Message Scheduling on a Wormhole-Switched Linear Client-Server Network

Bing Yang†, Ashwin Gumaste‡, Enyue Lu⋆ and S. Q. Zheng§

† Cisco Systems, 2200 East President George Bush Highway, Richardson, TX 75083-0688, USA
‡ School of Information Technology, Indian Institute of Technology, Bombay, India
⋆ Mathematics and Computer Science Department, Salisbury University, Salisbury, MD 21801, USA
§ Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083-0688, USA

***Abstract*—The advantage of wormhole switching in interconnection networks is its distance insensitivity of communication latency under light traffic. However, this property vanishes when traffic is heavy. We consider the performance of a linear wormhole-switched network used as a real-time client-server network. Messages generated by client hosts are periodically transmitted to a central server within a predicatable delivery time. We present two algorithms for generating feasible message transmission schedules and compare their performances. It is shown that trade-off exists between quality of schedules and the network utilization. Several open problems are posed.**

***Index Terms*—interconnection network, real-time application, wormhole switching, pipelining, client-server computing, scheduling, performance evaluation**

I. Introduction

Wormhole switching [5], [6] is an efficient switching method for communication in interconnection networks of multiprocessor parallel computing systems. For light traffic, the latency of wormhole-switched communication is distance insensitive due to tight but distributed synchronization of pipelined transmission of small data units (called *flow control units* or simply *flits*). Extensive research has been conducted on various aspects of wormhole switching. Several wormhole-switched interconnection networks have been implemented. These include Intel Cavallino [4], Network Design Frame [8], Cray T3D [14] and T3E [13], [15], Reliable Router [7], SGI SPIDER [11], Ariadne [2], and IBM SP2 [16]. These interconnection networks have regular topologies such as k-ary n-cubes (including tori and hypercubes) and meshes. Wormhole-switched routers and switches for constructing networks of workstations (NOWs) and system area networks (SANs) of arbitrary topologies, such as Myrinet [3] and ServerNet [12], have been also made available.

However, wormhole switching, as any other packet/cell switching methods, manifests of unpredictability under heavy traffic. One major problem is the possibility of costly deadlock. Various deadlock avoidance and prevention techniques based on virtual channels have been proposed and implemented [9]. Deadlock detection and recovery methods have also been proposed (e.g. [1]). Another problem is the unpredictable performance under heavy traffic due to non-deadlock blocking. This problem has not been rigorously investigated, and it is the subject addressed by this paper.

We consider a wormhole-switched linear network used in a real-time application domain. We choose a linear interconnection structure for the reason that it is the simplest structure and the results obtained may be generalized to dealing with more complicated structures. Also, interconnection networks of regular structures contain multiple linear substructures and the results obtained for linear network may be directly applied to such subnetworks. The application domain under consideration is real-time client-server computing with periodic client requests. We study the relationship among client message length, m period, and deliver time, and client locations. We show that under heavy traffic, client message periods and deliver times are message length sensitive and distance sensitive. That is, message periods and deliver times depend on message lengths and the locations where messages are originated. We provide two message scheduling algorithms and compare their performances. The first has a greedy feature and tends to have high network utilization, and larger message periods and deliver times. The second algorithm is less aggressive, resulting lower network utilization but much smaller message periods and deliver times. Our results have two implications. First, for some real-time applications suitable message schedules can be obtained by careful design. Second, wormhole-switched networks may have severe limitations for certain real-time applications. We conclude that, for real-time applications with heavy periodic traffic, higher network utilization does not imply better performance, and message communication must be carefully scheduled to achieve desired performance.

This paper is organized as follows. In the next section, we introduce the wormhole-switched linear client-server network, its operation mode and parameters, and some definitions to be used in the analysis. In Section III, we present a simple message schedule called greedy schedule. In Section IV, we present an improved message schedule called conservative schedule. In Section V, we compare the performances of the two schedules, explain the implications of this comparison, and point out related open problems.

