12-1993

Universal Wormhole Routing

Ronald I. Greenberg
Loyola University Chicago, Rgreen@luc.edu

Hyeong-Cheol Oh

Universal Wormhole Routing *
Prepub version for Fifth IEEE Symposium on Parallel and Distributed Processing, 1993, pages 56–63.

Ronald I. Greenberg and H.-C. Oh
Electrical Engineering Department
University of Maryland
College Park, MD 20742

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, 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, but that performance superior to our provable bound is attainable in practice.

1 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., [14, 11] 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 are composed of *flits* or *flow control digits*, and packets snake through the network one flit after another.

Only a few works have performed any theoretical analysis of wormhole routing or similar schemes. Leighton [13] performs average-case analysis of greedy cut-through routing on meshes. But cut-through routing [9] 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 [17] 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 [4] have obtained a simple, oblivious algorithm for wormhole routing of permutations on the butterfly and the mesh.

While previous 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 [14, 12, 11]. We require only that any two packet paths intersect in at most one contiguous sequence of edges. (This condition is always satisfied in networks that have a unique path between each pair of processors, and the condition can be easily satisfied in many other networks by choosing the paths for packets appropriately.)

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 [7]) of 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 [14, 11], such an approach

*Supported in part by NSF grant CCR-9109550

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 [7] 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 [8] for the store-and-forward model). A flit is an atomic objects, 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. 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 [14, 11], have no cycle in $\mathcal{D}$ for any set of packets. Also there are techniques for breaking cycles [3].)

2 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, and any two paths intersect in at most one contiguous sequence 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.

Following is a key lemma showing that sums of random variables with a binomial distribution are unlikely to greatly exceed their expected values.

Lemma 1 *Let X have a binomial distribution with density $f_X(x; K, p)$,[1] and let S_n be the sum $X_0 + X_2 + \cdots + X_{n-1}$ of n independent random variables distributed as X. Let m be a value greater than or equal to nKp. Then*

$$\Pr\{S_n \geq m\} \leq e^{-\frac{(m-nKp)^2}{2m}}.$$

Proof. The proof uses Chernoff's general bound on the sum of independent identically distributed random variables [2] and will appear in the full paper. ■

We now use a delayed-greedy approach similar to that of Felperin, Raghavan, and Upfal [4]. Each packet chooses an integral delay randomly and uniformly from the interval $[0, R-1]$, where R is to be determined later. Let $T_1 = \max\left\{\frac{d}{L}, \log P\right\}$ and $\eta = \min\{d, L\}$. A packet that is assigned delay x waits in its initial queue for xkT_1L steps and then proceeds to its destination, for some constant k.

Theorem 2 *Any set of P packets can be routed in $O(cd\eta + cL\eta \log P)$ flit-steps with high probability.*

[1] $f_X(x; K, p) = \begin{cases} \binom{K}{x} p^x (1-p)^{K-x} & \text{for } x = 0, 1, \cdots, K \\ 0 & \text{otherwise} \end{cases}$

Proof. We refer to the time from xkT_1L to $(x+1)kT_1L$ as the x-th phase, and we show that for any given worm W, the probability is at most $1/P^2$ that the worm fails to reach its destination by the end of the phase in which it enters the network (under the assumption that all worms dispatched in previous phases have been delivered). This will yield a probability of at most $1/P$ that there exists any worm that does not get delivered during its phase. Without loss of generality, we assume that W is sent in the phase starting at time 0, and we henceforth ignore any worms that are not dispatched in this phase.

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. Once worm W' delays worm W for at most L steps (not necessarily consecutive), worms W' and W take separate paths or W follows W', i.e., W' will not delay W again. Note also that when $r = 1$, we say that W delays itself, even though W actually moves in that case.

Now we count how many worms can delay W before it reaches its destination. Let Δ_t denote the set of all worms that delay W strictly before time t. Then let D_t be the union of Δ_t and all worms that traverse an edge where some worm in Δ_t blocks the previous worm in its delay chain. Also, let $D_0 = \{W\}$.

