International Journal of Foundations of Computer Science Vol. 9 No. 1 (1998) 39–56

A DYNAMIC SCHEDULING COMMUNICATION PROTOCOL AND ITS ANALYSIS FOR HYPERCUBE NETWORKS*

AYAN BANERJEE
University of California,
Department of Electrical and Computer Engineering, Santa Barbara, CA 93106
e-mail: ayan@mylos.ece.ucsb.edu

and

EMMANOUEL (MANOS) VARVARIGOS
University of California,
Department of Electrical and Computer Engineering, Santa Barbara, CA 93106
e-mail: manos@ece.ucsb.edu

Received 1 November 1995
Revised 11 July 1997
Communicated by D. F. Hsu, M. Palis and D. S. L. Wei

ABSTRACT

We propose a new protocol for one-to-one communication in multiprocessor networks, which we call the Dynamic Scheduling Communication (or DSC) protocol. In the DSC protocol, the capacity of a link is partitioned into two channels: a data channel, used to transmit packets, and a control channel used to make reservations. We initially describe the DSC protocol and the data structures needed to implement it for a general network topology. We then analyze the steady-state throughput of the DSC protocol for random node-to-node communication in a hypercube topology. The analytical results obtained are in very close agreement with corresponding simulation results. For the hypercube topology, and under the same set of assumptions on the node architecture and the routing algorithm used, the DSC protocol is found to achieve higher throughput than packet switching, provided that the size of the network is sufficiently large. We also investigate the relationship between the achievable throughput and the fraction of network capacity dedicated to the control channel, and present a method to select this fraction so as to optimize throughput.

Keywords: Communication protocols, routing, performance evaluation, hypercubes.

1. Introduction

The *Dynamic Scheduling Communication* (DSC) protocol, proposed in this paper, is a reservation-based protocol for organizing the transmission of data in a multiprocessor network. In a typical VLSI implementation of a parallel computer, there are usually many wires connecting a pair of neighboring nodes. In the DSC

*Research supported by NSF under grant NSF-RIA-08930554.

protocol, a subset of these wires is allocated to the transmission of data and the remaining wires are allocated to the transmission of control information, used to schedule future data.

In the DSC protocol, a packet enters the network only after having reserved the links and buffer space it requires to make it to the destination. An advantage of reservation-based communication protocols, such as the DSC protocol, is that they can provide lossless communication with minimal buffer space, even when the network is operating under heavy load conditions. Such protocols also avoid the waste of resources that arises when a packet is transmitted for several hops before being dropped (as may happen with packet switching), or the blocking of resources that occurs when a packet has to wait at some intermediate link, preventing other packets from using the link it occupies (as may happen with wormhole routing,[2] unless virtual channels are used). As a result, reservation-based protocols have the potential of using more efficiently the available capacity and buffer space than other switching formats, provided that the overhead associated with reservations can be kept low.

The most common example of a reservation-based scheme is circuit switching,[1,5,6,9] where the entire path from the source to the destination is reserved prior to the transmission of a message, and is released only when the message reaches its destination. Circuit switching has many well-known advantages, but it is inefficient for multiprocessors networks, because a link is reserved for more time than is required, and additional overhead is needed to "tear-down" a circuit when the transmission is completed. Another reservation-based protocol is the Conflict Sense Routing (CSR) protocol,[11] which is a hybrid of circuit and packet switching. In the CSR protocol, the time axis is divided into alternating control and data intervals, with reservations for data intervals being made during control intervals. A disadvantage of the CSR protocol is that the reservation overhead is completely determined by the network diameter and the length of the control and data packets, it is not flexible, and it is not under the control of the designer. As a result, at heavy loads, the reservation overhead may severely limit the achievable throughput. The CSR protocol also requires the time-division multiplexing of data and control information on the same wires, which complicates implementation, and introduces additional overhead to distinguish between the two kinds of information. A communication scheme that avoids the dropping of packets without using a reservation mechanism is deflection routing,[3,4,7] which, however, has several drawbacks,[8] the most serious of which is that packets may arrive at their destination out of order, or they may circulate indefinitely without reaching their destination (live-lock) if special precautions are not taken. The Dynamic Scheduling Communication (DSC) protocol proposed in this paper, offers an alternative way to provide lossless communication with minimal buffer requirements, and its performance compares favorably to the previously mentioned protocols.

In the DSC protocol, a resource (links and buffer space) is reserved for a packet only for the slot (or slots if it is a buffer) during which it will be used. This is more efficient than circuit switching, since resources are now used on a demand basis.

The DSC protocol is also more efficient than packet switching, under heavy load network conditions, because it avoids the dropping of packets and the associated waste of bandwidth. Our analysis for the hypercube topology will indicate that the improvements in performance obtained by using the DSC protocol instead of packet switching become more significant when the network diameter is large. The DSC protocol guarantees that a packet accepted into the network is eventually delivered to its destination even when minimal buffer space is available, and it does not require any additional feedback mechanism to acknowledge the receipt of packets, as required by packet switching. Moreover, data packets in the DSC protocol do not have to carry any routing information, since their transmission is scheduled in advance, and the information kept at the nodes is sufficient to perform the routing function. As we argue in Section 2, the control overhead required for the DSC protocol is only slightly more than that required for packet switching, while the improvements in the throughput can be significant.

The DSC protocol is also found to be more efficient than the CSR protocol[11] when the diameter of the network is large, or when the length of the data packets is small. An additional advantage of the DSC protocol over the CSR protocol, is that a node can easily distinguish between data and control packets, since they are transmitted along different wires, considerably simplifying the logic design at the switches. Also, in the DSC protocol, the control overhead is determined by the ratio of the control wires to the total number of wires, and it can be chosen so as to maximize the throughput. This is not possible with the CSR protocol, where the designer does not have such flexibility. The analysis that we will give for the hypercube network shows that the DSC network (when optimized) achieves larger throughput than that achieved by the CSR protocol.

