Universal Wormhole Routing

Ronald I. Greenberg, *Member, IEEE,* and H.-C. Oh, *Member, IEEE,*

To appear in IEEE Transactions on Parallel and Distributed Systems.

Universal Wormhole Routing

Ronald I. Greenberg, *Member, IEEE,* and H.-C. Oh, *Member, IEEE,*

***Abstract*— In this paper, we examine the wormhole routing problem in terms of the "congestion" c and "dilation" d for a set of packet paths. We show, with mild restrictions, that there is a simple randomized algorithm for routing any set of P packets in $O\left(cd\eta + cL\eta\log P\right)$ time with high probability, where L is the number of flits in a packet, and $\eta = \min\{d, L\}$; only a constant number of flits are stored in each queue at any time. Using this result, we show that a fat-tree network of area $\Theta(A)$ can simulate wormhole routing on any network of comparable area with $O(\log^3 A)$ slowdown, when all worms have the same length. Variable-length worms are also considered. We run some simulations on the fat-tree which show that not only does wormhole routing tend to perform better than the more heavily studied store-and-forward routing in this context, but that performance superior to our provable bound is attainable in practice.**

***Index Terms*—Wormhole routing, packet routing, randomized routing, greedy routing, area-universal networks, fat-tree interconnection network**

I Introduction

An efficient routing algorithm is critical to the design of most large-scale general-purpose parallel computers. One must move data between different locations in an appropriate routing network as quickly as possible and with as little queuing hardware as possible. Store-and-forward routing is the most extensively studied model and many asymptotically efficient algorithms have been proposed for this model (e.g., [15] and the references therein). Recently, increasing attention has been devoted to the wormhole routing model [3], since it can lead to a reduction in routing time and the storage requirements of intermediate nodes. In this model, packets (or *worms*) are composed of *flits* or *flow control digits*, and packets snake through the network one flit after another.

Few works have performed any theoretical analysis of wormhole routing or similar schemes. Leighton [17] performs average-case analysis of greedy cut-through routing on meshes. But cut-through routing [13] differs from wormhole routing in that it uses buffers that can store at least one full packet rather than a few flits. Makedon and Simvonis [20] give worst case bounds for cut-through routing of permutations on the mesh and the torus. Aiello, Leighton, Maggs, and Newman [1] give an efficient algorithm for wormhole routing of permutations on a dilated butterfly. Their algorithm is nonoblivious (may use information about other packets when routing a given packet). More recently, Felperin, Raghavan, and Upfal [6] have obtained a simple, oblivious algorithm for wormhole routing of permutations on the butterfly and the mesh. Some other works have described iterative methods for estimating the performance of certain networks under certain probabilistic models of message generation [4, 11, 2].

While the above analyses of wormhole routing have been applicable only to specific networks and/or specific message patterns, this paper takes a more general approach based on summary measures of the message traffic, as in [16, 15]. We require only that any two paths in the network intersect in at most a constant number of contiguous sequences of edges, a condition that is met by many networks used in practice. Recently, an intermediately broad class of networks has been considered by Ranade et. al. with particular application to improved routing of certain message distributions on the butterfly [26].

After deriving general bounds for wormhole routing, we apply the results to the construction of area-universal networks. In particular, when worms have a fixed length, a bounded-degree network (the butterfly fat-tree [9]) of

This work was supported in part by the National Science Foundation under grants CCR-9109550 and CCR-9321388 and by a Summer Research Award from the University of Maryland Office of Graduate Studies and Research.

R. I. Greenberg is with the Department of Mathematical and Computer Sciences Loyola University, 6525 N. Sheridan Rd., Chicago, IL 60626, rig@math.luc.edu.

H-C. Oh is with the Department of Information Engineering, Korea University, Chochiwon, Korea, hyeong@tiger.korea.ac.kr.

area $\Theta(A)$ using wormhole routing can simulate (on-line) any network of comparable area with $O(\log^3 A)$ slowdown. Though it has been proven that $O(\log A)$ slowdown suffices in the store-and-forward routing model [15], such an approach requires the universal network to queue full packets at each intermediate node and similarly limits the type of competing network that is considered. Also, the circuit-switching scheme of [9] could actually be used as a wormhole routing scheme, but with poorer overhead than we show here, since the earlier scheme locks down a routing path for more than the time required for a worm to pass.

We also extend the universality analysis to the case in which worms have varying lengths. In this case, each processor continuously generates and sends packets, where the packet length L is a random variable with mean $E[L] = \bar{L}$ and maximum value L_M. With mild restrictions, we show that a fat-tree network of area $\Theta(A)$ can simulate any network of comparable area with $O\left((L_M/\bar{L})\log^3 A\right)$ slowdown.

Before proceeding with the promised results, we give more detail on the model and terminologies used throughout this paper. We consider the routing of a set of P packets, each consisting of L flits. We follow the usual graph-based terminology; processors and switches are nodes in the graph and communication channels are represented by edges. We make the usual assumption that unit time suffices for a flit to cross any edge in the network (though it would also be desirable to extend the analysis to general edge delays as done in [10] for the store-and-forward model). A flit is an atomic object, which at each time step, either waits in a queue, or crosses an edge and enters the edge queue at the end of that edge. (In store-and-forward routing, packets are the atomic objects.) We call this unit time step a *flit-step*, while the corresponding unit time step for store-and-forward routing is a *packet-step*. We restrict attention to bounded-degree networks, so the time to make routing decisions at any given node does not affect the asymptotic time bounds.

