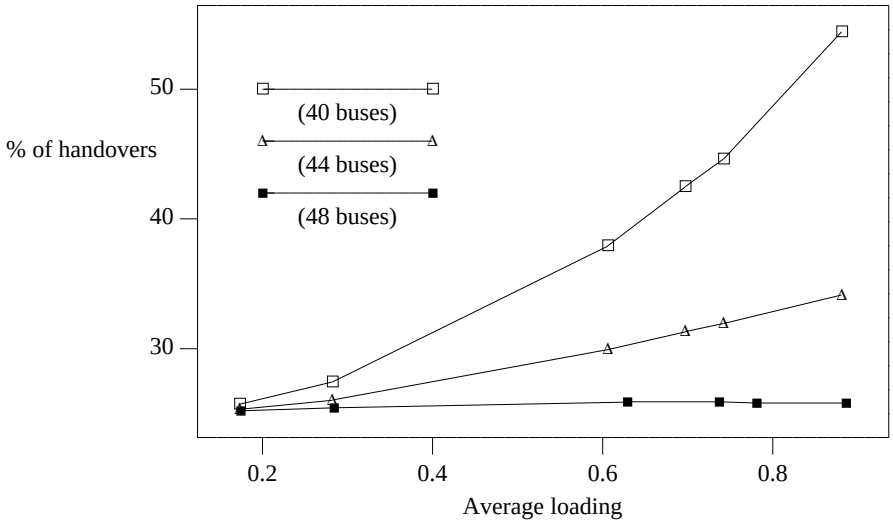
controller; no further handovers are allowed for such requests. Similarly, no handover is possible if the secondary controller assigned to a request is faulty. Requests routed to secondary controllers because of a failed primary controller are not counted among the number of handovers.

Figure 9(a) shows the percentage of connection requests that needed a handover as a function of the load under fault-free conditions. Figure 9(b) shows the percentage of blocked requests for the same case. When a total of 32 busses are used in the network with 90% load, approximately 30% of the requests needed a handover, and about 14% of the requests were blocked. When the number of buses was increased to 40, the number of handovers was reduced to about 2.2%, while the blocking probability dropped to about 0.05%. With 44 buses, the percentages further dropped to 0.5% and 0.003%, respectively. Recall that the chessboard scheme requires 64 buses to guarantee nonblocking operation with no handovers. Thus, the number of buses needed in practice is much less than the theoretical bound if a small amount of blocking is acceptable.

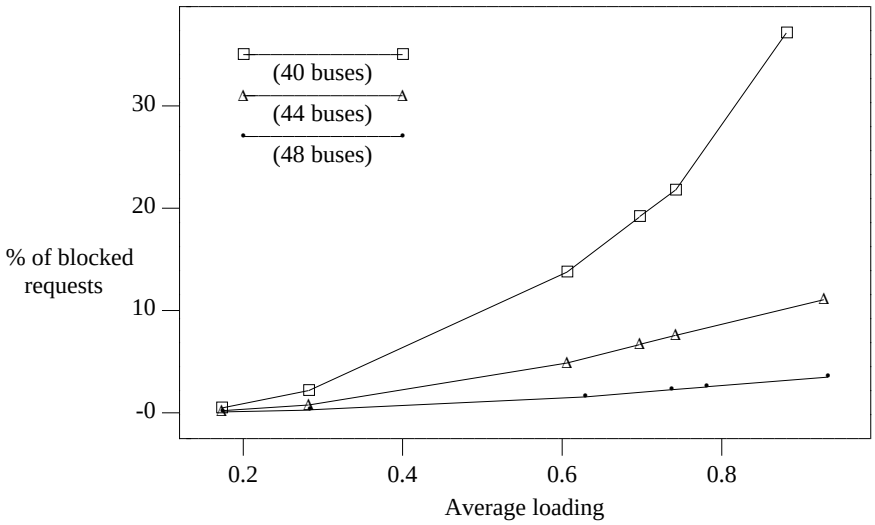
The graphs in Figures 10(a) and 10(b) show the results of simulating the chessboard scheme when one of the controllers is faulty. Again, the two parameters plotted are the percentage of secondary handovers required and the blocking probability. The performance as indicated by both metrics degraded significantly from the fault-free case. Thus, with 40 buses, the blocking probability was well above 30%. With 48 buses, however, the blocking probability dropped to under 3%; the number of handovers still remained high owing to the smaller number of buses available. Observe that only 36 of the 48 buses are available when one of the controllers is faulty. The blocking probability can be reduced further by adding more buses.

