A High Speed Scheduler/Controller for Unbuffered Banyan Networks *

C. Salisbury
Broadband Switching Systems
Lucent Technologies
Naperville, IL

R. Melhem
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA

Abstract

This paper describes the design of a centralized controller / scheduler for a communication switch with a Banyan switching fabric built using unbuffered switches. The controller accepts a set of connection requirements and identifies a non-conflicting subset that can be used to set the state of the switches for data transfer. The logic can be implemented using a relatively small number of gates and can be pipelined to provide rapid control of the switch fabric. The controller can be used with any replicated banyan network fabric that provides sufficient switching speed and data bandwidth.

Keywords: Communication switch architecture, circuit-switching, Banyan network

1 Introduction

The advent of packet switched techniques to integrate the transport of data from different applications and the concurrent increase in network bandwidth has heightened the need for high speed communication switches. Multistage interconnection networks (MINs) such as banyan networks have been widely investigated for use in these switches. A banyan network connecting N input ports to an equal number of output ports can be constructed using $O(N \log N)$ 2×2 switches. However, contention for these switches prevents a banyan network from providing simultaneous connections for all possible permutations of connections between input and output ports. This blocking characteristic limits the performance that can be obtained from a single banyan network.

A wide variety of banyan-based network architectures have been developed to overcome the performance limitations imposed by blocking[12]. Many designs place control logic in the switches to buffer and / or route packets. Packets which cannot be routed correctly due to contention may be dropped or passed through the switch fabric using *deflection routing*[13]. Packets which exit the network at an incorrect destination are not delivered, but are instead discarded or rerouted. A key performance measure of these designs is the rate at which packets must be discarded.

The loss rate can be reduced by using multiple banyan networks. For example, *replicated* architectures distribute the workload across several networks placed in parallel[1, 6]. *Tandem* architectures place banyan networks in a series[13]. *Dilated* architectures increase the number of links between switches in the banyan[6]. These architectures have been enhanced by interconnecting replicated or tandem networks using dilated links[15] or bypass connections[4]. An approach that can reduce packet loss with a minimal increase in the amount of hardware is to *recirculate* packets

*This work is supported in part by NSF award MIP-9633729 to the University of Pittsburgh

back into the network that would otherwise be lost[14]. Recirculation has been incorporated into many of these designs[11, 13].

Similarly, parallel networks have been used to increase the bandwidth of the switch fabric. Four switching planes with a one byte interface are used in [2] to handle input line speeds of up to 2.4 Gb/s. The switch described in [7] uses a selectable number of parallel one-bit wide cross-bars to improve switch throughput. Optical switch fabrics may provide another means of obtaining high bandwidth, where parallelism can be provided through wavelength division multiplexing (WDM). The use of electronic control logic in the switch elements may greatly reduce the performance of such an optical network, however.

The focus of this paper is to describe a high speed controller that can be used with a variety of banyan network switching fabrics, where the fabric provides the required bandwidth using simple 2×2 switches that do not have buffers or control logic. This means that connections between input ports and output ports are managed with centralized, circuit-switching techniques. The proposed controller could control a single high speed, highly reconfigurable optical banyan network built from electro-optical switches to provide all-optical paths through the switch fabric[10]. Alternatively, the switch fabric could be a replicated architecture consisting of electronic, unbuffered, banyan networks. Controller design parameters can be adjusted to provide operational and performance characteristics that are similar to those of parallel, tandem, and recirculating banyan architectures.

Central control has been used in other communication switch architectures. For example, a centralized scheduler for a cross-bar based switch fabric is described in [7], and a centralized algorithm for control of a MIN is described in [9]. With the centralized approach, the controller accepts a set of requests from the input ports for connections to output ports. The requests are processed to identify a contention-free subset. This information is returned to the input ports as authorization to transmit packets through the switch. Since packets are not transmitted until the desired connection is available, requests that are not granted are resubmitted[3], rather than recirculated.

During request processing, the controller can also determine the state required at each switch to establish the selected connections. When this *network state* information is used to set the states of the switches prior to data transmission, the proposed device acts as a switch controller. When it is not necessary to set the network state, the device acts as a transmission scheduler. The techniques used to determine the network state for unicast connections can be extended in a natural way to handle multicast.

To provide satisfactory performance, the controller must be capable of determining non-conflicting sets of connection requests at a rate sufficient to handle the requests arriving from the input ports. The controller can be pipelined to provide successive network states in constant time, independent of the number of ports. Since the states developed by the controller provide connections for a contention-free set of requests, the blocking characteristics of the banyan network affect the performance of the controller, rather than that of the switch fabric. The switch fabric can therefore be chosen based on the required bandwidth. Different controller implementations can be used to provide the switch with performance characteristics that are similar to those of replicated, tandem, and recirculating banyan architectures.

The rest of the paper is organized as follows. In section 2, we describe connection requests and their processing. We show how this can be extended to include multicast operations in section 2.5. We describe the hardware design and considerations for pipelining the controller in section 3, and present some performance results in section 3.2 before concluding in section 4.

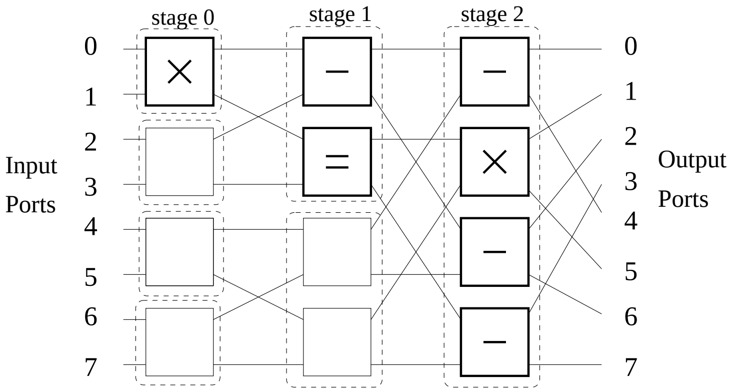


Figure 1: Reverse cube banyan network.

2 Centralized Control of a Banyan Network.

A banyan network is a class of multistage interconnection networks (MIN) that provides a unique path from any input port to any output port. A banyan with $N = 2^n$ input ports can be built with 2×2 switches using n stages of $N/2$ switches. An example of a banyan network with a reverse cube interconnection pattern for $N = 8$ and $k = 2$ is shown in Figure 1. Data transmitted by an input port may pass through a well-defined subset of the switches in the network. The switches that can be used for data transmitted by input ports 0 and 1 are highlighted in the figure. The banyan structure restricts the nature of the contention for switches that can take place at each stage of the network, and the controller is organized to take advantage of this.