We may view the packet routing problem as being comprised of two tasks, selecting a path through the network for each packet and setting a schedule for when packets move and wait. In the next section of this paper, we focus on the second task. Of course, the selection of paths affects the required routing time. For example, the maximum distance d, in number of edges, traveled by any packet is a lower bound on the routing time; this distance is often referred to as the *dilation* in the literature. (It may be noted that the dilation is typically at most as large as the network diameter, but this is not necessarily so if some packets do not traverse shortest paths.) Similarly, the routing time is lower bounded by cL, where the *congestion* c is the maximum over all edges of the number of packets that must traverse the edge over the entire course of the routing.

Once the set of packet paths has been determined, we can define a graph, $\mathcal{D}$, which has a vertex for each edge of the network and an edge (u, v) whenever there is a packet path in which network edge v immediately follows network edge u. We refer to this graph as the *dependency graph*. We ensure that deadlock cannot occur by assuming that the dependency graph of the paths is acyclic [3]. (Many networks, e.g., leveled networks [15], have no cycle in $\mathcal{D}$ for any set of packets, and there also are techniques for breaking cycles [3]. In addition, there are adaptive routing techniques for avoiding deadlock (e.g., see [5] and the references therein), and our analysis can be applied to the set of packet paths generated by such a technique.)

II A Simple wormhole routing algorithm

In this section, we give a simple delayed-greedy wormhole routing algorithm and its theoretical analysis, when all worms have the same length, L. Throughout this section, we only consider a set of paths such that the channel dependency graph is acyclic. We also assume that any two paths in the network intersect in at most one contiguous sequence of edges. (It will be easily seen that the results are also valid as long as any two paths intersect in at most a constant number of contiguous sequences of edges.) Each node has a queue, for each input edge, which can store at most one flit. It is sufficient for our analysis to have each node scan its input queues in a fixed order and send out a flit whenever the relevant outgoing edge is not occupied by another worm.

We say that a worm W' *blocks* W at t if the edge to which the head of W has to proceed at t is taken by W'. Worm W' *delays* worm W at t, if at t, there is a *delay chain* of $r(\geq 1)$ worms $W = W_1, W_2, \cdots, W_r = W'$ such that worm W_i is blocking worm W_{i-1}; worm W' is moving; and no other worm in the chain can move. Since we exclude any possibility of deadlock, any blockage will end at some time. Also, once worm W' delays worm W for at most L steps (not necessarily consecutive), W' will not delay W again because of our assumption that packet paths intersect in at most one contiguous sequence of edges.

The basic routing algorithm we use is a delayed-greedy approach similar to that of Felperin, Raghavan, and Upfal [6]. The analysis here simplifies, clarifies, and generalizes their argument. We begin with a worst-case bound on routing time that holds with high probability and then give a tighter bound on the expected time. (Throughout this paper we use the term "high probability" in the standard fashion that for any constant m, we can achieve probability $1-O(1/S^m)$, where S is an appropriate measure of problem size. In particular we focus on achieving time bounds for routing P packets with probability $1-O(1/P)$ and bounds for network simulation using networks of area A that hold with probability $1-O(1/A)$.) The starting point for both results is the following core routing procedure, expressed in terms of parameters k and T to be determined later:

Algorithm A Assign each packet an integral delay randomly and uniformly from the interval $[0, kcL-1]$. A packet that is assigned delay i waits in its initial queue for iT steps and then proceeds to its destination. We refer to the time between $(i-1)T$ and iT in as the i-th phase. In what follows, we assume T is large enough that worms dispatched in different phases do not interfere with each other and we analyze how large T needs to be for the worms dispatched in a phase to actually get delivered before the end of the phase.

We begin with a key lemma for bounding the tail of a binomial distribution:

Lemma 1 *Consider τ independent Bernoulli trials, each with probability p of success. The probability that the number of successes s is larger than the expectation $p\tau$ is at most $\left(\frac{ep\tau}{s}\right)^s$.*

Proof. The probability is at most $\binom{\tau}{s}p^s$ and the Lemma follows through the use of Stirling's approximation to the factorial. ■

Now, we can prove a general lemma about the number of delaying worms encountered during a specified set of edge traversals of some set of worms:

Lemma 2 *Consider a set of worm paths comprising a total of y edge traversals (by worm heads) that need to be accomplished in some phase of Algorithm A, and let $P(x,y)$ be the probability that delays by x worms interfere with that set of traversals. Then $P(w,x) \leq \left(\frac{1}{k'}\right)^{x-y/L}$, where $k' = \sqrt{\frac{k}{2e}} \geq 2$.*

Proof. We use induction on x and prove the result for each value of x by using induction on y. The base cases are trivial. For the induction step, we first consider $y > L$ and focus on the first L edge traversals of a particular worm; notice that they can be blocked by at most cL worms. If none of those worms is launched in the current phase, then $P(x,y)$ is just equal to $P(x, y-L)$. Otherwise, let S be the set of such worms launched in the current phase, and note that there is a probability of at most $\left(\frac{e}{k}\right)^i$ that S contains i worms, by using $\tau = cL$, $p = \frac{1}{kcL}$, and $s = i$ in Lemma 1. In the worst case, all the worms in S could act as delaying worms, and we must also worry about delays encountered by those worms during traversal of up to $L-1$ edges that have not already been considered. (We need not consider more than $L-1$ new edges for each worm in S, because once a worm has traversed so many edges not already under consideration, it has digressed far enough from the paths originally under consideration as to have no further effect.)

