Path hierarchies in interconnection networks

by P. A. Franaszek

This paper treats the problem of latency minimization in an interconnection network for a system of *N* high-performance devices. The networks considered here have data transport separated from control, with the data subnetwork designed so that each network function requires only a single control message, and thus only one contention-resolution delay. For sufficiently large *N* it is shown that (for an abstract hardware model) minimizing contention delays requires that each message subject to such delays have more than one way of reaching its destination (e.g., via a path hierarchy). The overall approach is discussed in the context of the processor-memory interconnection problem in parallel computing.

1. Introduction

Consider a network whose purpose is to interconnect a large number N of high-performance devices D_i, $i = 1, 2, \cdots, N$. The design of such a system entails trade-offs on issues which include the overall bandwidth, the performance of the network controller(s), and the expected time for message propagation. A need for high bandwidth may rule out broadcast systems such as buses or rings. At the other extreme, the usual type of rectangular crosspoint array or crossbar switch has often been thought unsuitable for large N due to a combination of cost (which grows quadratically with N), and performance, which is limited by the speed of the control mechanism. Such considerations have led to the investigation of a large variety of multistage networks composed of modules or nodes each of which is individually controlled [1–6]. These networks potentially provide high throughput, but also often entail problems of complexity and delay. For high-performance applications such as the interconnection of processors and memories, the delays, which grow logarithmically with N, are a potentially limiting factor for overall system size. That is, for desired values of N their latency may be too great for purposes such as connecting processors with main storage (or alternatively may require implementation in technologies whose switching speed is greater than that of the processors). Recent experience [6] suggests that this may be the case for N as low as 500. It is thus of interest to consider possible alternative organizations.

The control of a typical multistage network is distributed, with each internal node performing such functions as routing and pacing. An alternative to this is a structure with greater separation between data transport and control. Control here is of two kinds: a) the coordination of actions within the network (e.g., establishment of a link between D_i and D_j), and b) coordination of action between the D_i (e.g., those associated with specific requests for data items). Consider a network where path setup is separate from message transport. Here a device D_i might request a link over which to transmit a message, then proceed with such transmission. An action initiated by a D_i which required a total of ν separate transmissions (e.g., a data fetch, where $\nu = 2$) might entail ν such path requests. Alternatively, the network could be designed so that a single request could be made for sufficient network resources for the entire action. The advantage here is that only one round of contention resolution or controller participation is required for each action.

The configuration of such a network W is a function of the types of services to be performed. If the only actions to be taken by W are, say, the transmission of fixed-size data packets from a requestor D_i to a destination D_j, then the controller(s) need only know the identity of the source and destination pair, and only a single path need be established in W for a given request. On the other hand, if a request from D_i can be either for the retrieval of a data item or for permission to transmit to D_j, then the controllers need additional information (such as the identity and size of the packet to be transmitted). Here W can be viewed as consisting of both a control transport facility W_1 and a data transport facility W_2, where W_1 and W_2 are generally of substantially different form. In the general case, however, W_1 would include a capability for transmitting the particulars of each request. That is, W_1 would be capable of transmitting a control message.

If N is large, there will generally be a need for multiple network controllers, since a single controller becomes a serialization bottleneck. For the case of multiple controllers, the single request property alluded to above (each action requires a single control message) imposes strong constraints on the transport subnetwork W_2: Each controller is assigned responsibility for one of a set of essentially disjoint subnetworks. Such subnetworks, as is shown below, can be obtained via the use of what will be termed partitioned crossbars. Crossbar switches, although sometimes considered impractical for large N because the number of crosspoints grows as N^2, are with modern technology actually economical for quite high values of N, since the crosspoint matrix can be implemented with semiconductor arrays in which the individual switching elements are comparable in complexity to memory cells [7]. The number of chips required for such a switch is largely a function of the chip pin count (the number of input and output wires), but may be on the order of fifty to a hundred for $N = 500$, not excessive for such applications as large-scale parallel computing.

Much of the following discussion centers on the control network W_1. Questions relating to the structure of such networks are explored via a simple traffic model and an abstract model of the hardware in which it is assumed that the network components have no state information (i.e., information concerning the number and location of messages in W_1). The results indicate that under these conditions a short path length requires a multiplicity or hierarchy of paths, so that the effective path taken by a control message (the path that effects the control) is a function of the instantaneous traffic. The effective-path variation may be obtained either by sending multiple copies or by such means as retransmission given lack of acknowledgment.

The following is an outline of the paper. Section 2 discusses control issues which suggest the use of double-sided crossbars for the transport network W_2. Section 3 treats the problem of contention resolution and control message transport from an abstract viewpoint, as well as the design of a control network based on the results. A configuration for the control subnetwork W_1 is suggested based on what may be termed contention crossbars and delta networks. Included is a discussion of the use of combining, based on notions related to the Fetch-and-Add architecture proposed by the NYU ultracomputer project [1], to alleviate control network congestion. Finally, Section 4 considers application of the overall structure to the interconnection of processors and memories in parallel computing.

2. Data transport

As noted above, the network W is partitioned into control and data transport subnetworks, denoted respectively by W_1 and W_2. The data transport subsystems W_2 considered here act under the control of requests s_{ij} transmitted on W_1 and processed by one or more controllers $\{E_j\}$. If the rate of service requests is high, this translates into a requirement for parallel control even for small values of N, since a single controller becomes a serializing bottleneck. This section considers the problem of how to structure W_2 so as to minimize the number of control delays. More precisely, it treats the question of how W_2 should be structured so that each action to be taken via the network (e.g., a data fetch) requires the transmission of only a single control message. It is assumed that there are N controllers E_j, and that each E_j is associated with a device D_j, so that a control message s_{ij} may be considered as traveling between D_i and D_j.

W_2 can be viewed as consisting of a set of resources $\{R_i\}$, corresponding to items such as device ports, data links, and internal network nodes. The overall connection system (W_1 and W_2) may be considered as a means for processing a sequence of action or service requests (each initiated by a D_i). For simplicity, it is assumed that these are of two kinds, corresponding to operations on a shared memory:

a. A request for a data item such as a cache line to be transferred from D_j to D_i.
b. A request for permission to transfer a specified amount of data from D_i to D_j.

It is assumed that the order in which requests from different D_i are processed does not affect correctness. Here the service provided in response to a request is equivalent to what in database parlance is termed a *transaction* [8]. This may be viewed as a process in which a device gains access to an increasing subset of the resources required to satisfy its request. When all such resources have been claimed, action can proceed, corresponding to the *commit phase* in transaction processing. In general, this can require (as a function of the overall design) a substantial amount of interaction among the controllers. Partially completed

The configuration of such a network W is a function of the types of services to be performed. If the only actions to be taken by W are, say, the transmission of fixed-size data packets from a requestor D_i to a destination D_j, then the controller(s) need only know the identity of the source and destination pair, and only a single path need be established in W for a given request. On the other hand, if a request from D_i can be either for the retrieval of a data item or for permission to transmit to D_j, then the controllers need additional information (such as the identity and size of the packet to be transmitted). Here W can be viewed as consisting of both a control transport facility W_1 and a data transport facility W_2, where W_1 and W_2 are generally of substantially different form. In the general case, however, W_1 would include a capability for transmitting the particulars of each request. That is, W_1 would be capable of transmitting a control message.

