A Conflict Sense Routing Protocol and Its Performance for Hypercubes

Emmanouel A. Varvarigos and Dimitri P. Bertsekas, *Fellow, IEEE*

Abstract—We propose a new switching format for multiprocessor networks, which we call Conflict Sense Routing Protocol. This switching format is a hybrid of packet and circuit switching, and combines advantages of both. We initially present the protocol in a way applicable to a general topology. We then present an implementation of this protocol for a hypercube computer and a particular routing algorithm. We also analyze the steady-state throughput of the hypercube implementation for random node-to-node communications.

Index Terms—Packet and circuit switching, hypercubes, random one-to-one communications, throughput.

1 Introduction

There are two general switching formats, *circuit switching* and *packet switching*, that are used in network communications. Circuit switching combines many well-known advantages, but is seriously inefficient. The inefficiency is related to the allocation of a link to a message for more time than required. Packet switching on the other hand is efficient in terms of link utilization since a link is used whenever there is a packet that wants to cross it, but has a number of drawbacks especially when the buffer space per node is limited and packets must occasionally be dropped.

A solution that has been proposed is *deflection routing* (see [5], [2], [6], [10]). With deflection routing, packets are misrouted instead of dropped. This works well for several networks (for example, hypercubes, Manhattan street networks), but not for all (for example, its throughput for the shuffle exchange network is low; see [8]). Networks not having enough path redundancy will most probably be unsuitable for deflection routing. *Cut-through routing* [7] and, its variation, *wormhole routing* [4] is another interesting alternative for multiprocessor communications, but some theoretical problems are still unresolved. The possibility of deadlock cannot be ruled out for both deflection [9] and wormhole routing [4], unless special precautions are taken. In practice, most data networks and many multiprocessor systems currently use packet or circuit switching. However, for many applications, it is unclear which one of the two is preferable, since each has relative advantages at exactly the same areas where the other has disadvantages.

In this paper we introduce a new switching format, which we call *Conflict Sense Routing Protocol* (or CSR protocol), and is a hybrid of circuit and packet switching. With this protocol, a packet can enter the network only after having reserved its route (links and buffer space). This resembles circuit switching. A packet, however, reserves a resource only for the slot (or slots) during which the resource will be used. In particular, a link on a packet's route can be used by other packets while the given packet is using other links on its route. This resembles packet switching since the links and the buffer space are used on a demand basis.

The CSR protocol is more efficient than circuit switching, because in circuit switching the entire path of a packet is reserved as the packet is traveling on any one link of the path, and additional overhead is needed to "tear down" a circuit after all transmissions of the circuit have been completed. A major advantage of the CSR protocol over packet switching is that it avoids the waste of resources due to dropping packets that have been transmitted for several hops. In multiprocessor systems with thousands of processors the buffer space per node is usually small, making dropping of packets a serious problem when packet switching is used. To deal with the possibility of dropped packets it is necessary to use some acknowledgment system. In parallel computers it is typically impossible to piggyback acknowledgments on the opposite direction traffic (in a network of thousands of processors a particular pair of processors rarely communicates), while the use of separate acknowledgment packets increases the network load significantly. To make things worse, acknowledgments may themselves be dropped increasing the delay and complicating the implementation. The CSR protocol that we will propose does not use acknowledgments as a feedback mechanism. Once a packet enters the network, it knows that it will arrive at its destination because it has already reserved the resources required along the way.

Another advantage of the CSR protocol is that it provides a "built-in" flow control mechanism. Flow control is necessary in packet switching to slow down transmissions when congestion arises in order to reduce dropping of packets. Flow control protocols in a multiprocessor computer cannot be the same with the ones of a general data network, where the nodes are bigger and the buffering is

- *E.A. Varvarigos is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106.*
- *E-mail: manos@ece.ucsb.edu.*
- *D.P. Bertsekas is with the Laboratory for Information and Decision Science, Massachusetts Institute of Technology, Cambridge, MA 02139.*

Manuscript received Sept. 11, 1992; revised May 30, 1993.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C96048.

cheap. Such protocols will not work well for massively parallel computers with little buffer space per node. For example, a window flow control scheme [1] with a window of small size is inefficient when the roundtrip delay is large relative to the transmission time of a packet. On the other hand, making a window large requires too much buffer space for the storage of unacknowledged packets, which is a scarce commodity in parallel computers, and an estimate of the roundtrip delay, which is not always easy.

The livelock problem, where a packet circulates in the network without ever reaching its destination cannot occur with the CSR protocol; a packet that enters the network is guaranteed to arrive at its destination with a finite delay. Also, as described in Section 5, the CSR protocol does not suffer from the deadlock problem. A livelock (or a deadlock) may arise with deflection routing (or with wormhole routing, respectively), unless special measures are taken. Also, as explained in Section 5, the CSR protocol is more fair than other switching formats.

We initially present the CSR protocol in its generality. The description that we give is independent of the network topology, the routing algorithm used, and the buffer space available. We then specialize the CSR protocol to the case of a hypercube network of processors with buffer space only for the packet being transmitted. We focus on a particular routing algorithm, where packets traverse the hypercube dimensions in descending order. The node switches assumed by this routing algorithm are simple and inexpensive blocking switches, instead of cross-bar switches. The throughput of the unbuffered case is evaluated for various traffic loads through an approximate analysis, and is found very satisfactory. For the routing algorithm and the buffer that we assume, the protocol guarantees that every packet that enters the network arrives at its destination after exactly d slots, where d is the diameter of the hypercube.

The organization of the paper is the following. In Section 2, we describe the CSR protocol in its generality. In Section 3, we describe a particular CSR implementation for a hypercube network of processors. In Section 4, we evaluate the throughput of this implementation. In Section 5, we compare the CSR hypercube implementation to other switching formats and routing schemes, we discuss implementation issues, and we conclude the paper.

2 Description of the CSR Protocol

In this section we present the CSR protocol for a general topology, and describe the data structures that are necessary for its implementation. A pair (s, l) will represent the lth link of processor s, and d will represent the diameter of the network. We assume the existence of a routing algorithm which, for each source s and destination v, finds a path leading from s to v, and provides a way to resolve conflicts among packets requiring the same link. The only property that the routing algorithm has to satisfy is that the links traversed by a packet $\mathcal{P}$, and the time instants these links are used do not depend on packets that entered the network after $\mathcal{P}$. We call such a routing algorithm *future oblivious*. Future oblivious algorithms can be deterministic, probabilistic, distributed, or even adaptive. For example, an algorithm that gives priority to a given packet over packets that entered the network earlier is *not* future oblivious, while one that gives priority to a packet over packets that entered later is future oblivious.

Each link (s, l) of the network is assumed to have a *link buffer* and an *entry buffer*, denoted by $\mathcal{Q}_l^s$ and $\mathcal{E}_l^s$, respectively. Entry buffers form the interface of a processor with its router and can store only new packets. A packet *enters* (or *is accepted to*) the network when it moves from an entry buffer to the corresponding link buffer. A link buffer can be used only by packets already accepted to the network, and can hold up to K_l^s packets, in addition to the packet being transmitted. We assume that all packets require one time unit for transmission over a link, and a single buffer space for storage.

We define a *flit* (from "flow control unit") as the smallest number of bits which can contain routing information, or else the minimum number of bits which can be accepted or rejected by a link buffer (the term flit was used, with a slightly different meaning, by Dally [3]). A typical size of a flit is 64 bits. Flits are used in parallel computers that support wormhole routing (for example the J-machine, see [3]), and are similar to the set-up messages used in circuit switching. We assume that a node has a way to distinguish control flits from actual data.

