A High Speed Scheduler/Controller for Unbuffered Banyan Networks *

C. Salisbury and R. Melhem
Department of Computer Science
University of Pittsburgh

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 a replicated banyan network fabric that provides sufficient switching bandwidth.

1 Introduction

A wide variety of banyan-based network architectures have been developed to overcome the performance limitations imposed by blocking[9]. 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*. Packets which exit the network at an incorrect destination are undeliverable, and may be discarded or recirculated[11]. A key performance measure of these designs is the packet loss rate.

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, 5]. *Tandem* architectures place banyan networks in a series[10]. *Dilated* architectures increase the number of links between switches in the banyan[5]. These architectures have been enhanced by interconnecting replicated or tandem networks using dilated links[12] or bypass connections[4]. Recirculation has been incorporated into many of these designs[8, 10].

Similarly, parallel networks have been used to increase the bandwidth of the switch fabric. Four switching planes are used in the switch described by [2], and the switch described in [6] uses a set of of parallel one-bit wide cross-bars.

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

Optical switch fabrics may provide another means of obtaining high bandwidth, where parallelism is provided through wavelength division multiplexing.

The focus of this paper is to describe a controller that can be used with banyan network switching fabrics, where the fabric is designed to meet the bandwidth requirement using simple 2×2 switches that do not have buffers or control logic. For example, the switch fabric can be a replicated banyan architecture or a single high speed, highly reconfigurable banyan network.

A centralized scheduler for a cross-bar based switch fabric is described in [6], and a centralized algorithm for control of a MIN is described in [7]. 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 a packet through the switch. In this way the controller acts as a transmission scheduler. During request processing, the controller also determines the state required at each switch. This *network state* information can be used to control the network and establish the selected connections. Since packets are not transmitted until the desired connection is available, requests that are not granted are resubmitted[3], rather than recirculated.

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. The blocking characteristics of the banyan network affect the performance of the controller, rather than of the switch fabric. 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 describe the hardware design and considerations for pipelining the controller in section 3 before concluding in section 4.

2 Centralized control of a banyan.

Banyan networks are a class of multistage interconnection networks (MIN) that provide 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 **??**. Packets from any input port may only pass through a subset of the switches in the network. The switches that can be used for connections from input ports 0 and 1 are highlighted in the figure.

The switch architecture is pictured in Figure **??**. Incoming packets are queued 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 re-

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.

quest word is presented to the controller and processed with requests from other input ports to identify a contention-free subset.

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. An input port that can be connected to one switch in a switch group can be connected to all switches in that switch group. Switch groups are enclosed in dashed boxes in Figure ??. The set of input ports that can be connected to a switch group is referred to as an *input group*. The input group for the upper switch group at stage 1 consists of input ports 0, 1, 2, and 3.

In this paper we will consider only unicast messages. 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 $N-1$ switches. 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 needed for a connection require a "don't care" indication. 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 ??):

- 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 **??**. 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 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 to obtain the remaining switches and complete the connection.

Since a banyan network provides a unique path between an input port and an output port, two requests can contend for a switch at only one stage of the network. Thus the 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.

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. Unsuccessful requests must be resubmitted to the controller.

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 the externally set priority. The unsuccessful request is blocked and dropped from further processing.

Input port	Output port	Request Word	Level 0	
			Switch state	Merged word
0	5	`x -= -x--`	`x`	`x= -xx-`
1	6	`x x- --x-`		
2	5	`x -x -x--`	`x`	`xx =x--`
3	0	`x x- =---`		
4	5	`x -= -=--`	`x`	`-= -=--`
5	3	`= -x ---x`		
6	2	`= =- --x-`	`=`	`=- --x-`
7	0	`x x- x---`		

Table 2: Resolving contention (part 1).

In.	Level 1		Level 2	Final state seen by input port	Send ?
	Switch states	Merged word	Switch states		
0	`x=`	`=xx-`	`=xx-`	`x x= =xx-`	Yes
1				`x x= =xx-`	Yes
2				`x x= =xx-`	No
3				`x x= =xx-`	Yes
4	`==`	`-=x-`		`x == =xx-`	No
5				`x == =xx-`	No
6				`= == =xx-`	Yes
7				`= == =xx-`	No

Table 3: Resolving contention (part 2).

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. The ordering of the requested states within the request word allows us to do this. We can map the stage of a switch and its offset within its switch group to the stage and offsets of the switches connected to its upper and lower outputs. The mapping for the banyan network of Figure **??** is pictured in Figure **??**, where $S_{i,j}$ is the switch at stage i, offset j. As described in Section 3, this relationship is built into the circuitry at each CRU.

2.3 An example of resolving contention.

Let the desired port connections be those shown in the first two columns of Table 2. Using the reverse cube network of Figure **??**, the processing of request words at each level of the controller is shown in Tables 2 and 3. The corresponding switch settings are shown pictorially in Figures **??**, **??**, and **??**.

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 the port with lower address 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 **??**. The connections still being processed are shown as a dark solid line, while blocked connections are shown as dashed lines.