We now consider a given time step t. Let A be the worm which is in D_t and would delay W at t if W were not delayed by any worm outside D_t. Let e be the next edge which A has to traverse. Then, one of the following events will occur at t:

1. A worm, A', outside D_t delays A. (Note that e may or may not be the edge, e', which A' traverses at t.) In this case, we let D_{t+1} be comprised of the worms in D_t and all the worms that traverse e'. We make the conservative assumption that all of these new worms contribute to the delay of W one after another.

2. No worm outside D_t takes e. In this case, A will start traversing e.

Let B_t be the number of worms in $D_{t+1} - D_t$. Then B_t is dominated by a binomial distribution with density $f_B(b; c, \frac{1}{R})$.

Now, suppose W has reached its destination by time τ. Then $\tau \leq L\sum_{i=0}^{\tau-1} B_i + d + L$, since W is at most delayed for L steps by each of the $\sum B_i$ worms, and an additional $d + L$ steps suffices for all the flits of W to reach their destination. Let $k = 10$ and $\tau = 10T_1L$. The value of R depends on the condition on d and L.

When $L < d$, we choose $R = 5cL$. Since the B_i's are (at worst) distributed as B which has a binomial distribution with density $f_B(b; c', \frac{1}{R})$ with $c' \leq c$, by Lemma 1,

$$\Pr\left\{\sum_{i=0}^{\tau-1} B_i \geq 8T_1\right\} \leq e^{-2\log P} .$$

This implies that, with probability $1 - 1/P^2$, worm W is delayed by at most $8T_1$ worms during the phase, and all of its flits reach their destination by time $8T_1L + d + L \leq \tau$. Thus the routing can be done in $O\left(cdL + cL^2 \log P\right)$ flit-steps with high probability.

When $d \leq L$, we choose $R = 5cd$. For $d = L$, we can show that at most $8T_1$ worms delay a given worm W during the phase with high probability, because $\Pr\left\{\sum_{i=0}^{\tau-1} B_i \geq 8T_1\right\} \leq e^{-\frac{\left(8T_1 - 2T_1\frac{L}{d}\right)^2}{16T_1}} \leq e^{-2\log P}$ when $L \leq d$. This number of packets that delay W for $L = d$ does not change even if we increase the worm length, because any worm which delays W can be delayed by another worm only before its head reaches its destination. Since $8T_1$ worms can delay W for at most $8T_1L$ steps, the probability is at least $1 - 1/P^2$ that W reaches its destination within $10T_1L$ steps. Thus the routing can be done in $O\left(cd^2 + cLd \log P\right)$ flit-steps with high probability. ■

We also have the following corollary to Theorem 2, which is useful in Section 3.1:

Corollary 3 *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.* ■

3 Wormhole routing on fat-trees

Fat-trees constitute a class of routing networks for hardware-efficient parallel computation [15, 7, 14]. Figure 1 shows a layout of one variant of fat-trees, which uses 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 [16]. 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 *links*, each capable of transmitting one flit in unit time. We call the link from parent to child

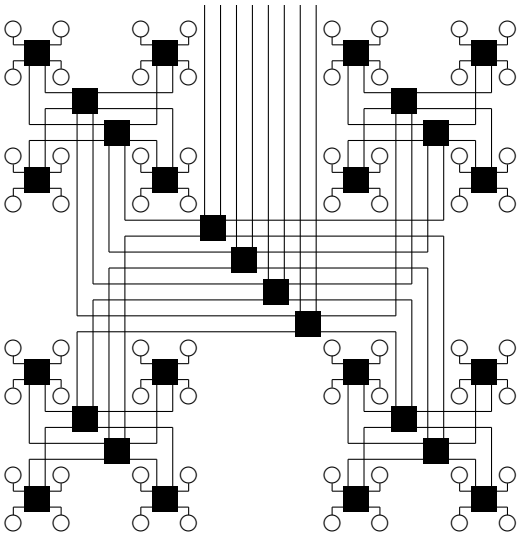

Figure 1: A butterfly fat-tree.

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 $c = O(\lambda + \log N)$ [14, 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. Also, any two paths selected in this way do not intersect in more than one contiguous sequences of edges. Hence the result of Section 2 can be applied.