Thus, for the induction step, we have

$$\begin{aligned} P(x,y) &\leq P(x,y-L) + \sum_{i=1}^{cL} \left(\frac{e}{k}\right)^i P(x-i, y-L+iL) \\ &\leq \left(\frac{1}{k'}\right)^{x-y/L+1} \left(1 + \sum_{i=1}^{cL} \left(\frac{e}{k}\right)^i \left(\frac{1}{k'}\right)^{-2i}\right) \\ &\leq \left(\frac{1}{k'}\right)^{x-y/L+1} \left(1 + \sum_{i=1}^{cL} \left(\frac{1}{2}\right)^i\right) \\ &\leq \left(\frac{1}{k'}\right)^{x-y/L} \end{aligned}$$

For $y \leq L$, the additive term preceding the summation above disappears, and we replace $P(x-i, y-L+iL)$ with $P(x-i, y+iL)$; the final result still holds. ∎

Now we can analyze the time after a worm is dispatched in Algorithm A that is required to traverse all d of its links:

Lemma 3 *Let $P'(z)$ be the probability that a given worm W requires at least $2d+Lz$ time to reach its destination under Algorithm A. Then, $P'(z) \leq \left(\frac{1}{k'}\right)^z$, where $k' = \sqrt{\frac{k}{2e}} \geq 2$.*

Proof. For W to require $2d + Lz$ time to reach its destination, it must be delayed by $\frac{d}{L} + z$ worms since any one worm can delay W for at most L steps. The result follows from Lemma 2. ∎

We are now ready for the main analytical result:

Theorem 4 *With $\eta = \min\{d, L\}$, any set of P packets can be routed in $O(cd\eta + cL\eta \log P)$ flit-steps with high probability.*

Proof. We consider first the case of $L \leq d$. In this case, we simply run Algorithm A with $k = 32e$ and $T = 2d + L\log_2 P$. By Lemma 3 (with $k' = 4$), the probability is $O(1/P^2)$ that any given worm is not delivered during the phase in which it is dispatched (under the assumption that all worms dispatched in previous phases have been delivered). This yields an overall probability of $O(1/P)$ that there exists any worm that does not get delivered. (The failure probability can be changed to any constant power of $1/P$ by changing the constant k.) The total time for the kcL phases of the algorithm is $O(cdL + cL^2 \log P)$. For $L > d$, we use a modified version of Algorithm A with kcd phases. Lemmas 2 and 3 still go through by replacing some appearances of L with d in the proofs, so we can proceed as above with $T = 2d + L\log_2 P = O(L \log P)$, and the total routing time is $O(cdL \log P)$ with high probability. ∎

It is interesting to note that we can also obtain a better expected routing time as well as the high probability result above.

Theorem 5 *Any set of packets can be routed in $O(cdL)$ expected time.*

Proof. As in the proof of Theorem 4, we actually use $O(32ec\eta)$ phases in Algorithm A, where $\eta = \min\{d, L\}$. Also, we initially run the algorithm with $T = T_0 = 2(d+L)$, and then run the whole algorithm through again with $T = 2T_0$, and then with $T = 4T_0$, etc. The high probability result of Theorem 4 still holds because we at most double the routing time through the process of building up towards a high enough value of T. In addition, the expected time per phase is at most $\sum_{z=0}^{\infty}(d+L)z\left(\frac{1}{4}\right)^z$ by Lemma 3, which is $O(d+L)$. So the total expected time is $O(c\eta(d+L)) = O(cdL)$. ∎

We also have the following corollary to Theorem 4, which is useful in Section A:

Corollary 6 *When $d \leq \log P$, any set of P packets can be routed in $O\left(cL\log^2 P\right)$ flit-steps with high probability.* ∎

III Wormhole routing on fat-trees

Fat-trees constitute a class of routing networks for hardware-efficient parallel computation [18, 9, 15]. Figure 1 shows a layout of one fat-tree variant using switches of constant size. A fat-tree in this style is usually referred as a *butterfly fat-tree*, of which a variation has been adopted in the CM-5 supercomputer of Thinking Machines Corporation [19]. In Figure 1, a set of N processors are placed at the leaves, represented by circles; the squares are switches. Each connection drawn between a pair of switches or a processor and a switch represents a pair of oppositely directed

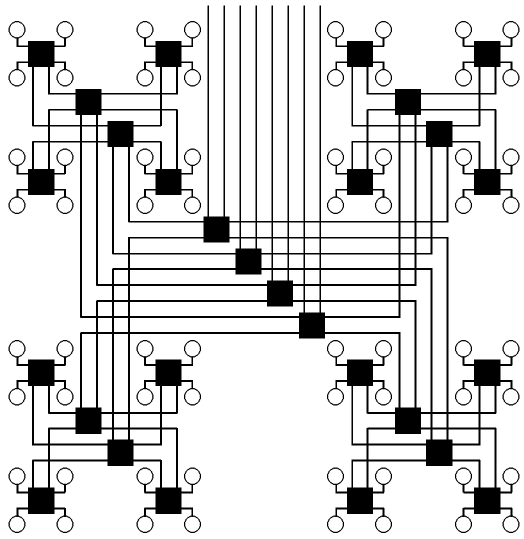

Fig. 1. A butterfly fat-tree.

links, each capable of transmitting one flit in unit time. We call the link from parent to child a *down link*, and the other an *up link*. The underlying structure of Figure 1 is a complete 4-ary tree. Each edge of the underlying tree consists of a group of links, called a *channel*. We call the channel from parent to child a *down channel*, and the other an *up channel*. The number of links in a channel is called its *capacity*. An important measure of the difficulty of routing a set of packets on a fat-tree is the *load factor*, the maximum ratio of the number of packets traversing a channel to the capacity of the channel. The load factor λ is closely related to the congestion c. We can always choose packet paths so that $c = O(\lambda + \log N)$ [15, Lemma 9].

We select a shortest path for each packet. The dependency graph for the paths selected in this way is free from cycles, because no shortest path proceeds from a down channel to any up channel. In fact, we can view the network as being partitioned into two halves, a network of up channels and a network of down channels, by duplicating each switch. In the network comprised of these two halves, any two paths intersect in at most two contiguous sequences of edges. Hence the result of Section II can be applied. (The bound on the number of steps during which a given worm can delay another given worm only increases from L to $2L$.)