The switch architecture is pictured in Figure 2. Each input port queues incoming packets in a packet buffer. Using routing information in the packet header, the input port determines the required output port and constructs a request word which describes the required routing through the switch fabric. The request word is presented to the controller and processed with requests from other input ports to identify a contention-free subset. In the next section we describe the contents and structure of a request word for unicast connections. In section 2.2 we present the architecture of the controller and describe how contention is resolved.

2.1 Requesting Connections.

The structure of a banyan network restricts the nature of the contention for each switch. We define a *switch group* at network stage i to be all the switches that can be connected to a given input port. The reverse cube network is an example of a bi-delta banyan network. One property of these networks is that the grouping of switches does not depend on the input port chosen to construct the groups. An input port that can be connected to one switch in a switch group can be connected to all switches in that group. Switch groups are enclosed in dashed boxes in Figure 1. The set of input ports that can be connected to a switch group is referred to as an *input group*. In Figure 1, the input group for the upper switch group at stage 1 consists of input ports 0, 1, 2, and 3.

An input port creates a request word that specifies the switch states required to form a connection to the desired output port. Each input port can be connected to a total of $\sum_{i=0}^{n-1} 2^i = N - 1$ switches so that connection requirements can be represented with $O(N)$ bits of information. Although a connection can use only one switch at each stage, the request word specifies the desired setting of all switches to which the input port can be connected. Thus, in addition to indicating switches required in the straight or cross states, switches which are not used for a connection re-

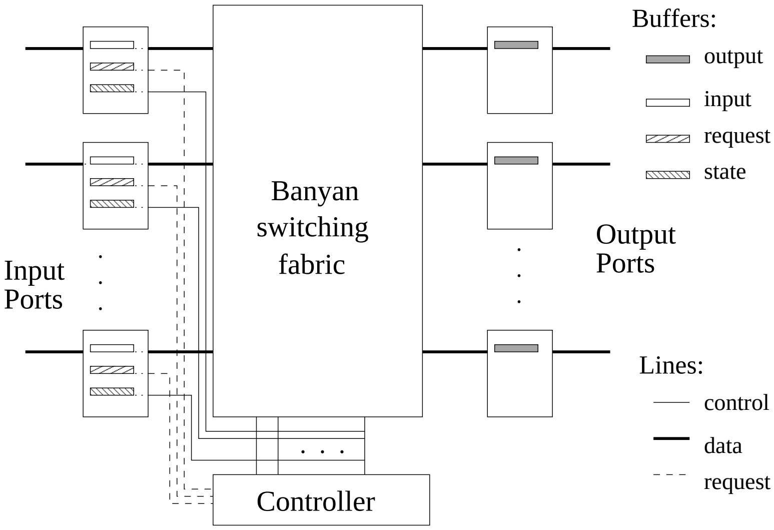


Figure 2: Switch architecture.

Binary Value		Notation	Meaning
c=Care	s=State		
1	1	`=`	Request straight state
1	0	`x`	Request cross state
0	0	`-`	Don't care

Table 1: Requesting a switch setting.

quire a "don't care" indication. Thus, switch requirements can be specified in two bits. Table 1 shows the binary representations we will use for these states and the notation that will be used in subsequent examples.

Within a request word, settings for each switch are ordered by stage and, within stage, by an offset reflecting relative physical position. For example, to connect input port 0 to output port 5 in the reverse cube network, the following switch states are required (see Figure 1):

- The switch at stage 0, row 0 must be in the cross state (`x`).
- The switch at stage 1, row 1 must be in the straight state (`=`).
- The switch at stage 2, row 1 must be in the cross state (`x`).
- The other switches to which port 0 could be connected do not affect this connection, and are marked with "don't care" (`-`).

The request word from input port 0 is therefore (`x -= -x--`), which can be represented in binary as (`10  00 11  00 10 00 00`).

2.2 Resolving contention.

The controller is built from Contention Resolution Units (CRUs) which are organized into a binary tree structure with n levels, corresponding to the n stages of the network. A controller for a switch with eight input ports is shown in Figure 3. Request words are sent to the CRUs at the lowest level of the tree. At each level of the tree, CRUs resolve contention for switches at one stage of

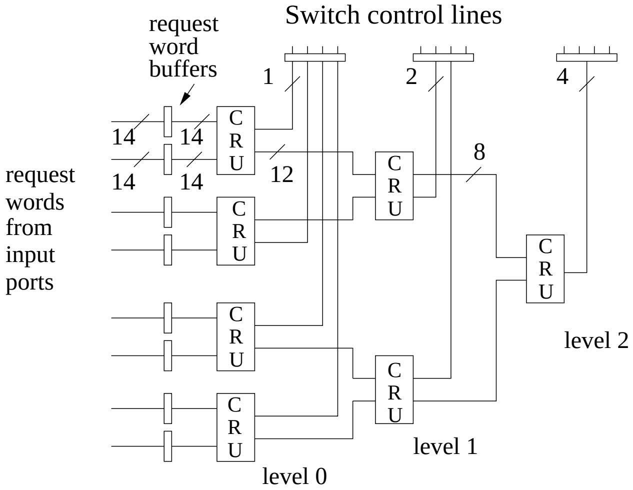


Figure 3: Controller architecture for an 8 $\times$ 8 Banyan.

the network. In the figure, the levels of the tree are labeled to indicate the corresponding stage of switches. Each CRU receives two requests and resolves contention for all the switches in a switch group, so that a CRU at level i resolves contention for 2^i switches. An externally controlled priority signal (not shown in the figure) is used to select the request word which prevails when contention occurs.

After resolving contention, the state required at each switch can be determined and placed on the signal line controlling the switch. The CRU then merges the two request words into a single word representing the additional switching requirements of all the successful requests. This merged request word is then passed to the CRU at the next level. The process is repeated at each level to resolve contention for the remaining switches and complete the connection.

When the controller has resolved all contention, the control signals may be used to set the switches to the desired state and establish the connections. Each input port monitors the switch control lines and determines if its request has been granted by comparing the request to the network state output by the controller. Input ports with successful requests transmit their packets through the switch fabric. Since requests flow through the controller rather than packets, unsuccessful requests are resubmitted rather than recirculated. An advantage of this is that the need to resequence packets at the output port can be avoided. Since a request is granted to provide a connection between an input port and an output port, the input port can queue packets by destination and send packets in the correct order, regardless of the order in which requests are granted. If desired, however, additional hardware could be used at the input and output ports to allow all packets to be sent and to provide deflection routing.