We initially present the DSC protocol in its generality, without assuming a particular network topology and routing algorithm. The control information that has to be exchanged, and the data structure required are also described. We next evaluate the packet delay for light load and the overhead due to reservations, for a general network. We also specialize the DSC protocol to the case of a hypercube network of processors, where each node has buffer space only for the packet being transmitted. We assume a particular node architecture and routing algorithm, where packets traverse the hypercube dimensions in descending order. The switches required by this routing algorithm are more simple and inexpensive than crossbar switches. We obtain analytical results on the throughput for random one-to-one (unicast) communication, for the case where the destinations of the packets are uniformly distributed. The analytical results obtained are in very good agreement (within 2%) with corresponding simulation results. Our results for the hypercube topology indicate that the throughput of the DSC protocol is superior to that of packet switching and the CSR protocol when the size of the network is sufficiently large, under the same set of assumptions on the node architecture and the routing algorithm. We also provide a graphical method to find the optimal fraction of capacity that should be dedicated to the transmission of control information in order to maximize the throughput. The optimal allocation of the capacity depends

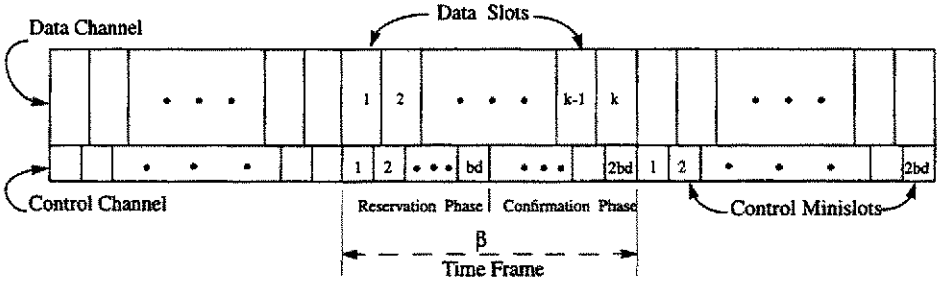

Fig. 1. Control information is transmitted over a link by using a dedicated fraction γ of the capacity of the link. Each control frame is subdivided into control minislots as explained in the text. During a frame, k data packets can be transmitted along the data wires using the remaining $(1-\gamma)C$ of the capacity.

on the diameter of the network and the length of the data and the control packets.

The remainder of the paper is organized as follows. In Section 2 we discuss the DSC protocol and the data structures needed to implement it in a general network topology. We also derive bounds on the packet delay under light load conditions, and obtain estimates for the overhead associated with reservations. In Section 3 we describe a particular implementation of the DSC protocol for a hypercube network of processors, evaluate its throughput through both analysis and simulations, and provide performance comparisons with other switching formats. Finally, in Section 4 we present our conclusions.

2. Description of the DSC Protocol

In this section we present the DSC protocol for a general network topology, and we describe the data structures necessary for its implementation. The wires of a link that are dedicated to the transmission of data (or control) information form the *data* (or *control*, respectively) *channel* of the link. We let γ be the fraction of wires of a link that are used for the transmission of control information. The time axis in the control channel is divided into *control frames* of duration β (see Fig. 1), where β is a parameter that will be specified shortly. We assume that the network is synchronized so that frames start synchronously at all nodes, and that all packets have equal length, and they require one data slot in order to be transmitted over the data channel of a link. The duration of a data slot is equal to

$$\frac{L}{(1-\gamma)C} + t_m, \tag{1}$$

where L is the length of the packet in bits, t_m is an upper bound on the propagation and processing delay of a packet, and $(1-\gamma)C$ is the capacity dedicated to the transmission of data, measured in bits/sec. The duration β of a control frame is assumed to be equal to an integer number k of data slots. We also assume that a routing algorithm exists that selects a path for each source-destination pair, and provides a way to resolve conflicts.

A packet enters the network only after having reserved all the resources (links and buffer space) along its route. These reservations are made through the transmission of special control packets, called *flow control units* (or *flits*[a]), over the control channel. A flit requires one minislot in order to be transmitted over a link, with the duration of a minislot being equal to

$$\frac{F}{\gamma C} + t_f,$$

where F is the length of the flit in bits, and t_f is an upper bound on the propagation and processing delay of a flit on a link.

A control frame is divided into two phases, which will be referred to as the *reservation phase* and the *confirmation phase.* During the reservation phase, a flit is sent towards the destination in order to reserve the required links and buffer space. As we will explain shortly, the reservation of a resource is made only for the data slots during which the resource will be used, and it is available for other packets for the remaining of the time. A flit that has been successful in making the appropriate reservations is sent, during the confirmation phase, on the reverse path from the destination to the source, in order to notify the source and confirm the reservations made at the intermediate nodes. A flit that was blocked at an intermediate node, because it could not make all necessary reservations, may be transmitted during the confirmation phase from the intermediate node to the source, in order to notify the source of this failure, and release any resources (links and buffer space) it may have reserved during the reservation phase; alternatively, blocked flits could be dropped, and resources whose reservations are not confirmed are automatically released.

Flits can start transmission from a source only during the first control minislot of a control frame, and, as we will see, acknowledgements are guaranteed to return to the source by the end of that frame. A flit generated at a source follows the path determined by the underlying routing algorithm of the network. Reservations for links and buffer spaces are made for data slots that start after the termination of the control frame in which the flit is transmitted. To make sure that a (positive or negative) acknowledgement returns to the source by the end of the frame in which the flit is transmitted, we request that the duration β of a frame is (greater than or) equal to $2bd$ minislots, where d is the diameter of the network, and b is the buffer size (measured in packets) per link. The durations of the reservation and the confirmation phase are both chosen to be equal to $\beta/2$. To see that the reservation phase is completed in time $\beta/2$, note that if a flit is not blocked, it can be delayed by at most $b-1$ other flits on a given link during the reservation phase. This is because if more than b flits request a link, at most b of them are allowed to proceed, and the remaining are blocked. Therefore, after time $\beta/2$, all flits have either arrived at their destinations or they have been blocked. To ensure that the confirmation phase is also completed in time $\beta/2$, we require that if a flit is transmitted on a link during minislot $i, i = 1, 2, \ldots, bd$, of the reservation phase, it is transmitted on the same link (but in the reverse direction) during

[a]Note that our definition of a flit differs from that given in Ref. [2].

minislot $2bd - i + 1$ of the subsequent confirmation phase. This ensures that flits do not collide during the confirmation phase, and feedback information arrives at the source reliably. Flits that are blocked during the reservation phase return to their source carrying negative acknowledgements (NACKs), while those that reach their destination return carrying positive acknowledgements (ACKs). A packet for which a NACK is received is not allowed to enter the network, and has to reattempt to make a reservation in one of the following frames.

