Distributed Path Management in Switched Optical Banyan Networks

C. Salisbury and R. Melhem
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260

C. Qiao
ECE Department
State University of New York at Buffalo
Buffalo, NY 14260

Abstract

This work describes a distributed technique to dynamically determine the state of an optical Banyan switching network that will provide the paths requested by a parallel application.

Distributed Path Management in Switched Optical Banyan Networks [1]

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

C. Qiao
ECE Department
State University of New York at Buffalo

1 Introduction

Optical networks for interconnecting massively parallel processor systems have the potential to provide high bandwidth, low latency communication paths. Circuit switching networks exploit this bandwidth by providing all-optical connections between a set of sources and destinations. This increases the complexity of managing the network, since the connections must be changed in a systematic way to provide the complete set of paths required by the application.

In a circuit switched network, time division multiplexing (TDM) shares resources by allocating time slices for a resource among the paths that use that resource. Communications are then coordinated with these time slices in a round-robin fashion. Thus, the paths required between a set of senders and receivers is established for a small time interval called a *time slot*. Each processor may communicate with another processor during the time slot when the network provides the communication path it needs. At the end of the time slot, another set of paths is established. When all different sets have been established, the cycle begins again with the first set. When the length of the cycle is short enough, the optical bandwidth is effectively shared between communicating processors.

Time division multiplexing requires the use of a global clock to synchronize the time slots for all the transmitters and receivers connected to the network. The set of paths provided during any time slot will determine the settings of various switches, registers, etc. in the network. The state of these network components at any time is collectively referred to as the *network state*. Dynamic control can be used to determine the set of paths to be provided by a network state in response to requests for paths from the application[1]. This requires the network state to be created while the application is executing. A distributed approach to network control allows each processor to determine the network state independently, but in a manner consistent with all other processors.

This work describes a distributed technique that can be used to dynamically determine the state of a circuit switched network in order to provide a requested set of paths. The technique is described for a multistage interconnection network, such as a Banyan network. The generated state can be used alone or with other similarly generated states as part of a time division multiplexed cycle.

2 Network Architecture and State Determination

We consider a multi-stage interconnection network built from 2x2 electro-optical switches, such as Lithium Niobate switches[2]. The switches may be set to the straight position, in which each input port (i.e. upper and lower) is connected to the corresponding output port, and the cross position, in which the input ports are connected to the opposite output ports. The switches are interconnected in a butterfly network, as shown in Figure 1. This network is topologically equivalent any Banyan network. It interconnects N processors using $\log N$ stages of switches, with $N/2$ switches in each stage. The stages are numbered from 0 to $\log N - 1$. Each switch is controlled by a register which is loaded by a processor responsible for that switch. The network state describes the setting of all the switches in the network at any instant of time. However, each processor only needs to know the setting of the switches through which it can communicate. For example, processor 0 in Figure 1 can use the switches drawn in boldface.

We considered a network with the following characteristics. First, the components of the network do not have any path management capability. All decisions about the state of the components are made by the attached processors. Based on the desired network state, the processors set the controls for the network

[1] This work is supported in part by NSF awards ASC-9318185 and MIP-9633729

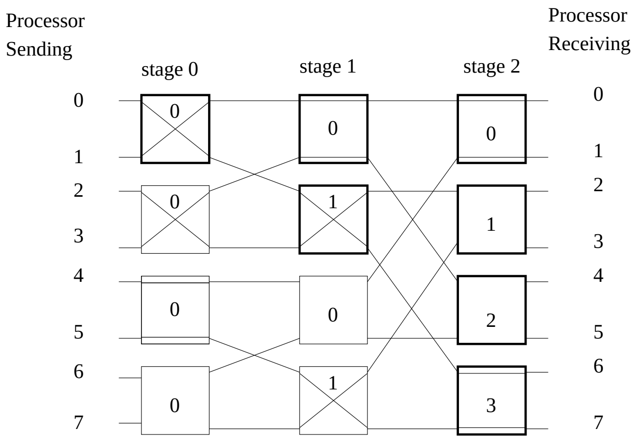

Figure 1: A butterfly network for eight processors. The switches accessible from processor 0 are highlighted.

components via registers, switch settings, etc. Second, the network is composed of two logical pieces, a data network and a control network. This may be implemented via two physical networks, or a single network may be shared. In a time division multiplexed network, it may be reasonable to reserve one or more time slots for control communications.

The network state creation algorithm develops a state by resolving conflicting requests for switch settings in a stepwise fashion. At step i, each processor exchanges information with the processor whose address differs by 2^i in order to resolve contention for the settings of the switches at stage i. Agreement on the setting of all switches is obtained after $\log N$ steps. The control network must therefore provide the ability for each processor to communicate with all processors whose address differs by 2^i, where i is a number from 0 to $\log N/2$. Thus, the processors are logically arranged as a $(\log N)$-dimensional array. At step i, information is exchanged along dimension i. Thus, the control network cycles through $\log N$ sets of paths.