A packet stored at an entry buffer sends a flit before entering the network in order to reserve the resources that it will need. This flit reserves a resource (link or buffer) only for the slots during which it will need it. A flit generated at an entry buffer follows the same path that the packet would follow, that is the path provided by the underlying routing algorithm of the network. Flits are treated one at a time by a link on a FCFS basis. If two flits arrive at the same time the order in which they are considered is found arbitrarily, for example at random. Flits that fail to reserve a link are blocked on the spot.

Let $\beta/2$ be an upper bound on the time required for a flit to travel a distance of d links, where d is the diameter of the network. We can take

$$\beta = 2kd\tau, \tag{1}$$

with

$$\tau = \frac{F}{W} + \gamma,$$

where F is the length of a flit in bits, W is the bandwidth of a link measured in bits per unit of time, γ is an upper bound on the propagation and processing delay of flits, and k is the buffer size for packets per link. This is because if a flit is not blocked, it can be delayed by at most $k-1$ other flits on a particular link. The parameter β is of critical importance, and it will be seen in what follows that the CSR protocol makes sense primarily when β is small relative to the transmission time of a packet.

The time axis is divided into alternating *control intervals* of length β units of time, where flits are routed and reservations are made, and *transmission intervals* of length equal to one unit of time where packet transmissions actually take place (see Fig. 1). A control interval is divided into a *forward* and a *backward* phase, each of length $\beta/2$. During the for-

ward phase flits travel from their source to their destination, reserving links and buffer space. After $\beta/2$ time units all flits have either arrived at their destination or have been blocked (see Appendix for a proof). In the backward phase flits travel in the opposite direction, carrying feedback information to the source. A way to ensure that flits will not collide on links in the backward phase, is to transmit a flit on a link at time $(2kd - i - 1)\tau$ in the backward phase, if it was transmitted on the same link (in the opposite direction) at time $i\tau$ in the forward phase, for $i = 0, 1, \ldots, kd - 1$ (see Appendix). In this way the feedback is 100% reliable. For a general network and a general routing algorithm, storage for at most $kd\delta$ flits per node, where δ is the in-degree of a node, is sufficient (see Appendix). If blocked flits do not have have to return to the origin (and therefore do not have to be stored), the storage requirements at the nodes for flits are considerably reduced. For the hypercube CSR implementation that we will give in Section 3, we will see that storage space for just one flit per link is adequate.

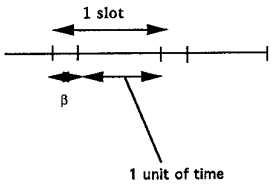


Fig. 1. The time axis divided into slots. A slot consists of a control interval of length β, and a transmission interval of equal to one unit of time.

Flits that have been blocked carry negative acknowledgments (or NACKs for brevity), while flits that have made all the necessary reservations carry positive acknowledgments (or ACKs). A NACK prevents the packet from entering the network during the transmission interval of the current slot. This saves bandwidth since such a packet would be dropped, if transmitted, at exactly the same link where the flit was blocked. This is the reason we call the protocol *conflict sense* routing protocol: it senses a conflict before it actually happens. The control interval serves as a "microscopic," inexpensive rehearsal of what would happen if the packet was transmitted. In this way, after a feedback delay of β time units, each entry buffer knows whether the packet (if any) that it holds can be transmitted without being dropped, or not.

The way the reservations are made and the data structures required for this purpose are described next. For every link queue $\mathcal{Q}_l^s$ there is a list $\mathcal{L}_l^s$, called *reservation list*, whose elements represent future transmission intervals. The first element represents the next transmission interval. At the end of a transmission interval, the first element of the list is deleted. The element of $\mathcal{L}_l^s$ which corresponds to the tth transmission interval (transmission intervals are counted with respect to the present control interval) is denoted by $\mathcal{L}_l^s[t]$, and is composed of two fields, denoted by $\mathcal{L}_l^s[t]\_link$ and $\mathcal{L}_l^s[t]\_buffer$. In case there is no buffer space at the links except for the packets currently under transmission, the two fields collapse into one. The field $\mathcal{L}_l^s[t]\_link$ is equal to one if the link (s, l) has already been reserved for the tth transmission interval, and zero otherwise. The field $\mathcal{L}_l^s[t]\_buffer$ takes integer values between zero and the buffer size K_l^s. It is equal to the number of buffer spaces of $\mathcal{Q}_l^s$ already reserved for the tth transmission interval.

Each flit f carries with it a *counter*, denoted by c_f. The counter of a flit generated at an entry buffer is originally set to one. In the forward phase of a control interval the flit travels on the path provided by the routing algorithm from the source to the destination. Let $\mathcal{L}_l^s$ be the reservation list of link (s, l), respectively, at the time when f is considered by link (s, l), where (s, l) is a link on the path of f. We define T as the minimum integer that satisfies

$$c_f \le T,$$
$$\mathcal{L}_l^s[T]\_link = 0,$$

and

$$\mathcal{L}_l^s[t]\_buffer < K_l^s \text{ for all } t \in \{c_f, c_f + 1, \ldots, T - 1\}.$$

If such a T exists, the link (s, l) is reserved for the Tth transmission interval by f. A buffer space at $\mathcal{Q}_l^s$ is also reserved for the intervals c_f up to $T - 1$. At the same time the reservation list of (s, l) and the counter of f are updated according to

$$c_f := T + 1,$$
$$\mathcal{L}_l^s[t]\_link := 1,$$

and

$$\mathcal{L}_l^s[t]\_buffer := \mathcal{L}_l^s[t]\_buffer + 1 \quad \forall t \in \{c_f, c_f + 1, \ldots, T - 1\}.$$

If a T that satisfies the previous relations does not exist, the flit is blocked, and the reservation fails. During the backward phase of the control interval such a flit returns to its source entry buffer by using the reverse path, carrying a negative acknowledgment (NACK) and freeing the links and buffer space it has reserved in the forward phase. A packet which receives a NACK does not enter the network at the next transmission interval and will retry to make the necessary reservations at some subsequent control interval. If on the other hand a flit manages to reach its destination reserving all the necessary resources, then at the backward phase it returns to its source entry buffer as a positive feedback. The corresponding packet will enter the network at the immediately following transmission interval, and will arrive at its destination after several transmission intervals by using the links and buffer space already reserved for it. If a packet that receives a NACK always retries at the next control interval, then the protocol preserves the order of the packets sent from a particular entry buffer to a destination node.

A last issue that has to be dealt with is the method of recording which packet reserved a link for a particular slot. One way is to store that information at the intermediate nodes, by having a third field at $\mathcal{L}_l^s[t]$ which will record the sequence number of the packet that reserved (s, l) for the tth slot. A different approach is to have the bookkeeping information attached to the packet. This is done by having the

flit record the sequence of values $c_f^{(i)}$ that it takes after each hop i (or, even better, the differences $c_f^{(i)} - c_f^{(i-1)}$). In the case where there is buffer space only for the packet being transmitted the book-keeping information is not needed. Note that it is not necessary to know which packet reserved which particular buffer space, since buffer spaces can be organized as a pool.