3.1 Area-universality of fat-trees

3.1.1 Worms with a fixed length

The algorithm analyzed in Section 2 allows us to extend to the wormhole routing problem universality theorems from [15, 7, 14, 6] 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 [6]. (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 [6, 5].) 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 [6, Lemma 2.1].

Lemma 4 *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 5 *A fat-tree F of area $\Theta(A)$ can simulate any network of area A with a factor of $O\left(\log^3 A\right)$ loss of runtime efficiency, using on-line wormhole routing.*

Proof. Consider the set of packets that moves during L flit steps in a competing network of area A. By extending Lemma 4 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 Theorem 3 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. [14, 11] 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 5, 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.

3.1.2 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 the mean $E[L] = \bar{L}$, the variance $var[L] = \sigma_L^2$, and the maximum L_M.

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. [18].

The full paper will prove the following theorem:

Theorem 6 *Consider a set, $\mathcal{R}$, of networks with area A. Suppose that each processor continuously generates and sends packets during a time interval of length $\Delta T \geq AL_M$. Let L_M be bounded above by a polynomial in A. Then a fat-tree F of area $\Theta(A)$ can simulate any network of area A with a factor of $O\left((L_M/\bar{L}) \log^3 A\right)$ loss of runtime efficiency, using on-line wormhole routing.*

3.2 Simulation

This section investigates the practical performance of wormhole routing algorithms on butterfly fat-trees. We only consider the case in which all packets have a fixed length L.

3.2.1 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 [20]. 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 $(l+1, \lfloor \frac{a}{2^{l+1}} \rfloor \cdot 2^l + (a + 2^{l-1}) \bmod 2^l)$.

3.2.2 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 [14] for leveled networks.

Algorithm *SPLIT* uses the independent-flit approach. Each packet is split into flits which are treated as independent packets and routed as in *STORE*.

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.

3.2.3 Simulation results

We consider only the static injection model in which every processor has a fixed number of packets to inject.

The communication patterns we consider are:

- 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 high congestion ($c = N/2$) with the same number of packets as for a permutation.

Four network sizes have been tested: $N = 16$, 64, 256, 1024. Experiments on networks of larger sizes are being conducted.

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 30 runs. The average values are connected with lines and the deviation at each point is indicated by an error bar. We also note in some figures the load factor λ.

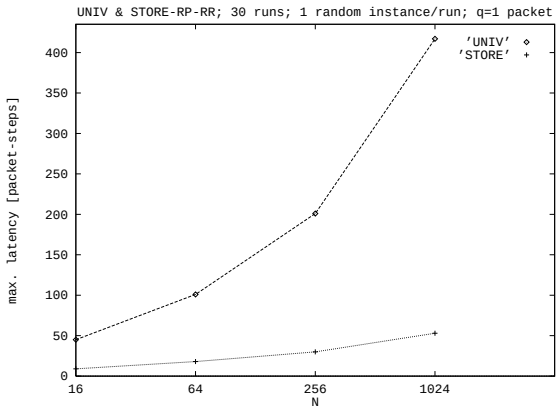

Figure 2: Performance of *STORE* (with RP and RR): Comparison with *UNIV*. Both algorithms used $R = \lg N$.

The queue size 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 compared *STORE* with *UNIV*. Even though *UNIV* is known to be asymptotically optimal ($O(c + \log N)$) on fat-trees, the 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.

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.

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 $L \geq d$, as we mentioned in the proof of Theorem 2. Therefore, except where otherwise noted, we do experiments for only one worm size $L = 32$ flits.

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

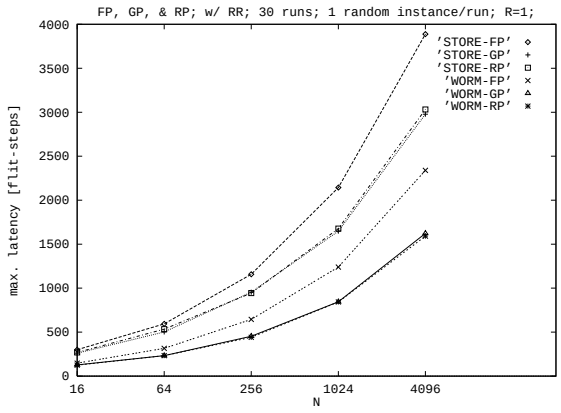

Figure 3: Comparison of routing schemes on selecting paths in *STORE* and *WORM* with RR scan.