If N is large, there will generally be a need for multiple network controllers, since a single controller becomes a serialization bottleneck. For the case of multiple controllers, the single request property alluded to above (each action requires a single control message) imposes strong constraints on the transport subnetwork W_2: Each controller is assigned responsibility for one of a set of essentially disjoint subnetworks. Such subnetworks, as is shown below, can be obtained via the use of what will be termed partitioned crossbars. Crossbar switches, although sometimes considered impractical for large N because the number of crosspoints grows as N^2, are with modern technology actually economical for quite high values of N, since the crosspoint matrix can be implemented with semiconductor arrays in which the individual switching elements are comparable in complexity to memory cells [7]. The number of chips required for such a switch is largely a function of the chip pin count (the number of input and output wires), but may be on the order of fifty to a hundred for $N = 500$, not excessive for such applications as large-scale parallel computing.

Much of the following discussion centers on the control network W_1. Questions relating to the structure of such networks are explored via a simple traffic model and an abstract model of the hardware in which it is assumed that the network components have no state information (i.e., information concerning the number and location of messages in W_1). The results indicate that under these conditions a short path length requires a multiplicity or hierarchy of paths, so that the effective path taken by a control message (the path that effects the control) is a function of the instantaneous traffic. The effective-path variation may be obtained either by sending multiple copies or by such means as retransmission given lack of acknowledgment.

The following is an outline of the paper. Section 2 discusses control issues which suggest the use of double-sided crossbars for the transport network W_2. Section 3 treats the problem of contention resolution and control message transport from an abstract viewpoint, as well as the design of a control network based on the results. A configuration for the control subnetwork W_1 is suggested based on what may be termed contention crossbars and delta networks. Included is a discussion of the use of combining, based on notions related to the Fetch-and-Add architecture proposed by the NYU ultracomputer project [1], to alleviate control network congestion. Finally, Section 4 considers application of the overall structure to the interconnection of processors and memories in parallel computing.

2. Data transport

As noted above, the network W is partitioned into control and data transport subnetworks, denoted respectively by W_1 and W_2. The data transport subsystems W_2 considered here act under the control of requests s_{ij} transmitted on W_1 and processed by one or more controllers $\{E_j\}$. If the rate of service requests is high, this translates into a requirement for parallel control even for small values of N, since a single controller becomes a serializing bottleneck. This section considers the problem of how to structure W_2 so as to minimize the number of control delays. More precisely, it treats the question of how W_2 should be structured so that each action to be taken via the network (e.g., a data fetch) requires the transmission of only a single control message. It is assumed that there are N controllers E_j, and that each E_j is associated with a device D_j, so that a control message s_{ij} may be considered as traveling between D_i and D_j.

W_2 can be viewed as consisting of a set of resources $\{R_i\}$, corresponding to items such as device ports, data links, and internal network nodes. The overall connection system (W_1 and W_2) may be considered as a means for processing a sequence of action or service requests (each initiated by a D_i). For simplicity, it is assumed that these are of two kinds, corresponding to operations on a shared memory:

a. A request for a data item such as a cache line to be transferred from D_j to D_i.
b. A request for permission to transfer a specified amount of data from D_i to D_j.

It is assumed that the order in which requests from different D_i are processed does not affect correctness. Here the service provided in response to a request is equivalent to what in database parlance is termed a *transaction* [8]. This may be viewed as a process in which a device gains access to an increasing subset of the resources required to satisfy its request. When all such resources have been claimed, action can proceed, corresponding to the *commit phase* in transaction processing. In general, this can require (as a function of the overall design) a substantial amount of interaction among the controllers. Partially completed

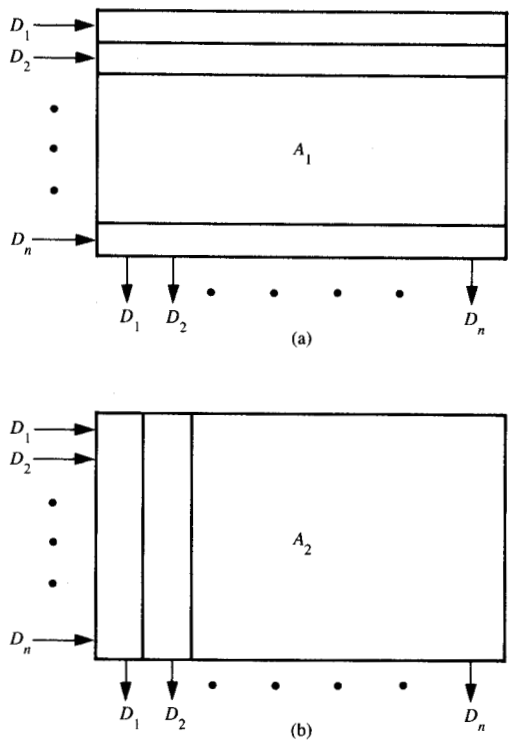

Figure 1

(a) Horizontally and (b) vertically partitioned double-sided crossbars.

transactions may have to be stopped or restarted due to phenomena such as deadlock. Reference [8] discusses how such interference can limit throughput in a parallel processing system.

Let $\{A_i\}$ be a set of actions to be performed by the network. An action A_i may interfere with another action A_j either because it requires access to one or more of the resources R_i claimed by A_j or because its control interferes with that of A_j. The former kind of interference can occur, for example, in a network that is not *nonblocking*, that is, in a network where a path from D_i to D_j may be blocked by a transmission between two other devices D_r and D_k. The latter type of interference, that between the controllers, may be illustrated by the following example. Suppose that the network W_2 is *single-sided* and that any device D_i may initiate a connection with any other. Let L_{i1}, L_{i2} be input and output ports from the W_2 network associated with D_i. Then a single-sided connection between D_i and D_j requires that D_i have simultaneous access to both L_{i2} and L_{j2}. One way for D_i to proceed in establishing such a connection is to reserve L_{i2} for its own use and to request L_{j2}. But L_{j2} may itself be reserved, say by D_j in the process of attempting a connection to D_i. This is an example of deadlock resulting from the interaction of two controllers. More complex cases may occur, which in general require extensive communication between controllers to resolve.

Suppose the design of W_2 is such that each controller E_j is responsible for allocating a disjoint subset of the network resources $\{R_{ij}\}$. Suppose further that each action involving W_2 (e.g., a data fetch or send) requires access to some subset of one $\{R_{ij}\}$. Then each action requires transmission of only a single control message s_{ij}. A structure with this property will be termed a *single-resource network* (SRN). It should be noted that the requirements to obtain an SRN are a function of the variety of services requested by control messages. Thus, if a request for a data item were partitioned into subrequests for a) permission to transmit a request for data, b) the data, and c) permission to transmit on the return path, the requirements on the network would be less stringent than for the case where such subrequests are combined, since three separate requests are then permitted for the service. Note, however, that the total delay may be considerably greater, since each subrequest is subject to a separate contention-resolution delay.