The number of data slots k contained in a control frame is an important design parameter and is related to β by the relation

$$\beta = 2bd\left(\frac{F}{\gamma C} + t_f\right) = k\left(\frac{L}{(1-\gamma)C} + t_m\right). \tag{2}$$

Solving with respect to γ, we get

$$\gamma = \frac{(1+u+v) - \sqrt{(1+u+v)^2 - 4uv}}{2u},$$

where

$$u = \frac{C}{Lk}(kt_m - 2bdt_f),$$

and

$$v = \frac{2bdF}{Lk}.$$

Assuming that the difference between $2bdt_f$ and kt_m is negligible (this happens, for example, when $t_m, t_f \approx 0$), the previous equations simplify to

$$\gamma = \frac{1}{1 + \frac{Lk}{2bdF}}, \tag{3}$$

and

$$\beta = \frac{Lk + 2bdF}{C}. \tag{4}$$

Equation (3) gives the relationship between the fraction of the capacity γ dedicated to the transmission of control information and the number k of data slots contained in a frame. In the remainder of the paper we use the notation DSC(k) to refer to a DSC protocol where the duration of control frame is equal to the time required to transmit k data packets.

The mechanism according to which reservations are made, the data structures required at the links, and the information that a flit has to carry are described next. Each link of the network is assumed to have a *link buffer* and an *entry buffer* (see also Fig. 2, for an implementation of the DSC protocol in a hypercube network). The entry buffer is used by packets that have not yet entered the network, while the link buffer is used by packets that have already been accepted in the network. A packet in an entry buffer sends out a flit, and, if it receives an ACK at the end of the frame, it enters the network at the beginning of the next frame by moving from the entry buffer to the corresponding link buffer. For every link queue $\mathcal{Q}_l$, there is a list $\mathcal{L}_l$, called *reservation list*, whose elements represent future data slots.

In particular, the t^{th} element of the list represents the t^{th} data slot *relative* to the start of the current frame. At the end of k data slots (or, equivalently, one control frame), the first k elements of $\mathcal{L}_l$ are deleted and the reservation list is updated. The element $\mathcal{L}_l(t)$ that corresponds to data slot t consists of two fields, denoted by $\mathcal{L}_l(t)\_link$ and $\mathcal{L}_l(t)\_buffer$. The field $\mathcal{L}_l(t)\_link$ is equal to one, if link l has already been reserved for data slot t, and is equal to zero, otherwise. The field $\mathcal{L}_l(t)\_buffer$ takes integer values between zero and the buffer size b, and it is equal to the number of buffer spaces of $\mathcal{Q}_l$ that have already been reserved for data slot t.

Each flit f carries with it a counter c_f, which signifies the data slot (relative to the start of the current control frame) for which the next link is requested. The counter c_f is initially set to $k+1$. This is because a packet enters the network only after receiving a positive confirmation, which happens at the end of the frame during which the flit is sent. Therefore, the first link on the path should be requested for the first data slot that starts after the current frame is completed, which is data slot $k+1$. Flits arriving at a node are processed in the order they arrive. If more than one flits arrive at a node during the same minislot, the order in which they are processed is determined in an arbitrary way (or in some way specified by the underlying routing/priority scheme).

We let c_f be value of the counter of flit f upon its arrival at link l, and $\mathcal{L}_l$ be the reservation list at the time flit f is processed at l. We also define T as the minimum integer satisfying

- $c_f \leq T$,
- $\mathcal{L}_l(T)\_link = 0$,
- $\mathcal{L}_l(t)\_buffer \leq b, \qquad$ for all $t \in \{c_f, c_f+1, \ldots, T-1\}$.

In other words, T is the first data slot, following the data slot for which the link is requested, for which no reservation has been made yet, and enough buffer space is available at the link to store the packet between its scheduled arrival and its departure from the link. If a T that satisfies the relations above does not exist, the flit f is blocked (the reservation fails). If, however, such a T exists, a reservation is made on l for data slot T, and the flit counter and the reservation list are updated according to

- $c_f = T+1$,
- $\mathcal{L}_l(T)\_link = 1$
- $\mathcal{L}_l(t)\_buffer = \mathcal{L}_l(t)\_buffer + 1, \qquad$ for all $t \in \{c_f, c_f+1, \ldots, T-1\}$.

Note that buffer space is reserved for the packet at link l starting at data slot c_f (this is when the packet is scheduled to arrive at l) and ending at data slot T (this is the slot for which reservation on l is made for the packet). During the confirmation phase, all reservations made by flits that were eventually blocked are cancelled, and the corresponding resources are released. This can happen either

explicitly (flits carrying NACKs follow the reverse path to the source freeing the links and buffer space), or implicitly (a resource is released if the reservation is not confirmed by the end of the frame during which it was made). A packet for which an ACK is received at the end of a frame, enters the network at the beginning of the next frame, and is forwarded to its destination using the links and buffer space that were reserved for it. If we also record (in a separate field in list $\mathcal{L}_l$ the slot and incoming link over which the packet is expected to arrive) then, when the packet is transmitted, it does not have to carry any information about its destination, and it can be routed based solely on the information kept locally in the list. This reduces the length of the packets, since no routing information has to be included in its header. Therefore, the only additional control overhead required by the DSC protocol over packet switching is the counter c_f; the routing information carried by a flit does not have to be included in the packet itself when it is transmitted. Note also that in the unbuffered case, where there is buffer space only for the packet being transmitted, the list elements consist of only one field (the fields $\mathcal{L}_l(T)\_link$ and $\mathcal{L}_l(t)\_buffer$ are then identical).