Figures 11(a) and 11(b) show the results of simulating the distribution scheme with local tables (Section 6). With a total of 32 buses and 90% load, about 33% of the requests needed a handover, and 18% were blocked. With 40 buses, the handovers dropped to about 2.5%, and the blocking probability to about 0.32%. The percentages further dropped to 0.32% and 0.067%, respectively, with 44 buses. These values are comparable to the the results for the chessboard scheme in Figures 9(a) and 9(b).

Performance of the distribution scheme with local tables under a controller failure is shown

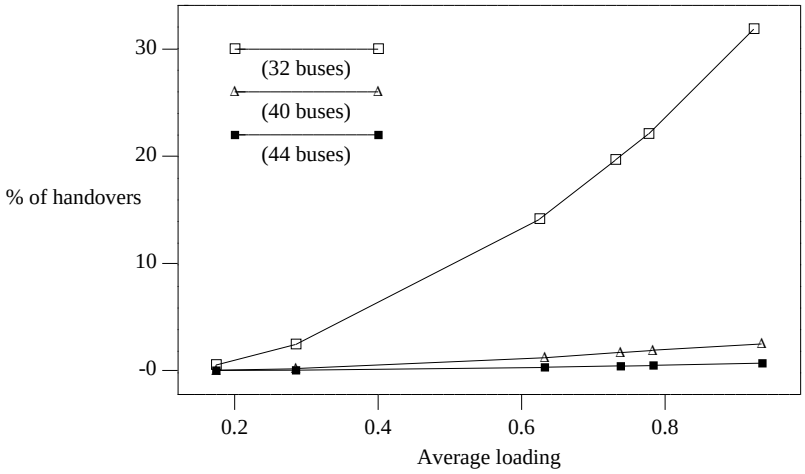