A Area-universality of fat-trees

A.1 Worms with a fixed length

The algorithm analyzed in Section II allows us to extend to the wormhole routing problem universality theorems from [18, 9, 15, 7] which state that a universal fat-tree of a given area (volume) can simulate (using circuit switching or store-and-forward packet routing) any other routing network of equal area (volume) with only a polylogarithmic factor increase in the time required. Throughout this section, we assume that all worms have a fixed length, L.

We construct a fat-tree on unit-size processors, which occupies area linear in the number of processors, as in [7]. (It is actually more reasonable to consider processors that are larger than constant-size, but we bypass this complication, since it can be handled as in [7, 8].) Then, a very simple one-to-one mapping of a competing network's processors to those of the fat-tree guarantees that any set of packets delivered in one packet-step by a competing network of comparable area does not induce too great a congestion on the fat-tree, as is shown by the following lemma, adapted from [7, Lemma 2.1]. (For example, in area A, we can construct a mesh or H-tree on $O(A)$ processors or a butterfly on $O(\sqrt{A}\lg A)$ nodes and do a straightforward geometric mapping to a fat-tree with A processors and appropriate channel capacities.)

Lemma 7 *Consider networks with unit-sized processors, and let R be the set of all networks of area A. Then, there exists a fat-tree F of area $\Theta(A)$ such that any set of packets delivered in one packet-step by a network in R induces a congestion of $O(\log A)$ on F.* ∎

We can immediately extend this lemma to the case in which the competing network uses wormhole routing; the set of packets that move during any window of L flit-steps in the competing network induce a congestion of $O(\log A)$. Then we can state our universality result for wormhole routing:

Theorem 8 *A fat-tree F of area $\Theta(A)$ can simulate any network of area A with a factor of $O(\log^3 A)$ loss of run-*

time efficiency, using on-line wormhole routing with high probability.

Proof. Consider the set of packets that moves during L flit steps in a competing network of area A. By extending Lemma 7 as suggested above, we know that the congestion created by this set of packets on a fat-tree of area $\Theta(A)$ is $O(\log A)$. Next we can restate Corollary 6 by substituting A for P as long as the number of packets is polynomial in A, as is true here. For a fat-tree, $d = O(\log A)$, so the set of packets can be delivered by F in $O\left(L \log^3 A\right)$ flit-steps. ■

It should be noted that under some circumstances, we can obtain an asymptotic bound that appears better than the above by splitting each packet into flits and essentially treating these flits as independent packets. Of course, we must then attach complete addressing information to each flit. If a flit is big enough to carry a full address, then we can think of each flit as being transformed into a packet of two flits and we could use the store-and-forward routing scheme for leveled networks of Leighton et. al. [15] to route the packets in $O(cL + d + \log P)$ time. This yields $O(\log A)$ overhead for fat-tree simulation. Of course, it is unfair to compare this result with Theorem 8, because this *independent-flit* approach would induce additional overhead, such as increased storage in the intermediate nodes and the overhead of splitting and reconstructing the packets.

A.2 Worms with variable lengths

In this section, we consider the situation in which each processor continuously generates and sends packets, where the packet length L is a random variable with mean $\bar{L}$, variance σ_L^2, and maximum value L_M.

The following lemma shows that, with realistic restrictions, the total length of a set of n packets is unlikely to greatly deviate from its expected value.

Lemma 9 *Let X be a random variable, with mean μ and variance σ^2, and let S_n be the sum of n independent random variables distributed as X. If $0 < \sigma \leq \epsilon\mu$, where ϵ is a constant and $0 < \epsilon < 1$, then $S_n = \Theta(n\mu)$ with probability at least $1 - \frac{1}{n}$.*

Proof. The proof follows from the Tchebycheff inequality [23, p 115]. ■

Next, we note a simple corollary to Corollary 6:

Corollary 10 *When $d \leq \log P$, any set of packets can be routed, in $O\left(cL_M \log^2 P\right)$ flit-steps with high probability.* ■

We assume that the standard deviation of the packet length satisfies $0 < \sigma_L \leq \epsilon\bar{L}$, for some constant ϵ such that $0 < \epsilon < 1$. This assumption is satisfied by the packet-length distributions, generated in typical concurrent computing applications, presented in the literature, e.g. [21].

When packets have varying lengths, the simulation overhead becomes more complicated to analyze, because the number of packets crossing a wire in competing networks may vary from wire to wire during an interval of time. We consider the situation in which each processor continuously generates and sends packets during a time interval of length $T \gg \bar{L}$. In the following theorem, we extend the result of Theorem 8 to this general setting:

Theorem 11 *Consider competing networks with area A in which processors continuously generate and send packets during a time interval of length at least $A\bar{L}$, and let L_M be bounded above by a polynomial in A. Then a fat-tree F of area $\Theta(A)$ can simulate any such network with a factor of $O\left((L_M/\bar{L}) \log^3 A\right)$ loss of runtime efficiency, using on-line wormhole routing with high probability.*

Proof. We break up the overall time period under consideration and consider separately the worms delivered by the competing network in consecutive time intervals of length $T = A\bar{L}$. Then we determine the overhead with which any set of message routed by any network during an interval I of length T can be delivered by a fat-tree of comparable area.

We construct a fat-tree on unit-size processors, as in Section A.1. Next we recursively bisect the competing network in the straightforward geometric fashion, as in [7, 8], and match the parts obtained in this bisection to pieces of F in accordance with the obvious recursive bisection of F. We consider a piece, $\mathcal{C}$, of area A' in the competing network. Our construction of F guarantees that the channel capacity, in F, corresponding to the perimeter of $\mathcal{C}$ is $O\left(\sqrt{A'}/\log A\right)$.