We next evaluate the worst case delay incurred by a packet under light load network conditions. This delay does not include queueing delays at the entry buffers or delays due to retrials, both of which are small under light load conditions. A newly generated packet first has to wait for at most β (and on the average, $\beta/2$) time units for the current frame to end, and for another β time units for the appropriate reservations to be made and for an acknowledgement to return to the source. Following the receipt of an acknowledgement, another d data slots are required for the packet to arrive at its destination (under light load conditions all links requested by a link are available and are reserved for consecutive data slots). Thus, the worst case packet delay under light-load conditions satisfies

$$D(\text{light load}) \leq \left(\frac{L}{(1-\gamma)C} + t_m \right) \cdot (2k + d), \tag{5}$$

where we have used Eq. (2).

Note that when the parameter k is increased, the light load delay increases, but the fraction of capacity that is available for the transmission of data

$$1 - \gamma = \frac{1}{1 + \frac{2bdF}{Lk}}$$

(and, therefore, the maximum possible throughput) also increases. In Section 3, we examine in more detail the tradeoffs involved, and we obtain methods to optimally choose the values of these parameters so as to maximize throughput.

3. The DSC(k) Protocol on the Hypercube

In this section we describe and analyze a particular implementation of the DSC protocol in a hypercube parallel computer. We start by describing the node model and the routing algorithm assumed. We then present an approximate analysis for the throughput, and compare it to simulation results. We also obtain results on the

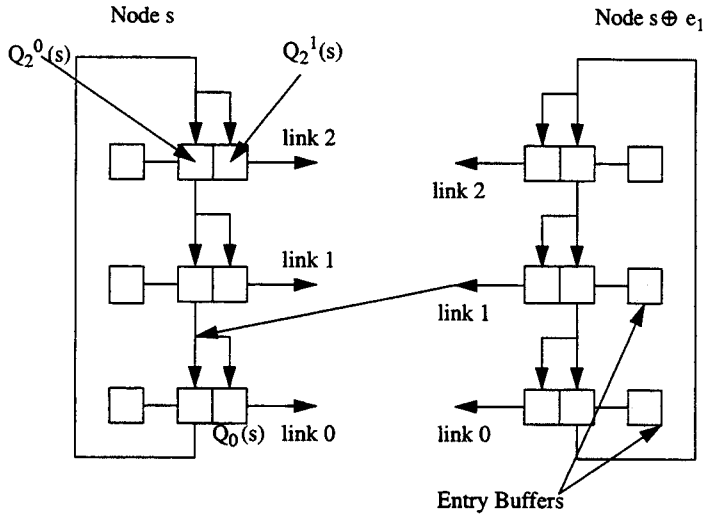

Fig. 2. Illustrates two neighboring descending-dimension switches in a 3-dimensional hypercube.

fraction of capacity that needs to be dedicated to the control channel to optimize the throughput.

3.1. Node Model and Implementation of the DSC(k) Protocol for the Hypercube

Each node of a $N = 2^d$-node hypercube is represented by a unique $d-$bit binary string, and the hypercube links connect nodes whose binary representations differ in exactly 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 let $e_i, i = 0, 1, \ldots, d-1$, be the binary string whose i^{th} 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 dimension i (or link i). Each link i of a node s has an entry buffer $\mathcal{E}_i(s)$, which can store up to one packet originating at that link, and a queue $\mathcal{Q}_i(s)$, which is used to store transit packets. The entry buffer can accept a new packet only if the previous packet has reserved its entire path to the destination, and a positive feedback has been received. An entry buffer holding a packet for which no feedback has been received is said to be *backlogged.* New packets arriving at backlogged entry buffers are discarded.

The queue $\mathcal{Q}_i(s)$ is composed of two buffers, denoted by $\mathcal{Q}_i^0(s)$ and $\mathcal{Q}_i^1(s)$, each of which can hold one packet. The first buffer $\mathcal{Q}_i^1(s)$ is called the *forward* buffer and is used by packets that need to cross the i^{th} dimension, that is, packets that have the i^{th} bit of their routing tag equal to one. The second buffer $\mathcal{Q}_i^0(s)$ is called the *internal* buffer and is used by packets that do not have to cross the i^{th} dimension. The internal buffer $\mathcal{Q}_i^0(s)$ is connected to 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 $\mathcal{Q}_{(i-1) \bmod d}(s)$ of the neighbour node $s \oplus e_i$ (see Fig. 2). This router architecture is called a *descending-dimensions*

switch, and it leads to switches that are simpler, faster, and less expensive than crossbar switches (see Ref. [2] and Ref. [10]).

In the routing algorithm that we assume, a flit traverses the hypercube dimensions in descending order starting with a random dimension l. When a flit that arrives at queue $Q_i(s)$ of node s, the i^{th} bit of its routing tag is checked. Depending on whether it is equal to one or zero, the flit claims buffer $Q_i^1(s)$ in order to be transmitted to queue $Q_{(i-1) \bmod d}(s \oplus e_i)$ of the neighbour node $s \oplus e_i$, or it claims buffer $Q_i^0(s)$ in order to be internally passed to the next link queue $Q_{(i-1) \bmod d}(s)$ of the same node. Since we assume that there is buffer space only for the packet being transmitted (that is, $b = 1$), flits that find a link already reserved are blocked. When two flits attempt to make a reservation for the same (available) data slot during the same control frame, one of them is selected at random and the other is blocked. The number of minislots in the reservation phase or the confirmation phase is equal to d, and the parameter β is equal to the time required by a flit to travel a distance of $2d$ links.

Flits that are successful in reserving all the links on the path to their destination, return during the confirmation phase to their origin, following the reverse path than the one followed in the reservation phase, and carrying a positive acknowledgement (ACK). A flit is transmitted over a link during control minislot $2d - i - 1, i = 0, 1, \ldots, d - 1$, of the confirmation phase if it was transmitted over the same link during control minislot i of the reservation phase; this guarantees that there are no collisions among flits during the confirmation phase. Flits that fail to make all the required reservations are discarded, and all unconfirmed reservations are cancelled at the end of a confirmation phase. Note that for the hypercube network and this particular implementation of the DSC protocol, each node needs to store at most two flits per link at any given time.

