# Distributed, Dynamic Control of Circuit-Switched Banyan Networks *

C. Salisbury and R. Melhem
Department of Computer Science
University of Pittsburgh
{salisbur,melhem}@cs.pitt.edu

## Abstract

*Circuit-switched Banyan interconnection networks can be built from simple switching elements that do not have logical processing or buffering capabilities. This paper describes a distributed technique for dynamic control of such a network, so that circuits can be established in response to the changing needs of a parallel application. Control information is interleaved with data, thus avoiding the need for a separate network to process control messages. These techniques are particularly useful in optical networks, where it may be desirable to provide all-optical circuit-switched connections.*

## 1. Introduction

A common technique used in banyan interconnection networks is packet switching, in which data is buffered at each switch, processed, and routed to the correct destination. Circuit-switched banyan networks can also be constructed, in which a direct connection is provided between the source of a message and its destination. Since neither packet processing nor buffering is needed, circuit switching can be implemented with switches that are simpler and faster than those required for packet switching. The circuits provided at any instant of time are determined by the states of the switches, and are collectively referred to as the *network state*.

One means of providing multiple network connections for each processor is to use time division multiplexing (TDM). With TDM, the network hardware automatically cycles through a sequence of network states. A control operation may, for example, load a register with a bit string representing a sequence of states for a switch. Rotating the bits in the register changes the control signal sent to the switch, providing a very rapid means of changing the network state. TDM can improve performance when the time to cycle between network states is much less than the time to perform the control operation[6]. Each network state is provided for a *time slot* sufficient to transmit a message. The number of network states in the sequence is the *multiplexing degree*. A global clock is required to synchronize all switches.

Techniques have been developed for dynamic control of circuit-switched networks. One approach is to use a central network controller to accept communication requests, determine the required network state, and notify the requester when the state is available[2]. Distributed approaches spread control functions across several devices, increasing fault tolerance and performance while also potentially increasing complexity. The distributed approach described in [5] uses a separate control network and control logic placed in each switch. A distributed approach for controlling an optical passive star network is described in [1].

This paper describes distributed, dynamic control of circuit-switched banyan networks comprised of simple switches, without the need for a control network. Multiplexing is used to share the network bandwidth for both control and data communication. Using a distributed algorithm, each processor attached to the network independently develops a network state that is consistent with the states developed by all other processors.

The rest of this paper is organized as follows. The principles of the distributed algorithm are reviewed in section 2. In section 3, we describe an implementation of the algorithm for a banyan MIN built from $2 \times 2$ switches. Section 4 describes how network control protocols can be built upon the contention resolution algorithm. Our conclusions are in section 5.

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

1063-7133/98 $10.00 © 1998 IEEE

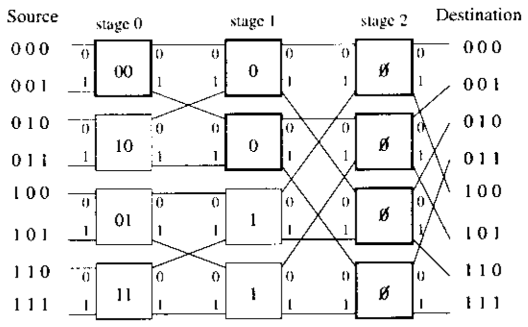

**Figure 1. A reverse cube network.**

## 2. Controlling banyan networks

A banyan network is a class of multistage interconnection network (MIN) that provides a unique path between any pair of nodes. A banyan network interconnecting $N = k^n$ processors is built with $n$ stages of $N/k$ switches, where each switch is a cross-bar switch of size $k \times k$. The distributed algorithm for creating network states is based on a commonly used banyan network structure called a Bi-Delta network[4].

### 2.1. Self-routing in banyan networks.

Banyan networks are distinguished by the interconnection pattern between the network stages. For example, the network shown in Figure 1 is called the "reverse cube" network for $N = 8$ and $k = 2$. The source and destination nodes are labeled with $n$ radix-$k$ digits.

The messages from each source node can only pass through a subset of the switches in the network. The switches that can be used for messages from inputs 0 and 1 are highlighted in Figure 1.

The path from a source node to a destination node can be described by the sequence of digits that label the successive outputs of the switches at each stage of the network. This sequence of $n$ digits is called a *path descriptor*. For example, using the switch port labels in Figure 1 we see that the descriptor for the path between input node 0 and destination node 5 is 101. We can also define a *reverse path descriptor* describing the route backwards through the network from a destination to an input node.