We finally note that the CSR protocol shares with other reservation schemes a generic drawback: for light load and large β it has larger delay than packet or circuit switching. For a packet that has to travel k hops this delay is nearly equal to k units of time with packet switching, $k + \beta$ with circuit switching, and $k(1 + \beta)$ with the CSR protocol. For heavy load or small β, the CSR protocol is expected to have smaller delay than circuit or packet switching, because it uses links more efficiently (which means higher throughput and smaller input queuing delay).

3 A Hypercube CSR Protocol

In this section we will describe a hypercube implementation of the CSR protocol. We introduce a particular routing algorithm, which is future oblivious, and superimpose on it the CSR protocol. This algorithm assumes simple inexpensive switches for the nodes, instead of cross-bar switches. We start by describing the model assumed for a hypercube node, and the routing algorithm used.

3.1 The Hypercube Node Model and the Routing Algorithm

Each node of an $N = 2^d$-node hypercube is represented by a unique d-bit binary string $s_{d-1}\, s_{d-2} \cdots s_0$. There are links between nodes whose representations differ in one bit. Given two nodes s and t, $s \oplus t$ denotes the result of their bitwise exclusive OR operation and is called the *routing tag* between the two nodes. We also denote by e_i the binary string whose ith bit is equal to one, and all other bits are equal to zero. A link connecting node s to node $s \oplus e_i$, $i = 0, 1, \ldots, d-1$, is called a *link of the* ith *dimension*. Note that if the ith bit of the routing tag of a packet is equal to one, then the packet must cross a link of dimension i in order to arrive at its destination.

Each link of a node has an entry buffer, which can hold one new packet. The entry buffer of link i of node s is denoted by $\mathcal{E}_i(s)$. The entry buffer is ready to accept a new packet only if the previous packet has reserved the links it will need, and positive feedback has been received. A packet that receives a NACK retries to make the reservations at the next control interval. An entry buffer holding a packet for which no positive feedback has been received is said to be *backlogged*. New packets arrive at the entry buffer of a link. New packets arriving at backlogged entry buffers are discarded.

Each node s has d link queues, each of them associated with a link of the node. The link queue of link i of node s is denoted by $\mathcal{Q}_i(s)$. A link queue is composed of two *buffers* which can hold only one packet each. The first buffer is called *forward buffer*, denoted by $\mathcal{Q}_i^1(s)$, and is used only by packets which must cross the ith dimension. The second buffer, denoted by $\mathcal{Q}_i^0(s)$, is called *internal buffer*, and is used only by packets which do not need to cross the ith dimension. The queues of the nodes are linked in the following way; the internal buffer $\mathcal{Q}_i^0(s)$ is connected to link queue $\mathcal{Q}_{(i-1) \bmod d}(s)$ of the same node and the forward buffer $\mathcal{Q}_i^1(s)$ is connected to queue $\mathcal{Q}_{(i-1) \bmod d}(s \oplus e_i)$ of the neighbor node $s \oplus e_i$ (see Fig. 2). This router organization results in node switches which are simpler, faster, and less expensive than cross-bar switches (see the comments on the node model in Section 5).

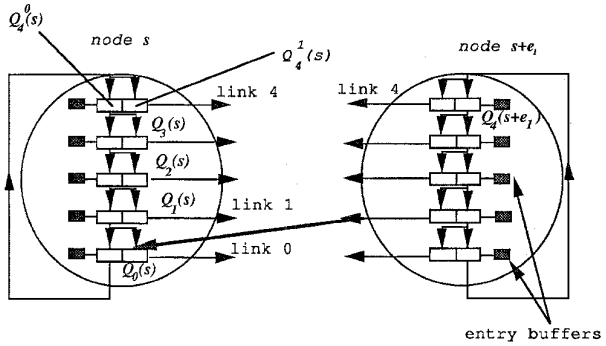


Fig. 2. Two neighbor nodes of a five-dimensional hypercube.

The routing algorithm is the following. A new packet generated at a node selects a link, say link l, with equal probability independently of its destination and competes for one of the two buffers of the lth link queue, $\mathcal{Q}_l^1(s)$ or $\mathcal{Q}_l^0(s)$, depending on whether it must use the lth dimension or not. The packet traverses the dimensions in descending (modulo d) order, starting from the random dimension l. In particular, consider a packet which arrives at queue $\mathcal{Q}_i(s)$ of node s, either from buffer $\mathcal{Q}_{(i+1) \bmod d}^0(s)$ of s, or from buffer $\mathcal{Q}_{(i+1) \bmod d}^1(s \oplus e_{i+1})$ of the neighbor node $s \oplus e_{i+1}$, or from the entry buffer $\mathcal{E}_i(s)$. The ith bit of its routing tag is checked, and depending on whether it is equal to one or zero, the packet claims buffer $\mathcal{Q}_i^1(s)$ in order to be transmitted to queue $\mathcal{Q}_{(i-1) \bmod d}(s \oplus e_i)$ of the neighbor node $s \oplus e_i$, or it claims buffer $\mathcal{Q}_i^0(s)$ in order to be internally passed to the next link queue $\mathcal{Q}_{(i-1) \bmod d}(s)$ of the same node. If two packets require the same link and there is not enough buffer space at the link, one of them is dropped. We will only analyze the case where each link buffer has space for only one packet, the one being transmitted. Note that when the CSR protocol is superimposed on the routing algorithm, packets are *not* dropped due to collisions: the flit of one of the packets that collide returns to its source carrying a NACK, preventing the packet from entering the network.

3.2 Superimposition of the CSR Protocol on the Hypercube Routing Algorithm

In this subsection we describe how the CSR protocol is superimposed on the hypercube routing scheme of the previous subsection. The unit of time is taken as the time required for the transmission of a packet over a link. The pa-

rameter β here is the time (in units of time) required by a flit to travel a distance of $2d$ links (recall our assumption that each link has buffer space for only one packet, so β is given by (1) with $k = 1$). The time axis is divided into slots, each of which has duration $1 + \beta$ units of time. During the first β time units, the flits are transmitted to make reservations for the new packets that want to enter the network. During the remainder of the slot the old packets and the new packets that have made the necessary reservations are transmitted one hop.

An entry buffer holding a packet that wants to enter the network (a new packet or one that is being retransmitted) sends a flit to the packet's destination, containing the packet's routing tag. A flit originated at link l is transmitted during the ith step of the forward phase, $i = 0, 1, \ldots, d - 1$, over the $l - i$ mod d dimensional forward (or internal) link of a node, if the $(l - i \bmod d)$th bit of its routing tag is a one (or a zero, respectively), provided that this link has not been reserved by another packet. At the same time the flit makes a reservation of that link for the ith subsequent transmission interval. When two flits try to reserve a link at the same time and for the same transmission interval, one of them is selected at random to make the reservation, and the other is blocked. Flits that find a link already reserved are also blocked.

When all flits have been blocked or have arrived at their destination, which happens after at most $\beta/2$ time units, a backward phase begins. In the backward phase each flit which was blocked follows the reverse route to its origin carrying a negative acknowledgment (NACK), and freeing the links that it had reserved. The NACK will prevent the corresponding packet from entering the network during the transmission interval of the current slot. Flits which reserve all the links to their destination, return in the backward phase to their origin following the reverse path than the one followed in the forward phase, and carrying a positive feedback (ACK). A flit is transmitted over a link at step $2d - i - 1$ in the backward phase ($i = 0, 1, \ldots, d - 1$) if it was transmitted on the same link at the opposite direction during step i in the forward phase. In this way there are no conflicts between flits in the backward phase. After a packet enters the network, it follows its path knowing that it will not collide with any other packet; the only information it needs is its routing tag. If an entry buffer receives a negative feedback, it tries to make the necessary reservations at one of the next control intervals. If we require only ACKs to return to their origins (in this case after a constant delay of β time units, a NACK is assumed and blocked flits can be discarded), then even with storage for just one flit per link, ACKs never get lost.