3.2. Throughput Analysis of the DSC(k) Protocol for the Hypercube

In this section we present an analysis of the DSC(k) protocol for a d–dimensional hypercube. We assume that the descending-dimension switches and the routing algorithm described in Section 3.1 are used. We also assume that d is a multiple of k, and, in particular, $d = kr$.

It is possible for a flit to reserve a link and release it later, during the same control frame, due to its failure to reserve the remainder of the path. We refer to such a reservation as a *ghost reservation*, as opposed to a *confirmed reservation*, where a flit is successful in reserving all the links on its path. We define $p(t, i, l)$ as the probability that during control frame t, a reservation (ghost or confirmed) is made for link l for the i^{th} data slot following the end of frame t, where $i = 1, 2, \ldots, d$. Assuming that the system eventually reaches steady state, the following limit exists and is independent of the link l

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

Thus, p_i is the steady-state probability, that, in a given control frame, a link is reserved for the i^{th} data slot following the end of that frame. We also denote by p_0

that the entry buffer of a link is nonempty immediately prior to the beginning of a new control frame (i.e., a new or a retrial packet is available to enter the network at that particular link); we call p_0 the *attempt probability*. Since a flit is successful only after having reserved all the d links on its path, the average number of successful reservations made per link per frame is equal to p_d. Therefore the steady-state throughput per node *per frame* is equal to $2dp_d$, and the throughput R per node *per data slot* is

$$R = 2dp_d/k.$$

We will evaluate p_d and then R by obtaining a recursive relationship on the steady-state probabilities p_i, $i = 1, 2, \ldots, d$.

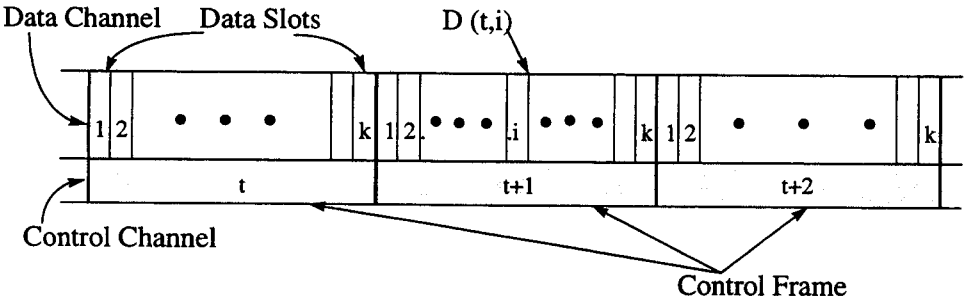

Fig. 3. The i^{th} data slot following the end of frame t is denoted by $D(t, i)$. Note that with this notation, $D(t, k+i) = D(t+1, i)$.

We denote by $D(t, i)$ the i^{th} data slot that follows the end of frame t (see Fig. 3). For a given link l, we let l_1 and l_2 be the internal and forward links, respectively, that lead to l (see Fig. 4). Link l may be reserved either by a flit f_1 arriving over link l_1, or by a flit f_2 arriving over link l_2. In particular, link l can be reserved during frame t for data slot $D(t, i)$ by a flit f_1 arriving over l_1 if:

- a reservation has been made by f_1 for link l_1 for the previous data slot $D(t, i-1)$; we call this event B,
- link l lies on the path of f_1 (given that l_1 lies on its path); this happens with probability $1/2$, and
- no confirmed reservation has been made for link l for data slot $D(t, i)$ during previous control frames, and no reservation (confirmed or unconfirmed) has been made for $D(t, i)$ during the current control frame t by any flit arriving over link l_2.

Thus, for $1 < i \leq d$, we have

$$p(t, i, l) = \frac{2 \times p(t, i-1, l_1)}{2}(1 - P(A/B)), \tag{6}$$

where A is the event that a confirmed reservation has been made on link l for data slot $D(t, i)$ during one of the previous frames $t-j, j = 1, 2, \ldots, r-1$, or a reservation has been made for $D(t, i)$ during the current frame t by a flit arriving on link l_2.

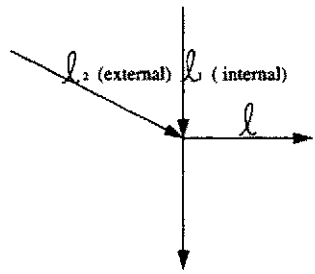

Fig. 4. Internal and forward link leading to link l.

The factor of 2 in Eq. (6) accounts for the fact that link l could be reserved by a flit arriving on link l_2, or by a flit arriving on link l_1 (the two cases are symmetric). The factor $1/2$ accounts for the probability that link l lies on the path of f_1, given that l_1 lies on its path.

The probability that a flit f_2 requests link l during frame t for data slot $D(t,i)$ and is granted the link can be approximated by

$$\frac{1}{4} \cdot p(t, i-1, l_2). \tag{7}$$

This is because for flit f_2 to reserve link l for data slot $D(t,i)$, it should have already reserved link l_2 for data slot $D(t,i-1)$. The factor of $1/4$ is the probability that link l lies on the path of flit f_2 (given that l_2 lies on its path), and f_2 wins over flit f_1 when the conflict arises.