II. Wormhole-Switched Linear Client-Server Network

A linear whormhole-switched client-server network is a network consisting of N client hosts H_i, $1 \leq i \leq N$, that are connected as a linear array. Each H_i has an interface connecting it to the input I_i^1 of a wormhole-switched 2×1 switch S_i. The output of S_i, denoted by O_i, is connected to the input I_{i-1}^2 of S_{i-1}. The output O_1 of S_1, which is considered as the network output, is connected to the central server. Figure 1 shows the configuration of this structure. A client message sent from H_i to the network output has to traverse switches $S_i, S_{i-1}, \cdots, S_1$ to reach the output. We assume that all switches have a small buffer space for one flit.

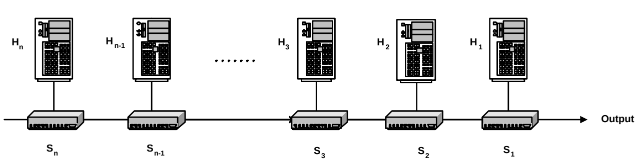

Fig. 1. A simple network

Messages sent from clients to the server (network output) are real-time periodic messages. A periodic message from H_i is denoted as $M_i = \{p_i, e_i, d_i\}$, where p_i, e_i and d_i are the period, message delay time at any switch on the way to the network output, and the sufficient message deliver time (an upper bound) for the message to reach the network output starting from the time its first bit is injected into the network to the time its last bit reaches the network output. All messages from H_i have the same p_i, e_i and d_i. By the nature of wormhole switching, e_i is proportional to the length of M_i. We assume that when two messages arrive at the same switch at the same time, the message from an upstream (left) client host has the higher priority of *owning the switch*, meaning that it has the higher priority to go through the switch. To make this linear client-server network complete, we may need another linear wormhole-switched network in reversed direction to send the results computed by the central server back to clients. Assume that the response to each client message can be computed by the central server in constant time in the first-come-first-serve order, and the response message for client message $M_i = \{p_i, e_i, d_i\}$ owns e_i time to go through any switch. Then, the response messages can be smoothly sent back to client hosts in pipelined fashion in the order of their corresponding client messages received by the server. In Figure 1, the reverse linear wormhole-switched network is not shown. One possible application of this network is to use it as a wired sensor networks with client hosts being sensors for monitoring and periodic reporting. Clearly, for $M_i = \{p_i, e_i, d_i\}$, p_i and d_i are functions of p_j, e_j and d_j, $1 \leq j < i$. A feasible schedule is a specification of p_is and d_is for $1 \leq i \leq N$ such that periodic messages M_is can reach the network output using period p_i and the "deadline" d_i can be met. The problem we are concerning about is how to determine p_is and d_is that are as small as possible. An implication of this objective is minimizing the average value of p_is and d_is.

In what follows we present some notations and definitions to be used in the rest of the paper:

- H_i Host i.
- S_i Switch i.
- M_i The periodic message from H_i: $M_i = \{p_i, e_i, d_i\}$.
- b_i^k The blocking time for an $M_k (k > i)$ blocked at switch S_i. This is the time that the blocked header flit of an M_k waits at the input port I_i^2 of S_i.
- B_i The maximum blocking time for an $M_k (k > i)$ blocked at switch S_i. That is, $B_i = \max\{b_i^k\}$.
- th_i^k The time required for the header flit of an $M_k (k > i)$ to travel through switches $S_i, S_{i-1}, \cdots, S_1$, given the header flit has already reached the input port I_i^2 of S_i.
- TH_i The least sufficient time required for the header flit of an $M_k (k > i)$ to travel through switches $S_i, S_{i-1}, \cdots, S_1$, given the header flit has already reached the input port I_i^2 of S_i. That is, $TH_i = max\{th_i^k\}$.
- ps_i The time required for an M_i to completely pass through the network, given M_i owns S_i immediately after its release.
- PS_i The least sufficient time for H_i to send an M_i to completely pass through the network, given M_i

owns S_i immediately after its release. That is, $PS_i = max\{ps_i\}$.