The preceding hypercube implementation indicates some of the typical advantages of a CSR protocol. First, packets that are going to be dropped are not allowed to enter the network. This prevents congestion from feeding on itself. Second, the feedback is obtained as soon as possible. This makes the use of a window of size one possible and efficient at the same time, and the storage of packets not yet acknowledged minimal. All packets accepted to the entry buffer (router) arrive at their destination with constant delay. Whenever a packet is successfully transmitted, the corresponding entry buffer enables the processor to insert a new packet, if it has one. Resources are reserved for as long as they are needed and yet all the advantages of circuit switching are maintained. The hypercube CSR protocol example indicates that by using capabilities available in multiprocessor systems, namely the possibility to efficiently route flits through the knowledge of the topology, the flow control mechanism becomes easy and efficient.

Note that the control flits can also be transmitted "off" channel. In a VLSI implementation of parallel computers, there are usually many wires for each link, and the bandwidth of the link is proportional to the number of these wires. In such systems several bits are transmitted in parallel over a link during a clock cycle. In an implementation of the CSR protocol, one would probably choose to dedicate one of these wires to the control flits in order to simplify the design. This corresponds to a kind of FDMA multiplexing as opposed to a TDMA multiplexing of control information and data.

4 Performance Analysis of the CSR Protocol for Hypercubes

In this section we present an approximate analysis of the throughput of the hypercube CSR protocol described in the previous section. We assume that packets having a single destination are generated at each node, and the destinations of the packets are uniformly distributed over all the hypercube nodes. Packets are being generated over an infinite time horizon, and require one unit of time for transmission over a link. We are interested in the average throughput when the network reaches steady state. We are also interested in the associated stability issues. We will limit our attention to the case where the link buffers have space only for the packet being transmitted.

Assuming that both the control flits and the data packets use the same channel, a slot is defined to be equal to $1 + \beta$ units of time. The probability that the entry buffer of a link tries to insert a new packet during a slot is called *attempt rate* and is denoted by p_0. The attempt traffic is the result of the merging of newly generated packets and retransmissions. Let m be the steady-state average value of the ratio of the number of backlogged entry buffers to the total number of entry buffers. Let also q_a be the probability of a new packet arrival at an entry buffer of a link, and q_r be the probability with which a blocked packet retries to enter the network (by making the necessary reservations) during a control interval. Then the attempt rate is

$$p_0(m) = (1 - m)q_a + mq_r$$

If retransmissions are sufficiently randomized, it is plausible to approximate the process of attempted reservations from an entry buffer, by an independent Bernoulli process with parameter $p_0(m)$. If retransmissions are not attempted from the same entry buffer, but from another available entry buffer of the same node, then more randomization is added, and the Bernoulli approximation is expected to be more accurate even for $q_r = 1$. This approximating assumption is reminiscent of the approximating assumption used in the analysis of various multiaccess systems (for example

the Aloha protocol, see [1]), where the aggregate traffic of new arrivals and retransmissions is modeled as a Poisson process.

The system that we analyze has some similarities with a multiaccess system (for example, an Aloha system). Conflicts over links or buffer space correspond to collisions in a multiaccess system. An important difference is that packets in the CSR protocol collide when they request the same link for the same slot, while in Aloha whenever two nodes transmit simultaneously there is always a conflict. Another difference is that in our system whenever a conflict occurs, one of the conflicting packets is granted the link (or buffer), while in Aloha whenever a collision happens all transmissions are destroyed. The feedback in our system requires β time units, while in multiaccess systems it is usually assumed instantaneous. The CSR protocol also has similarities with the Carrier Sense Multiaccess protocol, since they both "sense the channel" before transmitting, in order to avoid collisions.

It is possible that a flit reserves a link l during some control interval and frees it later in the same control interval due to its failure to reserve the remainder of its path. We will refer to such a reservation as a *ghost reservation*, as opposed to a *confirmed reservation* where the flit after reserving link l, it also reserves the rest of its path. Let $p(t-i+1, t:l)$, $i = 1, 2, \ldots, d$, be the probability that a particular link l is reserved (by a confirmed or a ghost reservation) on the $t-i+1$th control interval for the tth transmission interval. Assuming that the system eventually reaches steady state, the following limit exists and is independent of l:

$$p_i = \lim_{t\to\infty} p(t-i+1, t:l), \quad i = 1, 2, \ldots, d.$$

Thus p_i is the steady-state probability that in a given control interval a link is reserved for the ith subsequent transmission interval. Note that we have $p_{i+1} \le p_i$, for all $i \in \{0, 1, 2, \ldots, d-1\}$. Note that $p(t-i+1, t:l)$ is independent of l for any t (and not only in steady state); we use the index l to clarify the meaning of some of the subsequent calculations, but we will also sometimes omit it.

Consider two flits f_1 and f_2 corresponding to packets $\mathcal{P}_1$ and $\mathcal{P}_2$, which try to make the necessary reservations during control intervals t_1 and t_2, starting from entry buffers of dimensions d_1 and d_2, respectively. Packets $\mathcal{P}_1$ and $\mathcal{P}_2$ may request link l for the same slot t [$t \ge \max(t_1, t_2)$] only if l is on their path, all links needed prior to slot t have been reserved, and $t_1 + d_1 \bmod d = t_2 + d_2 \bmod d$. If $t_1 < t_2$ then $\mathcal{P}_1$ is not affected by the presence of $\mathcal{P}_2$, since its attempt to make the reservations is made at a control interval prior to $\mathcal{P}_2$'s arrival. In this case, f_2 can reserve link l only if f_1 fails to reserve all of its path. If $t_1 = t_2$ (and $d_1 = d_2$) then f_1 and f_2 will claim the same link provided that it is on their path and they have reserved all other links they need prior to l. If the link is free, then it is allocated to one of them arbitrarily.

We want to calculate p_i for $i > 1$. Let l_1 and l_2 be the internal and forward links, respectively, that lead to l. Link l may be reserved either by a flit f_1 coming on l_1, or by a flit f_2 coming on l_2. Link l can be reserved during the $t-i+1$th control interval for transmission interval t by a flit coming on l_1 only if:

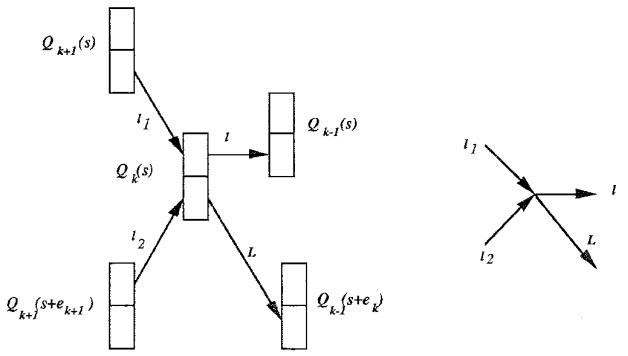


Fig. 3. A link l, and the two links leading to it. In this figure, link l corresponds to an internal link connecting the internal buffer $\mathcal{Q}_k^0(s)$ to the link queue $\mathcal{Q}_{k-1}(s)$ of the same node. Link L corresponds to a forward link connecting the forward buffer $\mathcal{Q}_k^1(s)$ to the link queue $\mathcal{Q}_{k-1}(s \oplus e_k)$ of the neighbor node $s \oplus e_k$. Similarly, l_1 is the internal link that connects the internal buffer $\mathcal{Q}_{k+1}^0(s)$ to the link queue $\mathcal{Q}_k(s)$ of the same node, and l_2 is a forward link connecting the forward buffer $\mathcal{Q}_{k+1}^1(s \oplus e_{k+1})$ of the neighbor node $s \oplus e_{k+1}$ to the link queue $\mathcal{Q}_k(s)$ of node s. Flits (as well as packets) cross dimensions in descending modulo d order; thus flits arriving on links l_1 or l_2 are transmitted, if not blocked, on links l or L, depending on their destination (or equivalently on their routing tag).