Since there is an attempt by flit f_1 arriving over l_1 to reserve link l for data slot $D(t,i)$, any confirmed reservations that have been made on link l for data slot $D(t,i)$ should have been made by flits arriving over link l_2. The number n of control frames during which reservation can be made for $D(t,i)$ is given by

$$n = \begin{cases} r-1-(i/k) & \text{if } i \bmod k \neq 0 \\ r-(i/k) & \text{if } i \bmod k = 0 \end{cases}.$$

Hence, the probability that a confirmed reservation for $D(t,i)$ was made during a previous frame by a flit f_2 arriving over link l_2 is given by

$$\frac{1}{2}\sum_{j=1}^{n} p(t-j, i+jk-1, l_2) \cdot p(\text{flit } f_2 \text{ reserves } l \text{ given it requests it}) \frac{p(t-j,d)}{p(t-j,i+kj,l)}, \tag{8}$$

where the factor of $\frac{1}{2}$ is the probability that f_2 chooses link l, and the ratio

$$\frac{p(t-j,d)}{p(t-j,i+kj,l)}$$

is the probability that the reservation made by f_2 was finally confirmed. To evaluate the probability that f_2 reserved link l during frame $t-j$ given that it requested it, note that f_2 would only lose out to a flit arriving over link l_1 that made a *ghost* reservation during frame $t-j$. This is because we know that a reservation is

requested by flit f_1 for data slot $D(t,i)$, and, therefore, any flit that had reserved data slot $D(t,i)$ during frame $t-j$ coming from link l_1, should have eventually been blocked (that is, such a reservation should have been a ghost reservation). Thus,

$$p \text{ (flit } f_2 \text{ reserves } l \text{ given it requests it)} = 1 - \frac{1}{4}p(t-j, i+jk-1, l_1)\left(1 - \frac{p(t-j,d)}{p(t-j,i+kj,l)}\right), \tag{9}$$

where $\frac{1}{4}$ accounts for the probability that a flit arriving over link l_1 choses link l and wins over a flit arriving over l_2, and the term

$$\left(1 - \frac{p(t-j,d)}{p(t-j,i+kj,l)}\right)$$

is the probability that such a reservation is not finally confirmed.

Taking the limit $t \to \infty$ and using the symmetry with respect to the links (all steady-state probabilities are independent of the links), Eqs. (6)-(9) give

$$p_i = p_{i-1}\left(1 - \frac{p_{i-1}}{4} - \frac{1}{2}\sum_{j=1}^{n}\left[p_{i+jk-1}\frac{p_d}{p_{i+kj}}\left(1 - \frac{p_{i+jk-1}}{4}\left(1 - \frac{p_d}{p_{i+jk}}\right)\right)\right]\right) \quad \text{for } i = 2,3,\ldots,d. \tag{10}$$

By solving Eq. (10) with respect to p_{i-1}, and keeping only the solution that corresponds to $p_{i-1} < 1$, we obtain

$$p_{i-1} = s_i - \sqrt{s_i^2 - 4p_i} \qquad \text{for } i = 2,3,\ldots,d, \tag{11}$$

where

$$s_i = 2 - p_d\sum_{j=1}^{n}\left[\frac{p_{i+jk-1}}{p_{i+kj}}\left(1 - \frac{p_{i+jk-1}}{4}\left(1 - \frac{p_d}{p_{i+jk}}\right)\right)\right]. \tag{12}$$

In order to relate p_1 with p_0, we observe that a link l may be reserved for the first data slot of a given frame during one of the $r-1$ frames that precede it. During each of these frames, a confirmed reservation on l for that data slot is made with probability p_d. Since confirmed reservations made for the same link and data slots are disjoint events, the probability that the link is unreserved is $1-(r-1)p_d$, and we have

$$p_1 = p_0(1-(r-1)p_d). \tag{13}$$

Note that Eq. (11) gives p_{i-1} in terms of p_i and s_i (which itself depends only on p_j's with $j \geq i$). Therefore, using Eqs. (11)-(13) it is possible to compute the throughput for the DSC(k) protocol by running a backward recursion. This can be done by choosing a particular value of p_d and then successively obtaining $p_{d-1}, p_{d-2}, \ldots, p_0$, using the above equations. A curve can then be plotted for the average throughput $R = 2dp_d/k$ per node per data slot as a function of the attempt rate p_0.

3.3. Analytical and Simulation Results

In this section we present the analytical results obtained for the throughput of the DSC protocol using the analysis of Section 3.2, and compare these results with corresponding simulation results. We also evaluate graphically the fraction of the capacity that should be dedicated to the transmission of control information in order to maximize throughput. Simulations results are given only for hypercube dimensions $d \leq 8$, since simulations for $d > 8$ are too intensive computationally. We do, however, present results for dimensions $d > 8$ using the analytical method.

Fig. 5 compares the analytical and simulation results obtained for the average

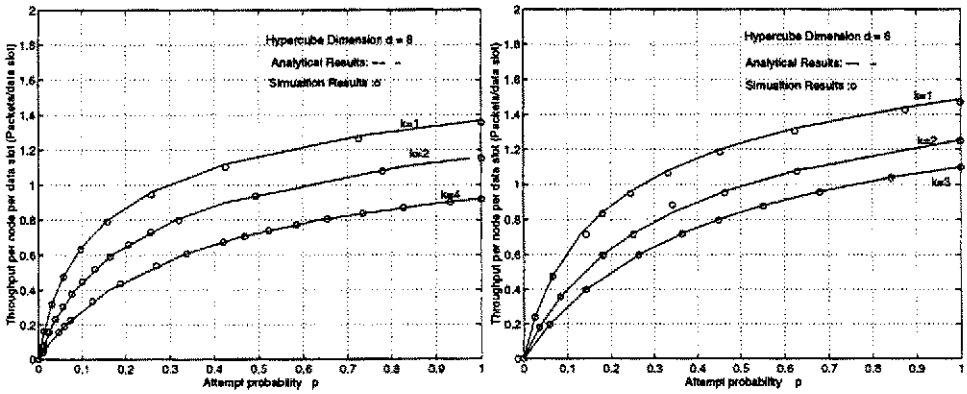

Fig. 5. (a) Analytical and simulation results for the average throughput $2dp_d/k$ per node per data slot of the hypercube implementation of the DSC(1), DSC(2) and DSC(4) protocols for dimension $d = 8$. (b) Analytical and simulation results for the average throughput of the hypercube implementation of the DSC(1), DSC(2) and DSC(3) protocols for dimension $d = 6$. Note that the duration of a data slot is not the same for the different versions of the DSC(k) protocol, since it depends on the value of k (see also Fig. 6).

throughput per node and data slot of the DSC(k) protocol on hypercubes of dimension 6 and 8 for several values of k, assuming that there is no buffer space at the nodes, except for the packets currently under transmission. The agreement between analytical and simulation results is excellent; the difference is consistently smaller than 2%. Note that the maximum throughput that can be achieved in a hypercube for uniform distribution of the destinations assuming infinite buffer space per node is at most equal to 2 packets per node and data slot.