The Lemmas below are a consequence of the fact that a banyan network provides a unique path between an input port and an output port. They can be used to show that two requests can contend for a switch at only one stage of the network. Thus a merged request word contains no unresolved contention, and the CRU at the next level of the tree need only resolve contention between the two merged request words that it receives. This also means that CRU operations can be performed independently and in parallel for all switches in the group.

Lemma 1: Consider a banyan MIN and two different connections, C_1 and C_2, that require switch S to be set in state s. If C_1 passes through switch S', $(S' \neq S)$, then C_2 does not pass through S'.

Proof: If C_2 passes through S', then there are two paths that connect S and S'. One path is used by C_1 and one is used by C_2. This means that C_1 (and C_2, as well) could be established using two different paths through the network. This, however, contradicts the fact that a banyan network provides a unique path between each input and output. Thus, C_2 does not pass through S'.
□

Lemma 2: Consider a banyan MIN and two different connections, C_1 and C_2, that require switches S_1 and S_2 respectively ($S_1 \neq S_2$) and which are in the same switch group at stage i. Then if C_1 requires a switch S' at stage j, $j > i$, then C_2 does not require S'.

Proof: By definition of a switch group, the input port at the source of C_2 can be connected to both switches S_1 and S_2. If connections exist from both S_1 and S_2 to S', then there are two ways to form connection C_2 (and connection C_1, as well). This contradicts the uniqueness of paths in a banyan MIN.
□

These lemmas tell us that given requests for switches in a group at stage i, if there is contention between two of these requests for a switch at stage j, where $j \geq i$, then the two requests must also contend for a switch in the switch group at stage i. After the CRU resolves contention for a switch at stage i, the winning request will not contend at any stage of the network with the winning request from any other switch in this switch group at stage i. The results from each switch in the switch group can be merged independently into a combined request for switches at the following stages. Further, if two requests for a switch specify the same state, then the switch states contained in both requests do not conflict at any stage and the connections can be provided simultaneously.

A CRU processes the two requests as follows:

- If both requests specify the same setting for a switch in the current stage, there is no contention, and the switch can be set as requested. The merged request word for the next level CRU will request all switches required for both connections at subsequent stages of the network.
- If one request word specifies "don't care" for the switch, there is no contention. If the other request specifies a switch setting, it is successful and its requirements are placed in the merged request word.
- If two requests specify "don't care" for the switch, the switch can be set arbitrarily.
- If the requests are for different switch settings, there is contention. One request is chosen to be successful based on some set priority. The unsuccessful request is blocked and dropped from further processing. The additional requirements for switches from the successful request are included in the merged request word.

To merge requests, the CRU must obtain information about physical switch interconnections directly from the request word. This information is provided by the ordering of the requested states within the request word. The stage of a switch and its offset within its switch group can be mapped to the stage and offsets of the switches connected to its upper and lower outputs.

Let $S_{i,j}$ be the switch at stage i and offset j. In the banyan network of Figure 1, this switch is connected through its upper output port to switch j at stage $i+1$, $S_{i+1,j}$. We denote this $\mathcal{C}_u(S_{i,j}) = S_{i+1,j}$. Similarly, the connection through the lower port is to switch $\mathcal{C}_l(S_{i,j}) = S_{i+1,j+2^i}$. These mappings are pictured in Figure 4. From each switch at stage i in a request word, these

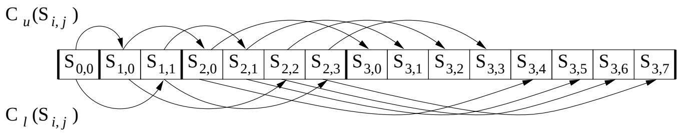


Figure 4: Physical connections between switches in a request word.

Input port	Output port	Request Word	Level 0		Level 1		Level 2	Final state observed by input port	Request Satisfied
			Switch state	Merged request	Switch states	Merged request	Switch states		
0	5	`x -= -x--`	`x`	`x= -xx-`	`x=`	`=xx-`	`=xx-`	`x x= =xx-`	Yes
1	6	`x x- --x-`						`x x= =xx-`	Yes
2	5	`x -x -x--`	`x`	`xx =x--`				`x x= =xx-`	No
3	0	`x x- =---`						`x x= =xx-`	Yes
4	5	`x -= -=--`	`x`	`-= -=--`	`==`	`-=x-`		`x == =xx-`	No
5	3	`= -x ---x`						`x == =xx-`	No
6	2	`= =- --x-`	`=`	`=- --x-`				`= == =xx-`	Yes
7	0	`x x- x---`						`= == =xx-`	No

Table 2: Resolving contention for switches.

mappings can be applied recursively to find all switches in subsequent stages which could be required as part of a connection passing through switch $S_{i,j}$. The set of all such switches will be called the *merge set* for $S_{i,j}$. From Lemma 2, the merge sets for switches in the same switch group at stage i are disjoint. As will be described in Section 2.4, the circuitry within a CRU will associate each switch with its merge set.

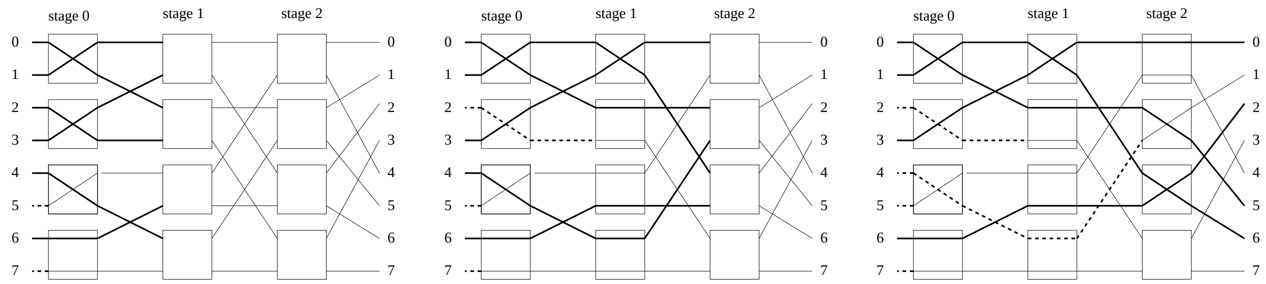
2.3 An example of contention resolution.

Let the desired port connections be those shown in the first two columns of Table 2. Using the reverse cube network of Figure 1, the processing of request words at each level of the controller is shown in the remaining columns of Table 2. The corresponding switch settings are shown pictorially in Figure 5.