We now consider the set of wires, S, connecting $\mathcal{C}$ to the remaining part of the competing network. Let P' denote the maximum, over all the wires in S, of the number of worms crossing the wire during I. Since the sum of the lengths of the P' packets cannot exceed T, we know from Lemma 9 that $P' = O\left(T/\bar{L}\right)$ with probability at least $1 - \frac{1}{T/\bar{L}} = 1 - \frac{1}{A}$. Since the perimeter of $\mathcal{C}$ is $O\left(\sqrt{A'}\right)$, the total number of worms crossing the wires in S during T is $O\left(\sqrt{A'}P'\right)$. Thus the congestion induced, on the channel in F, by the worms crossing the wires in S is $O\left(P' \log A\right)$. By Corollary 10, all the worms routed by the competing network during I can be delivered by F in $O\left(L_M P' \log A \log^2 P\right)$ flit-steps, where P is the total number of worms to be delivered.

Since $P = O\left(NT\right) = O\left(NAL_M\right)$, P is bounded above by a polynomial in A, from which the theorem follows.

■

B Simulation

This section investigates the practical performance of wormhole routing algorithms on butterfly fat-trees. For simplicity, we focus on the case in which all worms have a fixed length L. (The overhead for fat-trees simulating other networks is not much worse with variable length worms in terms of provable bounds, and we expect the same to be true in practice.)

B.1 Description of the butterfly fat-tree

We use the butterfly fat-tree with N processors in the style of Figure 1. Each node has an address which is expressed as a pair (l, a) of integers, where l represents the level of the node in the butterfly fat-tree and a represents the address of the node in that level. Let the level of a node be its distance from the leaves. At the 0-th level ($l = 0$) are N processors which are addressed from 0 to $N-1$. In Figure 1, we arrange the processors in a similar fashion to the *shuffled row-major indexing* in [27]. These processors are connected to $N/4$ switches at the 1-st level such that the processor at $(0, a)$ is connected to the switch $(1, \lfloor a/4 \rfloor)$. At the l-th level, for $l = 2, \cdots, \log_4 N$, there are $m_l = \frac{m_{l-1}}{2}$ switches. The connections of a switch are determined by the switch's address as follows: (l, a) is connected to $(l+1, \lfloor \frac{a}{2^{l+1}} \rfloor \cdot 2^l + a \bmod 2^l)$ and $(l+1, \lfloor \frac{a}{2^{l+1}} \rfloor \cdot 2^l + (a + 2^{l-1}) \bmod 2^l)$.

B.2 Routing algorithms and strategies

Algorithm *STORE* is a (delayed) greedy store-and-forward routing algorithm. Each packet chooses an integral delay randomly and uniformly from the interval $[0, R-1]$. A packet that is assigned delay x waits in its initial queue for x time steps and then proceeds to its destination. At each step, each node scans its input queues once and sends out available packets greedily (whenever the corresponding output edge is idling and the queue at the end of that edge is not full).

Algorithm *WORM* is a (delayed) greedy wormhole routing algorithm. Each packet consists of L flits. Each packet chooses an integral delay randomly and uniformly from the interval $[0, R-1]$. A packet that is assigned delay x waits in its initial queue for $xL \log N$ time steps and then proceeds to its destination. At each flit step, each node scans its input queues once. If the flit is a head flit, the node sends it out according to the flit's path only when the output edge is not being used by any other packet and the queue at the end of that edge is not full. If the flit is not a head flit, the node sends it out to where the flit's head was sent out, whenever the queue at the end of that edge is not full.

Algorithm *UNIV* is the universal store-and-forward routing algorithm of [15] for leveled networks. In this scheme packets choose a random priority from $[1, R]$ that is used to order the passage of packets through any given switch.

Algorithm *SPLIT* uses the independent-flit approach.

Each packet is split into flits which are treated as independent packets and routed as in *STORE*. This approach is also called the *multipacket routing approach* [14]. Note that this approach requires a replication of addressing information so that each of the independent packets can be routed to the correct location.

In the butterfly fat-tree, there is more than one shortest path between a pair of leaves. More specifically, at a switch, a packet can take any one of two up links, when its destination is not one of the leaves of the subtree rooted at the switch. (There is no redundancy for down links.) We can use this redundancy in selecting paths.

- Fixed-Path (FP) selection: For each packet, we select a shortest path randomly and uniformly before the packet leaves its source.
- Random-Path (RP) selection: When a packet needs to go up, it selects an up link randomly. If the link is blocked, the packet waits. The selection is oblivious, i.e., each time a packet seeks to go up, it makes a selection randomly.
- Greedy-Path (GP) selection: The packet seeking to go up scans up links and chooses the first one which is not blocked.

When more than one incoming packet is to be routed to an outgoing link, the way of selecting one may affect the results. The following schemes have been tested:

- Fixed-Order (FO) scan: At each time step, a switch scans its incoming links in a fixed order and chooses the first pertinent packet for each outgoing link.
- Random Round-robin (RR) scan: This scheme is similar to FO scan, except that a switch selects the first incoming link randomly and scans around from that link.
- Farthest-First (FF) selection: In this scheme, a switch scans its input queues in a RR fashion, except that priority is given to packets heading to the farthest destinations for up links, and packets from the farthest sources for down links.

