Randomized Routing on Fat-Trees

Ronald I. Greenberg
Charles E. Leiserson

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

June 13, 1996

Abstract

Fat-trees are a class of routing networks for hardware-efficient parallel computation. This paper presents a randomized algorithm for routing messages on a fat-tree. The quality of the algorithm is measured in terms of the *load factor* of a set of messages to be routed, which is a lower bound on the time required to deliver the messages. We show that if a set of messages has load factor λ on a fat-tree with n processors, the number of delivery cycles (routing attempts) that the algorithm requires is $O(\lambda + \lg n \lg\lg n)$ with probability $1 - O(1/n)$. The best previous bound was $O(\lambda \lg n)$ for the off-line problem in which the set of messages is known in advance. In the context of a VLSI model that equates hardware cost with physical volume, the routing algorithm can be used to demonstrate that fat-trees are universal routing networks. Specifically, we prove that any routing network can be efficiently simulated by a fat-tree of comparable hardware cost.

1 Introduction

Fat-trees constitute a class of routing networks for general-purpose parallel computation. This paper presents a randomized algorithm for routing a set of messages on a fat-tree. The routing algorithm and its analysis generalize an earlier *universality* result by showing, in a three-dimensional VLSI model, that any network can be efficiently simulated by a fat-tree of comparable volume. The result had been proved only for *off-line* simulations [12], where the communication pattern is known in advance; this paper extends it to the more interesting *on-line* case, where messages are spontaneously generated by processors.

As is illustrated in Figure 1, a fat-tree is a routing network based on Leighton's tree-of-meshes graph [8]. A set of n processors are located at the leaves of a complete binary

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014–80–C–0622 and in part by the Office of Naval Research under Contract N00014–86–K–0593. Ron Greenberg is supported in part by a Fannie and John Hertz Foundation Fellowship. Charles Leiserson is supported in part by an NSF Presidential Young Investigator Award.

Figure 1: The organization of a fat-tree. Processors are located at the leaves, and the internal nodes contain concentrator switches. The channels between internal nodes consist of bundles of wires.

tree. Each edge of the underlying tree corresponds to two *channels* of the fat-tree: one from parent to child, the other from child to parent. Unlike a normal tree which is "skinny all over," in a fat-tree, each channel consists of a bundle of wires. The number of wires in a channel c is called its capacity, denoted by $\mathrm{cap}(c)$. Each internal node of the fat-tree contains circuitry that switches messages from incoming to outgoing channels.

The channel capacities of a fat-tree determine the amount of hardware required to build it. The greater the capacities of the channels, the greater the communication potential, and also, the greater the hardware cost of an implementation of the network. The idea of fat-trees is to take advantage of a principle of locality in much the same way as does the telephone network: by using only slightly more hardware than that required to support fast *nonlocal* communication among a set of processors, much additional *local* communication among a larger set of processors can be supported.

An issue that any routing algorithm for a fat-tree must face is that some communication patterns among the processors are harder than others. For example, suppose the channel capacity between the two halves of a fat-tree is $\Theta(n^{2/3})$, where n is the number of processors, and suppose each processor sends a message to a processor in the other half. Since the number of messages that must pass through the root is n and the capacity is $\Theta(n^{2/3})$, the time required by the network to deliver all the messages is $\Omega(n^{1/3})$ because of congestion. In contrast, there are many communication patterns among n processors with less nonlocal communication that can be implemented in subpolynomial time.

The routing algorithm for fat-trees presented in this paper is a randomized algorithm which we analyze in terms of a measure of congestion called the *load factor*. The load factor of a set of messages is the largest ratio over all channels in the fat-tree of the number of messages that must pass through the channel divided by the capacity of the channel. The load factor of a set of messages is thus a lower bound on the time to deliver the messages.

We show in this paper that our routing algorithm can deliver a set of messages with load factor λ in $O(\lambda + \lg n \lg\lg n)$ delivery cycles (routing attempts) with high probability. The best previous bound for a problem of this nature was an $O(\lambda \lg n)$ bound for the off-line situation where the set of messages is known in advance [12] and the problem is to schedule

their delivery. The analysis of our randomized routing algorithm makes no assumptions about the statistical distribution of messages, except insofar as it affects the load factor. Moreover, the algorithm is not restricted to permutation routing or situations where each processor can only send or receive a constant number of messages, as is common in the literature. We consider the general situation where each processor can send and receive polynomially many messages.