Delta networks are a class of banyan network with the property that the path descriptor for a destination node is a permutation of the $n$ radix-$k$ digits of the node address[4, 7]. This property is known as *self-routing.* This means that the output port used at a switch does not depend on the address of the node originating the message.

The self-routing property allows each switch to be labeled with a single descriptor for the path used to reach it. This descriptor is independent of the input node from which the path originates. At a given stage, several switches can have the same descriptor. For switches at stage 0, no routing is needed and the descriptor is empty.

A Bi-Delta network is a Delta network in both directions. That is, the path from a destination node back through the network to an input node can also be obtained by self-routing based on the address of the input node. The switches in Figure 1 are labeled with reverse path descriptors. Many of the commonly studied banyan networks are Bi-Delta networks, including the Omega, cube, baseline, flip, and butterfly networks.

### 2.2. Distributed network control

We define a *switch group* to be a set of switches with the same reverse path descriptor. Switches at different stages of the network have different length descriptors and thus are in different switch groups. Let $G_i$ be the set of all switch groups at stage $i$. Each switch group $S \in G_i$ consists of switches with the same reverse path descriptor. Clearly, these switch groups are disjoint. For every switch group $S \in G_i$, there is a corresponding set of input nodes $I_S$ such that every node in $I_S$ has a path to every switch in $S$. Define this group of nodes to be the *input group* for that switch group.

As shown in Figure 1, all switches in stage 2 have an empty path descriptor and form a single switch group. The corresponding input group consists of all eight input nodes. At network stage 1, the two switches with the descriptor (0) form a group for the group of input nodes 0, 1, 2, and 3. Since each switch at stage 0 has a unique descriptor, each switch is a group for the two attached input nodes.

The following theorem describes an efficient distributed algorithm for resolving contention for switches and developing a network state. The proof of the Theorem is based on the self-routing properties of Bi-Delta networks.

**Theorem 1:** In a Bi-Delta network, the following distributed procedure can be used to develop a network state. For each network stage $i$ from 0 to $n-1$,

1. Exchange switch state requirements for switch group $S$ at stage $i$ with all processors whose address differs only in the position corresponding to the reverse path routing descriptor for stage $i$.

2. Resolve contention for the switches in $S$ and determine the combined requirements from input group

$I_S$ for switches at subsequent stages of the network.

□

The required state of all switches can therefore be determined in a *control cycle* of $n$ steps, proceeding from stage 0 to stage $n-1$.

## 3. Developing the network state.

In this section, we consider an implementation of the above algorithm for a network with $k=2$. We describe how processors specify their circuit requirements for unicast connections and resolve contention to create network states that satisfy these requirements, subject to the blocking characteristics of the network. The discussion and example are based on the reverse cube network shown in Figure 1 with eight processors.

### 3.1. Overview

Each processor requests a circuit by placing its switch state requirements into a *control message.* At step $i$ of the algorithm, processors exchange control messages. Contention is resolved for the switches in the switch group at stage $i$ and the resulting states are retained for later use in setting the network state. Since each message contains the requirements from an entire input group, it may contain requests for several circuits. Each circuit requires exactly one switch in each switch group. For circuits that can be provided through stage $i$, the switch states required at later stages of the network are merged into a control message for step $i+1$. Requests that are not satisfied are dropped and must be resubmitted later by the originating processor.

When $k=2$, information is exchanged between pairs of processors. To set the switches at stage $i$ in the eight node reverse cube network, processor $o=(o_2o_1o_0)$ exchanges its request message with the processor that has a different value of bit $o_i$.

For example, to resolve contention for switches at stage 1, nodes 1 and 3 exchange request messages since their addresses differ in bit position 1. The message from processor 1 contains the combined requirements of its input group at stage 0, which consists of processors 0 and 1. Similarly, processors 0 and 2 exchange messages. After the exchanges, all four processors will have complete information about the states requested for the switches in the group at stage 1. Thus, all processors in the input group can resolve contention and merge control messages in a consistent manner.

### 3.2. Request messages.

Each processor creates a control message that specifies, for all switches to which it can be connected, the states required to form a circuit to the desired destination. While the network contains $\frac{N}{2}\log N$ switches, each processor can be connected to only $N-1$ of them.