- a reservation was made for l_1 for the $t-1$ transmission interval during control interval $t-i+1$; this happens with probability $p(t-i+1, t-1:l_1)$,
- link l is on the flit's path (given that l_1 is on the flit's path); this happens with probability 1/2,
- no confirmed reservation has been made for l during a previous control interval, and no reservation (confirmed or ghost) has been made by a flit coming on l_2 during the same control interval for transmission interval t.

Thus,

$$p(t-i+1, t:l) = 2\frac{p(t-i+1, t-1:l_1)}{2}\left(1 - \Pr(A \mid B)\right), \quad i = 2, 3, \ldots, d, \tag{2}$$

where A is the event that a confirmed reservation has been made for link l for transmission interval t during a previous control interval, or a reservation has been made during the current control interval for transmission interval t by a flit coming on l_2, and B is the event that f_1 reserved link l_1 for the transmission interval $t-1$ during control interval $t-i+1$.

The factor 2 in (2) accounts for the fact that l can be reserved either by a flit coming on l_1, or by a flit coming on l_2. Given that event B occurred, we know that no confirmed reservation has been made for l for the transmission interval t by a flit coming from l_1. Therefore, the probability of the event A is equal to the (conditional on B) probability that some flit f_2 reserved l_2 for the transmission interval $t-1$ during control interval $t-j+1$ with $j = i+1, \ldots, d$, it chose link l, and its reservation was finally confirmed. Ignoring the conditional on B, this probability can be approximated by

$$\frac{1}{2}\sum_{j=i+1}^{d} p(t-j+1, t-1:l_2)\frac{p(t-j+1, t-j+d)}{p(t-j+1, t)}. \quad (3)$$

The ratio

$$p(t-j+1, t-j+d)/p(t-j+1, t)$$

in (3) is the probability that the reservation of l by f_2 was finally confirmed.

The probability that a flit f_2 claims link l during the *same* control interval $t-i+1$ with f_1, and for the same transmission interval t, and it is granted the link can also be approximated, ignoring the conditional on B, by

$$\frac{1}{4}p(t-i+1, t-1:l_2). \quad (4)$$

The factor 1/4 is the probability that f_2 requests link l (given that it reserved l_2), and it is selected instead of f_1. Combining (3) and (4) we get

$$\Pr(A \mid B) = \frac{1}{2}\sum_{j=i+1}^{d} p(t-j+1, t-1:l_2)\frac{p(t-j+1, t-j+d)}{p(t-j+1, t)} + \frac{1}{4}p(t-i+1, t-1:l_2).$$

The preceding equation, together with (2) gives

$$p(t-i+1, t:l) = p(t-i+1, t-1:l_1)\cdot \left(1-\frac{1}{2}\sum_{j=i+1}^{d} p(t-j+1, t-1:l_2)\frac{p(t-j+1, t-j+d)}{p(t-j+1, t)} - \frac{p(t-i+1, t-1:l_2)}{4}\right)$$
$$i = 2, 3, \ldots, d. \quad (5)$$

Taking the limit $t \to \infty$ and using the symmetry with respect to the links we obtain from (5) that

$$p_i = p_{i-1}\left(1-\frac{1}{2}\sum_{j=i}^{d-1} p_j \frac{p_d}{p_{j+1}} - \frac{p_{i-1}}{4}\right) \quad \text{for } i = 2, 3, \ldots, d, \quad (6)$$

which yields

$$p_{i-1} = 2 - p_d\sum_{j=i}^{d-1}\frac{p_j}{p_{j+1}} - \sqrt{\left(2 - p_d\sum_{j=i}^{d-1}\frac{p_j}{p_{j+1}}\right)^2 - 4p_i} \quad \text{for } i = 2, 3, \ldots, d. \quad (7)$$

To relate p_1 and p_0 we first observe that at the beginning of a control interval, the steady-state probability that a link is reserved for the ith subsequent transmission interval, $i = 1, 2, \ldots, d-1$, is equal to p_d (note that these are confirmed reservations, since all ghost reservations of previous control intervals have been canceled, and that no reservations for the dth subsequent transmission interval have been made yet). Thus the probability that a link is unreserved is $1-(d-1)p_d$, and, therefore,

$$p_1 = p_0(1-(d-1)p_d). \quad (8)$$

For a particular value of p_d, we can use (7) to find p_i in terms of $p_{i+1}, \ldots, p_d$ for each i, and then (8) to find the corresponding p_0. Repeating this for various values of p_d we obtain a curve that gives p_d as a function of the attempt rate p_0. It is possible to prove inductively that p_d is a monotonically increasing (and 1-1) function of p_0.

Fig. 4 illustrates the results obtained for $d = 11$. The horizontal axis corresponds to both the fraction m of backlogged entry queues and the attempt rate p_0, which are related through the linear equation $p_0(m) = (1-m)q_a + mq_r$. The vertical axis corresponds to the throughput per node (curve), which is

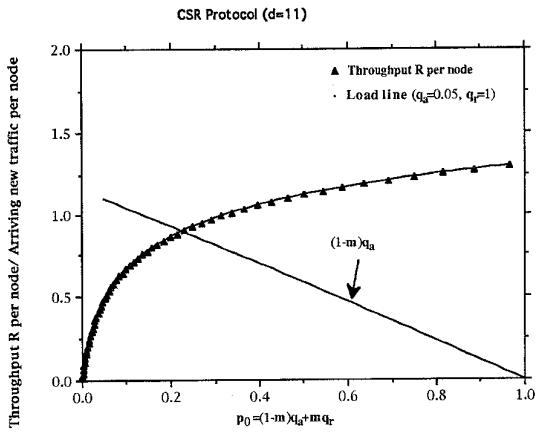


Fig. 4. Throughput and stable point of the hypercube implementation of the CSR protocol for $d = 11$. The probability with which a new packet is available at the entry buffer during a slot is q_a, and the probability of a retrial is q_r. The fraction of backlogged entry queues to the total number of entry queues is denoted by m. The throughput per link cannot be greater than min (2, $2dq_a$) (= 1.1 for $q_a = 0.05$).

$$R = 2dp_d,$$

and the arrival rate of new packets per node (straight line). The throughput and the arrival rate are measured in packets per $1+\beta$ units of time. For each value of the probability of a new arrival q_a, the maximum throughput is obtained for retransmission probability $q_r = 1$ (modulo the approximating assumptions). From Fig. 4 we see that there is a single stable point in a CSR system, which corresponds to quite high throughput. The straight line corresponds to $q_a = 0.05$, which represents rather heavy load.