B.3 Simulation results

Since most real parallel computations tend to be dominated by communication time, and algorithms typically can be viewed as consisting of alternating phases of computation and communication, we focus on the static injection model. Here each processor has a fixed number of packets to send, and we measure the time to deliver the full set of messages. This scenario corresponds to the situation analyzed theoretically in Section A.1, to which we compare our empirical results, and constitutes an important general model as argued in [28], for example. We consider three communication patterns representative of the range of likely patterns in real parallel computations:

- Random Instance: Each packet chooses a destination randomly and uniformly.
- Complement Permutation: Each processor $(0, a)$ sends a packet to processor $(0, N-1-a)$. This permutation induces as high a congestion on the fat-tree as any other permutation. The congestion created by this permutation is $\sqrt{N}/2$.
- Many-to-1 Instance: Packets are sent from processors $(0,0), \cdots, (0, N/2-1)$ to processor $(0, N-1)$, and packets from processors $(0, N/2), \cdots, (0, N-1)$ are sent to processor $(0,0)$. This pattern gives us a high congestion ($c = N/2$) with the same number of packets as for a permutation.

The random pattern is probably the most common in other simulation studies and arises in many practical contexts. For example, a recently studied variation on sample sort begins by randomly redistributing the keys to be sorted [12]. We focus on this pattern in the graphical results presented below. Permutations have also been included in our studies, however, since they comprise a common communication primitive, but some are trivial, whereas the complement is a natural permutation that presents a high congestion. Finally, the many-to-1 pattern models the common operations of broadcast or census, which involve "hot-spot contention" that can give substantially different behavior than more uniform traffic patterns [25].

Five network sizes have been considered: $N = 16$, 64, 256, 1024, 4096. For each run, we measure the *maximum communication latency* which is the time elapsed after the routing has begun until the tail of the last packet arrives at its destination. In figures 2 – 5, each point represents the average of 30 runs. Error bars showing the 99% confidence interval for the true average value of the maximum latency are also included in Figure 2, but they are omitted from the other plots to ease readability. In all cases where we draw distinctions in performance, there is little or no overlap of the error bars. We describe these plots and the principle conclusions below; additional plots including other combinations of routing strategies, different parameters (such as queue size and worm size), and error bars can be found in [22].

The queue size q of *WORM* was chosen experimentally. For random instances, little was gained by increasing the queue size beyond 2 flits, and this choice generally yielded better performance than *STORE* with queues of any size tested. In the following, we use queues for 2 flits in *WORM*, and queues for 1 packet in *STORE*. (*STORE* improves somewhat with larger queues, but we are already using more buffer space than for *WORM*.)

First, we compare *STORE* with *UNIV*. Even though *UNIV* is known to achieve an asymptotically optimal time of $(O\,(c + \log N))$ on fat-trees, the delayed greedy routing algorithm *STORE* performed better than *UNIV*, for all of the communication patterns considered. A comparison on random instances is shown in Figure 2. The marked difference here is somewhat surprising since both algorithms use the same value of R and the use of random priorities in *UNIV* seems intuitively somewhat similar to imposing random initial delays.

We tested the effects of initial delays on the latency of *STORE* and *WORM*. We found that the initial random delays can decrease the latency, but we did not find any cases in which they provided much advantage, so we do not use them henceforth.

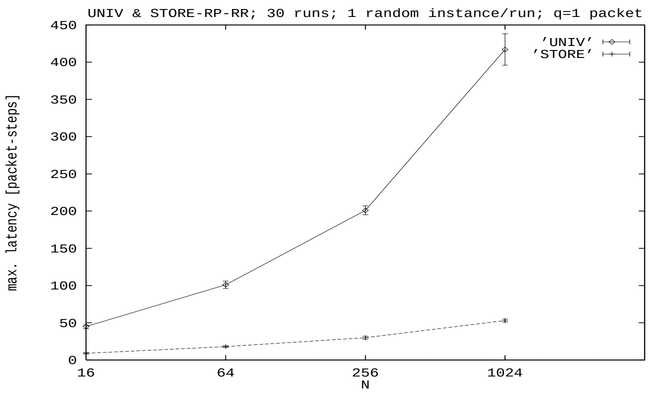

Fig. 2. Performance of *STORE* (with RP and RR): Comparison with *UNIV*. Both algorithms used $R = \log_2 N$.

We also found that the average latency tends to depend linearly on the worm size L. This is consistent with the observation that the total number of packets which may delay a given packet is not a function of L once $L \geq d$. Therefore, except where otherwise noted, we do experiments for only one worm size $L = 32$ flits, a typical value in the literature [24].

Figure 3 compares the path selection schemes for both store-and-forward routing and wormhole routing. Adaptive schemes significantly outperform the fixed-path scheme for the cases we considered. Similar results were obtained with the other packet selection schemes.

Using the best path selection scheme, RP, Figure 4 compares the packet selection schemes. It shows that RR slightly outperforms FO (by 4–8% for most cases). (FF performed similarly to FO.) We henceforth show most of our results with the RR and RP routing schemes. (The GP-FO combination may also be a good choice, though RP-RR outperforms it by 5–9% for *STORE* and 12–15% for *WORM* with $N = 1024$ and $N = 4096$. With GP-FO, we don't have to worry about the difficulty of implementing good randomization schemes, and some programmers prefer deterministic systems.)