For unicast communication, a $2\times 2$ switch can be set in either the "straight" or "cross" state. Since all switches accessible to a processor appear in the control message, switches which are not required for a connection are indicated with a "don't care" marking. In the following examples we will use the notation "x" to indicate a cross state, "=" to indicate a straight state, and "-" to denote "don't care".

The desired state of each switch can easily be computed from the source and destination addresses using the self-routing properties of the network. The request for each switch is placed in the control message in a location (*i.e.* a stage and offset) that allows the physical relationship between switches to be determined. From the location of the state in the control message, a simple mapping function can be used to locate the requested states of the switches connected to the upper and lower output ports.

For example, to connect processor 0 to processor 5 requires the following (see Figure 1):

- The top switch at stage 0 must be in the cross state (**x**).
- The second switch at stage 1 must be in the straight state (=).
- The second switch at stage 2 must be in the cross state (**x**).
- The state of the other four switches to which processor 0 could be connected do not affect this connection. They are marked as "don't care" (-).

These switch states are placed in a control message and ordered by switch position within stage, so that the control message from processor 0 is (`x -= -x--`).

### 3.3. Resolving contention

At step $i$ of the algorithm, contention is resolved for each of the $2^i$ switches in a switch group. Circuits from the successful request(s) are traced through the control message and switch requirements are inserted into a merged control message that will be used at step $i+1$. Switches for which there is no specific request are marked in the merged message with "don't care". A circuit is traced through the control message using the mapping function. For each switch, the following contention is possible.

| Source | Dest. | Request Message | Step 0 | |
|---|---|---|---|---|
| | | | Switch setting | Merged request |
| 0 | 5 | `x -= -x--` | `x` | `x= -xx-` |
| 1 | 6 | `x x- --x-` | `x` | `x= -xx-` |
| 2 | 5 | `x -x -x--` | `x` | `xx =x--` |
| 3 | 0 | `x x- =---` | `x` | `xx =x--` |
| 4 | 5 | `x -= -=--` | `x` | `-= -=--` |
| 5 | 3 | `= -x ---x` | `x` | `-= -=--` |
| 6 | 2 | `= =- --x-` | `=` | `=- --x-` |
| 7 | 0 | `x x- x---` | `=` | `=- --x-` |

**Table 1. Resolving contention (part 1).**

| S. | Step 1 | | Step 2 | Final State | Need Met |
|---|---|---|---|---|---|
| | Switch setting | Merged request | Switch setting | | |
| 0 | `x=` | `=xx-` | `=xx-` | `x x= =xx-` | Yes |
| 1 | `x=` | `=xx-` | `=xx-` | `x x= =xx-` | Yes |
| 2 | `x=` | `=xx-` | `=xx-` | `x x= =xx-` | No |
| 3 | `x=` | `=xx-` | `=xx-` | `x x= =xx-` | Yes |
| 4 | `==` | `-=x-` | `=xx-` | `x == =xx-` | No |
| 5 | `==` | `-=x-` | `=xx-` | `x == =xx-` | No |
| 6 | `==` | `-=x-` | `=xx-` | `= == =xx-` | Yes |
| 7 | `==` | `-=x-` | `=xx-` | `= == =xx-` | No |

**Table 2. Resolving contention (part 2).**

- Both requests specify the same setting for a switch in the current stage. There is no contention, and both requests can be satisfied. Note that these two requests can not have conflicting requirements at any subsequent stage of the network.
- One request contains "don't care" for the switch. Again, there is no contention. If the other request specifies a switch setting, it is successful.
- Two requests indicate "don't care" for the state of the switch. The switch can be set arbitrarily.
- The requests are for different switch settings, indicating contention. One request is chosen to be successful, based on the contention resolution algorithm. The unsuccessful request has become blocked and is dropped from further processing. The circuit from the successful request is traced and the switch requirements are inserted into the merged control message.

## 3.4. An example.

Let the connections desired by the processors and the associated control messages be those shown in the first three columns of Table 1. Contention resolution

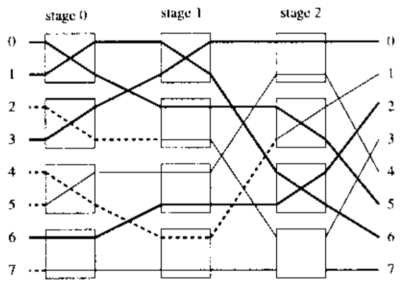

**Figure 2. After step 2.**

at step 0 produces the results shown in the last two columns of the table. For example:

- Both processors 0 and 1 request the (`x`) setting for the stage 0 switch. There is no contention. The cross state is selected for the switch at stage 0 and the merged message for the remaining switches is (`x= -xx-`).
- Processors 6 and 7 have conflicting requests for the stage 0 switch. For this example, we will use a fixed priority scheme with the request from the processor with lowest address having the higher priority. The request from processor 7 is blocked, and the stage 0 switch will be used in the straight state. The merged message for the remaining switches is (`=- --x-`).