Clearly,

$$\begin{cases} th_i^k &= b_i^k + b_{i-1}^k + b_{i-2}^k + \cdots + b_1^k \\ ps_i &= e_i + th_{i-1}^i \\ PS_i &= e_i + TH_{i-1} \\ B_i &= PS_i \end{cases} \quad (1)$$

Because the header flit of a message M_k can only be directly blocked by an M_i at switch S_i ($i < k$), so the last equation holds.

III. Greedy Message Schedules

By the definitions given in the last section, we can see that in the worst case, $TH_i = B_i + \cdots + B_1$. Then,

$$\begin{aligned} B_i &= PS_i \\ &= e_i + (B_{i-1} + \cdots + B_1) \\ &= e_i + (e_{i-1} + TH_{i-2}) + (B_{i-2} + \cdots + B_1) \\ &= e_i + e_{i-1} + 2(B_{i-2} + \cdots + B_1) \\ &= e_i + e_{i-1} + 2e_{i-2} + 4(B_{i-3} + \cdots + B_1) \\ &= \cdots \\ &= e_i + e_{i-1} + 2^1 e_{i-2} + 2^2 e_{i-3} + ... + 2^{i-2} e_1 \end{aligned}$$

Since $B_i = e_i + TH_{i-1}$, we have

$$TH_i = B_{i+1} - e_{i+1} = e_i + 2^1 e_{i-1} + \cdots + 2^{i-1} e_1.$$

Hence,

$$\begin{cases} B_i &= e_i + \sum_{k=0}^{i-2} 2^k e_{i-1-k} \\ PS_i &= e_i + \sum_{k=0}^{i-2} 2^k e_{i-1-k} \\ TH_i &= \sum_{k=0}^{i-1} 2^k e_{i-k}. \end{cases} \quad (2)$$

Now, consider d_i. Obviously, the worst case occurs when an M_i is blocked by a $M_m (m > i)$ at switch i, and M_m is also blocked by a M_k at each switch S_k for $k < i$. Then,

$$\begin{aligned} d_i &\geq \underbrace{(e_m + TH_{i-1})}_{time\ for\ M_m} + \underbrace{PS_i}_{time\ for\ M_i} \\ &= (e_m + TH_{i-1}) + (e_i + TH_{i-1}) \\ &= e_m + e_i + 2(TH_{i-1}) \\ &= e_m + e_i + 2(\sum_{k=0}^{i-2} 2^k e_{i-k}) \\ &= e_m + \sum_{k=0}^{i-1} 2^k e_{i-k}. \end{aligned}$$

As m can be any number greater than i, we have

$$d_i \geq \max_{k>i}\{e_k\} + \sum_{k=0}^{i-1}(2^j e_{i-k})$$

Thus, if we let $p_i = d_i = \max_{k>i}\{e_k\} + \sum_{k=0}^{i-1}(2^j e_{i-k})$, then each message can be delivered within time d_i.

We call the schedule satisfying

$$p_i = d_i = \max_{k>i}\{e_k\} + \sum_{k=0}^{i-1}(2^j e_{i-k}) \quad (3)$$

a *greedy schedule*, because, according to it, every client host tries to send messages as many as possible, except occasionally giving chances to the messages at farther locations without lossing any network bandwidth. In what follows we show that we could not make any of above d_i and p_i smaller, without increasing other d or p values. Let $e = e_1 = e_2 = \cdots = e_N$, then

$$\begin{aligned} p_i = d_i &= e + e + 2^1 e + 2^2 e + \cdots + 2^{i-1} e \\ &= 2^i e \end{aligned}$$

Then, the total network utilization is:

$$\begin{aligned} u &= u_1 + u_2 + \cdots + u_N \\ &= \frac{e}{2e} + \frac{e}{2^2 e} + \cdots + \frac{e}{2^N e} \\ &= \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^N} \\ &\rightarrow 1 \end{aligned}$$

Though the greedy schedule may cause long blocking time, it still has some benefits in addition to full bandwidth utilization. It is very simple. After adding a restriction on the size of longest messages allowed in the network, the values of p_i and d_i are only related to message lengths of the downstream hosts (which are located closer to the network output). In the situation that all messages have the same length, the values of p_is and d_i only depend on the hosts' positions on the network. And in the real world, a message may not always be sent in every period cycle. The blocking time for message could be much smaller than B_i of Equation (2).