Our routing algorithm also differs from others in the literature in the way randomization is used. Unlike the algorithms of Valiant [18], Valiant and Brebner [19], Aleliunas [2], Upfal [17] and Pippenger [14], for example, it does not randomize with respect to paths taken by messages. For example, Valiant's classic scheme for routing on a hypercube sends each message to a randomly chosen intermediate destination and, from there, to its true destination. On a fat-tree, such a technique would likely convert communication patterns with good locality into ones with much global communication. Instead of choosing random paths for messages to traverse, our algorithm repeatedly attempts to deliver a randomly chosen subset of the messages. A by-product of this strategy is that our algorithm requires no intermediate buffering of messages.

The remainder of this paper is organized as follows. Section 2 describes fat-trees in more detail. Section 3 presents the randomized algorithm for efficiently routing messages on the fat-tree network, and Section 4 contains the full analysis of the algorithm. Section 5 proves that fat-trees are universal in VLSI models. Specifically, we use the randomized routing algorithm to show that a fat-tree with properly chosen channel capacities can efficiently simulate any other network of comparable hardware cost, where cost is measured as the area or volume of the circuitry. Section 6 gives an existential lower bound for a class of naive greedy routing algorithms which shows that the greedy strategy is inferior to our randomized algorithm for worst case inputs. Section 7 contains several additional results. These include a modification of the routing algorithm that achieves better bounds when each channel has capacity $\Omega(\lg n)$, a new, simpler fat-tree design, and results on off-line routing. Finally, Section 8 contains some concluding remarks.

2 Fat-trees

This section describes fat-trees in more detail. We give more specific implementation details on our routing strategy and the hardware required to support it. We precisely define the *load factor* of a set of messages on a general network in terms of cuts of the network, and we prove that it suffices in a fat-tree to consider only the load factors on channels.

The implementation of fat-trees described here follows that of [12]. We consider communication through the fat-tree network to be synchronous, bit serial, and batched. By synchronous, we mean that the system is globally clocked. By bit serial, we mean that the messages can be thought of as bit streams. Each message snakes its way through the wires and switches of the fat-tree, with leading bits of the message setting switches and establishing a path for the remainder to follow. By batched, we mean the messages are grouped into *delivery cycles*. During a delivery cycle, the processors send messages through the network. Each message attempts to establish a path from its source to its destination. Since some messages may be unable to establish connections during a delivery cycle, each

successfully delivered message is acknowledged through its communication path at the end of the cycle. Rather than buffering undelivered messages, we simply allow them to try again in a subsequent delivery cycle. The routing algorithm is responsible for grouping the messages into delivery cycles so that all the messages are delivered in as few cycles as possible.

The mechanics of routing messages in a fat-tree are similar to routing in an ordinary tree. For each message, there is a unique path from its source processor to its destination processor in the underlying complete binary tree, which can be specified by a relative address consisting of at most $2 \lg n$ bits telling whether the message turns left or right at each internal node. Within each node of the fat-tree, the messages destined for a given output channel are *concentrated* onto the available wires of that channel. This concentration may result in "lost" messages if the number of messages destined for the output channel exceeds the capacity of the channel. We assume, however, that the concentrators within the node are ideal in the sense that no messages are lost if the number of messages destined for a channel is less than or equal to the capacity of the channel. Such a concentrator can be built, for example, with a logarithmic-depth sorting network [1]. A somewhat more practical logarithmic-depth circuit can be built by combining a parallel prefix circuit [7] with a butterfly (i.e., FFT, Omega) network. With switches of logarithmic depth, the time to run each delivery cycle is $O(\lg^2 n)$ bit times, making the natural assumption that messages are $O(\lg n)$ bits long.[1] (Section 7 contains another fat-tree design where the time to run a delivery cycle is $O(\lg n)$ bit times.)

The performance of any routing algorithm for a fat-tree depends on the locality of communication inherent in a set of messages. The locality of communication for a message set M can be summarized by a measure $\lambda(M)$ called the *load factor*, which we define in a more general network setting.