Column 5 of Table 1 shows the control message developed for stage 1 after resolving contention for switches at stage 0. At step 1, consider the requests exchanged between processors 0/1 and 2/3. These requests must be processed for each switch in the switch group.

- Both requests for the uppermost switch at stage 1 specify the cross state. There is no contention, and circuits for both requests are inserted into the merged control request. This stage 1 switch will be set in the cross state, and both paths through it will be used. The merged message for stage 2 switches based on this stage 1 switch is (`=-x-`).
- The requests conflict for the lower switch at stage 1. Contention is resolved and the switch will be set in the straight state. The circuit requested by processors 0 and 1 is traced and (`-x--`) is inserted into the merged message. The complete control message for the next step is therefore (`=xx-`).

In the final step, the state of all switches in the final stage is determined. There is no need to merge requests. The control register of each switch can then be loaded with the required state by a processor associated with the switch. For example, processors with even (or odd) numbered addresses may be assigned to load the registers for the $n$ switches in the corresponding row of the network. By comparing the network state after stage 2 to its initial request, each processor can determine if the circuit it requested will be established. The final network state information contained in each processor is shown in column 5 of Table 2. For this example, four circuits will be established from processor 0 to 5, 1 to 6, 3 to 0, and 6 to 2, as shown in Figure 2. The successful requests are shown as a dark solid line, while requests that have become blocked are shown as a dashed line extending to the switch where the blocking occurred.

Since control message processing does not require a large amount of computation or memory, it could be performed by network interface hardware.

## 4. Implementation alternatives.

The number of network states required for control communication depends on the topology of the banyan network. We can multiplex these states for network control together with states for data communication to implement dynamic, distributed control of the circuit switched network. Various network control protocols can be developed by describing sets of rules which govern how control cycles can alter the states used for data communication.

### 4.1. Allocating network bandwidth.

To allow the control communication pattern to be implemented without blocking the network must be partitionable into contention-free and channel-balanced disjoint $k$-ary cubes, as described in [3]. The processors that must communicate in each step of the distributed procedure form a $k$-ary 1-cube. From [3], a cube banyan network (and by symmetry, the reverse cube network of Figure 1) has the required property. Thus, the communication required for each step of the distributed algorithm can be accomplished with a single network state.

Control communication therefore requires $n$ network states corresponding to the $n$ steps of the algorithm. In addition, data transmission requires one or more network states. The required states can be provided in a sequence using TDM. In an optical network they can also be provided using wavelength division multiplexing (WDM), or with a combination of WDM and TDM. In the remainder of this paper we will illustrate the use of the algorithm with TDM. In general, the length of a time slot for data communication may differ from the length of a time slot for control communication.

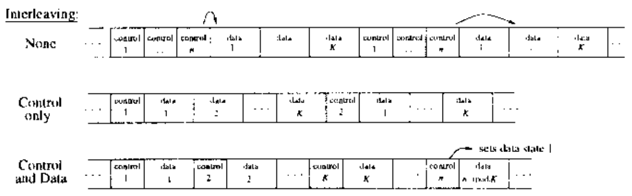

Figure 3. Interleaving data and control.

A sequence of $n$ network states for control can be interleaved with a sequence of $K$ states for data in many different ways. One approach is to determine, after each time slot, whether the state in the following time slot is to be taken from the same sequence or not. We call the use of a control state following each data state *data interleaving*. Similarly, when a data state always follows a control state, we call it *control interleaving*. The time slot following the final state in each sequence always contains a state of the opposite type. This approach can be used to produce the sequences shown in Figure 3. For example, control interleaving alone produces a sequence in which every control state and every data state (except for the $K^{th}$ one) is followed by a data state. The $K^{th}$ data state is always followed by a control state.