(a) Percentage of handovers vs average loading of the network

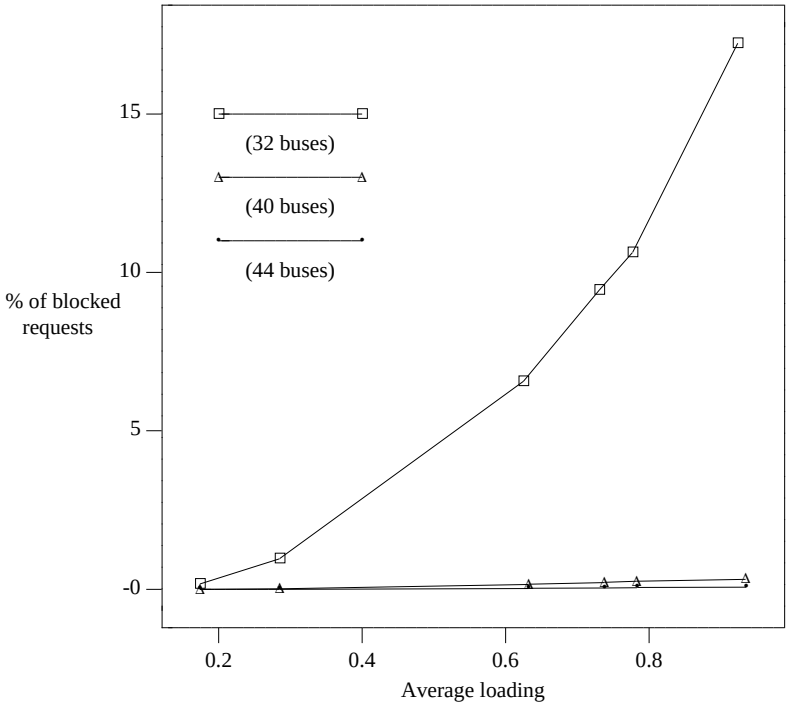


(b) Percentage of blocked requests vs average loading of the network

Figure 10: Simulation results for the chessboard scheme in a 64-port switch when one of the four controllers is faulty.

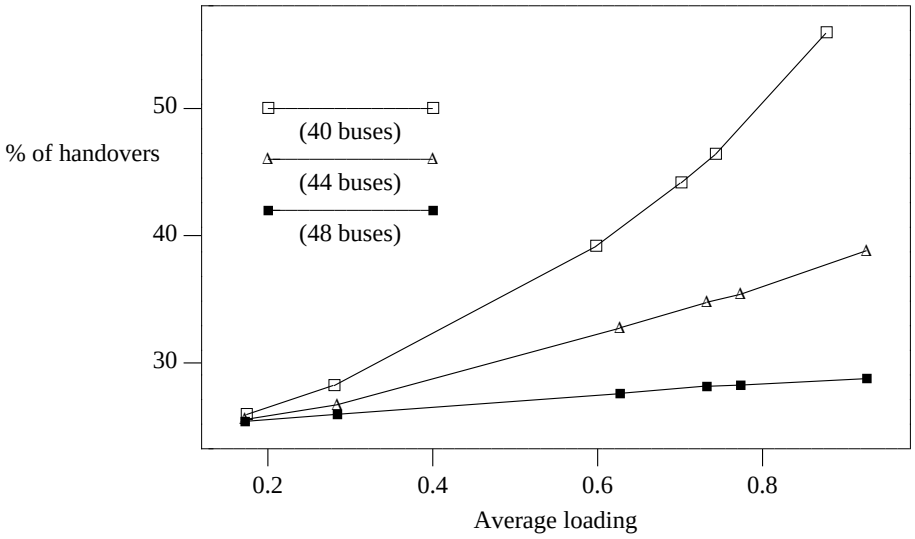


(a) Percentage of handovers vs average loading of the network

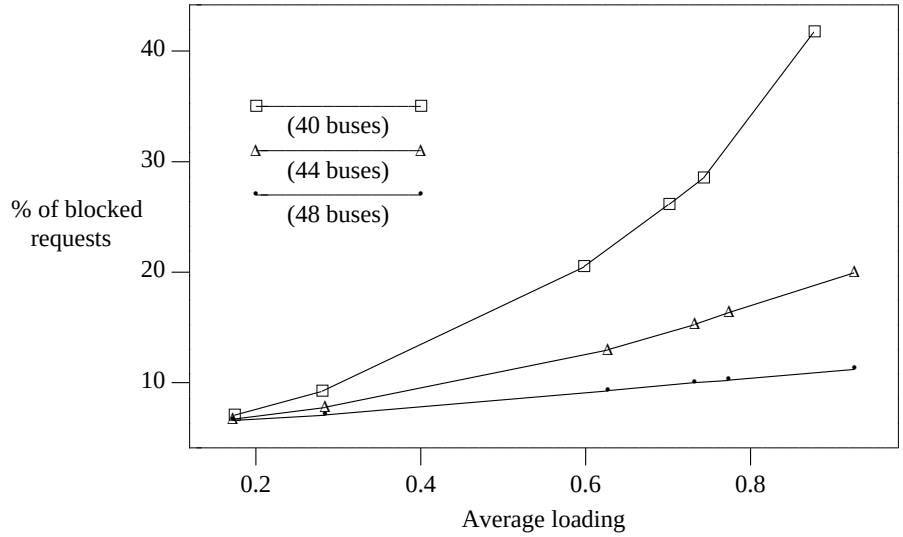


(b) Percentage of blocked requests vs average loading of the network

Figure 11: Simulation results for the distribution scheme based on local tables (Section 6) on a 64-port switch with four controllers.