> **Definition:** Let R be a routing network. A set S of wires in R is a (directed) *cut* if it partitions the network into two sets of processors A and B such that every path from a processor in A to a processor in B contains a wire in S. The *capacity* $\mathrm{cap}(S)$ is the number of wires in the cut. For a set of messages M, define the *load* $\mathrm{load}(M, S)$ of M on a cut S to be the number of messages in M from a processor in A to a processor in B. The *load factor* of M on S is
>
> $$\lambda(M, S) = \frac{\mathrm{load}(M, S)}{\mathrm{cap}(S)} \ ,$$
>
> and the *load factor* of M on the entire network R is
>
> $$\lambda(M) = \max_S \lambda(M, S) \ .$$

The load factor of a set of messages on a given network provides a lower bound on the time required to deliver all messages in the set.

For fat-trees, only cuts corresponding to channels need be considered to determine the load factor, as is shown by the following lemma.

[1] In this paper, we measure time in terms of bit operations, rather than word operations, to better reflect actual costs.

Lemma 1 *The load factor of a set M of messages on a fat-tree is*

$$\lambda(M) = \max_c \lambda(M, c) \ ,$$

where c ranges over all channels of the fat-tree.

Proof. Any cut S must entirely contain at least one channel. Let us partition the wires in S into $S = c_1 \cup \cdots \cup c_l \cup w$, where $c_1, \ldots, c_l$ are the complete channels in S and w is the set of remaining wires in S. For convenience, let $x_i = \text{load}(M, c_i)$ and $y_i = \text{cap}(c_i)$. Assume without loss of generality that $\lambda(M, c_1) \geq \lambda(M, c_i)$ for $i = 1, \ldots, l$, which implies $x_1 y_i - x_i y_1 \geq 0$ for all i. The load factor of M on S is therefore

$$\begin{aligned}
\lambda(M, S) &= \frac{x_1 + \cdots + x_l}{y_1 + \cdots + y_l + |w|} \\
&= \frac{x_1}{y_1} - \frac{(x_1 y_2 - x_2 y_1) + \cdots + (x_1 y_l - x_l y_1) + (x_1 |w|)}{y_1 (y_1 + \cdots + y_l + |w|)} \qquad (1) \\
&\leq \frac{x_1}{y_1} \\
&= \lambda(M, c_1)
\end{aligned}$$

since each term in the numerator of the second term of (1) is nonnegative. ■

3 The routing algorithm

This section gives our randomized algorithm for routing a set M of messages on a fat-tree. The algorithm *RANDOM*, which is based on routing random subsets of the messages in M, is shown in Figure 2. It uses the subroutine *TRY-GUESS* shown in Figure 3. Section 4 provides a proof that on an n-processor fat-tree, the probability is at least $1 - O(1/n)$ that *RANDOM* delivers all messages in M within $O(\lambda(M) + \lg n \lg\lg n)$ delivery cycles, if the two constants k_1 and k_2 appearing in the algorithm are properly chosen.

The basic idea of *RANDOM* is to pick a random subset of messages to send in each delivery cycle by independently choosing each message with some probability p. This type of message set merits a formal definition.

> **Definition:** A *p-subset* of M is a subset of M formed by independently choosing each message of M with probability p.

We will show in Section 4 that if p is sufficiently small, a substantial portion of the messages in a p-subset are delivered because they encounter no congestion during routing. On the other hand, if p is too small, few messages are sent. *RANDOM* varies the probability p from cycle to cycle, seeking random subsets of M that contain a substantial portion of the messages in M, but that do not cause congestion.

The algorithm *RANDOM* varies the probability p because the load factor $\lambda(M)$ is not known. The overall structure of *RANDOM* is to guess the load factor and call the subroutine *TRY-GUESS* for each one. The subroutine *TRY-GUESS* determines the probability p based on *RANDOM*'s guess λ_{guess} and a parameter r, called the *congestion parameter* of

```
     Algorithm RANDOM
1    send M
2    U ← M − {messages delivered}
3    λ_guess ← 2
4    while k_1 λ_guess < k_2 lg n and U ≠ ∅ do
5        TRY-GUESS(λ_guess)
6        λ_guess ← λ²_guess
7    endwhile
8    λ_guess ← (k_2/k_1) lg n lg lg n
9    while U ≠ ∅ do
10       TRY-GUESS(λ_guess)
11       λ_guess ← 2λ_guess
12   endwhile
```