In these methods, the control cycle is repeated for each data state to establish circuits for requests that are new or were blocked. Once established, circuits for data communication are provided in the same time slot until a control cycle of $n$ steps has been completed for each of the $K$ data states. After this time the network state may change, as determined by the rules of the protocol. Many characteristics of the network can be computed from the manner in which the two sequences are interleaved and the lengths of the data and control time slots. This includes, for example, the minimum latency to build a network state, the minimum number of times a state can be used before it may be rebuilt, and the percent of network bandwidth used for control.

Determining the optimal multiplexing method is a complex task beyond the scope of this paper. Performance is affected by many factors, including the rate and burstiness of communication requests, switch contention within the communication pattern, the use of blocking communication or other synchronization, the use of variable length messages, higher level protocol requirements (*e.g.* reverse paths required for acknowledgments), the frequency with which processors require new circuits, and the relative size of data and control time slots. Further, the time between the receipt of a control message and transmission of the merged message must allow for contention resolution processing. Depending on the interleaving method and data slot size, the control slot size may need to be extended to allow sufficient processing time.

These same factors also influence the choice of the optimal multiplexing degree. This value may change as the communication activity of the program changes. The value can be dynamically chosen using a distributed algorithm that adds information to the request messages. It may be easy to add an additional state to the sequence of data states being multiplexed. The difficulty of removing a state from the sequence depends on the network control protocol.

### 4.2. Network control protocols.

Network control protocols provide rules governing how data states are created. One approach is to begin each control cycle with an empty set of circuits. Once established, each circuit is available for a fixed amount of time that can be computed from the network parameters. We call this Reservation with Fixed Expiration (RFE). Connection requests may be processed in any control cycle. Circuits are released automatically when the state that provided them has been rebuilt. Even when the communication requirements of the program do not change, RFE requires control operations to rebuild states as they expire. RFE is suited to programs that use fixed length messages which can be transmitted in a single time slot and to programs with frequently changing communication patterns.

Another approach is to update each network state, and process only requests that do not alter existing straight or cross switch settings. This allows processors to request circuits only once and to keep them indefinitely. As a consequence, there may be restrictions on the states into which a request can be processed. By selecting this state carefully, it may be possible to optimize the number of circuits used in each network state. To avoid the need to continually increase the multiplexing degree to accommodate new circuits, an explicit release message is required. Thus, this protocol is called Reservation with Explicit Release (RER). A "use count" must be kept for each switch to accurately reflect available ("don't care") switches. The contention resolution process can be extended with a *roll-back* algorithm to maintain this use count. RER depends on processors to be well-behaved and release unneeded circuits. RER may be most appropriate for programs that send variable length messages, and for programs that require infrequent changes to the communication pattern.

The control communication pattern allows additional information to be added to a control message and used to make other decisions in a distributed fashion. For example, a flag could be added to support an algorithm to change the protocol used based on the communication needs of the program. The choice of RFE or RER could then be made dynamically to adjust the performance of the network control protocol to suit the program's needs.

## 5. Conclusions

We have presented a distributed algorithm for resolving contention and developing network states in Bi-Delta banyan networks built from simple switches. The algorithm can be used as the basis of dynamic network control protocols which provide unicast communications.

Multiplexing can be used not only to interleave control and data in the same network, but to increase the number of paths that can coexist in a network and thus reduce the frequency of control operations. Various techniques can be used to multiplex the control protocol with data communication in a single network. In future work, we will investigate how the choice of control protocol and multiplexing technique can be made to match the capabilities of the network to a program's communication requirements.

## References

[1] D. M. Chiarulli, S. P. Levitan, R. G. Melhem, J. P. Teza, and G. Gravenstreter. Multiprocessor interconnection networks using partitioned op tical passive star (POPS) topologies and distributed control. In *Proceedings of the First International Workshop on Massively Parallel Processing Using Optical Interconnections*, pages 70–80. IEEE, April 1994.

[2] 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.

[3] L. Ni, Y. Gui, and S. Moore. Performance evaluation of switch-based wormhole networks. *Journal of Parallel and Distributed Computing*, 8(5):462–474, May 1997.

[4] J. H. Patel. Performance of processor-memory interconnections for multipr ocessors. *IEEE Transactions on Computers*, C-30:771–780, October 1981.

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

[6] C. Salisbury and R. Melhem. Modeling communication costs in multiplexed optical switchin g networks. In *11th International Parallel Processing Symposium, (IPPS)*. IEEE, April 1997.

[7] S. Sibal and J. Zhang. On a class of banyan networks and tandem banyan switching fa brics. *IEEE Transactions on Communications*, 43(7):2231–2239, July 1995.