Consider a link in W_2 which lies on a path between D_i and D_j. Suppose W_2 is an SRN. Then, if D_i has access to a network output port at D_j, D_i also has access to a link to D_j, and there is no contention on this link. Suppose that (as is assumed below) E_j is the controller which allocates ports at D_j. Then E_j was also responsible for allocating the path between D_i and D_j. It follows that no link on this path can be allocated by any other controller. That is, all shared links (i.e., links that carry messages from more than one origin) lead to the same destination. The network W_2 then consists of a set of essentially disjoint (for shared links) subnetworks, one for each destination. This is a stronger condition than merely having the network be nonblocking.

An example of a network which satisfies the above condition is one consisting of a set of trees, with one tree for each destination, which is also the root. A way to implement a network of this form is as a double-sided crossbar, that is, a crossbar with N input and N output ports, where each connection is between a single input and output port. Each output port may then be regarded as the root of a subnetwork tree whose leaves are the input ports. Note that such a crossbar (illustrated in **Figure 1**) might consist of a rectangular array of crossbar chips, each of which forms a connection between some subset of network input ports and a subset of buses leading to network output ports. Here each message traverses two shared paths: a bus within a chip and a bus external to the chips, corresponding to a tree of depth two. But each crossbar chip is itself a double-sided crossbar, which could be implemented via buses and subchips, so that a crossbar may represent a tree of depth greater than two.

Figure 1 shows two examples of doubled-sided crossbars, configured so that crosspoints in each row (horizontal

partition) or column (vertical partition) of the array are set independently. For example, in the horizontal partition, each transmitting device D_i has the capacity to establish a path to any destination D_j. For the application considered here, a horizontal partition is generally more convenient.

Note that if two or more data packets are transmitted simultaneously to the same destination, they will collide. If the means for control does not prevent such collisions, the switch is termed a *collision* or *contention* crossbar. Collisions may be avoided by such means as contention resolution at the inputs or via reservations, i.e., allocations of specific times at which a given device D_i can transmit to another device D_j. The configuration of such crossbars into a W_2 network is discussed in Section 4.

3. Contention resolution and control message transmission

This section considers the structure of the control subnetwork W_1. This subnetwork must provide two types of services: transport and contention resolution. Contention resolution needs to be included because there may, for example, be more than a single message headed for a given destination, and in general there can be other network resources subject to such contention. Two design alternatives are

a. Incorporate contention resolution in the W_1 transport mechanism.
b. Separate contention resolution from control message transmission.

Alternative (a) is a subnetwork W_i in which data transport (here transmission of the s_{ij}) is not separated from control. That is, there is no subnetwork W_0 used to control W_1. Alternative (b) requires a separate means for contention resolution. Here transmission of an s_{ij} might be done by (i) contention resolution, (ii) notifying each D_i when s_{ij} can be transmitted, and finally (iii) transmitting s_{ij}. If all s_{ij} are of one type (e.g., requests for permission to transmit a packet of fixed length), then transmission of control information represented by step (iii) is not necessary, and in general a system of type (b) may often be desirable. The discussion below first treats the problem of contention resolution in a general framework. Signals originating at a D_i whose destination is associated with D_j are for simplicity termed s_{ij} without distinguishing option (a) from (b). Discussions of possible implementation, however, focus on alternative (a), partly because of its compatibility with the consideration of Fetch-and-Add as a means for switch control.

Contention resolution may be viewed in a general sense as a process in which a number of messages attempt to traverse a network to a point where they can claim the desired resource. Each node in this network may perform such functions as routing and resolution of contention for transmission to the next node. The structure of such networks is here studied via the combination of a traffic model and an abstract model for the nodes. These are assumed to be devices G_i, each of which can "accept" a limited number of inputs and produce a limited number of outputs in unit time. The G_i, as well as the devices D_i served by the network, are assumed to have no knowledge of the network state (i.e., the number and location of messages in the system). The latter constraint can be expected to hold for large N, where device limitations make it difficult to distribute such information rapidly.

Proposition 1

Suppose i) that W_1 is composed of a set of devices G_i each of which can accept no more than V_1 inputs and produce no more than V_2 outputs in a given cycle, where it is assumed that an input is lost if not accepted. Suppose also ii) that the G_i as well as the D_i have no information concerning the network state, and finally iii) that all messages reach their destinations. Then there exists a set of paths in W_1, one for each (D_i, D_j) pair, such that the average number of G_i encountered on a path in this set is at least $\log_{V_1} N$ (for simplicity, it is assumed that $\log_{V_1} N$ is an integer).

Proof Each device D_i must transmit its request s_{ij} to at least one device G_k which is guaranteed to have no more than $(V_1 - 1)$ other inputs in this cycle. But D_i has by assumption no information concerning other messages which may be transmitted to G_k. Thus, each such G_k must have a potential input from at most a total of V_1 devices, so that, for each destination D_j, there exists a path from each D_i on which each intermediate node G_k can receive a message from at most V_1 devices. Suppose only the shortest such paths from each D_i to a given D_j are considered. These paths form a tree, rooted at D_j, with an average path length (in terms of devices) no smaller than $\log_{V_1} N$. To see that this is so, note that $\log_{V_1} N$ is the path length in a balanced tree which contains one path from each origin. This balanced tree produces a minimum average depth: If the tree were not balanced, exchange of a leaf at depth less than $\log_{V_1} N$ with a subtree at greater depth reduces the average path length. □

The above result concerns the average path length, where this average is taken over the paths. If the traffic is nonuniform, device pairs whose interactions occur with more than average frequency can be placed "closer" within the network, so that the path length averaged over the messages s_{ij} (as opposed to the paths) need not necessarily grow as $\log N$. This, however, requires substantial traffic nonuniformity.

Network performance can also be improved by reducing traffic congestion. Such congestion, and thus the expected delay in transmission for a message s_{ij}, can be decreased if messages with the same destination can be combined within the network, for example via the use of Fetch-and-Adds [1].

N_r	$N = 512$	$N = 1024$
$P(0)$	0.8187	0.8187
$P(1)$	0.1638	0.1637
$P(2)$	0.0164	0.0136
$P(>2)$	0.0015	0.004

(a)

N_r	$N = 512$
$P(0)$	0.9048
$P(1)$	0.0905
$P(2)$	0.0045
$P(3)$	0.0001
$P(4)$	<0.0001

(b)

Figure 2

(a) $P(N_r)$ for $N = 512$ and $N = 1024$ with $p = 0.2$; (b) $P(N_r|N_a)$ for $N_a = N/2$ and $N = 512$ with $p = 0.2$.

This possibility is discussed in Section 4 and in the Appendix.

An alternative approach to shortening the average path length, suggested by Proposition 1, is to provide potentially more than one path between each origin-destination pair. Here transmission on an individual path may be unsuccessful due to a violation of the V_1 input constraint, so that the effective path length is a function of the instantaneous traffic. Such multipath networks are studied below, first under the assumption that no state information is made available to the G_i and D_k, then given limited information (e.g., acknowledgment of successful transmission). Let $N_{req}(j)$ be the number of messages initiated to D_j in a given cycle. Suppose it is known that $N_{req}(j)$ is bounded by some value less than N. The above discussion suggests that one may then construct a network with shorter path lengths than would be required without this bound. One approach to the overall design is to combine several subnetworks, each appropriate for a particular value of $N_{req}(j)$. The resulting performance will then depend on the traffic statistics as well as the available hardware.