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. 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-`).
- The requests conflict for the lower switch at stage 1. If contention is resolved in favor of the input port with the lowest address, the switch will be set in the straight 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 merge requests. 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 each switch 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.

3 CRU implementation.

Figure **??** depicts the organization of a CRU at level 0 in the switch controller for eight input ports. A CRU consists of two components: conflict detection and merge control. The conflict detection component signals, for each switch at the current stage, when there is contention between the two request words. The result from each switch controls the merge logic for all switches which can be part of the connection at later stages of the network. Since the level 0 CRU resolves contention for only a single switch, a single output signal controls the merge logic for all remaining switches in the two requests. For CRU's at higher levels, the signals control merging of requests for switches physically accessible at later stages of the network, as determined by the mapping function of Figure **??**. This mapping is reflected in the way the contention signals are connected to the merge control logic. Figure **??** shows how the connections are made in a CRU at level 1.

A priority signal line allows an external source to determine which request word receives the highest priority. This signal is connected to the merge control logic for all switches. Merge control determines the state of each switch in the current switch group. The "care" bit for these switches can be discarded. For the remaining switches, the merged request word is passed to the next level CRU.

The logic required to perform conflict detection is shown in Figure **??**. Contention is indicated when both request words care about the state of a switch, and the specified states are different. Contention can be detected in the time required by three logic gates. As shown in Figure **??**, the priority and contention signals control the merging of requests. When there is no contention and one of the requests has the "care" bit set, that state will be output to the next level. If both requests specify "don't care", so will the merged request word. When there is contention, only states from the high priority request word are passed to the next level.

The initial merge control gate operations can be overlapped with the contention detection logic. Assuming 2 logic gates per selector, a CRU can detect and resolve contention and merge requests with a total of 5 gate delays. We call this time a *CRU cycle*. To determine the state of every switch in the network requires n levels of CRU. The time to resolve contention among a set of requests thus depends on both hardware technology and network size. Depending on the speed of the input lines, the controller may be able to identify non-conflicting subsets of connections fast enough to provide connections for all arriving packets.

3.1 Pipelining the controller

To increase the speed with which contention-free network states are generated, additional hardware may be used to pipeline the controller. For example, latches would need to be added between CRU's and along switch control lines.

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 $\frac{t_p}{n}$. With pipelining, one state can be produced every pipeline cycle up to a maximum of t_p states. Since each state is produced in constant time, the number of states between packets no longer depends on switch size.

In general, the number of levels of CRU included in a pipeline stage is a design decision. We call this number the pipeline parameter, u. The depth of the pipeline, d, is thus $\lceil \frac{n}{u} \rceil$. Pipeline cycle time is defined to be the time per pipeline stage, which equals u. The controller can create $\frac{t_p}{u}$ states per packet time.

Another design decision is the policy for handling unsuccessful requests. Since requests rather than packets flow through the controller, 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 so they can be sent in the correct order, regardless of the order in which requests are granted. If desired, however, additional hardware at the input and output ports could be used to provide deflection routing[10]. Three policies can be considered 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 can be directed to submit requests during different pipeline cycles. In this manner, the controller will develop non-conflicting sets of connections that are similar to the sets developed by a replicated banyan architecture[5]. The controller can thus be used with $\lfloor \frac{t_p}{u} \rfloor$ replicated banyans or a single highly reconfigurable (possibly optical) banyan with the required bandwidth.

A second policy is to *always* resubmit unsuccessful requests. Packets received by an input port are never discarded or lost. 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[11].

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. 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 with fewer requests. In this case, the *only-if* policy emulates a tandem network[10]. Combining this with the submission of requests from different input ports into different pipeline cycles results in the emulation of replicated tandem networks.

To fully exploit the pipeline, each input port could be allowed to submit multiple requests. By allowing input ports to look ahead at up to d packets in the input buffer and to submit a request in each pipeline cycle, pipeline utilization may be increased and the effect of head of line blocking may be reduced. The values of t_p, n, u, and the resubmission and look-ahead policies determine when the controller emulates a familiar banyan architecture. The emulation can be improved with the addition of hardware to provide deflection routing. The optimal controller design will depend on the operational requirements of the switch.

Controller operation was simulated to evaluate performance with different values of the pipeline parameter u for a switch with $N = 64$. The CRU cycle time was assumed to be 30 ns and packet arrival rates were simulated for line speeds up to 2.4 Gb/s. Packets arrive in time slots that are synchronized at all input ports. Packet destinations were randomly distributed across all output ports, and the "always resubmit" policy was used. The result shown in Figure **??** confirms that increasing the amount of pipelining increases the throughput of the switch, and that a completely pipelined controller can support very high line speeds.

4 Conclusions

We have described an architecture for a central controller for 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 choice of switch fabric can be based on bandwidth requirements, rather than the blocking characteristics of the banyan network. Simple switches without buffers or control logic can be used. While not described in this paper, 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. 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 replicated, tandem, or recirculating banyan networks.

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, 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] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.