Since (by Eqs. (1) and (2)) the duration of a data slot is different for different values of k, the results given in Fig. 5 are not directly comparable. This is because they do not take into account the control overhead, which is larger for small values of k. To compare the performance of the DSC(k) protocol for different values of k, we define the *normalized throughput* as the percentage of the total capacity that is efficiently used (capacity is not used efficiently if it is used to transmit control bits, or if it stays idle). In Fig. 6(a) we plot the normalized throughput of the DSC protocol for a hypercube of dimension 8 and several values of k, as a function of the ratio $\frac{F}{L}$ of the length F of a flit over the length L of a data packet. In Fig. 6(b) we also plot the fraction γ of the capacity allocated to the transmission of control

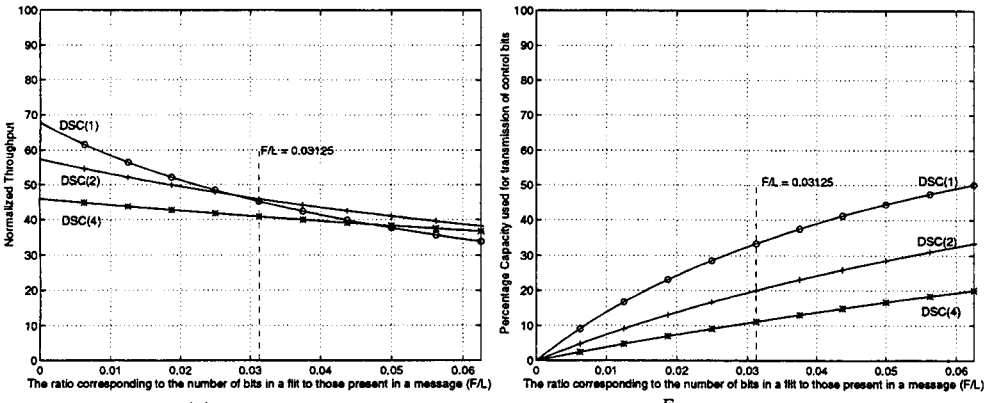

Fig. 6. (a) The normalized throughput as a function of $\frac{F}{L}$ for a 8−dimensional hypercube and several values of k (or equivalently γ). (b) We also illustrate the variation of the fraction γ of the capacity dedicated to the control channel, obtained by Eq. (2) as a function of $\frac{F}{L}$ and k.

information, given by Eq. (3), as a function of $\frac{F}{L}$ for several values of k.

Given the values of the parameter $\frac{F}{L}$ and the hypercube dimension d, it is possible to find the value of k (or equivalently, the fraction γ of capacity to be allocated to the control channel) that maximizes the normalized throughput. For example, for hypercube dimension $d = 8$, flit size $F = 64$, and packet size $L = 2048$ (in bits), Fig. 6(a) shows that the normalized throughput is maximized when $k = 2$ (for these values of F and L, we have $\frac{F}{L} = 0.03125$). From Fig. 6(b) we see that this corresponds to $\gamma = 0.2$, which implies that in the VLSI design we should allocate 1 in every 5 wires of a link to the transmission of control information. In general, for different values of the parameters F, L and d, a different version of the DSC(k) protocol maximizes the normalized throughput. For example, if $L = 2048$ and $d = 8$, the DSC(2) protocol is optimal for $F > 57$ bits, while the DSC(1) protocol is optimal for $F < 57$ bits. The larger the length F of the flits is (or the smaller the length L of the packets), the more advantageous it becomes to use larger values of k.

In Fig. 7, we illustrate the normalized throughput of the DSC(k) protocol for different values of k as a function of the dimension d of the hypercube. Note that for a fixed k, the normalized throughput per node decreases when the dimension d increases. This is mainly due to the descending-dimension switches used, and has also been observed in other works.[2,10] Note that for hypercubes of larger dimensions, larger values of k are preferable. For example, when $F = 64$ and $L = 1600$, protocol DSC(1) is better for $d = 4$, DSC(2) is better for $d = 8$ and 12, and DSC(4) is better for $d \geq 16$. In general we expect that the larger the diameter of a network is, the larger the parameter k should be in order to obtain the optimal balance between the capacity allocated to the control channel and the capacity available for the transmission of data.

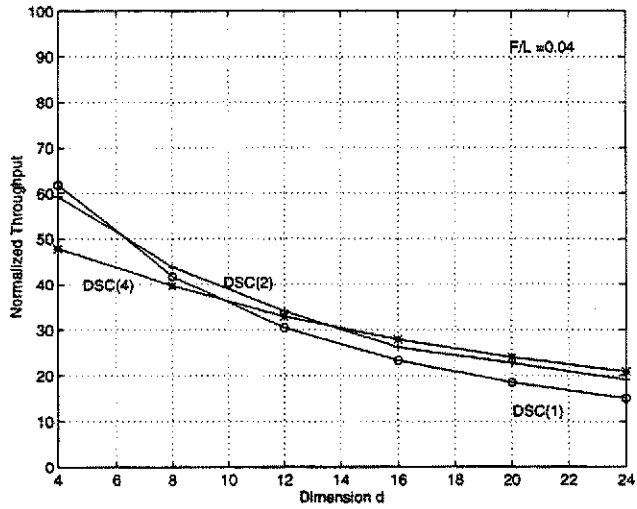

Fig. 7. Illustrates the normalized throughput of the DSC(k) protocol for different values of the parameter k as a function of the dimension d of the hypercube, for $F = 64$ and $L = 1600$.