Figure 2: The randomized algorithm *RANDOM* for delivering a message set M on a fat-tree with n processors. This algorithm achieves the running times in Figure 4 with high probability if the constants k_1 and k_2 are appropriately chosen. Since the load factor $\lambda(M)$ is not known in advance, *RANDOM* makes guesses, each one being tried out by the subroutine *TRY-GUESS*.

```
     procedure TRY-GUESS(λ_guess)
1    λ ← λ_guess
2    while λ > 1 do
3        for i ← 1 to max {k_1 λ, k_2 lg n} do
4            independently send each message of U with probability 1/rλ
5            U ← U − {messages delivered}
6        endfor
7        λ ← λ/2
8    endwhile
9    send U
10   U ← U − {messages delivered}
```

Figure 3: The subroutine *TRY-GUESS* used by the algorithm *RANDOM* which tries to deliver the set U of currently undelivered messages. When $\lambda_{guess} \geq \lambda(U)$, this attempt will be successful with high probability, if the constants k_1 and k_2 are appropriately chosen. (The value r is the congestion parameter of the fat-tree defined in Section 4, which is typically a small constant.) In that case, λ is always an upper bound on $\lambda(U)$, which is at least halved in each iteration of the **while** loop. When the loop is finished, $\lambda(U) \leq 1$, so all the remaining messages can be sent.

load factor	delivery cycles
$0 \le \lambda(M) \le 1$	1
$1 \le \lambda(M) \le 2$	$O(\lg n)$
$2 \le \lambda(M) \le \lg n \lg\lg n$	$O(\lg n \lg(\lambda(M)))$
$\lg n \lg\lg n \le \lambda(M) \le n^{O(1)}$	$O(\lambda(M))$

Figure 4: The number of delivery cycles required to deliver a message set M on a fat-tree with n processors. All bounds are achieved with probability $1 - O(1/n)$. The bounds on the number of delivery cycles can be summarized as $O(\lambda(M) + \lg n \lg\lg n)$.

the fat-tree, which is independent of the message set and which will be defined in Section 4. If λ_{guess} is an upper bound on the true load factor $\lambda(M)$, then with high probability, each iteration of the **while** loop in *TRY-GUESS* halves the upper bound on the load factor $\lambda(U)$ of the set U of undelivered messages, as will be shown in Section 4. When the loop is finished, we have $\lambda(U) \le 1$, and all the remaining messages can be delivered in one cycle. The number of delivery cycles performed by *TRY-GUESS* is $O(\lg \lambda_{guess} \lg n)$ if $2 \le \lambda_{guess} \le \Theta(\lg n)$, and the number of cycles is $O(\lambda_{guess} + \lg n \lg\lg n)$ if $\lambda_{guess} = \Omega(\lg n)$.

RANDOM must make judicious guesses for the load factor because *TRY-GUESS* may not be effective if the guess is smaller than the true load factor. Conversely, if the guess is too large, too many delivery cycles will be performed. Since the amount of work done by *TRY-GUESS* grows as $\lg \lambda_{guess}$ when λ_{guess} is small, and as λ_{guess} when λ_{guess} is large, there are two main phases to *RANDOM*'s guessing. (These phases follow the handling of very small load factors, i. e., $\lambda(M) \le 2$.)

In the first phase, the guesses are squared from one trial to the next. Once λ_{guess} is sufficiently large, we move into the second phase, and the guesses are doubled from one trial to the next. In each phase, the number of delivery cycles run by *TRY-GUESS* from one call to the next forms a geometric series. Thus, the work done in any call to *TRY-GUESS* is only a constant factor times all the work done prior to the call. With this guessing strategy, we can deliver a message set using only a constant factor more delivery cycles than would be required if we knew the load factor in advance.

4 Analysis of the routing algorithm

This section contains the analysis of *RANDOM*, the routing algorithm for fat-trees presented in Section 3. We shall show that the probability is $1 - O(1/n)$ that *RANDOM* delivers a set M of messages on a universal fat-tree with n processors in $O(\lambda(M) + \lg n \lg\lg n)$ delivery cycles. Figure 4 gives the tighter bounds that we actually prove.

We begin by stating two technical lemmas concerning basic probability. One is a combinatorial bound on the tail of the binomial distribution of the kind attributed to Chernoff [4], and the other is a general, but weak, bound on the probability that a random variable takes on values smaller than the expectation.