Request statistics

Two models for the request traffic are considered. In the first, each device D_i can present a request in a given cycle t_n with probability p, and the request is to any E_j with equal probability. The second model includes a restriction that each device have no more than one outstanding request. This could correspond to a system in which the $\{D_i\}$ are processors requesting lines or pages from a common distributed memory. Here, given a page or line fault, the requesting processor is halted until the underlying operation (i.e., a fetch or a send) is completed.

Suppose the $\{D_i\}$ are a set of processors which access the network for line fetches due to cache faults and for line sends. Suppose further that there is on average less than one line send for each fault, as is generally the case for a machine with a write-in cache. The network cycle is assumed to be no longer than half of a full processor cycle, so that p is upper-bounded by the cache fault probability. For simplicity, it is assumed that $p \leq 0.2$, which represents a rather large value in this context. Note that p may be decreased by decreasing the network cycle time, e.g., by increasing the path width of W_1.

The probability that there are $N_r(i)$ requests for the ith resource in the first model is given by the following binomial distribution:

$$P[N_r] = \frac{N!}{N_r!(N - N_r)!}\left(\frac{p}{N}\right)^{N_r}\left(1 - \frac{p}{N}\right)^{N-N_r}. \quad (1)$$

Note that $\{P[N_r]/P[N_r - 1]\}$ is an increasing function of p:

$$\frac{P[N_r]}{P[N_r - 1]} = F(N, N_r)\left(\frac{p}{N}\right)\left(1 - \frac{p}{N}\right)^{-1}. \quad (2)$$

Figure 2(a) shows $P[N_r]$ for $N = 512$ and 1024, with $p = 0.2$. Note that $P[N_r = 0]$ and $P[N_r = 1]$ vary slowly with N. Increasing N from 512 to 1024 changes these quantities only slightly. This, as will be seen, means that the expected performance of the networks discussed below will vary little as N is increased.

We now consider the steady-state probabilities for N_r given that each D_i has only one pending request s_{ij}. There are N potential requestors, of which N_a have requests pending. Suppose that on the average it takes u cycles to complete the servicing of a request. Then, in the steady state,

$$up(N - N_a) = N_a, \quad (3)$$

so that

$$N_a = \frac{N(u)p}{1 + (u)pN}. \quad (4)$$

If $p = 0.2$ and $u = 5$, then $N_a = N/2$. The request probability for N_r, given that N_a potential requestors are occupied, is given by

$$P[N_r | N_a] = \frac{(N - N_a)!}{N_r!(N - N_a - N_r)!}\left(\frac{P}{N}\right)^{N_r}\left(1 - \frac{P}{N}\right)N - N_a - N_r. \quad (5)$$

Figure 2(b) shows values of $P[N_r | N_a]$ for $N_a = N/2$, $p = 0.2$, and $N = 512$.

A hardware model

As mentioned above, the W_2 network is composed of a set of devices G_i whose capabilities are intended to model the speed and fanout restrictions of the hardware. Specifically, each G_i accepts no more than V_1 inputs and produces no

more than V_2 outputs with a delay of "a few" cycles. A set of such devices is described below. This device set is by no means exhaustive: Other devices may be formulated which plausibly have comparable performance. Thus the results should not be regarded as absolute proofs of optimal structures, but rather as indications of directions for system design.

The device types are as listed below and illustrated in **Figure 3.**

1. Network input node. One such node is associated with each D_i. It is assumed that it can produce $V_2 \geq \log_2 N$ simultaneous outputs (these are multiple copies of a control message s_{ij}) in a given cycle. This avoids the need to define a special node for the purpose of replicating messages.
2. Network output node. One such node is associated with each D_i. It accepts messages s_{ij} from W_2. One such message is accepted in each cycle.
3. Single-output bus (SO-bus). This has $2 \leq V_1 \leq N$ inputs and produces, with a delay of Z_1 cycles, a single output whose value is either 0 (if there is no message or a collision) or s_{ij} (if s_{ij} is the single input). That is, it accepts only the case of $V_1 = 1$. Unlike a standard collision bus, this has no collision sensing by the input devices, as this would violate the fanout restriction V_2.
4. Contention-resolution node. This accepts up to two inputs (if internal storage is available). It puts out one output with delay Z_2 and the next with delay $2Z_2$ (provided that storage is available at the destination). The storage restrictions imply the existence of input and output control lines which are not shown in Figure 3.
5. Routing node. This operates similarly to a contention-resolution node, but can emit two outputs simultaneously (one to each destination).

A horizontally partitioned $N \times N$ collision crossbar (defined in Section 2) provides an approximation to an idealized single-output bus for each of its N output lines. The Z_1 delay here corresponds to a) transmission of the address to the appropriate chip(s), which establishes the desired path, b) setting the crosspoint, and c) transmission of the signal. The delay associated with a) and c) is essentially proportional to the inverse of the signal path width, which can be increased as necessary.

Delays associated with contention-resolution nodes may also be decreased by increasing the path width. Here the lower limit to Z_2 is due to such delays as those required for checking control lines and performing contention resolution, in addition to housekeeping functions such as the local storage and retrieval of messages and the setting of control lines to preceding nodes. The result is that the delay Z_2 may not be vastly different from Z_1.

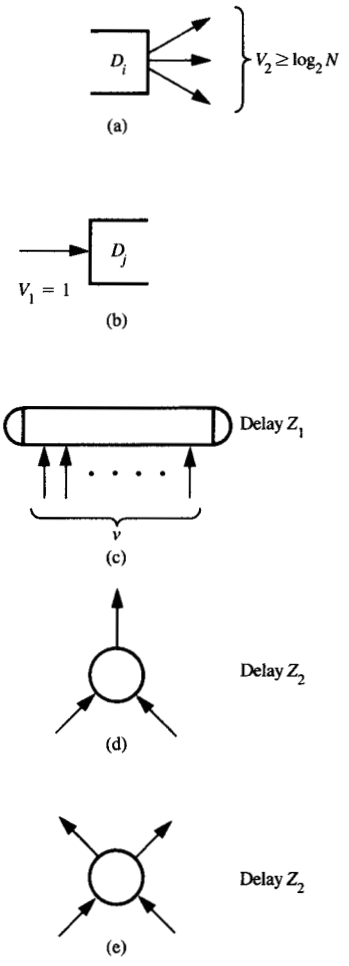

Figure 3

Network devices, where (a) is an input node to a network, (b) is an output node, (c) is an SO-bus, (d) is a contention-resolution node, and (e) is a routing node.