Each input port creates a request word specifying the switch settings required to set up a connection to the desired output port, as shown in column 3 of Table 2. The requests from ports 0 and 1, 2 and 3, *etc.* are combined at the lowest level CRU in the controller to determine settings for the stage 0 switches. For example:

- Both ports 0 and 1 request the (`x`) setting for the stage 0 switch. There is no contention. The stage 0 switch will be set in the cross position. The merged request for the remaining switches is (`x= -xx-`).
- Ports 6 and 7 have conflicting requests for the stage 0 switch. If the request from port 6 is given higher priority, then the stage 0 switch will be set in the straight position and the request from port 7 is blocked. The merged request for the remaining switches is (`=- --x-`).

Column 5 of Table 2 shows the merged requests developed as a result of resolving contention for switches at stage 0. The intended state of the stage 0 network switches at this point is shown in Figure 5a. The connections still being processed are shown as a dark solid line, while blocked connections are shown as dashed lines.



a) After step 0 b) After step 1 c) After step 2

Figure 5: Setting switches to establish connections.

The merged requests from level 0 are passed on to level 1 CRU's and processed for each stage 1 switch in the request word. Consider the requests from input ports 0/1, and 2/3.

- Both requests for the uppermost switch at stage 1 specify the cross state. This switch is located in the request message at stage $i = 1$, offset $j = 0$. $S_{1,0}$ is connected to switches $\mathcal{C}_u(S_{1,0}) = S_{2,0}$ and $\mathcal{C}_l(S_{1,0}) = S_{2,2}$. The merged request is constructed from the settings of these switches as specified in the two requests. Each request will specify a value for one of these switches, and "don't care" for the other. The remaining switch requirements for each request are placed into the merged request by the level 1 CRU. The merged request for stage 2 switches based on this stage 1 switch is (`=-x-`). This stage 1 switch will be set in the cross state, and both paths through this switch will be used.

- The requests conflict for the lower switch at stage 1. This switch has offset $j = 1$. If contention is resolved in favor of input ports 0 and 1, the switch will be set in the straight state. This connection requires switches $\mathcal{C}_u(S_{1,1}) = S_{2,1}$ and $\mathcal{C}_l(S_{1,1}) = S_{2,3}$. Since $S_{2,1}$ is requested in the cross state, the merged request for stage 2 switches must contain (`-x--`). The complete, merged request is therefore (`=xx-`).

The final level works in the same manner as the lower levels, except that there is no need to construct a merged request. At this point, the controller can set the state of all the switches. Switches which remain in the "don't care" state at the end of the process may be set arbitrarily. The state of all switches can be represented with a single bit and placed on a bus $\frac{N}{2}\log N$ bits wide. Each input port is connected to the signal lines for switches present in its request message. By comparing the signals to the request sent to the controller, the port can determine if its request has been granted. If so, it transmits the packet at the head of the queue. In this example, four connections are established connecting port 0 to 3, 1 to 5, 3 to 0, and 6 to 4.

2.4 CRU Implementation.

A CRU has three types of components: conflict detection, switch control, and merge control. Conflict detection signals when two requests have conflicting requirements for a switch at the current stage. Switch control places the correct signal on the switch control line. Merge control combines two requests into a merged request word for input to the next level CRU.

A CRU at level i has a conflict detection component for each of the 2^i switches in the switch group at stage i for which it resolves contention. These components signal when the two requests have conflicting requirements for a switch. The logic to perform conflict detection is shown in Figure 6. The upper portion of the figure determines when both requests care about the switch

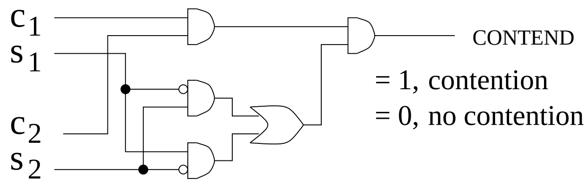


Figure 6: Conflict detection logic.

state ($c_1 = c_2 = 1$), and the lower portion performs an exclusive-or operation to detect when the two specifications are different ($s_1 \neq s_2$). Contention occurs when both of these conditions are true. The output of contention resolution is the CONTEND signal.

Each request contending for switch $S_{i,j}$ contains a set of requirements for the additional switches at later stages of the network that are required to complete a circuit. The CONTEND signal controls whether one or both sets of requirements are placed into the merged request word. The switches that may be required to complete the circuit must be part of the merge set for switch $S_{i,j}$. The CONTEND signal is connected to the merge control components of all switches in the merge set. These switches are located in specific positions within the request word as determined by the mapping relationship shown in Figure 4. The CONTEND signal is also connected to the switch control component for $S_{i,j}$.

Figure 7 depicts the organization of a CRU at level 0 in the switch controller for eight input ports. Since the level 0 CRU resolves contention for only one switch, a single output signal controls the merge logic for all remaining switches in the two requests. Figure 8 shows how the connections are made in a CRU at level 1, and Figure 9 shows the CRU at level 2.

The CONTEND signal is connected to a switch control component for each switch $S_{i,j}$ in the current switch group, and to a merge control component for each switch in the merge set of $S_{i,j}$. The merge control logic uses an external priority signal and the CONTEND signal to combine successful requests into a single request word. The logic to perform merge control is shown in Figure 10. This logic operates on requests for switches not in the current switch group. Consider first the case when there is no contention. From Lemma 1, the requested circuits cannot again contend for a switch. Thus whenever one request word contains a request for a switch, the other request word must contain a "don't care" for that switch ($c_i = s_i = 0$), and the merger of requests can be performed by OR gates. These outputs are used when CONTEND = 0, indicating no contention.

When there is contention, the switch requirements from the unsuccessful request must not be placed in the merged request word. When CONTEND = 1, the merged output is obtained from multiplexers which are controlled by the priority signal to select the high priority request. A priority of 0 selects c_1 and s_1. Requirements from the unsuccessful request will not appear in the merged output. In most cases, they will be replaced in the merged request by a "don't care" value copied from the successful request.

Requests for switches in the current switch group are sent to switch control logic as well as the conflict detection logic. The logic for switch control is identical to that for merge control. When there is no contention, the required state can be determined by an OR since the two requests are either identical, or at least one of them is a "don't care". The "state" bit can be used to control the switch. The "care" bit is not used and can be discarded.

Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs) are good targets for the implementation of a CRU-based controller. An 8 port controller as shown in Figure 3 requires approximately 130 pins and 675 gates. The logic density on today's devices is high enough that the I/O pin count is likely to be the constraining factor in the choice

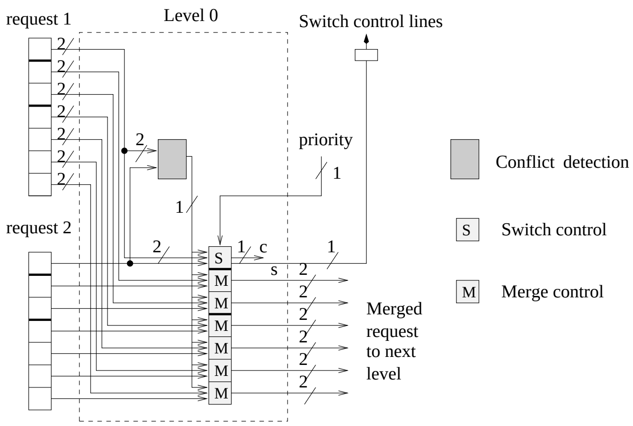


Figure 7: Level 0 CRU block diagram.

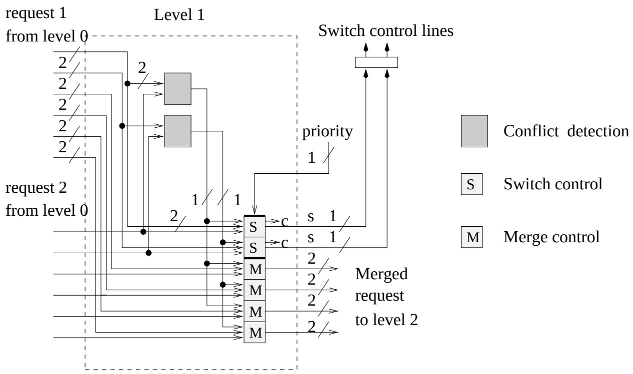


Figure 8: Level 1 CRU block diagram.

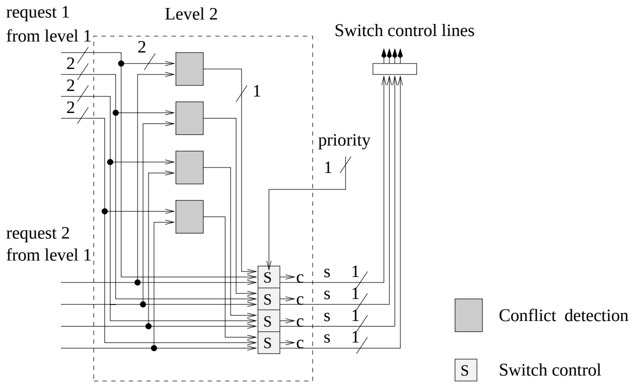


Figure 9: Level 2 CRU block diagram.

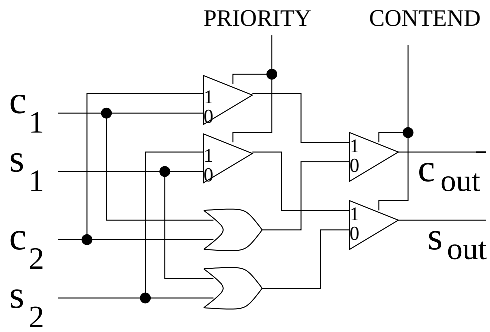


Figure 10: Merge control logic.

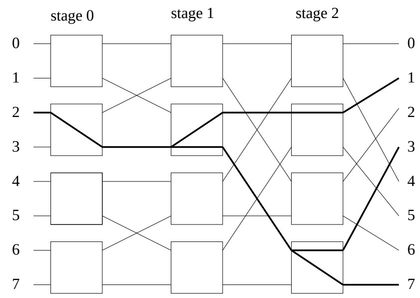


Figure 11: A multicast connection.

of device.

2.5 Multicast connections.

Two approaches can be taken with the establishment of multicast connections, either *partial* multicast or *full* multicast[8]. With full multicast, all requested connections must be established or the multicast request fails and no data is transmitted. Full multicast is difficult to handle for several reasons. To provide for consistent processing, a switch requested as part of a multicast should be distinguishable from a switch requested as part of a unicast. Further, to avoid mutual interference between multicast requests, requests for switches cannot be processed independently. To handle these requirements adds a great deal of complexity to the control algorithms and hardware that is required.

On the other hand, partial multicast is a work conserving approach to multicasting[5]. With partial multicast, a message can be sent over any portion of the connections that can be established. Any unsatisfied portion of the multicast request is simply requested again. With partial multicast, contention can be resolved for each switch independently. Thus, the approach from the previous sections can be extended in a natural way to handle partial multicast communications.

As shown in Figure 11, multicast connections can be provided when the 2×2 switches in the network provide broadcast settings. The "upper broadcast" setting connects both output ports to the upper input port, and the "lower broadcast" setting connects them to the lower input port. The existence of four switch states means that two signal lines are required to control the state of each switch. For example, the "care" bit which is discarded when determining a unicast switch state could be retained and a `0` value used to represent a multicast setting on the additional control line. We will use a value of `01` to indicate lower broadcast, and a value of `00` to indicate upper broadcast.

To continue to request a switch state using only two bits, the `01` bit combination which is not used in Table 1 can represent a request for a broadcast state. The switch control component of a CRU can determine which of the two broadcast states is needed based on the origin of the request. An `01` in the lower request word represents a request for the lower broadcast state, while an `01` in the upper request word represents a request for the upper broadcast state. With this approach, the representation of a switch request for upper broadcast would be different from the representation of the upper broadcast switch state when placed on the switch control lines. Thus for multicast, the output of a switch control component of a CRU is no longer identical to that of a CRU merge control component.

A multicast request is structured the same as a unicast request. To build it, the input port develops a routing request for each destination individually. These requests are then merged into a combined request word. "Contention" for a switch is resolved during this merger by requesting the switch in a broadcast setting. The resulting multicast request is sent to the controller.

When a CRU resolves contention between two requests, a request for a broadcast state conflicts with a request for any state other than "don't care". This may be handled with modifications to the conflict detection hardware, as shown in Figure 12. The unicast conflict detection logic continues to signal contention when the care bits of both requests are `1`. Additional logic is used to detect broadcast states in either request word. When one request is for a broadcast state and the other request has the care bit set to `1`, there is contention. In these cases, a `1` will be present on one of the lines labeled d_u and d_l in the figure when it involves upper or lower broadcast, respectively. Contention also occurs when both requests are for a broadcast, and is signalled on the line labeled d_b. The signals for these four cases are combined through a series of OR gates to produce the CONTEND signal. In addition, the UPPER signal is passed to the switch control components to indicate the presence of an upper broadcast request.