IV. Conservative Message Schedules

In the last section we showed that in the worst case values of p_i and d_i are exponential numbers of 2 (i^{th} power), which are very big numbers for for large i. In this section, we present a scheduling algorithm with reduced values of p_i and d_i and slightly smaller network utilization. We notice that the hosts closer to the network output contribute most blocking to messages. Since we could not reduce any d_i or p_i for a host without increasing the values of other hosts (proved in last section), we could decrease the values of d_i and p_i of upstream hosts by increasing the values of d_i and p_i of some downstream hosts (their values are small anyway). The new scheduling algorithm greatly reduces the values of d_i and p_i for all but first two hosts.

Let F_i be the ith Fibonacci number. That is,

$$F_i = \begin{cases} 1 & i = 1 \\ 1 & i = 2 \\ F_{i-1} + F_{i-2} & i > 2 \end{cases}$$

Define:

$$S(n) = F_1 e_n + \cdots + F_n e_1 = \sum_{i=1}^{n} F_i e_{n-i+1} \qquad (4)$$

The following two lemmas are useful.

Lemma 1:

$$S(n+2) = S(n+1) + S(n) + e_{n+2}$$

Proof:

$$\begin{aligned} & S(n+1) + S(n) + e_{n+2} \\ = & \sum_{i=1}^{n+1} F_i e_{n-i+2} + \sum_{i=1}^{n} F_i e_{n-i+1} + e_{n+2} \\ = & \sum_{i=0}^{n} F_{i+1} e_{n-i+1} + \sum_{i=1}^{n} F_i e_{n-i+1} + e_{n+2} \\ = & \sum_{i=1}^{n} (F_i + F_{i+1}) e_{n-i+1} + F_1 e_{n+1} + e_{n+2} \\ = & \sum_{i=1}^{n} F_{i+2} e_{n-i+1} + e_{n+2} + e_{n+1} \\ = & \sum_{i=3}^{n+2} F_i e_{n-i+3} + e_{n+2} + e_{n+1} \\ = & \sum_{i=1}^{n+2} F_i e_{n-i+1} \\ = & S(n) \end{aligned}$$

■

Lemma 2: $\{S(n)\}$ is a strictly increasing sequence.

Proof: Directly from Lemma 1. ■

Based on above two lemmas, we have the following theorem.

Theorem 1: Each message $M_i = (p_i, e_i, d_i)$ can meet the deadline if:

$$\begin{cases} p_i > \max_{k>i}\{e_k\} + S(i+1) \\ d_i = \max_{k>i}\{e_k\} + S(i) \end{cases} \qquad (5)$$

Proof: Let t_i be the period between release time and receiving time (at the network output) of an M_i. Define $\delta_i = p_i - d_i$. Then,

$$\begin{aligned} \delta_n &> S(n+1) - S(n) \\ &= S(n-1) + e_{n+1} \qquad (Lemma\ 1) \end{aligned}$$

We use induction to prove that for any i,

$$\begin{cases} TH_i \leq S(i) \\ t_i \leq d_i. \end{cases}$$

When $i = 1$, $TH_1 = B_1 = PS_1 = e_1 = S(1)$, and $t_1 \leq e_1 + max_{k>1}\{e_k\} = d_1$. Hence, there exists a $k \leq 1$, for $\forall i \leq k$, $TH_i \leq S(i)$ and $t_i \leq d_i$. Now we consider $i = k + 1$.

First, we prove that $TH_{k+1} \leq S(k+1)$. Since $TH_{k+1} = PS_{k+2} - e_{k+2}$, we only need to prove that $ps_{k+2} - e_{k+2} \leq S(k+1)$ for every M_{k+2}. Let M_{k+2} be a message released at time r_{k+2}. It owns switch S_{k+2} immediately. We need to consider two cases.