The first lemma is the Chernoff bound. Consider t independent Bernoulli trials, each with probability p of success. It is well known [5] that the probability that there are at

least s successes out of the t trials is

$$B(s,t,p) = \sum_{k=s}^{t} \binom{t}{k} p^k (1-p)^{t-k} \ .$$

The lemma bounds the probability that the number of successes is larger than the expectation pt.

Lemma 2

$$B(s,t,p) \leq \left(\frac{ept}{s}\right)^s \ .$$

Proof. The lemma follows from [18, p. 354]. ■

The second technical lemma bounds the probability that a bounded random variable takes on values smaller than the expectation.

Lemma 3 *Let $X \leq b$ be a random variable with expectation μ. Then for any $w < \mu$, we have*

$$\Pr\{X \leq w\} \leq 1 - \frac{\mu - w}{b - w} \ .$$

Proof. The definition of expectation gives us

$$\mu \leq w \Pr\{X \leq w\} + b(1 - \Pr\{X \leq w\}) \ ,$$

from which the lemma follows. ■

We now analyze the routing of a p-subset M' of a set M of messages. If the number $\mathrm{load}(M', c)$ of messages in M' that must pass through c is no more than the capacity $\mathrm{cap}(c)$, then no messages are lost by concentrating the messages into c. We shall say that c is *congested by* M' if $\mathrm{load}(M', c) > \mathrm{cap}(c)$. The next lemma shows that the likelihood of channel congestion decreases exponentially with channel capacity if the probability of choosing a given message in M is sufficiently small.

Lemma 4 *Let M be a set of messages on a fat-tree, let $\lambda(M)$ be the load factor on the fat-tree due to M, let M' be a p-subset of messages from M, and let c be a channel through which a given message $m \in M'$ must pass. Then the probability is at most $(ep\lambda(M))^{\mathrm{cap}(c)}$ that channel c is congested by M'.*

Proof. Channel c is congested by M' if $\mathrm{load}(M', c) > \mathrm{cap}(c)$. There is already one message from the set M' going through channel c, so we must determine a bound on the probability that at least $\mathrm{cap}(c)$ other messages go through c. Using Lemma 2 with $s = \mathrm{cap}(c)$ and $t = \mathrm{load}(M, c)$, the probability that the number of messages sent through channel c is greater than the capacity $\mathrm{cap}(c)$ is less than

$$\begin{aligned} B(\mathrm{cap}(c), \mathrm{load}(M,c), p) &\leq \left(\frac{ep\,\mathrm{load}(M,c)}{\mathrm{cap}(c)}\right)^{\mathrm{cap}(c)} \\ &\leq (ep\lambda(M))^{\mathrm{cap}(c)} \ . \end{aligned}$$

■

The next lemma will analyze the probability that a given message of a p-subset of M gets delivered. In order to do the analysis, however, we must select p small enough so that it is likely that the message passes exclusively through uncongested channels. The choice of p depends on the capacities of channels in the fat-tree. For convenience, we define a parameter of the capacities that will enable us to choose a suitable upper bound for p.

Definition: The *congestion parameter* of a fat-tree is the smallest positive value r such that for each simple path $c_1, c_2, \ldots, c_l$ of channels in the fat-tree, we have

$$\sum_{k=1}^{l} \left(\frac{e}{r}\right)^{\mathrm{cap}(c_k)} \leq \frac{1}{2} .$$

The congestion parameter is generally quite small. For any fat-tree based on a complete binary tree, the longest simple path is at most $2 \lg n$, where n is the number of processors, and thus we have $r \leq 4e \lg n$. For universal fat-trees (discussed in Section 5), the congestion parameter is a constant because the capacities of channels grow exponentially as we go up the tree. (All we really need is arithmetic growth in the channel capacities.) The congestion parameter is also constant for any fat-tree based on a complete binary tree if all the channels have capacity $\Omega(\lg \lg n)$. Our analysis of *RANDOM* treats the congestion parameter r as a constant, but the analysis does not change substantially for other cases.

We now present the lemma that analyzes the probability that a given message gets delivered.