We have performed simulations in order to assess the accuracy of the analysis. The simulation results obtained for $d = 7$, together with the corresponding analytical results, are shown in Table 1. The relative difference between the simulation and the analytical results is less than 2%.

TABLE 1
SIMULATION AND ANALYTICAL RESULTS FOR THE HYPERCUBE IMPLEMENTATION OF THE CST PROTOCOL FOR $d = 7$

p_0	Throughput/node (analytical)	Throughput/node (simulations)
0.011666	0.140000	0.142795
0.027465	0.280000	0.283746
0.048996	0.420000	0.418328
0.078620	0.560000	0.558200
0.119931	0.700000	0.693059
0.178584	0.840000	0.831379
0.263852	0.980000	0.965929
0.391796	1.120000	1.104581
0.592309	1.260000	1.242851
0.927213	1.400000	1.388006
1.000000	1.422100	1.409178

5 Comparison with Other Switching Formats and Routing Schemes

The CSR protocol can be applied to various topologies and

routing algorithms as a way to perform scheduling and resource management in a synchronous multiprocessor computer. In this section we will compare the hypercube CSR implementation of Section 3 to some other switching formats and schemes. The results concerning these switching formats and schemes are not always directly comparable. Therefore, the comparison is not intended to be a rigorous one, but we believe it will give insight into the relative advantages and disadvantages of each scheme.

The routing schemes that will be compared are the following:

1) The hypercube implementation of the CSR protocol.
2) The simple and the priority deflection schemes described in [10]. In these schemes each node has a queue which can hold up to d packets. When conflicts over a link arise, then packets are misrouted instead of being dropped. During each slot the nodes transmit all the packets that they hold, either by transmitting them on links that take them closer to their destination, or by simply transmitting them on any available link. When assigning packets to links, the priority deflection scheme gives priority to packets which are closer to their destination (see [10], Section 5.5). A common characteristic between deflection schemes and CSR schemes is that they do not drop packets.
3) The unbuffered simple and priority schemes introduced in [10], Sections 5.3-5.4. The switches used by these schemes are the same with those assumed for the hypercube implementation of the CSR protocol (see Fig. 2). The results for these two schemes are directly comparable to those of Section 4 because the feasible switching assignments are in both cases the same; one can use the results to see the improvement obtained by using the CSR protocol instead of packet switching. The priority scheme differs from the simple scheme in the way that contention over buffer space is resolved: in the priority scheme, packets that have been in the network longer have priority.

We will refer to the *saturation point* of a routing scheme as the ratio of the maximum throughput of the scheme for uniformly distributed traffic over the maximum possible throughput that the network can sustain. In other words, the saturation point is the maximum fraction of the capacity of the network that performs useful work, where the maximum is taken over all possible loads. A link is not doing useful work when

1) it is idle,
2) the packet transmitted on it will be eventually dropped,
3) the packet transmitted on it is being deflected, or it had previously been deflected on a link of the same dimension.

Fig. 5 indicates the saturation point for the hypercube implementation of the CSR protocol (Sections 3 and 4), the simple and the priority deflection schemes ([10], Section 5.5), the unbuffered simple scheme ([10], Section 5.3), and the unbuffered priority scheme ([10], Section 5.4) for various dimensions d of the hypercube.

In order to interpret Fig. 5 the following comments are necessary:

CSR Protocol: The saturation point for each hypercube size is obtained from the approximate analysis of Section 4 (which is within 2% from simulation results), and correspond to the unbuffered implementation of Section 3. The routing algorithm on which the CSR protocol is superimposed assumes a very simple switch at the nodes. Other implementations would probably give a higher saturation point, but they would require more expensive nodes (see the comments below on the node cost).

Deflection Routing: The results of Fig. 5 on the simple and the priority deflection schemes have been obtained through simulations. For the evaluation of the saturation point of both deflection schemes we have taken the probability of access p_0 to be equal to one, that is, we have assumed that packets are always available and enter the hypercube whenever there is an available empty slot. This does not necessarily result in the maximum possible throughput, but the difference is of the order of 2-3% (see [10]), which is within the statistical error of our simulations, and is in any case negligible.

Simple and Priority Schemes: The results of Fig. 5 for the simple and the priority schemes have been obtained from an approximate analysis of [10]. The approximate analysis is in very good agreement with simulation results.

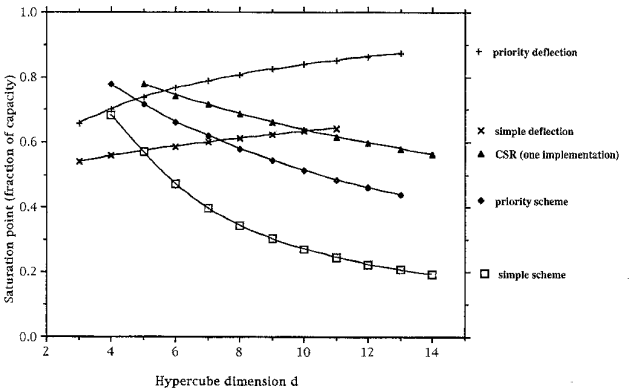


Fig. 5. The saturation point as a function of the dimension of the hypercube for: 1) the hypercube implementation (Section 3) of the CSR protocol for $\beta = 0$ (for other values of β the saturation point should be divided by $1 + \beta$), 2) the priority deflection scheme, 3) the simple deflection scheme, 4) the (unbuffered) priority scheme, and 5) the (unbuffered) simple scheme.

The remainder of the section is devoted in examining advantages and disadvantages of each scheme when applied to the hypercube network, and to other topologies.

5.1 Saturation Throughput and Congestion

The schemes that are the most interesting in terms of saturation throughput are the two deflection schemes (especially the priority deflection scheme), and the CSR

scheme. One reason the CSR scheme does not achieve 100% utilization of the capacity of the network even for heavy load is source blocking: if a packet fails to make the necessary reservations, then the corresponding entry buffer is backlogged and the packets behind it do not have access to the network. A second reason is related to the "segmentation" of the available link capacity: some links may be free, but if put together they may not form a whole path (d links may form a path only when their link dimensions appear in descending order). This segmentation is not inherent in the CSR protocol, and is mainly due to the simple node switches assumed, which are not cross-bar switches, and do not permit arbitrary switching assignments. The saturation throughput of the CSR scheme shown in Fig. 5 has to be divided by $1 + \beta$, since each slot is equal to $1 + \beta$ units of time. For example, if $\beta = 0.5$ then the curve that corresponds to the CSR protocol has to be multiplied by 2/3. It is, however, important that the CSR protocol does not require additional acknowledgment packets, while the simple and the priority schemes do require. The two deflection schemes also require the use of acknowledgments for reasons to be explained later, but to a lesser extent. Therefore, for all the schemes examined there is some overhead not taken into account in Fig. 5 (one can view the parameter β as the cost of the acknowledgments).

A disadvantage of the unbuffered simple scheme is that, after some point, increasing the offered load decreases the throughput (see [10]). This makes necessary the existence of a mechanism for controlling the transmission rate of the nodes. This is less of a problem for the priority scheme (buffered or unbuffered), the buffered simple scheme, and the simple deflection scheme. In the latter schemes the throughput at the saturation point is somewhat smaller than the throughput when the attempted traffic is the maximum possible, but this difference is small (less than 5% for hypercube dimension less than 13; see [10]). The simulations of the priority deflection scheme, and the approximate analysis of the CSR scheme have indicated that their throughput increases monotonically when the offered load increases.