• *Network optimization*

Let $X_n(j)$ denote the set of requestors to a particular D_j in the cycle t_n. The following discussion restricts attention to a single D_j, and $X_n(j)$ is represented simply as X_n. Consider a message $s_{ij} \in X_n$. The lower bound on the time required for s_{ij} to reach D_j is that required for the first member of X_n to reach D_j, given that this message encounters no delays due to messages which are not in X_n, which is equivalent to the case where W_1 is otherwise empty. Let this time be denoted by $\psi(X_n)$, or $\psi(X)$, since it is assumed that the network is time-invariant. The quantity

$$\Sigma\psi(X)P(X) = \psi \qquad (6)$$

is then a lower bound for the expected delay. This section

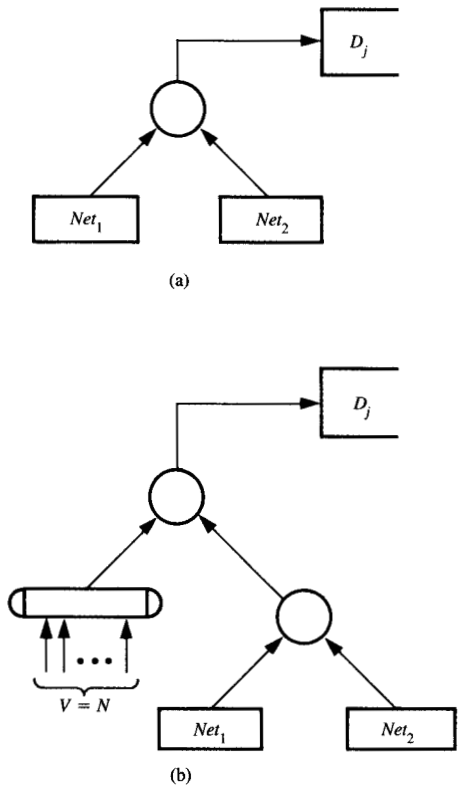

Figure 4

The original network (a), altered as shown at (b).

considers the problem of structuring a network so as to minimize ψ subject to the constraint that it must be capable of *resolving* (that is, accepting and successfully transmitting) any X.

A message s_{ij} is presented to W_1 at one or more *entry points* each of which is an SO-bus, a contention-resolution node, or a routing node. Note that W_1 must resolve the case of $|X| = N$, that is, the case where all D_i transmit to D_j in a given cycle. Thus, each s_{ij} must be presented to at least one entry point which is not an SO-bus, so that there are at least $N/2$ contention-resolution or routing nodes to provide entry points.

Similar arguments lead to the conclusion that each s_{ij} has at least one path from D_i to D_j which does not include an SO-bus. That is (corresponding to Proposition 1), there is a path tree leading to D_j composed of contention-resolution and routing nodes. This tree is of average depth no smaller than $\log_2 N$. Note that the node immediately preceding the D_j network output node is not an SO-bus. The subnetwork being considered here has just one such node, so without loss of generality this may be assumed to be a contention-

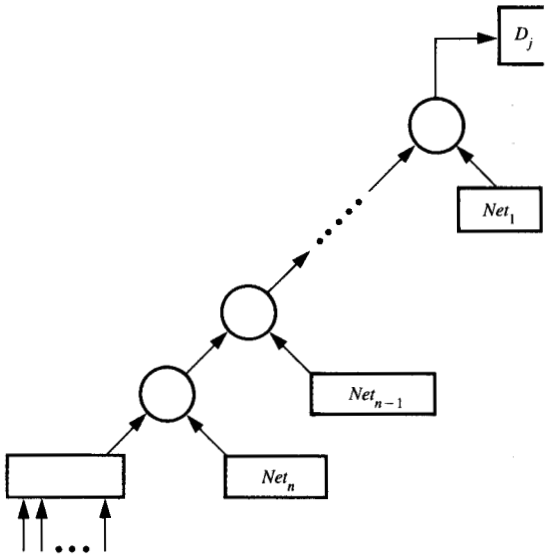

Figure 5

A configuration with an SO-bus.

resolution node. Let Net_1 and Net_2 be the two subnetworks whose outputs are inputs to this node, as illustrated in **Figure 4**.

Proposition 2
Suppose neither Net_1 nor Net_2 contains an SO-bus. Then, if

1. $p \leq 0.2$

and

2. $Z_1 \leq Z_2(\log_2 N - 2)$,

ψ may be reduced by modifying the network so that there is an additional contention-resolution node, immediately before the output node, whose inputs are (a) the output from an SO-bus with $V_1 = N$ (i.e., all s_{ij}) and (b) the original contention-resolution node, as shown in Figure 4. That is, under the above conditions, it pays to have an SO-bus in the system.

Proof A lower bound for the time required to resolve $|X| = 1$ in the original network is $Z_2 \log_2 N$ (corresponding to a balanced contention-resolution tree). Introducing the change reduces the delay for $|X| = 1$ by $(Z_2 \log_2 N - Z_2 - Z_1) \geq Z_2$. The delay for $|X| > 1$ is increased by Z_2. But with $p \leq 0.2$, $p(|X| = 1) > p(|X| > 1)$. □

Proposition 3
Suppose p, Z_1, and Z_2 are as in Proposition 2. Then minimizing the delay for $|X| = 1$ requires that there be an

SO-bus whose output is an input to the final contention-resolution node leading to D_j.

Proof Proposition 2 indicates that the minimum delay configuration for $|X| = 1$ includes an SO-bus. Suppose the optimal (for $|X| = 1$) configuration is as in **Figure 5**, containing an SO-bus with V direct inputs. Then the delay for the SO-bus is greater than for Net_n, Net_{n-1}, $\cdots$. Otherwise V could be increased to include the $|X| = 1$ inputs resolved by these networks, since for $|X| = 1$ the SO-bus performance is independent of V. But V (those $|X| = 1$ resolved by the bus) must be greater than the number resolved by Net_n, Net_{n-1}, $\cdots$, as otherwise the bus could be replaced by a subnetwork of that form, which is faster. But this implies that exchanging the location of the SO-bus with a network Net_i at lower depth decreases the expected delay for $|X| = 1$. □

Proposition 4
Suppose p, Z_1, Z_2 are as in Proposition 2, and that $Z_1 = KZ_2$, where K is an integer. Then the lower bound on the expected delay ψ is

$$\psi_{\min} \geq P(|X| = 1 \mid |X| > 0)\left[\left(\frac{N - 2^K}{N}\right)(Z_1 + Z_2) + \frac{2^K}{N}(2Z_2)\right] + P(|X| > 1 \mid |X| > 0)(Z_2).$$

Proof The delay for $|X| > 1$ is trivially no lower than Z_2. The lower bound for $|X| = 1$ is obtained via the configuration of Proposition 3. At most $(2^k/N)$ of the $|X| = 1$ requests do not use the SO-bus, and the delay for these is at least $2Z_2$. □

The above bound is very loose, especially for K small. However, as is shown below, it is sufficient to indicate that a particular configuration is reasonably close to optimal.

Proposition 5
Let p, Z_1, Z_2 be as in Proposition 2. Now consider the network shown in **Figure 6**, consisting of an SO-bus with $V \leq N$ inputs and a balanced tree of contention-resolution nodes, both of whose outputs lead to a final contention-resolution node. For this configuration, $V = N$ is optimal.