Lemma 5 *Let M be a set of messages on a fat-tree that has congestion parameter r, let $\lambda(M)$ be the load factor on the fat-tree due to M, and let m be an arbitrary message in M. Suppose M' is a p-subset of M, where $p \leq 1/r\lambda(M)$. Then if M' is sent, the probability that m gets delivered is at least $\frac{1}{2}p$.*

Proof. The probability that $m \in M$ is delivered is at least the probability that $m \in M'$ times the probability that m passes exclusively through uncongested channels. The probability that $m \in M'$ is p, and thus we need only show that, given $m \in M'$, the probability is at least $\frac{1}{2}$ that every channel through which m must pass is uncongested. Let $c_1, c_2, \ldots, c_l$ be the channels in the fat-tree through which m must pass. The probability that channel c_k is congested is less than $(e/r)^{\mathrm{cap}(c_k)}$ by Lemma 4. The probability that at least one of the channels is congested is, therefore, less than

$$\sum_{k=1}^{l} \left(\frac{e}{r}\right)^{\mathrm{cap}(c_k)} \leq \frac{1}{2} ,$$

by definition of the congestion parameter. Thus, the probability that none of the channels are congested is at least $\frac{1}{2}$. ■

We now focus our attention on *RANDOM* itself. The next lemma analyzes the innermost loop (lines 3–6) of *RANDOM*'s subroutine *TRY-GUESS*. At this point in the algorithm, there is a set U of undelivered messages and a value for λ. The lemma shows

that if λ is indeed an upper bound on the load factor $\lambda(U)$ of the undelivered messages when the loop begins, then $\lambda/2$ is an upper bound after the loop terminates. This lemma is the crucial step in showing that *RANDOM* works.

Lemma 6 *Let U be a set of messages on an n-processor fat-tree with congestion parameter r, and assume $\lambda(U) \leq \lambda$. Then after lines 3–6 of RANDOM's subroutine TRY-GUESS, the probability is at most $O(1/n^2)$ that $\lambda(U) > \frac{1}{2}\lambda$.*

Proof. The idea is to show that the load factor of an arbitrary channel c remains larger than $\frac{1}{2}\lambda$ with probability $O(1/n^3)$. Since the channel c is chosen arbitrarily out of the $4n-2$ channels in the fat-tree, the probability is at most $O(1/n^2)$ that any of the channels is left with a load factor larger than $\frac{1}{2}\lambda$.

For convenience, let C be the subset of messages that must pass through channel c and are undelivered at the beginning of the innermost loop in *RANDOM*. Let $C_0 = C$, and for $i \geq 1$, let $C_i \subset C_{i-1}$ denote the set of undelivered messages at the end of the ith iteration of the loop. Notice that we have $\lambda(C_i, c) = |C_i| / \mathrm{cap}(c)$, since we have $|C_i| = \mathrm{load}(C_i, c)$ by definition.

We now show there exist values for the constants k_1 and k_2 in line 3 of *TRY-GUESS* such that for $z = \max\{k_1\lambda, k_2 \lg n\}$, the probability is $O(1/n^3)$ that $\lambda(C_z, c) > \frac{1}{2}\lambda$, or equivalently, that

$$|C_z| > \frac{1}{2}\lambda \mathrm{cap}(c) \ . \tag{2}$$

It suffices to prove that the probability is $O(1/n^3)$ that fewer than $\frac{1}{2}|C|$ messages from C are delivered during the z cycles under the assumption that $|C_i| > \frac{1}{2}\lambda \mathrm{cap}(c)$ for $i = 0, 1, \ldots, z-1$. The intuition behind the assumption $|C_i| > \frac{1}{2}\lambda \mathrm{cap}(c)$ is that otherwise, the load factor on channel c is already at most $\frac{1}{2}\lambda$ at this step of the iteration. The reason we need only bound the probability that fewer than $\frac{1}{2}|C|$ messages are delivered during the z cycles is that inequality (2) implies that the number of messages delivered is fewer than $|C| - \frac{1}{2}\lambda \mathrm{cap}(c) \leq |C| - \frac{1}{2}\lambda(C, c)\mathrm{cap}(c) \leq \frac{1}{2}|C|$.