Figure 5 compares two approaches for treating the flits in a packet: ordinary wormhole and independent-flit (*SPLIT*) approaches. The performance of *SPLIT* is pretty sensitive to the selection of routing schemes. For example, *SPLIT* with GP-FF uniformly outperforms *SPLIT* with

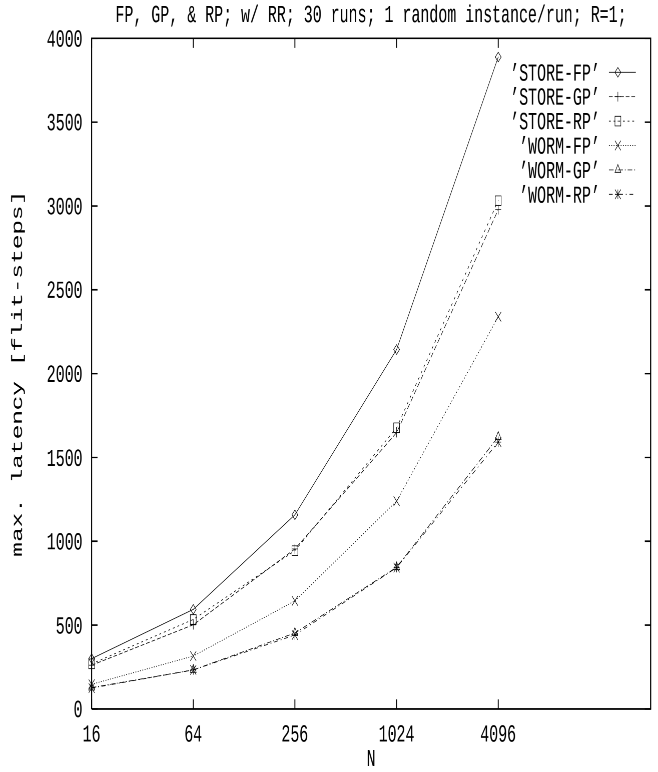

Fig. 3. Comparison of routing schemes on selecting paths in *STORE* and *WORM* with RR scan.

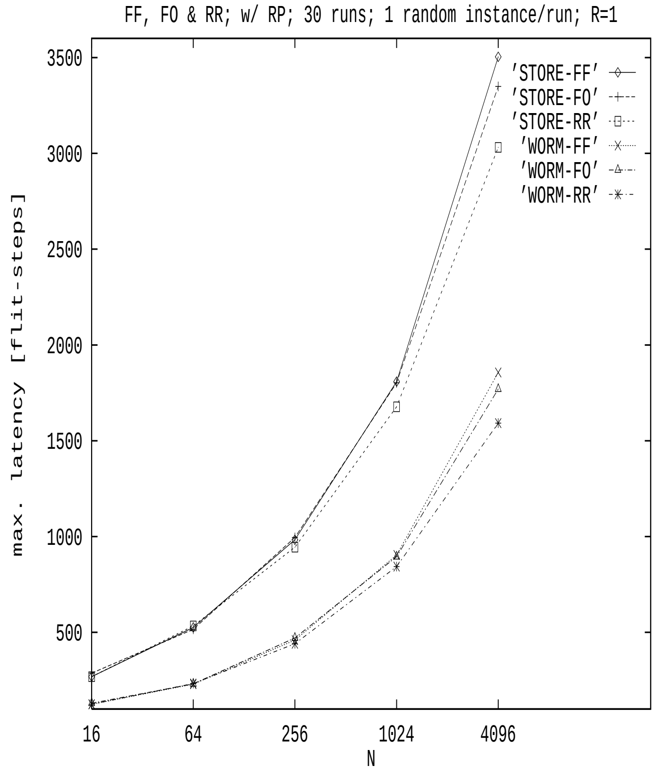

Fig. 4. Comparison of routing schemes on scanning input queues in *STORE* and *WORM* with RP selection.

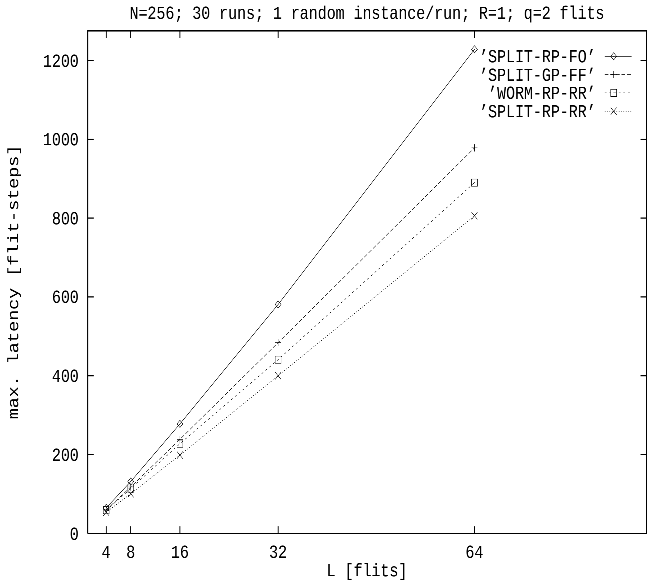

Fig. 5. Comparison of routing schemes on treating flits: *WORM* (with RP and RR) and *SPLIT* (with various strategies).

RP-FO, which was not observed for *STORE*. We found that *WORM* with RP-RR outperforms *SPLIT* with RP-FO and *SPLIT* with GP-FF, and *SPLIT* with RP-RR performs slightly better than *WORM* with RP-RR. This comparison is, however, made without considering the addressing information to be added to each individual flit in the original packet. From Figure 5, we can expect that even a slight increase in the number of flits sent by *SPLIT* (due to the replication of addressing information) would cause *WORM* to outperform *SPLIT*.

Table I compares the average latencies of *WORM* and *STORE* for various conditions. For all cases considered, *WORM* outperforms *STORE*.