Case 1: M_{k+2} is not blocked at switch S_{k+1}.

Then, $b_{k+1}^{k+2} = 0$ and

$$\begin{aligned} ps_{k+2} - e_{k+2} &= b_{k+1}^{k+2} + b_k^{k+2} + \cdots + b_1^{k+2} \\ &= b_k^{k+2} + \cdots + b_1^{k+2} \\ &= th_k^{k+2} \\ &\leq TH_k \\ &\leq S(k) \\ &\leq S(k+1) \qquad (Lemma\ 2) \end{aligned}$$

Case 2: M_{k+2} is blocked at switch S_{k+1} by an M_{k+1} release at time r_{k+1}.

There are two subcases.

Subcase 2.1: M_{k+1} is not blocked at switch k.

Then, $b_k^{k+1} = 0$ for M_{k+1} and

$$\begin{aligned} ps_{k+2} - e_{k+2} &\leq b_{k+1}^{k+2} + th_k^{k+2} \\ &\leq ps_{k+1} + TH_k \\ &\leq (e_{k+1} + b_k^{k+1} + \cdots + b_1^{k+1}) + TH_k \\ &= (e_{k+1} + b_{k-1}^{k+1} + \cdots + b_1^{k+1}) + TH_k \\ &= (e_{k+1} + th_{k-1}^{k+1}) + TH_k \\ &\leq (e_{k+1} + TH_{k-1}) + TH_k \\ &\leq e_{k+1} + S(k-1) + S(k) \\ &= S(k+1) \qquad (Lemma\ 1) \end{aligned}$$

Subcase 2.2: M_{k+1} is blocked at switch S_k by an M_k released at time r_k.

We show that M_{k+2} will not be blocked at switch S_k. Since H_k releases a message at r_k, the next message release time is

$$r_k + p_k = r_k + d_k + \delta_k.$$

As $t_k \leq d_k$, at time $r_k + d_k$, M_k has already passed S_k. So M_{k+1} owns S_{k-1} before or at time $r_k + d_k$. The additional time for M_{k+1} pass S_k is

$$\begin{aligned} e_{k+1} + th_{k-1}^{k+1} &\leq e_{k+1} + TH_{k-1} \\ &\leq e_{k+1} + S(k-1) \\ &< \delta_k \end{aligned}$$

Hence, before the next M_k release, M_{k+1} has already passed S_k and M_{k+2} owns switch S_k. Hence M_{k+2} will not be blocked at switch S_k. Thus,

$$\begin{aligned} ps_{k+2} - e_{k+2} &\leq b_{k+1}^{k+2} + b_k^{k+2} + \cdots + b_1^{k+2} \\ &\leq b_{k+1}^{k+2} + 0 + th_{k-1}^{k+2} \\ &\leq B_{k+1} + TH_{k-1} \\ &\leq e_{k+1} + TH_k + TH_{k-1} \\ &\leq e_{k+1} + S(k) + S(k-1) \\ &= S(k+1) \qquad (Lemma\ 1) \end{aligned}$$

By now we have proved that $TH_{k+1} \leq S(k+1)$. We proceed to prove that $t_{k+1} \leq d_{k+1}$. Let M_{k+1} be a message released from H_{k+1} at time r_{k+1}. We need to consider two cases.

Case 1: M_{k+1} is not blocked by another message at S_{k+1}.

Then,

$$\begin{aligned} t_{k+1} = ps_{k+1} &\leq e_{k+1} + TH_k \\ &= e_{k+1} + S(k) \\ &= S(k+1) - S(k-1) \\ &< d_{k+1}. \end{aligned}$$

Case 2: M_{k+1} is blocked by a message M_m $(m > k+1)$ at S_{k+1}, and M_m is blocked at switch S_l, $l \leq k$.

We have two subcases.

Subcase 2.1: $l < k$.