Proof The SO-bus resolves a member of $|X|$ if it receives a single input, with a probability which is denoted as $P_r(V)$. For $V \leq N$,

$$\frac{P_r(V+1)}{P_r(V)} = \frac{V+1}{V}\left(1 - \frac{P}{N}\right). \tag{7}$$

But $[1 + (1/V)]\,[1 - (P/N)] > 1$ for $p < 0.2$ and $1 \leq V \leq N$. Hence, the probability of $|V| = 1$ (i.e., the probability that the SO-bus has exactly one input) is maximized if $V = N$. □

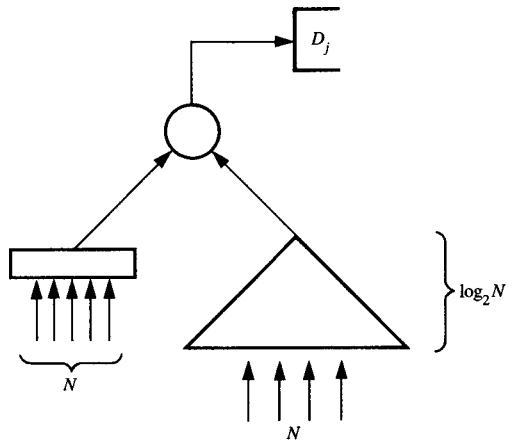

Figure 6

A multipath network.

Consider the network shown in Figure 6. It will now be shown that for the range of parameters given here, this network is reasonably close to optimal. The delay ψ for this configuration is

$$\psi = P(|X| = 1 \mid X > 0)(Z_1 + Z_2) + P(|X| > 1 \mid X > 0)(1 + \log_2 N)(Z_2). \tag{8}$$

Suppose $N = 512$ and $p = 0.2$; then, using the bound for $\psi_{\min}$ of Proposition 4, one can obtain the following:

a. If $Z_1 = Z_2$, $\psi/\psi_{\min} < 1.5$.
b. If $Z_1 = 3Z_2$, $\psi/\psi_{\min} < 1.25$.

Moreover, $\psi/\psi_{\min}$ decreases as p decreases [as a consequence of Equation (2)]. The bound for $\psi_{\min}$ is rather loose, which suggests that this configuration should provide good performance for $p \geq 0.2$. If $p > 0.2$, it might be desirable to introduce additional SO-buses so that, for example, $|X| = 2$ could be resolved in a substantial fraction of the time.

• *Retransmission*
It was assumed above that no state information (i.e., information relating to the location and number of messages in the network) is distributed. However, it may often be feasible to provide limited information by means such as additional circuitry, which, with some delay, informs the requestor of a collision on a particular path. Alternatively, successful transmissions could be acknowledged. Lack of such acknowledgment would then trigger retransmission, possibly on an alternate path. Potential advantages include

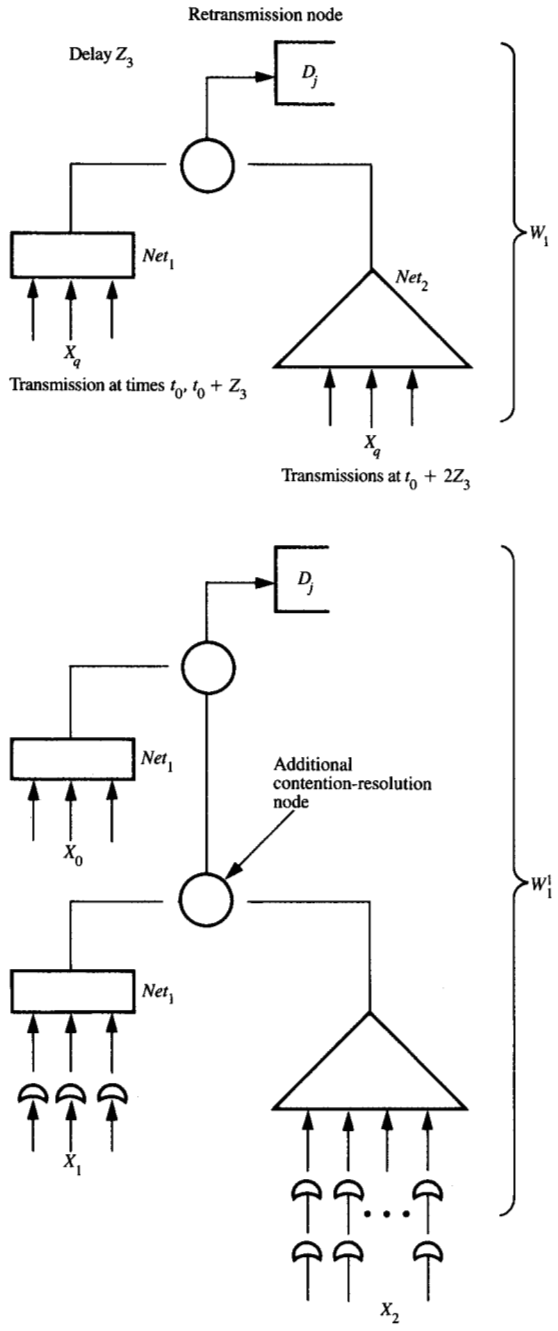

Figure 7

A network W_1 and its retransmission version W_1^1.

a. The elimination of duplicate messages. Such duplicates could be removed by various means at the destination, but this would lead to added complexity.
b. Reduction of traffic delays due to congestion, not included in the above analysis.

Suppose retransmission is included. How then should one design a network? It is shown below that the lower bound for the transmission delay ψ obtained in the previous section is close to that for networks with retransmission (if the time interval between successive transmissions is no less than for a contention-resolution node). This suggests a straightforward modification of the structure shown in Figure 6, the performance of which (in relation to $\psi_{\min}$) remains close to optimal.

Let ψ be, as above, the delay experienced on the average by the first $s_{ij} \in X$ to reach its destination, given that the set X of simultaneous requests comprise the only messages in W_1. It is convenient to introduce what is termed a *retransmission node.* This has a single input and output, and a delay Z_3 corresponding to the time between retransmissions, where it is assumed that Z_3 is greater than the cycle time of W_1.

A network W_1 is then composed of SO-buses and of nodes which perform contention resolution, routing, and retransmission. Consider a particular device D_i and a control message s_{ij}. This message will be transmitted at times $t_1, t_2, \cdots, t_q$, where t_q is the first transmission that is guaranteed to succeed (either because of acknowledgment or because its path consists entirely of, say, contention-resolution devices, which accept all inputs). The sequence $\{t_i\}$, of which $t_1, t_2, \cdots, t_q$ is a prefix, is independent of the network state, so that it can be considered fixed.

Each member of a set X has its retransmissions occur at times which are dependent only on whether transmission is successful. That is, X determines only the point at which each retransmission schedule stops. Let X_q^r be that subset of X which is retransmitted on the qth cycle, that is, at $(t_0 + q)$. Note that the cycle is assumed for convenience to be of unit length. Let Net_q be the subnetwork of W_1 on which X_q^r is transmitted. A new network W_1^1 can be obtained from W_1 by linking the outputs of Net_i by a contention-resolution tree, as shown in **Figure 7**. Each X_q^r then travels through at most $(q + 1)$ additional contention-resolution nodes when compared to the corresponding path in W_1. It also traverses V retransmission nodes. A path in W_1^1 is longer than that in W_1 by some number of additional contention-resolution nodes. But if $Z_2 < 1$ (the network cycle time), removing the retransmission nodes produces a network with no retransmission with a value of ψ within Z_2 of that for W_1. We then have the following.