3.4. Comparison with Packet Switching and the CSR Protocol

Since the CSR protocol[11] can be viewed as the time division multiplexed version of DSC(1), its performance is inferior to that of the (optimized) DSC protocol, when the diameter of the network is large, or when the ratio F/L is large. In Fig. 8 we also compare the throughput of the DSC(k) protocols with that of a simple packet switching scheme, analyzed in Ref. [10]. In the packet switching scheme, packets are transmitted without making any reservations, and they may be dropped at intermediate nodes due to buffer overflow. As illustrated in Fig. 8, the DSC(k) protocols outperforms the simple packet switching scheme for hypercube dimensions $d > 6$ (the figure has been drawn assuming $F/L = 0.04$). Note that the results presented in Fig. 8 for the DSC and the packet switching scheme are directly comparable, since they both use the same switch architecture and routing algorithm, and they assume the same buffer space per node. We believe that these results are indicative of the performance improvement that can be obtained by superimposing the DSC protocol on other routing algorithms, and under different assumptions on the topology and switch architecture.

4. Conclusions

We have developed a new dynamic scheduling communication protocol that is easily amenable to a VLSI implementation for any multiprocessor network. The DSC protocol and the data structures required to implement it were initially presented in their generality, without assuming a particular network topology, routing algorithm, or buffer size. We then specialized the DSC protocol to the case of a hypercube multicomputer, and computed its throughput using analysis and simulation. The analytical results closely match the simulation results obtained. For

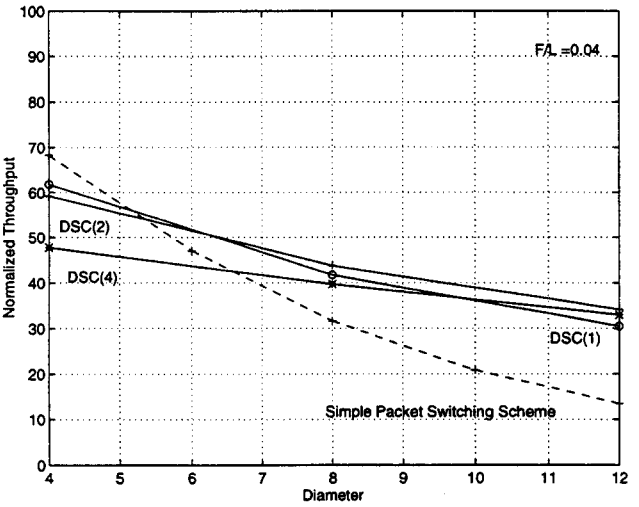

Fig. 8. Illustrates (a) the normalized throughput of the DSC(k) protocol for different values of the parameter k as a function of the dimension d of the hypercube, for $F = 64$ and $L = 1600$, (b) the throughput obtained by the simple packet switching scheme .

the hypercube topology, our results indicate that the DSC protocol outperforms the corresponding packet switching scheme (for the same node architecture and routing algorithm) when the size of the network is sufficiently large. We also provided a graphical method for obtaining the fraction of the capacity that should be dedicated to the control channel to maximize the throughput.

The DSC protocol is actually a family of protocols parameterized by an integer k, the optimal choice of which depends on the flit lengths, the packet lengths, and the diameter of the network. For the hypercube implementation of the DSC protocol, we used the analytical results on the throughput to evaluate the optimal choice of k, for different values of the above quantities. The DSC protocol compares favorably to other communication schemes (including packet switching, circuit switching, wormhole routing, and the CSR protocol), and we believe that it is a promising communication protocol for multiprocessor computers.

References

1. Broomell, G., and Heath, J. R., "Classification categories and historical development of circuit switching topologies", *Computing Surveys*, (June 1983) vol. 15, no. 2, pp. 95-133.
2. Dally, W. J., "Network and Processor Architecture for Message-Driven Computers," in *VLSI and Parallel Computation*, R. Suaya, and G. Birtwistle (Eds.), (Morgan Kaufmann Publishers, San Mateo, CA, 1990) pp. 140-222.
3. Greenberg, A. G., and Goodman J., "Sharp Approximate Models of Adaptive Routing in Mesh Networks," *Teletraffic Analysis and Computer Performance Evaluation* (Elsevier, Amsterdam, 1986, revised 1988), pp. 255-270.
4. Greenberg, A. G., and Hajek, B., "Deflection Routing in Hypercube Networks," *IEEE Transactions on Communications*, (June 1992) vol. COM-35, no. 6, pp. 1070-

1081.

5. Grunwald, D. C., and Reed, D. A., "Analysis of backtracking routing in binary hypercube Computers," Research Report UIUCDCS-R-89-1486, Univ. of Illinois, Urbana-Champaign, Dept. of Computer Science, February 1989.
6. Kelly, F. P., "Blocking Probabilities in large circuit switched networks," *Adv. Appl. Prob.*, (1986) vol. 18, pp. 473-505.
7. Krishna A., "Communication with Few Buffers: Analysis and Design," Ph.D. Thesis, Department of ECE, University of Illinois at Urbana-Champaign, December 1990.
8. Maxemchuk, N. F., "Problems Arising from Deflection Routing: Live-lock, Lock-out, Congestion and Message Reassembly," *Proceedings of NATO Workshop on Architecture and High Performance Issues of High Capacity Local and Metropolitan Area Networks*, (France, June 1990).
9. Sharma, V., and Varvarigos, E. A., "Circuit switching with input queueing: An analysis for the $d-$dimensional wraparound mesh and the hypercube," *IEEE Transactions on Parallel and Distributed Systems*, (April 1997) vol. 8, no. 4, pp. 349-366.
10. Varvarigos, E. A., and Bertsekas, D.P., "Performance of Hypercube Routing Schemes With or Without Buffering," *IEEE/ACM Transactions on Networking*, (June 1994) vol. 2, no. 3, pp. 299-311.
11. Varvarigos, E. A., and Bertsekas, D. P., "A Conflict Sense routing Protocol and its Performance for Hypercubes," *IEEE Transactions on Computers*, (June 1996) vol. 45, no. 6, pp. 693-706.