The switch control logic must determine when a request for upper broadcast wins the contention and then set the upper broadcast state correctly. Appropriate modifications to the merge control logic of Figure 10 are shown in Figure 13. The first additional OR gate outputs a `1` unless the lower word contains "don't care" for the switch. When the lower word does contain a request and the lower request word receives high priority, the output of the following NAND gate will be a `0`. In this case, the request from the upper request word will not be successful. The output of the AND gate that controls the multiplexer will be a `0`, and the output of the switch control component will be that developed by the unicast logic. In all other cases, the request in the upper word will be successful, and the multiplexer is controlled by the presence or absence of an upper broadcast request. Only when an upper broadcast is also present (UPPER= 1) will the output of the AND gate be `1` controlling the multiplexer to select `0` for the s_{out} state bit. In the absence of a successful upper broadcast request, the signal produced by the unicast merge logic can be used directly. Note that when a request for lower broadcast is successful, the unicast switch control logic outputs the correct state, $c_{out} = 0$ and $s_{out} = 1$, taken from the lower request word.

The unicast merge control component for switches in the merge set can be used for multicast without change. When a broadcast state wins the contention for a switch, the merged request for the next level CRU must contain requests for circuits extending from both upper and lower output ports of the switch. The merge control logic of Figure 10 and the CRU wiring previously described will correctly perform this function, placing into the merged request all the requested switch states along all branches of the multicast. Similarly, a successful unicast request will be placed in a merged request word, removing all branches of an unsuccessfull multicast request.

Ports must again monitor the switch control lines to determine the success or failure of their requests. Each destination must be checked individually against the final network state to verify

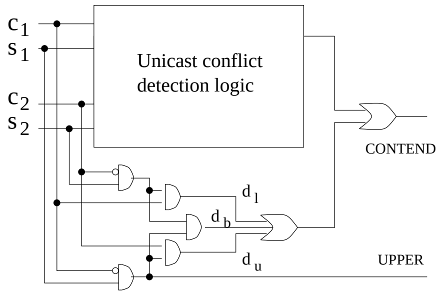


Figure 12: Conflict detection logic for multicast.

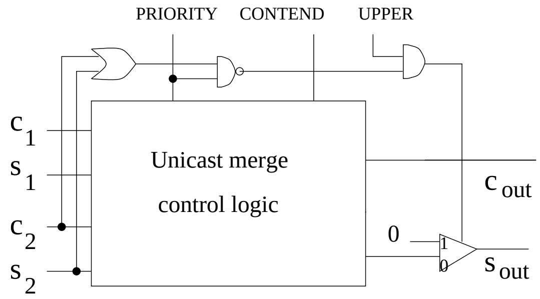


Figure 13: Switch control logic for multicast.

that at least one circuit has been obtained and data transmission is required, and to identify the unsatisfied portion of the multicast request.

Note that switch state 00 will place the switch in the upper broadcast setting, although it will also be the result of "don't care" requests. This ambiguity will not affect the detection of success or failure, however. This is because a broadcast request never loses when contending with "don't care", so that when the port obtains all other switch settings required for the connection, the 00 setting must indicate the broadcast state.

3 Controller performance

The controller's design distributes the logic used to resolve contention and establish connections. There is no single, central component which has knowledge of all the connection requests and which can optimize the number of connections provided by a network state. The key to obtaining good performance with this design is for the controller to resolve contention and develop network states quickly and for the switch fabric to control the internal switches and transmit data at a commensurate rate. A key feature of this design is that the controller can be used with different banyan switch fabrics. An electronic switch fabric could consist of several replicated banyan networks. An optical switch fabric could be a single high speed, highly reconfigurable network, possibly using wavelength division multiplexing. With a suitable switch fabric, performance is determined by the rate at which the controller can develop contention-free network states relative to the rate at which data arrives at the input ports.

From Figure 6, we see that contention can be detected in the time required to propagate a signal through three logic gates. The initial merge control gate operations can be overlapped with the contention detection logic. Assuming 2 logic gates per multiplexer, a CRU can detect and resolve contention and merge requests with a total of 5 gate delays. Assuming a delay of 4 ns per gate, each level of the CRU hierarchy requires 20 ns to resolve contention. We call the time required by a CRU to resolve contention a *CRU cycle*. To determine the state of every switch in the network requires a controller with $\log N$ levels of CRU. The time to resolve contention among a set of requests thus depends on both hardware technology and network size. The controller should be designed to be fast enough to identify non-conflicting subsets of connections for all arriving packets.

With the above assumptions, resolving contention for a switch with 512 input ports requires 180 ns. An input port receiving 53 byte packets at a rate of 2.4 Gb/s must process a packet in 177 ns before the next packet arrives. Thus, this design could create roughly one network state per time slot, and network blocking characteristics would lead to unacceptably high packet losses. To obtain satisfactory performance would require either slower line speeds or fewer input ports, or both. For 32 ports, roughly two network states could be created per time slot, allowing the use of a recirculating strategy to reduce packet loss. With 150 Mb/s input lines, a controller for a switch with 512 input ports could create approximately 16 network states, which could reduce packet losses greatly.

Another approach to increase the speed with which contention-free network states can be generated is to use additional hardware to pipeline the controller. This can allow the controller to produce a new contention-free state as often as every CRU cycle.

3.1 Pipelining the controller

Additional hardware can be used to increase the speed with which the controller generates contention-free network states. Latches can be inserted between CRUs so that requests can be submitted to the controller in a pipelined fashion. Latches must also be added to the switch control lines to delay the control signals so that they arrive after all levels of contention have been resolved. We will assume that the addition of these latches increases the CRU cycle time to no more than 30 ns.

If time is measured in units of CRU cycles and the time between packet arrivals is t_p, then without pipelining the number of states produced between packet arrivals is $\frac{t_p}{n}$, where $n = \log N$. With a fully pipelined design, one state can be produced every CRU cycle providing t_p network states between packet arrivals. Since each state is produced in constant time, the number of states between packets no longer depends on switch size. With a 30 ns CRU cycle time, the number of states produced ranges from six states at an input line speed of 2.4 Gb/s up to roughly 100 states for a 150 Mb/s line. For comparison, to obtain packet loss rates of 10^{-6} a tandem banyan network of size $N = 1024$ input ports may require 14 states[12].