We shall establish the $O(1/n^3)$ bound on the probability that at most $\frac{1}{2}|C|$ messages are delivered in two steps. For convenience, we shall call a cycle *good* if at least $\mathrm{cap}(c)/8r$ messages are delivered, and *bad* otherwise. In the first step, we bound the probability that a given cycle is bad. The expected number of messages delivered in any given cycle is the product of the number of messages that remain to be delivered and the probability that any of these messages is successfully delivered. Using Lemma 5 with $p = 1/r\lambda \leq 1/r\lambda(U) \leq 1/r\lambda(C_i)$ in conjunction with the assumption that $|C_i| > \frac{1}{2}\lambda \mathrm{cap}(c)$, we can conclude that the expected number of messages delivered in any given cycle is greater than $\frac{1}{2r\lambda}\frac{1}{2}\lambda \mathrm{cap}(c) \geq \mathrm{cap}(c)/4r$. Then by Lemma 3, the probability that a given cycle is bad is at most $1 - 1/(8r-1) < 1 - 1/8r$.[2]

The second step bounds the probability that a substantial fraction of the z delivery cycles are bad. Specifically, we show that the probability is $1 - O(1/n^3)$ that at least some

[2]We use the weak bound of Lemma 3 because we cannot assume that the probability that a message is delivered in a given cycle is independent of the probabilities for other messages. In practice, one would anticipate that the dependencies between messages are weak, and that the algorithm would be effective with much smaller values for the constants k_1 and k_2 than we prove here.

small constant fraction q of the z cycles are good. By picking $k_1 = 4r/q$, which implies $z \geq 4r\lambda/q$, at least $qz\mathrm{cap}(c)/8r \geq \frac{1}{2}|C|$ messages are delivered.

We bound the probability that at least $(1-q)z$ of the z cycles are bad by using a counting argument. There are $\binom{z}{(1-q)z}$ ways of picking the bad cycles, and the probability that a cycle is bad is at most $1 - 1/8r$. Thus, the probability that at most $\frac{1}{2}|C|$ messages are delivered is

$$\begin{aligned}
\Pr\left\{\leq \tfrac{1}{2}|C| \text{ messages delivered}\right\} &\leq \binom{z}{(1-q)z}\left(1-\frac{1}{8r}\right)^{(1-q)z} \\
&\leq \left(q^q(1-q)^{1-q}\right)^{-z}\left(1-\frac{1}{8r}\right)^{(1-q)z} \qquad (3) \\
&\leq 2^{-z/12r}\ , \qquad (4)
\end{aligned}$$

where (3) follows from Stirling's approximation (for sufficiently large z), and (4) follows from choosing $q = 1/100r\ln r$ and performing algebraic manipulations. Since $z = \max\{k_1\lambda, k_2\lg n\}$, if we choose $k_2 = 36r$, the probability that fewer than $\frac{1}{2}|C|$ messages are delivered is at most $1/n^3$. ■

Now we can analyze *RANDOM* as a whole.

Theorem 7 *For any message set M on an n-processor fat-tree, the probability is at least $1 - O(1/n)$ that RANDOM delivers all the messages of M within the number of delivery cycles specified by Figure 4.*

Proof. First, we will show that if $\lambda_{guess} \geq \lambda(M)$, the probability is at most $O(1/n)$ that the loop in lines 2 through 8 of *TRY-GUESS* fails to yield $\lambda(U) \leq 1$. Initially, $\lambda \geq \lambda(U)$, and we know from Lemma 6 that the probability is at most $O(1/n^2)$ that any given iteration of the loop fails to restore this condition as λ is halved. Since there are $\lg\lambda_{guess}$ iterations of the loop, we need only make the reasonable assumption that λ_{guess} is polynomial in n to obtain a probability of at most $O(1/n)$ that $\lambda(U)$ remains greater than 1 after all the iterations of the loop.

Now we just need to count the number of delivery cycles that have been completed by the time we call *TRY-GUESS* with a λ_{guess} such that $\lambda(M) \leq \lambda_{guess}$. Let us denote by λ^*_{guess} the first λ_{guess} that satisfies this condition, and then break the analysis down into cases according to the value of $\lambda(M)$.

For $\lambda(M) \leq 1$, we do not actually even call *TRY-GUESS*. We need only count the one delivery cycle executed in line 1 of *RANDOM*.

For $1 < \lambda(M) \leq 2$, we need add only the $k_2\lg n$ cycles executed when we call *TRY-GUESS*(2).