The results of Fig. 5 assume uniform traffic. If the traffic is not uniform then congestion may become a serious problem, especially for deflection routing, as results in [9] indicate. Congestion feeds on itself since it forces packets to take longer paths, increasing the utilization, and making other packets to take even longer paths. If the topology is not regular, congestion may become an even more serious problem. Even in regular topologies which have a severe penalty for deflections (for example, the shuffle exchange network; see [8] and [9]) deflection routing can be very inefficient in terms of throughput. The CSR protocol behaves better in congestion, and is apparently least affected by the choice of the topology.

5.2 Node Cost

Deflection routing requires a $d \times d$ cross-bar switch with $\Theta(d^2)$ wires at each node of a hypercube. The simple and the priority schemes, as well as the hypercube implementation of Section 3 of the CSR protocol require a much simpler switch. This switch, which we call *descending-dimensions switch*, is illustrated in Fig. 6 (see also Fig. 2). The number of wires of a descending-dimensions switch is only $\Theta(d)$. A cross-bar router is larger and slower, and results in a slower network (the processing time at a node and the clock cycle is larger). The switching assignments possible with the descending-dimensions switch are of course more restricted, and suffer from internal message collisions (the collisions on the internal links of the node model of Fig. 2). This results in a degradation in performance, which in the case of the CSR protocol was not severe. Since the descending-dimensions switch uses simple 2 : 2 switch/merge switches, it can be made to operate very quickly, which may offset the degradation in the performance due to the restrictions in the routing algorithm (see [3]). If the CSR protocol were used with a cross-bar switch, it would probably outperform deflection routing (for small enough β); however, we believe that the improvement would not be worth the additional cost. An advantage of the CSR protocol is that it performs well even with simple switches.

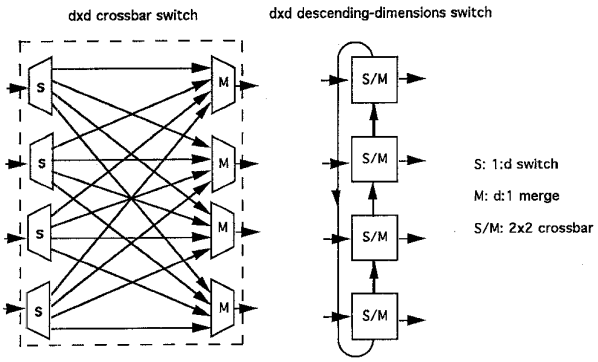


Fig. 6. A $d \times d$ cross-bar switch, a $d \times d$ descending-dimensions switch, and the modules out of which they are composed.

5.3 Livelock/Deadlock

The livelock problem is unique to deflection routing (see [9]). It occurs when packets are transmitted continuously without any chance of reaching their destination. This problem cannot be removed by an end-to-end control scheme, since such packets do not reach their destination. If routing decisions are made deterministically then scenarios can be found where a livelock persists forever. Possible solutions to the livelock problem exist (see [9]), but complicate the implementation. The other switching schemes examined do not suffer from this problem. In particular, in the CSR protocol a packet that is accepted in the network is guaranteed to arrive at its destination with a finite delay (upper bounded by d, if no buffering is used). This is because all resources that will be used by the packet are reserved in advance. Another problem that may arise in switching schemes for parallel computers (e.g. in wormhole routing, see [4]) is that of a deadlock. A deadlock arises when there is a cyclic dependency where a packet cannot proceed without being granted some resource, and this resource is being held by another packet, and cannot be freed before the first packet proceeds. This situation may cause important problems for some switching schemes (such as

wormhole routing; see [4]), and special mechanisms have to be devised to avoid it, or to resolve it if it occurs. The CSR protocol is deadlock-free, because all reservations made by a flit are canceled in the backward phase if the flit fails to reserve all the necessary resources along its path.

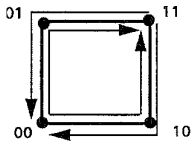


Fig. 7. Node 00 continuously sends two packets per slot to node 11, and node 11 sends two packets per slot to node 00. Then, if the (unbuffered) priority scheme or one of the two deflection schemes is used, both nodes 01 and 10 cannot insert any packets. If at some slot node 00 does not send a packet via node 01 to node 11, then node 01 will be able to insert a packet, but it will again be blocked out when node 00 resumes sending packets to node 11. If the CSR protocol (without buffers) is used, the senario shown in the figure will also cause nodes 01 and 10 to be blocked out. However, if at some slot node 00 does not send a packet via node 01 to node 11, node 01 may take this slot, and if it has more packets to send, it will also take the subsequent slots. Therefore, the fairness problem is less important for the CSR protocol, and is easily solved, for example, by forcing from time to time a node to leave empty slots.

5.4 Fairness

The priority scheme, and the two deflection schemes, can cause the system to operate unfairly. This is because packets that are on transit have priority over packets that are trying to enter the network. The first source that has access to the empty slots make take all the slots that it requires, while the source that follows takes what is left over. Fig. 7 illustrates how a source can be locked out with the priority deflection scheme (even with randomized decisions). The CSR protocol is more fair as the caption of Fig. 7 explains.

5.5 Packet Resequencing

The CSR protocol can easily guarantee that packets arrive at their destination in sequence. On the other hand, the need for resequencing packets is inherent in deflection routing, and cannot be avoided. An implication of that is that resequencing buffers may overflow, dropping packets, and making the use of acknowledgments necessary.

5.6 Processing at the Nodes

The simple and the priority schemes are the easiest to implement. The hardware required for these schemes is very simple (see Fig. 2 for the node model). Deflection routing requires more processing at each node, especially if we want to address the livelock and the fairness problems. Also, a cross-bar switch is slower than a descending-dimensions switch. The priority deflection scheme requires slightly more processing time at the nodes than the simple deflection scheme. The CSR protocol can be implemented fairly easily in a synchronous system. In the unbuffered case, the state of each link can be described by a binary number of length d (at any time reservations may exist for the next d slots at most), which should not be a problem.

5.7 Synchronization

The CSR scheme, the priority scheme, and the two deflection schemes are best suited for synchronous systems. The simple scheme can also be implemented in an asynchronous system. Synchronous systems have a number of advantages which have resulted in an almost universal use (see [3]). Asynchronous systems are potentially faster (the slower component does not have to dominate the speed of the system), and avoid the problem of the distribution of the clock to all the chips with as small a clock-skew as possible, at the expense of a much more complicated implementation.

Appendix

In Section 2, we claimed that the duration of the forward phase of the control interval is at most equal to $kd\tau$ time units, and we described a rule that when followed ensures that there are no collisions among flits in the backward phase. We also claimed that for a general network storage space for at most $kd\delta$ flits is required, where d is the diameter of the network, and δ is the in-degree of the node. In the following lemma we prove these claims.

LEMMA 1. The forward phase of a control interval requires at most $kd\tau$ time units. If the rule of Section 2 is followed, at most one flit is scheduled on a link at any time in the backward phase, so the backward phase is collision-free. Finally, storage for at most $kd\delta$ flits is required per node.