In a fully pipelined design, latches are placed between all connected CRUs. More generally, the placement of latches is a design decision, and several levels of CRU may be contained within a single pipeline stage. We call this number the pipeline parameter, u. The number of pipeline stages, also called the pipeline depth d, is thus $\lceil \frac{n}{u} \rceil$. Normally u is chosen to be a factor of n, so that $n = du$. The pipeline cycle time is defined to be the time per pipeline stage. We will include latching time in the CRU cycle time, so that the pipeline cycle time is equal to the pipeline parameter u. The total time to process a request is du, and the controller can create $\lfloor \frac{t_p}{u} \rfloor$ states between packet arrivals. Thus, we require $u \leq t_p$.

The basic operation of an input port with the pipelined controller is as follows. The port submits a request to transmit a newly arrived packet into the pipeline at a cycle determined by the request submission policy. It then waits until the request has been processed. If the request was successful, it transmits the packet. If it was not, the request is resubmitted or dropped as determined by the request resubmission policy.

A simple request submission policy is that all input ports submit requests into the pipeline at the same time. Performance will be representative of that obtained from a single banyan network. Another policy is to place the input ports into $\lfloor \frac{t_p}{u} \rfloor$ groups, and to have each group submit at a different time into different pipeline cycles. In this case, contention in each cycle is between a subset of all the requests and performance will be representative of a replicated architecture. To fully exploit the pipeline, each input port could be allowed to submit a request into each pipeline cycle. Input ports could look ahead at up to $\min\{\frac{t_p}{u}, d\}$ packets in the input buffer and submit a request for each one. This will increase pipeline utilization and may reduce the effect of head of line blocking. Performance will depend on several factors, including the presence of requests being resubmitted after a previous failure.

We will consider three policies for resubmitting requests. The first policy is that requests are *never* resubmitted. Requests not satisfied after a single pass through the controller are lost. With this policy, a single set of connections is developed from each set of requests. To reduce contention, input ports are partitioned and each partition is directed to submit requests during a different pipeline cycle, as described above. The controller will develop non-conflicting sets of connections that are similar to the sets developed by a replicated banyan network[6]. The controller can configure $\lfloor \frac{t_p}{u} \rfloor$ replicated banyan networks or a single highly reconfigurable (possibly optical) banyan network with the required bandwidth.

A second policy is to *always* resubmit unsuccessful requests. Packets received by an input port are never discarded or lost. Packets are buffered at the input port until they can be transmitted. Depending on packet arrival rates, network flow control may be needed to eliminate input buffer overflows. Unsuccessful requests are resubmitted immediately and the controller emulates a recirculating banyan network[14].

A third policy is to resubmit requests subject to some condition. One approach is to resubmit requests *only if* no new packets have arrived at the input port. Packets which can no longer be resubmitted are discarded, thus limiting the required size of the input buffer and eliminating the need for flow control. In the special case where $t_p = mn$, for some integer m, each request can be resubmitted up to m times before being discarded. Each time it is submitted, it contends only with other requests which have been equally unsuccessful. In this case, the *only-if* policy emulates a tandem network[13]. Combining this with the submission of requests from different input ports into different pipeline cycles results in the emulation of replicated tandem networks. Another approach to conditional resubmission is to delay resubmissions that contend with new requests until the next available pipeline cycle. While making full use of the pipelining capabilities, it may require flow control to keep input queues from building up and may alter the performance characteristics of the controller.

3.2 Pipeline performance.

While the design choices discussed in the previous section can be used to cause a pipelined controller to emulate well-known architectures, there are some operational differences. These differences can cause the network states developed by the controller to provide different sets of connections than would be provided by implementing the tandem and recirculating architectures directly. The differences include the following:

- The controller emulates the tandem banyan architecture by resubmitting requests into the pipeline from the input port that received the request, rather than from an address determined by deflection routing during contention resolution. This results in a different distribution of requests prior to each cycle of contention resolution and may affect performance.

- To simplify the CRU hardware, the controller uses the same contention resolution priority for all switches in a switch group, rather than a priority determined at each switch individually.

- When the controller emulates a recirculating network, the packets being recirculated are not given priority over new packets during contention resolution. This function would require additional hardware to maintain a priority for each request.

Simulation experiments were performed to investigate the value of pipelining and the effect of each of these differences. Controller operation was simulated for pipelines with different depths for a network switch with 64 input ports. Packets arrive in time slots that are synchronized at all input ports. The CRU cycle time was 30 ns. Packet destinations were randomly distributed across all output ports.

While the *only-if* resubmission policy provides the most accurate emulation of a replicated tandem network, the actual number of times packet retransmission may be attempted is governed by the timing relationship between pipeline cycles and packet arrivals. To avoid the complexity introduced by packet interarrival times that are not a multiple of pipeline cycle time, the *always* resubmit policy was used.

In the first experiment, the time between packet arrivals, t_p, was chosen to be six representing a 2.4 Gb/s line speed, as noted in the previous section. The value of pipelining is clearly shown in Figure 14, which shows the percentage of successfully transmitted requests increases with the depth of the pipeline. Simulations were also made to evaluate the effect of each of the three differences described above. Figure 15 shows the difference between the throughput of each alternative and the throughput of the current design. Throughput can be increased with each of the alternatives. Relatively little throughput can be gained, however, by increasing the granularity of the priority for contention resolution or by providing the resubmitted requests with higher priority. The gain from deflection routing is larger, and is more significant with higher degrees of pipelining where there are a larger number of states per time slot. This suggests an increasing interdependence between the requests that are ultimately resubmitted for processing into these additional states. The randomizing effect of deflection routing appears to reduce this interdependence. A fully pipelined controller with depth six produces twice as many network states as a controller with depth three. At a depth of six, the controller has enough capacity to create network states that the efficiencies of the alternatives are not required in order to obtain good performance.

To show how the pipeline depth can be chosen to match controller performance with the requirements of a given line speed, packet arrival rates were simulated for speeds up to 2.4 Gb/s. The results are shown in Figure 16. With slower line speeds, pipelining is not necessary for satisfactory performance. A fully pipelined controller is capable of supporting line speeds up to 2.4 Gb/s.

4 Conclusions

We have described an architecture for the central controller of a network switch based on a banyan architecture. The controller uses a hierarchical approach to resolving contention for the state of the switches in the network. The logic required to resolve contention can be built into fast hardware, and can be easily pipelined. This means the controller can create network states that provide contention-free connections through the switch in constant time, independent of the number of switch ports. The basic structure and operation of the controller can be extended in a natural way to support multicast traffic.