Proposition 6

If Z_2 is no greater than the cycle time of the network, then for every retransmission network W_1, there is a network with no retransmissions W_1^1 whose delay $\psi^1 \leq \psi + Z_2$.

Consider the network of Figure 6. Suppose all requestors in a given cycle attempt transmission over the SO-bus, and (given lack of acknowledgment), retransmit over a contention-resolution tree. The increment to ψ due to

retransmission is $P(|X| > 1 \mid X > 0)Z_3$, which for $p \leq 0.2$ implies an additional delay of less than ten percent, so that this configuration should yield reasonably good performance.

Note that the property of retransmission networks given by Proposition 6 is for the special case where retransmission is based solely on information equivalent to the lack of acknowledgment. Here the retransmission schedule for each input is essentially independent of X, which only determines the schedule termination. This is quite different from the situation in, for example, a bus with collision detection, where knowledge of X can influence the times at which retransmission is attempted. This dependency of the transmission schedules on X precludes a straightforward equivalent network with no retransmission.

• *Design alternatives*

If N is large, say on the order of several hundred, then a separate contention-resolution tree leading to each D_j may be impractical. An alternative is to employ an arbitration structure of complexity less than N^2, for example a delta network (2, 3), whose complexity is $O(N \log N)$. For $N = 512$, this yields a reduction of the number of nodes by a factor of more than fifty. An example of a delta network based on shuffle-exchange interconnections (4, 3) is shown in **Figure 8**. An interesting property of such networks is that the connections between stages d and $d + 1$ are the same for all $d < D$, the maximum depth, which is potentially advantageous for a modular implementation of the hardware.

The reduction in complexity from $O(N^2)$ via substitution of a delta network for N contention-resolution trees is, however, obtained at the expense of blocking. Here the reduction in traffic due to retransmission is especially important, but may not fully alleviate the problem if traffic is not uniform. Heavy demand to one port can, for example, lead to blocking of the network subtree that has this node as the root. This result, due to Pfister and Norton [9], was motivated by their investigation of delta networks as the transport mechanism between processors and memories, but their conclusions also apply here. A further finding of theirs is that the Fetch-and-Add architecture proposed by the NYU Ultracomputer project [1] provides an effective though expensive solution to this problem via the combining of messages. It is shown below that this is also potentially the case here. The Appendix describes the use of message combining in the control of a crossbar network.

It was observed above that if the traffic load is low, the average path length can be shortened by including paths which successfully transmit a message only in the case of limited contention. Attention was restricted to the case where a short path is available for $|X| = 1$, but additional SO-buses could be included (with limited input) to resolve, say, $|X| = 2$ with high probability. Given limited state information, it is necessary to provide paths which are

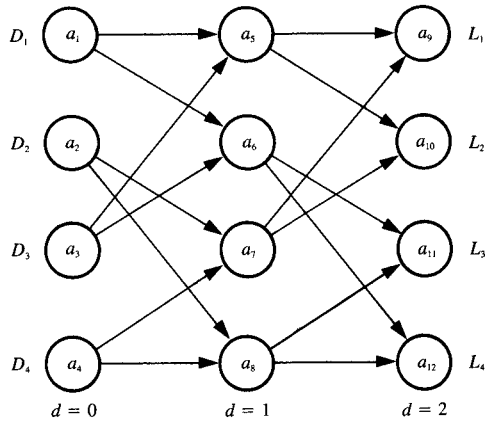

Figure 8

An $N \log N$ complexity delta network.

collision-free, since it is not feasible to incorporate standard contention-resolution methods, which generally include such features as exclusion of new traffic during contention-resolution cycles (and thus require notification of potential requestors). In the above treatment, such paths were provided by multistage networks (composed of routing and contention-resolution nodes), but other possibilities exist, including networks (such as crossbars with contention resolution), where such resolution is separate from message transmission. Alternatively, it may be feasible to design structures where state information is made available (with delay) to the set of requestors under high-traffic conditions.

4. Interconnection of processors and memories

The notions developed above are illustrated here by considering a processor-memory interconnection system. Let $\{D_i\}$ be a set of N processors, each with an associated memory module. It is assumed that the network supports actions of two kinds:

1. Requests by a device D_i for a specific line or page from a memory module associated with D_j.
2. Requests by a D_i for permission to transmit a specific number B of bytes to D_j.

It is assumed that each D_i has no more than one unacknowledged request pending.

The requirements imposed on W_2 by the single-resource constraint are now considered. Suppose that D_i wishes to obtain a line from D_j. This requires resources which include a transmission port at D_j and a reception port at D_i. These are also the resources necessary if D_j wishes to transmit some number of bytes to D_i. But the former operation is under the

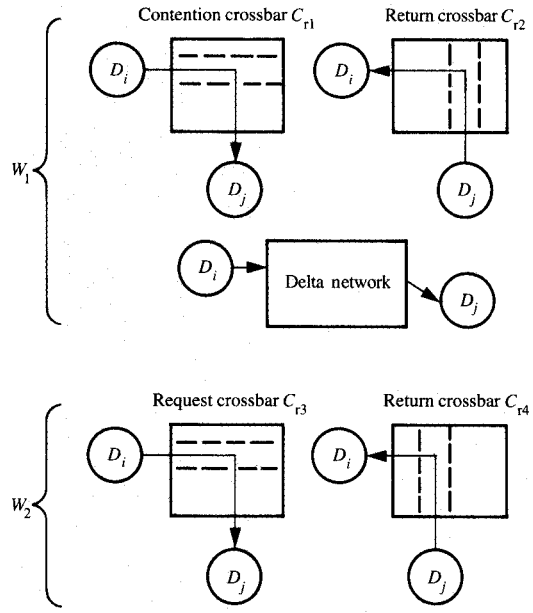

Figure 9

Crossbar network with hierarchical control.

control of D_j and the latter is controlled by D_i. It follows that there must be two sets of such ports, such as may be obtained by two double-sided crossbars, as shown in **Figure 9**.

The control network W_1 might be comprised of a contention crossbar (each column of which is an SO-bus), a one-way delta network for contention resolution, and a noncontention crossbar for acknowledgments, as shown in Figure 9. This may be viewed as providing a two-level path hierarchy. Operation for data fetches and stores could be as follows:

1. *Request for data by D_i from D_j.*
   Here a control message s_{ij} is transmitted on W_1, specifying the address of the desired line. The initial transmission is on the SO-bus whose output leads to D_j. To accomplish this, D_i selects and sets the appropriate crosspoint(s) in C_{r1}, then transmits the message. If transmission is successful, D_j immediately returns an acknowledgment on C_{r2}. There is no contention on C_{r2} due to the constraint that each D_i has at most one outstanding request s_{ij}. If D_i does not receive an acknowledgment, it retransmits on the delta network (perhaps after again attempting to use the path on C_{r1}). Once s_{ij} arrives at D_j, this device retrieves the appropriate data from memory, then returns it on C_{r4}. Note that delays associated with setting the appropriate crosspoint(s) in C_{r4} can be overlapped with memory latencies.
2. *Request for permission to transmit data from D_i and D_j.*
   A control message is transmitted by D_i on W_1 asking for a reservation (i.e., one or more time slots) for transmitting data on C_{r3}. The reservation is returned on C_{r4}, which then permits D_i to transmit on C_{r3}.