Clearly,

$$\begin{aligned} t_{k+1} &= ps_{k+1} + e_m + th_l^m \\ &\leq ps_{k+1} + e_m + TH_l \\ &\leq ps_{k+1} + e_m + S(l) \\ &\leq e_{k+1} + TH(k) + e_m + S(k-1) \\ &\leq e_{k+1} + S(k) + e_m + S(k-1) \\ &= S(k+1) + e_m \qquad (Lemma\ 1) \\ &\leq d_{k+1} \end{aligned}$$

Subcase 2.2: $l = k$.

M_m is blocked at S_k. Then, M_{k+1} will not be blocked at S_k and

$$\begin{aligned} t_{k+1} &= e_m + th_k^m + e_{k+1} + th_{k-1}^{k+1} \\ &\leq e_m + TH_k + e_{k+1} + TH_{k-1} \\ &\leq e_m + S(k) + e_{k+1} + S(k-1) \\ &= e_m + S(k+1) \qquad (Lemma\ 1) \\ &\leq d_{k+1} \end{aligned}$$

This completes the proof of $t_{k+1} \leq d_{k+1}$ and the proof of the theorem. ■

Based on Theorem 1, we obtain the following scheduling algorithm:

1) For $i = 1, 2, \cdots, N$, calculate $S(i)$ according to Equation (4).
2) For $i = 1, 2, \cdots, N$, do:
   a) Calculate $e^* = \max_{k>i}\{e_k\}$.
   b) Set $p_i = e^* + S(i-1)$.

This algorithm guarantees that the time $d_i = e^* + S(i)$ is sufficient for delivering message M_i to the network output. We call the message schedule produced by this algorithm a *conservative schedule*. As in the last section, we evaluate the network utilization by assuming that $e_i = e$ for all M_is. Then, we have network utilization

$$\begin{aligned} u &= u_1 + u_2 + \cdots + u_N \\ &= \frac{e}{p_1} + \frac{e}{p_2} + \cdots + \frac{e}{p_N} \\ &= \frac{1}{F_4} + \frac{1}{F_5} + \cdots + \frac{1}{F_{N+3}} \\ &\rightarrow 0.8599 \end{aligned}$$

V. Comparisons and Concluding Remarks

To simplify comparisons, we assume that all messages have the same length, i.e., $e_i = e$, for any i. Then we have:

$$Conservative: \left\{ \begin{array}{lcl} d_i &=& e + S(i) \\ &=& (1 + F_1 + \cdots + F_i) \cdot e = F_{i+2} \cdot e \\ p_i &=& e + S(i+1) \\ &=& (1 + F_1 + \cdots + F_{i+1}) \cdot e = F_{i+3} \cdot e \end{array} \right.$$

$$Greedy: \left\{ \begin{array}{lcl} d_i &=& (2^0 + \cdots + 2^{i-1}) \cdot e = 2^i \cdot e \\ p_i &=& (2^0 + \cdots + 2^{i-1}) \cdot e = 2^i \cdot e \end{array} \right.$$

Comparing the two schedules based on the coefficients of e, we have:

$$d_i: \left\{ \begin{array}{rrrrrrrrrrr} i: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \cdots \\ Conservative: & 2 & 3 & 5 & 8 & 13 & 21 & 34 & 55 & 89 & \cdots \\ Greedy: & 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 & \cdots \end{array} \right.$$

$$p_i: \left\{ \begin{array}{rrrrrrrrrrr} i: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \cdots \\ Conservative: & 3 & 5 & 8 & 13 & 21 & 34 & 55 & 89 & 144 & \cdots \\ Greedy: & 2 & 4 & 8 & 16 & 32 & 64 & 128 & 256 & 512 & \cdots \end{array} \right.$$