In the first step of the algorithm, each processor creates a request message that specifies how each switch that it can reach should be set to handle its communications requirements. Two bits are required to contain switch information that is used during the control cycle and to represent the final network state. We will use a '0' to denote a straight switch setting, '1' to denote a cross setting, and 'x' to indicate "don't care". Given that each processor can have access to $N-1$ switches (2^i switches at stage i, for $i = 0, \ldots, \log N/2$), each processor requires $2(N-1)$ bits to represent the network state. For example, processor 0 can represent a request for switch 0 at stage 0 to be set in the cross state, switch 1 at stage 1 to be set in the cross state, and switch 3 at stage 2 to be set in the straight state using the string "*1 x1 xxx0*". This setting of switches connects processor 0 to processor 6 (see Figure 1).

To see how the algorithm works, consider an example where processors 0, 1, 2, and 5 request a path to processors 6, 0, 7, and 7 respectively. The complete set of interactions between processors is shown in Table 1. After each processor creates a request message, processors connected along dimension 0 exchange request messages and resolve conflicts for the switches at stage 0. Column 2 in Table 1 shows the requests sent by each processor. Column 3 shows the result of resolving the conflicts for switches at stage 0. We will describe the process as seen by processor 0. Since processors 0 and 1 need the same switch setting for the stage 0 switch, there is no conflict. Because of this, there can be no conflict for any other switches in their requests, so the requests can be merged in a straight-forward manner. The stage 0 switch that they share will be set in the cross position.

To resolve conflicts for the stage 1 switches, all processors will exchange their merged requests across dimension 1. In the example, we can see that there is a conflict in the requested setting of the second switch of stage 1. Processors 0 and 1 requested a cross state, and processors 2 and 3 requested a straight state. This conflict can be resolved in any arbitrary, but consistent manner. For example, we can resolve the conflict in favor of the processors with the lowest addresses. In this simple case, we can again merge the requests by replacing "don't care" switch settings with required settings. In general, the "don't care" switch positions that change must correspond to switches along the path that won the conflict resolution. In the simplest

Development of the Network State

Processor	Request Message	After Step 0	After Step 1	(Final State)
0	1 x1 xxx0	1 01 0xx0	1 01 0xx0	1 01 0xx0
1	1 0x 0xxx	1 01 0xx0	1 01 0xx0	1 01 0xx0
2	1 x0 xxx0	1 x0 xxx0	1 01 0xx0	1 01 0xx0
3	x xx xxxx	1 x0 xxx0	1 01 0xx0	1 01 0xx0
4	x xx xxxx	0 x1 xxx0	0 x1 xxx0	0 x1 0xx0
5	0 x1 xxx0	0 x1 xxx0	0 x1 xxx0	0 x1 0xx0
6	x xx xxxx	x xx xxxx	x x1 xxx0	x x1 0xx0
7	x xx xxxx	x xx xxxx	x x1 xxx0	x x1 0xx0

Table 1: Resolving contention for switches.

implementation of the algorithm, we do not release switches that have already been reserved for the newly blocked path, although it is possible to do this. Since the switch setting is harmless, we allow it to persist until it is reset during the negotiation for subsequent requests. The network state seen by each processor after this step is shown in column 4 of Table 1.

For the final round, processors exchange information across dimension 2 to resolve the contention for the stage 2 switches. There is no contention for switch settings and the requests are merged. Note that both paths through switch 3 of stage 2 will be used. This is a consequence of merging identical requests for the setting of a switch. The final network state is pictured in Figure 1, and the final network state information contained in each processor is shown in column 5 of Table 1. Since each processor has a view of only part of the network, in general the network state information will be identical only between processors that share stage 0 switches.

The control cycle is then repeated to respond to new communication requests and to allow new requests to be honored as paths in the current state are no longer needed. To provide more fairness in conflict resolution, we can use a predetermined sequence of resolution strategies in successive control cycles. For example, alternate control cycles may favor higher and lower addresses.

In a TDM network, the switch settings for each time slot may be negotiated in a round-robin manner. After the network states for all time slots have been determined, the process repeats, beginning with the first time slot. To allow processors to request a path through a switch that is already set in the correct position, we need to add the capability of releasing switches on a path that is no longer required, and releasing switches on paths that become blocked during conflict resolution. This capability can be provided by using an explicit switch-release message.

3 Conclusions

Dynamic algorithms for establishing circuit switching states are effective when the number of changes to the network state is sufficiently low. Due to communication locality, multiplexing not only offers a way to share optical bandwidth, but can increase the number of paths provided by a network and thus further reduce the number of changes that are required. We have described a distributed, dynamic algorithm to establish network states in an all-optical Banyan network. The algorithm can be implemented in a variety of ways as determined by the capabilities of the network, and can be extended to handle more complex communications requirements.

References

[1] D. M. Chiarulli, S. P. Levitan, R. G. Melhem, J. P. Teza, and G. Gravenstreter. Multiprocessor interconnection networks using partitioned optical 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] E. Murphy, T. Murphy, A. Ambrose, R. Irvin, B. Lee, P. Peng, G. Richards, and A. Yorinks. 16x16 strictly nonblocking guided-wave optical switching system. *IEEE Journal of Lightwave Technology*, 1996.