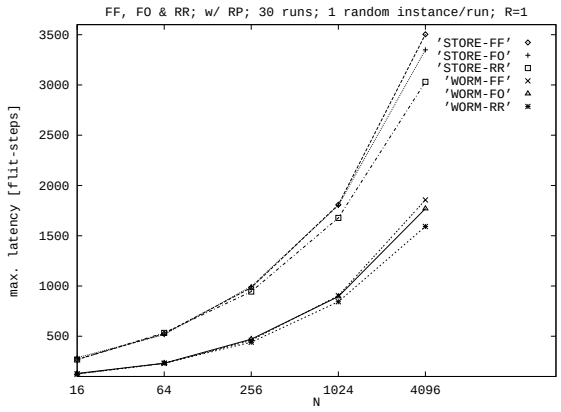

Figure 4: Comparison of routing schemes on scanning input queues in *STORE* and *WORM* with RP selection.

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 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*.

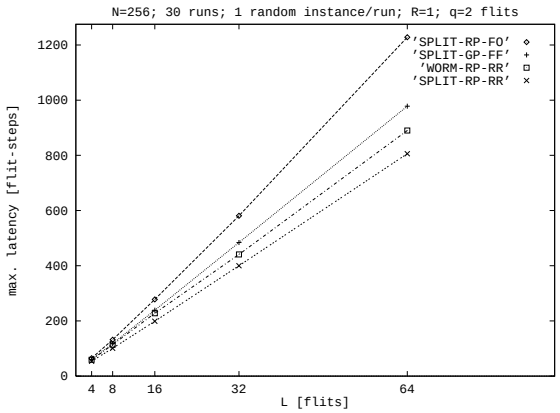

Figure 5: Comparison of routing schemes on treating flits: *WORM* (with RP and RR) and *SPLIT* (with various strategies).

Table 1 compares the average latencies of *WORM* and *STORE* for various conditions. For all cases considered, *WORM* outperforms *STORE*.

Using the measured congestion and average latencies for *WORM* with RP and RR on the random message patterns, we sought a best fit to the routing time in the form $kcL \log_4^p N$ for constants k and p. Using data for $N = 16$, 64, 256, 1024, and 4096, the best least-squares fit was obtained with $p = 1.7$, which is a better growth rate than would be expected from our proven bound.

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] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of

N	STORE-RP-RR			WORM-RP-RR		
	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

Table 1: Average latency, in flit-steps, of the greedy store-and-forward (*STORE* with $R = 0$ and $q = 1$) and the greedy wormhole (*WORM* with $R = 0$ and $q = 2$) algorithms. Each value represents an average of 30 experiments. The average load factors of the random instances are $2.9, 4.4, 6.9, 12.9$, for $n = 16, 64, 256, 1024$, respectively.

observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[3] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1:187–196, 1986.

[4] S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. Manuscript, 1993. Earlier version in *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*.

[5] R. I. Greenberg. The fat-pyramid and universal parallel computation independent of wire delay. *IEEE Trans. Computers*. To appear.

[6] 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, Apr. 1990.

[7] 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.

[8] R. I. Greenberg and H.-C. Oh. Packet routing in networks with long wires. In *Proceedings of the 30th Annual Allerton Conference on Communication, Control, and Computing*, pages 664–673, 1992. Revised version (University of Maryland technical report CS-TR-3044 and UMIACS-TR-93-22) submitted to *Journal of Parallel and Distributed Computing*.

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

[10] 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.

[11] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. Manuscript, 1991. To appear in *Journal of Algorithms*.

[12] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. Manuscript, 1991. To appear in *Combinatorica*.

[13] 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.

[14] T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 256–269. IEEE Computer Society Press, 1988.

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

[16] 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.

[17] 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.

[18] 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.

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

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