The improvement of conservative schedule over greedy schedule is obviously significant. For d_is, no value is larger, and all but the first are reduced. For p_is, all but the first three are reduced. These results suggest that for linear wormhole-switched client-server network real-time applications, client hosts requiring more frequent communications with the server should be placed closer to the server (network output). In some applications, uniform period is preferred for all client hosts. Consider the case that $e_i = e$ for $1 \leq i \leq N$. A simple feasible message schedule for this case is letting $p_i = N^2 \cdot e$ and $d_i = i \cdot e$. For $N = 9$, we have

i :	1	2	3	4	5	6	7	8	9
d_i :	1	2	3	4	5	6	7	8	9
p_i :	81	81	81	81	81	81	818	81	81

The network utilization is

$$\begin{aligned} u &= u_1 + u_2 + \cdots + u_N \\ &= \frac{e}{p_1} + \frac{e}{p_2} + \cdots + \frac{e}{p_N} \\ &= \frac{1}{N^2} + \frac{2}{N^2} + \cdots + \frac{N}{N^2} \\ &\to 0.5 \end{aligned}$$

In our linear network, we assumed that host H_N is connected to S_N. Since there is no traffic from the left of H_N, H_N can be directly connected to the input of S_{N-1}. In such a situation, slightly smaller values of p_i and d_i with more complicated proofs can be achieved. But such an improvement is insignificant. We assume that H_N is connected to S_N for the sake of obtaining cleaner closed-form expressions and simpler proofs.

Our study shows that under heavy traffic, p_is and d_is are communication distance sensitive. There is a trade-off between network utilization and values of p_is and d_is. A couple of related problems arise. For example, how to obtain better schedules in terms of smaller p_i and d_i values? How to design message schedules with smaller uniform periods? How to generalize our techniques to networks of other topologies (such as trees)? These problems may deserve further investigation.

REFERENCES

[1] Anjan K. V. and T.M. Pinkston, "DISHA: A Deadlock Recovery Scheme for Fully Adaptive Routing," *Proceedings of the 9th International Parallel Processing Symposium*, pp. 201-210, 1995.

[2] J.D. Allen et al., "Ariadne - An Adaptive Router for Fault-Tolerant Multicomputers," *Proceedings the 21st International Symposium on Computer Architecture*, pp. 278-288, 1994.

[3] N.J. Boden et al., "Myrinet - A Gigabit Per Second Local Area Network," *IEEE Micro*, pp. 29-36, 1995.

[4] J. Carbonaro and F. Veroorn, "Cavallino: The Terafbps Router and NIC," *Proceedings of Hot Interconnects Symposium IV*, 1996.

[5] W.J. Dally and C.L. Seitz, "The torus Routing Chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187-196, 1986.

[6] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547-553, 1987.

[7] W.J. Dally et al., "Architecture and Implementation of the Reliable Router," *Proceedings of Hot Interconnects Symposium II*, 1994.

[8] W.J. Dally and P. Song, "Design of Self-Timed VLSI Multicomputer Communication Controller," *Proceedings of International Conference on Computer Design*, pp. 230-234, 1897.

[9] J. Duato, S. Yalamanchili and L. Ni, *Interconnection Networks, An Engineering Approach*, Morgan Kaufmann, 2003.

[10] J. Duato, A. Robles and F. Silla, "A Comparison of Router Architectures for Virtual Cut-Through and Wormhole Switching in a NOW Environment,", *Journal of Parallel and Distributed Computing*, 61, pp. 224-253, 2001.

[11] M. Galles, "Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SPIDER Chip," *Proceedings of Hot Interconnects Symposium IV*, 1996.

[12] R. Horst, "ServerNet Deadlock Avoidance and Fractahedral Topologies," *Proceedings of the 10th International Parallel Processing Symposium*, pp. 274-280, 1996.

[13] S.L. Scott, "Synchronization and Communication in the T3E Multiprocessor," *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 26-36, 1996.

[14] S.L. Scott and G. Thorson, "Optimized Routing in the Cray T3D," *Proceedings of the Workshop on Parallel Computer Routing and Communication*, pp. 281-294, 1994.

[15] S.L. Scott and G. Thorson, "The Cray T3E Network: Adaptive Routing in a High-Performance 3D Torus," *Proceedings of Hot Interconnects Symposium IV*, 1996.

[16] C.B. Stunkel et al., "The SP2 Communication Subsystem," Technical Report, IBM T.J. Watson Research Center, 1994.