(a) Percentage of handovers vs average loading of the network



(b) Percentage of blocked requests vs average loading of the network

Figure 12: Simulation results for the distribution scheme based on local tables (Section 6) when one of the four controllers is faulty.

in Figures 12(a) and 12(b). Both the percentage of handovers and the blocking probability are degraded considerably over the fault-free case. The degradation in this case is worse than in the case of the chessboard scheme. With 40 buses, about 55% of the requests were handed over to another controller, and about 41% were rejected. Even with 48 buses, the blocking probability was still 11%.

Comparing the schemes, we observe that the performance of the distribution scheme with local tables (Section 6) is not much different from that os the chessboard scheme in the fault-free case. When one controller is faulty, the performance degradation is worse for the former scheme. For small switches, however, the performance of the scheme with local tables is better; the best performance is obtained when $N = K^2$. Though the number of handovers required for this scheme is slightly higher as compared to the chessboard scheme, the blocking is considerably less. The chessboard scheme, however, has the advantage of simplicity of implementation and therefore may be faster for most applications.

8 Concluding Remarks

A number of schemes were presented for distributing the setup function of a one-sided crossbar switch. In the symmetric triangular scheme, the simplest among all of them, each request is serviceable only by one of the controllers, chosen based on the addresses of the ports involved in the request. In the event of a controller-failure the assignment of port-pairs to controllers must be modified, thus causing a temporary disruption of service. With this scheme, the number of buses needed to support nonblocking operation with K controllers is approximately $N\left(1 - \frac{1}{2\sqrt{K}}\right)$.

In the symmetric triangular scheme, the failure of a controller causes at lease a temporary service interruption. A number of distributions schemes that eliminate this problem were presented in Sections 4, 5, and 6. These schemes allow the switch to continue operating operating in the event of a controller failure, albeit with increased blocking. In the *chessboard scheme*, presented in Section 4, every request is serviceable by one of *two* predetermined controllers — a primary controller and secondary controller. A request is routed to the secondary controller if the primary

controller is not available. The distribution scheme with global balance, in Section 5, allows load-balancing among the primary–secondary pairs. The minimum number of buses for nonblocking operation in this case is $N\left(1-\frac{1}{2^{\lfloor \log_2 K \rfloor}}\right)$. This scheme requires the controllers to share the load information among themselves. The distribution scheme with handovers, presented in Section 6, avoids the need for controllers to access global load information. Instead, the primary controller may hand over the request to any of the remainining controllers based on local information. The number of buses needed for nonblocking operation in this case is $N-K$.

The distributed control mechanisms presented range in complexity. By using global information, the scheme in Section 5 attains the lower bound on the number of buses required, but requires more time for arbitration. At the other extreme, the simple chessboard scheme provides fast arbitration, but requires N buses for an N-port switch.

Although the lower-bounds we derived on the number of buses are tight, simulation results indicate that the percentage of blocked requests is quite low even when the number of busses used is significantly less than the corresponding lower bound. A small blocking probability is acceptable for most applications, allowing a signifcant reduction in the overhead introduced by the distributed control schemes.

The best choice among these schemes depends on the size of the network and the characteristics of the application. Simulation results show that the distributed scheme using local tables is best for small networks ($N \leq 36$), and when the optimality condition $N = K^2$ is satisfied. The chessboard scheme performs better for larger networks and high traffic rates, and is also the simplest. The performance degradation with a controller–failure is also lower for the chessboard scheme for large switches.