PROOF. Consider a control interval t. Let f be a flit in this interval, $c_f^{(i)}$, $i = 1, 2, \ldots, d$, be the content of its counter after the ith transmission, and $c_f^{(0)} = 0$ be the initial content of the counter. As it can be seen from the description of Section 2, the ith transmission of a flit occurs during the $c_f^{(i)}$th step of the forward phase. A flit that successfully reserves all necessary resources completes the forward phase during the $c_f^{(d)}$th step. By convention, if a flit is blocked after its ith transmission, we will say that the value of its final counter is equal to $c_f^{(i)}$. The forward phase is completed by a flit when it is blocked or when it is transmitted for the dth time; this happens at a step equal to the final value of its counter. To prove the first claim of the lemma it is enough to show that a flit counter cannot take a value greater than kd. To see that, note that a flit arriving at a node with counter $c_f^{(i-1)}$ attempts to reserve a link for transmission interval $t + c_f^{(i-1)}$. Since its counter upon departure is $c_f^{(i)}$, the flit reserves the link for transmission interval $t + c_f^{(i)} - 1$, and one buffer space for all transmission intervals between $t + c_f^{(i-1)}$ and $t + c_f^{(i)} - 1$. This means that the link has already been reserved by other flits for transmission intervals between $t + c_f^{(i-1)}$ and $t + c_f^{(i)} - 2$; let S be this set of flits. Since all flits in S were processed before flit f, the content of their counter upon arrival at the node was

less than or equal to $c_f^{(i-1)}$. Therefore, the $c_f^{(i)} - c_f^{(i-1)} - 1$ flits in S have all reserved a buffer space for interval $t + c_f^{(i-1)}$. This is because when a flit is granted a different interval than it asked for, it must also be granted buffer space for all intervals in between. Thus, there are a total of $c_f^{(i)} - c_f^{(i-1)}$ buffer reservations (including the reservation by flit f) for interval $t + c_f^{(i-1)}$. Since the buffer size per link is equal to k (a link can store up to k packets including the packet under transmission), we have $c_f^{(i)} - c_f^{(i-1)} \leq k$. This combined with the fact that $c_f^{(0)} = 0$ gives $c_f^{(i)} \leq kd$ for all $i = 1, 2, \ldots, d$. Since all packets complete the forward phase at a step equal to the final value of their counter, the forward phase is completed in at most kd steps, and requires at most $kd\tau$ time units, where τ is the time required for the transmission of a flit over a link.

It is easy to see that since the forward phase is completed by time $kd\tau$, it does not interfere with the backward phase (which starts after that time). According to the rules of Section 2, two flits will be scheduled for transmission at the same link at time $(2kd - i - 1)\tau$ in the backward phase, only if they were both transmitted over that link at time $i\tau$ of the forward phase. Clearly, this cannot happen because at most one flit is transmitted over a link at any time in the forward phase.

The forward phase requires at most kd steps, and during each step at most δ flits may arrive. Even if all arriving flits are blocked at that node, the node will not have to store more than $kd\delta$ flits. This proves the third claim of the lemma. □

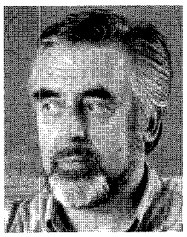
ACKNOWLEDGMENTS

The authors would like to gratefully thank the (anonymous) reviewers for their helpful suggestions. Research supported by NSF under Grants NSF-DDM-8903385 and NSF-RIA-08930554, and by the ARO under Grants DAAL03-86-K-0171 and DAAL03-92-G-0309.

REFERENCES

[1] D.P. Bertsekas and R. Gallager, *Data Networks*. Prentice Hall, 1987.
[2] J.T. Brassil, "Deflection Routing in Certain Regular Networks," PhD thesis, Univ. of California, San Diego, 1991.
[3] W.J. Dally, "Network and Processor Architecture for Message-Driven Computers," *VLSI and Parallel Computation*, R. Suaya and G. Birtwhistle, eds., pp. 140-222. San Mateo, Calif.: Morgan Kaufmann, 1990.
[4] W.J. Sally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers*, vol. 36, pp. 547-553, 1987.
[5] A.G. Greenberg and J. Goodman, "Sharp Approximate Models of Adaptive Routing in Mesh Networks," *Teletraffic Analysis and Computer Performance Evaluation*, J.W. Cohen, O.J. Boxma, and H.C. Tijms, eds., pp. 255-270. Amsterdam: Elsevier, 1988.
[6] A.G. Greenberg and B. Hajek, "Deflection Routing in Hypercube Networks," *IEEE Trans. Comm.*, vol. 35, no. 6, pp. 1,070-1,081, June 1992.
[7] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communicating Switching Technique," *Computer Networks*, vol. 3, pp. 267-286, 1979.
[8] N.F. Maxemchuk, "Comparison of Deflection and Store-and-Forward Techniques in the Manhattan Street and Shuffle-Exchange Networks," *Proc. INFORCOM '89*, vol. 3, pp. 800-809, Apr. 1989.
[9] N.F. Maxemchuk, "Problems Arising from Deflection Routing: Livelock, Lock-Out, Congestion and Message Reassembly," *Proc. NATO Workshop Architecture and High Performance Issues of High Capacity Local and Metropolitan Area Networks*, France, June 1990.
[10] E.A. Varvarigos, "Static and Dynamic Communication in Parallel Computing," PhD thesis, Dept. of EECS, Massachusetts Inst. of Technology, Aug. 1992.

Emmanouel A. Varvarigos received a Diploma (1988) in electrical engineering from the National Technical University of Athens, Greece, and the MS (1990), Engineer (1991), and PhD (1992) degrees in electrical engineering and computer science from the Massachusetts Institute of Technology. In 1990, he worked on optical fiber communications at Bell Communications Research, Morristown, New Jersey. He is currently an assistant professor of electrical and computer engineering at the University of California, Santa Barbara. His research interests are in the areas of parallel and distributed computation, data networks, and mobile communications. Dr. Varvarigos received the first panhellenic prize in the Greek Mathematic Olympiad in 1982 and received the Technical Chamber of Greece award four times (1984-1988). He is a member of the Technical Chamber of Greece.



Dimitri P. Bertsekas received a combined BSEE and BSME from the National Technical University of Athens, Greece, in 1965, the MSEE degree from George Washington University in 1969, and the PhD degree in system science from the Massachusetts Institute of Technology in 1971.

Dr. Bertsekas has held faculty positions with the Engineering-Economic Systems Department at Stanford University (1971-1974) and the Electrical Engineering Department of the University of Illinois, Urbana-Champaign (1974-1979). He is currently a professor of electrical engineering and computer science at the Massachusetts Institute of Technology. He consults regularly with private industry and has held editorial positions in several journals. He was elected a fellow of the IEEE in 1983.

Professor Bertsekas has done research in the areas of estimation and control stochastic systems, linear, nonlinear, and dynamic programming, data communication networks, and parallel and distributed computation and has written numerous papers in each of these areas. He is the author of *Dynamic Programming and Stochastic Control* (Academic Press, 1976), *Constrained Optimization and Lagrange Multiplier Methods* (Academic Press, 1982), *Dynamic Programming: Deterministic and Stochastic Models* (Prentice Hall, 1987), and *Linear Network Optimization: Algorithms and Codes* (MIT Press, 1991) and is a coauthor of *Stochastic Optimal Control: The Discrete-Time Case* (Academic Press, 1978), *Data Networks* (1987), and *Parallel and Distributed Computation: Numerical Methods* (Prentice Hall, 1989).