A variety of alternative designs for W_1 are possible, as discussed in Section 3. That shown in Figure 9 has the advantage that it can be modified so as to illustrate two approaches to shortening the average transmission delay: multiple (i.e., hierarchical) paths and combining. Combining of control messages via Fetch-and-Adds, as described in the Appendix, may be viewed as a means for alleviating blockage in the delta network resulting from heavy demand for resources controlled by a small subset of the controllers (note that this does not alleviate the load on W_2).

Fetch-and-Add control may be incorporated in the configuration shown in Figure 9 by replacing the one-way delta network with a two-directional network that permits combining and decombining. Requests for permission to transmit data would result in the return, as described in the Appendix, of reservation for using the desired path in W_2. Fetch requests, however, are not directly combinable. One possibility here would be to have them be of the form (data line requested; time). Two or more requests could then be combined into a data line request (for the highest-priority D_i) and a time at which the other requests could be transmitted to D_j on C_{r3}. A request could then travel on any of a variety of paths: the SO-bus given no collision and the delta network otherwise for the forward path, with those requests that resulted in combining resulting in use of the return paths of the delta network.

The incorporation of Fetch-and-Adds substantially increases the cost and complexity of the hardware, so that it is unlikely to be an attractive option unless it is in any event required to prevent serialization delays for access to systemwide shared variables, the original motivation for such combining. Here the reservation times for switch output ports can be viewed as simply another set of shared variables. The hierarchical or multipath approach to construction of the control network is then a means for speeding access to all such variables, not merely those associated with transmission.

5. Discussion and conclusion

This paper has considered a class of networks characterized by a) separation of data transport from control so as to permit control messages to flow over a lightly loaded W_1 subnetwork and b) a data subnetwork W_2 configured so that its resources could be allocated by a set of independently acting controllers. The question of minimal expected path

length in W_1 was studied for the case in which no state information is available to the network components. Two ways of reducing transmission delays given uniform traffic were considered: the use of combining (such as by Fetch-and-Add instructions), and the notion of multiple paths. It was shown, for a simple model of the traffic, that the use of multiple paths has the effect that the overall network speed is close to that of its fastest component. The notion of multiple paths and combining instructions was illustrated via an example of a processor-memory interconnection network.

Appendix: Fetch-and-Add switch control

Consider a double-sided crossbar with output ports L_k at each D_k, $k = 1, 2, \cdots, N$. Suppose D_i requests permission to transmit to D_k for a period of time of length T_i. Let Y_k be a *delay parameter* associated with L_k, where Y_k denotes the time after which L_k becomes free to accept new transmissions. Granting D_i its request increases Y_k to $(Y_k + T_i)$.

This corresponds to the definition of Fetch-and-Add. Specifically, a Fetch-and-Add to a variable α with parameter β, which is denoted as $FA(\alpha, \beta)$, has the effect that

1. The value of α is incremented by β.
2. The original value of α is returned.

Thus $FA(Y_k, T_i)$ would return Y_k, the time at which L_k is free, and increment Y_k by T_i, the time required for transmission. That is, the requestor would be granted a reservation for use of L_k. Unlike the usual application of FA, however, a reservation is useless unless it reaches the requestor before expiration. Thus it is necessary to incorporate some estimate of the return delay. Note that this estimate need only be of sufficient accuracy so as not to waste an appreciable percentage of the available reserved bandwidth.

The control of a crossbar via use of a combining network is illustrated via an example. Consider the network shown in Figure 8. Suppose that D_2 and D_4, respectively, issue instructions $FA(Y_1, 1)$ and $FA(Y_1, 2)$. These could be represented in the network, respectively, as tuples $(D_2, n_2, t_2, Y_1, 1)$ and $(D_4, n_4, t_4, Y_1, 1)$, where the components are, respectively, the identity of the originating node, a parameter to be used in the combining process, the time at which the request is made, Y_k, and α_k. The two requests would intersect at node a_7, where they might be combined into the tuple $[a_7, n_7, \min(t_2, t_4), Y_1, 3]$, and the original tuples stored for use in the decombining process. Suppose $Y_1 = 6$. The combined tuple could then be processed to produce a tuple $[a_7, n_7, (6 + b)]$, which would be returned to a_7. Y_1 would then be changed to $(9 + b)$. Here b is an increment which could be used to account for the expected delay on the return path, obtained, for example, by comparing $\min(t_2, t_4)$ with the time of arrival. Node a_7 could then use the parameter n_7 to obtain correspondence with the original requests, which would then be processed to obtain, say, $[D_2, n_2, (6 + b)]$ and $[D_4, n_4, (7 + b)]$, indicating that D_2 can transmit its message of duration 1 at time $(6 + b)$ and D_4 its message of duration 2 at time $(7 + b)$.

References

1. A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer . . . Designing a MIMD, Shared Memory Parallel Machine," *IEEE Trans. Computers* **C-32**, 175–189 (February 1983).
2. J. H. Patel, "Performance of Processor-Memory Interconnections for Multiprocessors," *IEEE Trans. Computers* **C-30,** No. 10, 771–780 (October 1981).
3. D. M. Dias and J. R. Jump, "Analysis and Simulation of Buffered Data Networks," *IEEE Trans. Computers* **C-30,** No. 4, 273–282 (April 1981).
4. H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers* **C-30,** No. 2, 153–161 (February 1981).
5. R. H. Kuhn and D. A. Padua, Eds., *Tutorial on Parallel Processing*, IEEE Computer Society presentation, 1981, pp. 168–177.
6. G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proceedings of 1985 International Conference on Parallel Processing*, pp. 764–769.
7. C. J. Georgiou, "Fault-Tolerant Crosspoint Switching Networks," *Research Report RC-10446*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1984; *Proc. Int. Conf. Fault-Tol. Computers* **FTS-14,** 245–249 (June 1984).
8. P. A. Franaszek and J. T. Robinson, "Limitations of Concurrency in Transaction Processing," *Research Report RC-10151,* IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1983; to appear in *ACM Trans. on Data Base Syst.*
9. G. P. Pfister anbd V. A. Norton, " 'Hot Spot' Contention and Combining in Multistage Interconnection Networks," *Proceedings of 1985 Conference on Parallel Processing*, pp. 790–795.

Received January 22, 1985; accepted for publication August 26, 1986

Peter A. Franaszek *IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Dr. Franaszek is manager of the Advanced System Structures and Analysis group in the Computer Sciences Department at the Thomas J. Watson Research Center. His interests include analytical problems associated with storage hierarchies, computer organization, magnetic recording, and digital communications. He received his B.Sc. degree from Brown University, Providence, Rhode Island, in 1962, and the M.A. and Ph.D. degrees from Princeton University in 1964 and 1965. During the academic year 1973–1974, he was on sabbatical leave at Stanford University as a Consulting Associate Professor of Electrical Engineering and Computer Science. Prior to joining IBM in 1968, he was a member of the technical staff at Bell Telephone Laboratories. Dr. Franaszek is a member of the Institute of Electrical and Electronics Engineers, Sigma Xi, and Tau Beta Pi.