All the simulation results in this paper assume a uniform traffic-distribution among the ports. An area of further research is to investigate these schemes for non-uniform communication patterns. Another important aspect to be considered is the packaging technology to be used in the fabrication of the one-sided crosspoint chips along with the controllers. It is interesting to study how the controllers can be integrated with the crosspoint chips, resulting in faul-tolerance in both

data– and control–paths. VLSI

References

[1] Special Issue on Large-Scale ATM Switching Systems for B-ISDN, *IEEE Journal of Selected Areas in Communications*, Vol. 9, No. 8, October 1991.

[2] J. Beetem, M. Denneau, D. Weingarten, "The GF11 Supercomputer," *Proc. 12th Annu. Int. Symp. on Computer Architecture*, June 1985, pp. 108–115.

[3] B. F. Colbry et al., "High Performance Crosspoint Switch Implemented in CMOS on AVP," *Proceedings of the IEEE International Conference on Computer Design (ICCD '87)*, 1987, pp. 113–119.

[4] G. Ditlow and A. Brown, "The Delta-I Simultaneous Switching Problem," Research Report RC 10015, IBM Research Division, May 1985.

[5] C. J. Georgiou. "Fault-Tolerant Cross-Point Switching Networks," *Proceedings of the 14th International Symposium on Fault-Tolerant Computing*, July 1984, pp. 240–245.

[6] C. J. Georgiou. *Controller for a Crosspoint Switching Matrix*, U.S. Patent 4 630 045, 1987.

[7] C. J. Georgiou, T. A. Larsen, P. W. Oakhill, and B. Salimi, "The ESCON Director: A Dynamic Switch for 200 Mbits/sec Fiber-Optic Links," Research Report RC 16475, IBM Research Division, January 1991.

[8] C. J. Georgiou and A. Varma, *One-Sided Crosspoint Switch with Distributed Control*, U.S. Patent 5 072 217, December 10, 1991.

[9] J. Ghosh and A. Varma, "Reduction of Simultaneous-Switching Noise in Large Crossbar Networks," *IEEE Transactions on Circuits and Systems*, Vol. 38, No. 1, January 1991, pp. 86–99.

[10] J. Ghosh, A. Hukkoo, and A. Varma. "Neural Networks for Fast Arbitration and Switching Noise Reduction in Large Crossbars," *IEEE Transactions on Circuits and Systems*, Vol. 38, No. 8, August 1991, pp. 895–904.

[11] R. F. Miracky et al., "Design of a 64-processor by 128-memory Crossbar Switching Network," *Proceedings of the IEEE International Conference on Computer Design (ICCD '88)*, October 1988, pp. 526–532.

[12] A.J. Rainal, "Computing Inductive Noise of Chip Packages," *AT&T Bell Laboratories Technical Journal*, Vol. 63, 1984, pp. 177–192.

[13] K. K. Saluja, R. Sharma, and C. R. Kime, "A concurrent testing technique for digital circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 7, No. 12), Deceomber 1988, pp. 1250–1259.

[14] A. Varma, J. Ghosh, and C. J. Georgiou. "Reliable Design of Large Crosspoint Switching Networks," *Proceedings of the 18th International Symposium on Fault-Tolerant Computing*, Tokyo, June 1988, pp. 320–325.

[15] A. Varma, J. Ghosh, and C. J. Georgiou. "Rearrangeable Operation of Large Crosspoint Switching Networks," *IEEE Transactions on Communications*, Vol. 38, No. 9, September 1990, pp. 1616–1624.

[16] A. Varma and S. Chalasani, "Fault-Tolerance Analysis of One-sided Crosspoint Switching Networks," *IEEE Transactions on Computers*, Vol. 41, No. 2, February 1992, pp. 143–158.

[17] A. Varma, "Combinatorial Design of Bus-based Interconnection Structures," Research Report RC 12550, IBM Research Division, 1986.

[18] W. D. Wallis, *Combinatorial Designs*, Marcel Dekker, Inc., 1988.