The controller can be used in networks with either electronic or optical switch fabrics built with simple switches that do not have buffers or control logic. The choice of switch fabric can be based on bandwidth requirements, rather than the blocking characteristics of the banyan network.

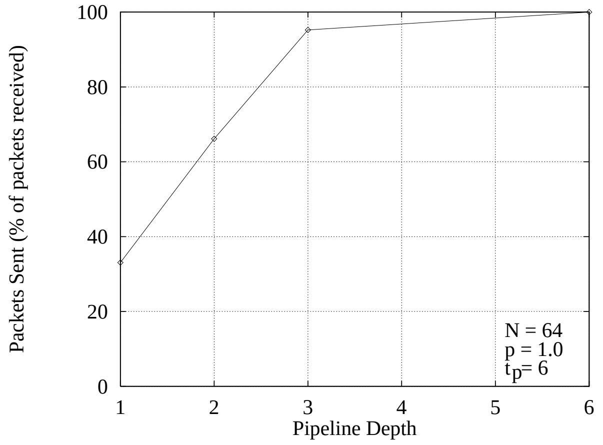


Figure 14: Throughput.

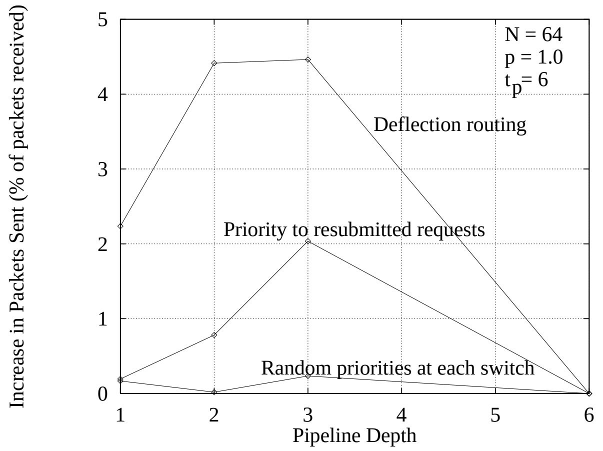


Figure 15: Effect of alternatives on throughput.

When control of the switch states is not required, the design may be used as a scheduler to signal input ports when packets may be transmitted. Based on the operational requirements of the switch, the designer may choose the pipeline structure, switch size, and recirculation policy to provide performance characteristics typical of parallel, tandem, or recirculating banyan networks.

Performance simulations show that the simplifications made to the design result in small performance penalties that could be overcome with the addition of more hardware. Pipelining can greatly increase the throughput of the controller, and a fully pipelined controller can support line speeds up to 2.4 Gb/s.

References

[1] C. Catania and A. Pattavina. Analysis of replicated banyan networks with input and output queuing for ATM switching. In *Proceedings of the IEEE International Conference on Communications (ICC '96)*, volume 3, pages 1685–1689. IEEE, 1996.

[2] T. Chaney, J. A. Fingerhut, M. Flucke, and J. S. Turner. Design of a gigabit ATM switch. In *INFOCOM '97: 16th Joint Conference of the Computer and Communication Societies*. IEEE,

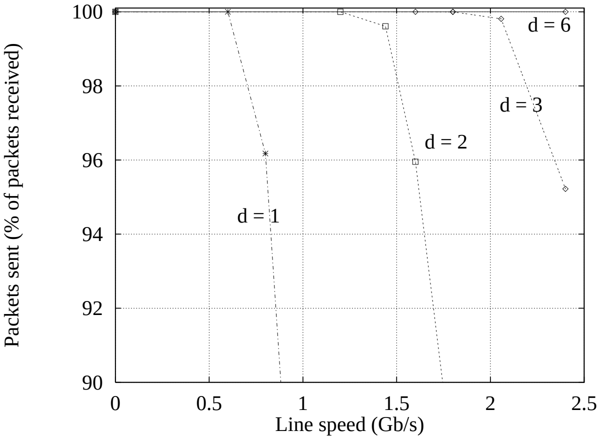


Figure 16: Pipelining to increase throughput.

April 1997.

[3] P. Dietrich and R. R. Rao. Request resubmission in a blocking, circuit-switched interconnection network. *IEEE Transactions on Computers*, 45(11):1282–1293, November 1996.

[4] C.-A. Fun and J. A. Silvester. A new parallel banyan ATM switch architecture. In *Proceedings of the IEEE International Conference on Communications (ICC '95)*, volume 1, pages 523–527. IEEE, 1995.

[5] J. F. Hayes, R. Breault, and M. Mehmet-Ali. Performance analysis of a multicast switch. *IEEE Transactions on Communication*, 39(4):581–587, April 1991.

[6] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C-32(12):1091–1098, December 1983.

[7] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz. The Tiny Tera: A packet switch core. *IEEE Micro*, pages 26–33, Jan/Feb 1997.

[8] N. McKeown and B. Prabhakar. Scheduling multicast cells in an input-queued switch. In *INFOCOM '96: 15th Joint Conference of the Computer and Communication Societies*, volume 3, pages 271–278. IEEE, March 1996.

[9] C. Qiao and R. Melhem. Reconfiguration with time division multiplexing MINs for multiprocessor communications. *IEEE Transactions on Parallel and Distributed Systems*, 5(4):337–352, 1994.

[10] C. Salisbury, R. Melhem, and C. Qiao. Distributed path management in switched optical banyan netwo rks. In *Optics in Computing*, volume 8 of *OSA Technical Digest Series*, pages 195–197, Washington, DC, March 1997. Optical Society of America.

[11] N. K. Sharma. Space and time redundant broadband packet switch. In *Proceedings of the 15th Annual IEEE International Phoenix Conference on Computers and Communications*, pages 194–200. IEEE, 1996.

[12] F. A. Tobagi. Fast packet switch architectures for broadband integrated services digital networks. *Proceedings of the IEEE*, 78(1):133–167, January 1990.

[13] F. A. Tobagi, T. Kwok, and F. M. Chiussi. Architecture, performance, and implementation of the tandem banyan fast packet switch. *IEEE Journal on Selected Areas in Communications*, 9(8):1173–1193, October 1991.

[14] H. Uematsu and R. Watanabe. Architecture of a packet switch based on banyan switching network with feedback loops. *IEEE Journal on Selected Areas in Communications*, 6(9):1521–1527, December 1988.

[15] K. L. Yeung. A new ATM switch design using wrapped around multiple WAM banyan network. In *Proceedings of the IEEE International Conference on Communications (ICC '95)*, volume 1, pages 523–527. IEEE, 1995.