Finally, we investigated the experimental upper bound of the (maximum) latency for *WORM* with RP-RR. Table II summarizes, for the random instances, the average values over 30 runs of c and of latency divided by c. We sought the best least-squares fit to the latency divided by c, in the form of $k \log_4^p N$ for constants k and p. The best fit was obtained with $p \approx 0.22$. It would be rash to conclude that latency fits very closely to $\Theta(cL\log^{0.22} N)$, since we have neglected lower order terms, and the network sizes used may be too small to observe the proper asymptotic bound. Nonetheless, it appears that performance superior to our provable bound in Corollary 6 and close to the obvious lower bound ($\Omega(cL+d)$) is attainable for random instances

	STORE-RP-RR			WORM-RP-RR		
N	random	complement	many-to-1	random	complement	many-to-1
16	269	198	544	125	68	258
64	534	442	2144	233	161	1028
256	944	829	8352	441	301	4102
1024	1677	1565	32992	843	583	16392
4096	3031	2896	131360	1592	1123	65546

Table I. Average latency, in flit-steps, of the greedy store-and-forward ($STORE$ with $R = 1$ and $q = 1$) and the greedy wormhole ($WORM$ with $R = 1$ and $q = 2$) algorithms. Each value represents an average of 30 independent experiments.

N	c	Latency/c
16	3.5	35.6
64	5.6	41.9
256	10.2	43.4
1024	18.6	45.3
4096	34.3	46.4

Table II. The average values of c and of the latency divided by c, for $WORM$ with RP-RR. Each value represents 30 independent experiments with $L = 32$, $R = 1$, and $q = 2$ flits.

in practice.

References

[1] B. Aiello, T. Leighton, B. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 55–64. Association for Computing Machinery, 1990.

[2] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. Computers*, 39(6):775–785, June 1990.

[3] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Computers*, C-36(5):547–553, May 1987.

[4] J. T. Draper and J. Ghosh. A comprehensive analytical model for wormhole routing in multicomputer systems. *Journal of Parallel and Distributed Computing*, 23:202–214, 1994.

[5] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. In D. P. Agrawal, editor, *Proceedings of the 1994 International Conference on Parallel Processing*, pages I-142–I-149. CRC Press, 1994.

[6] S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. *IEEE Trans. Computers*, 45(6):704–713, June 1996.

[7] R. I. Greenberg. The fat-pyramid: A robust network for parallel computation. In W. J. Dally, editor, *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*, pages 195–213. MIT Press, 1990.

[8] R. I. Greenberg. The fat-pyramid and universal parallel computation independent of wire delay. *IEEE Trans. Computers*, 43(12):1358–1364, Dec. 1994.

[9] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In S. Micali, editor, *Randomness and Computation.* Volume 5 of *Advances in Computing Research*, pages 345–374. JAI Press, 1989.

[10] R. I. Greenberg and H.-C. Oh. Packet routing in networks with long wires. *Journal of Parallel and Distributed Computing*, 31(2):153–158, Dec. 1995.

[11] F. T. Hady. *A Performance Study of Wormhole Routed Networks Through Analytical Modeling and Experimentation.* PhD thesis, University of Maryland Electrical Engineering Department, 1993.

[12] D. R. Helman, D. A. Bader, and J. JáJá. A parallel sorting algorithm with an experimental study. Technical Report UMIACS-TR-95-102, University of Maryland Institute for Advanced Computer Studies, Dec. 1995.

[13] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, Sept. 1979.

[14] M. Kunde and T. Tensi. Multi-packet-routing on mesh connected arrays. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*,

pages 336–343. Association for Computing Machinery, 1989.

[15] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, July 1994.

[16] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–180, 1994.

[17] T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 2–10. Association for Computing Machinery, 1990.

[18] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Computers*, C-34(10):892–901, Oct. 1985.

[19] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak. The network architecture of the connection machine CM-5. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285. Association for Computing Machinery, 1992.

[20] F. Makedon and A. Simvonis. On bit-serial packet routing for the mesh and the torus. In J. JaJa, editor, *Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation*, pages 294–302. IEEE Computer Society Press, 1990.

[21] J. Y. Ngai. A framework for adaptive routing in multicomputer networks. Technical Report CS-TR-89-09, Department of Computer Science, California Institute of Technology, 1989.

[22] H.-C. Oh. *Efficient Communication Schemes for massively Parallel Computers.* PhD thesis, University of Maryland Electrical Engineering Department, 1993.

[23] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, 1984.

[24] M. J. Pertel. A critique of adaptive routing. Technical Report CS-TR-92-06, Department of Computer Science, California Institute of Technology, 1992.

[25] G. F. Pfister and V. A. Norton. Hot spot contention and combining in multistage interconnection networks. *IEEE Trans. Computers*, C-34(10):943–948, Oct. 1985.

[26] A. Ranade, S. Schleimer, and D. S. Wilkerson. Nearly tight bounds for wormhole routing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 347–355. IEEE Computer Society Press, 1994.

[27] C. Thompson and H. Kung. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 20(4):263–271, Apr. 1977.

[28] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, Aug. 1990.

Ronald I. Greenberg (S'87-M'90) received the A.B. degree in Mathematics, the B.S. degree in Computer Science and the B.S. and M.S. degrees in Systems Science and Mathematics all from Washington University, St. Louis, MO in 1983. He received the Ph.D. degree in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology in 1989.

He is currently an Associate Professor in the Department of Mathematical and Computer Sciences at Loyola University Chicago. His research interests include parallel computation and algorithms for computer-aided design of integrated circuits.

Dr. Greenberg is a member of ACM and SIAM.

Hyeong-Cheol Oh (M'95) received the B.S. degree in Electronics Engineering from Seoul National University, Seoul, Korea in 1982 and the M.S. degree in Electrical and Electronic Engineering from Korea Advanced Institute of Science and Technology, Seoul, Korea in 1984. He received the Ph.D. degree in Electrical Engineering at the University of Maryland, College Park in 1993.

He is currently an Assistant Professor in the Department of Information Engineering, Korea University, Chochiwon, Korea. He also worked for three years at Goldstar Semiconductor Ltd, Korea, where he designed NMOS full-custom and CMOS Gate-Array ICs. His research interests include parallel computation and VLSI design.

Dr. Oh is a member of the Korea Information Science Society.