For $2 < \lambda(M) < (k_2/k_1)\lg n$, the number of delivery cycles involved in each execution of *TRY-GUESS* is $O(\lg\lambda_{guess}k_2\lg n)$, since we perform $O(\lg\lambda_{guess})$ iterations of the loop in lines 2–8 of *TRY-GUESS*, each containing $k_2\lg n$ iterations of the loop in lines 3–6. The value of λ^*_{guess} is at most $(\lambda(M))^2$, so the number of delivery cycles is $O(\lg n\lg(\lambda(M))^2)$ for the last guess, $O(\lg n\lg\lambda(M))$ for the second-to-last guess, $O(\lg n\lg\sqrt{\lambda(M)})$ for the

third-to-last guess, and so on. The total number of delivery cycles is, therefore,

$$\begin{aligned}\sum_{0\le i\le 1+\lg\lg\lambda(M)} O(\lg n \lg(\lambda(M))^{2^{1-i}}) &= \sum_{0\le i\le 1+\lg\lg\lambda(M)} O(2^{1-i}\lg n \lg(\lambda(M))) \\ &= O(\lg n \lg\lambda(M)),\end{aligned}$$

since the series is geometric.

For $\lambda(M) > (k_2/k_1)\lg n$, the number of delivery cycles executed by the time we reach line 8 of *RANDOM* is $O(\lg n \lg\lg n)$ according to the preceding analysis, and then we must continue in the quest to reach λ^*_{guess}. If $\lambda(M) \le (k_2/k_1)\lg n \lg\lg n$, then we need only add the $O(\lg n \lg\lg n) = O(\lg n \lg\lambda(M))$ delivery cycles involved in the single call *TRY-GUESS*$((k_2/k_1)\lg n \lg\lg n)$.

If $\lambda(M) > (k_2/k_1)\lg n \lg\lg n$, the number of delivery cycles executed before reaching line 8 is $O(\lg n \lg\lg n)$ as before, which is $O(\lambda(M))$. We must then add $O(\lambda_{guess})$ cycles for each call of *TRY-GUESS* in line 10. Since λ^*_{guess} is at most $2\lambda(M)$, the total additional number of delivery cycles is

$$\sum_{0\le i\le t} O(2^{1-i}\lambda(M)) = O(\lambda(M)),$$

where $t = 1 + \lg(k_1\lambda(M)/k_2 \lg n \lg\lg n)$. The total number of delivery cycles is thus $O(\lambda(M))$. ■

The $1 - O(1/n)$ bound on the probability that *RANDOM* delivers all the messages can be improved to $1 - O(1/n^k)$ for any constant k by choosing $k_2 = 12(k+2)r$, or by simply running the algorithm through more choices of λ_{guess}.

We can also use *RANDOM* to obtain a routing algorithm that guarantees to deliver all the messages in finite time, and whose expected number of delivery cycles is as given in Figure 4. We simply interleave *RANDOM* with any routing strategy that guarantees to deliver at least one message in each delivery cycle. If the number of messages is bounded by some polynomial n^k, then we choose k_2 such that *RANDOM* works with probability $1 - O(1/n^k)$.

5 Universality

The performance of the routing algorithm *RANDOM* allows us to generalize the universality theorem from [12] which states that a universal fat-tree of a given volume can simulate any other routing network of equal volume with only a polylogarithmic factor increase in the time required. The original proof assumed the simulation of the routing network was off-line. In this section we show that the simulation can be carried out in the more interesting on-line context. We first discuss VLSI models briefly and state how channel capacities can be chosen for area and volume-universal fat-trees. We then give a simple universality result that requires no routing algorithm. Finally, we give a stronger universality theorem based on *RANDOM*.

VLSI models provide a means of measuring hardware costs quantitatively in terms of area or volume [8, 10, 11, 12, 16]. These models are interesting from an engineering point of view because "pin-boundedness" is modeled directly as the limitation on communication

imposed by the perimeter of a two-dimensional region or by the surface area of a three-dimensional region. In VLSI models, the processors and wires of a network are the vertices and edges of a graph. The graph must be embedded in a two or three-dimensional grid such that vertices are mapped to gridpoints and edges are mapped to disjoint paths in the grid. The area or volume of the network is the number of gridpoints occupied by either vertices or edge segments. These assumptions implicitly restricts the number of connections to a processor to at most 4 in two dimensions and 6 in three dimensions, but generalizations to larger processors are straightforward.

In order for a fat-tree to be universal for area or volume, the channel capacities must be picked properly. Let us consider area, instead of volume, for simplicity. Intuitively, we wish the processors to be densely packed in the region required by the network, The bandwidth of communication to a region is constrained by the perimeter of the region, however, and thus if the channel capacity to a subtree is too large relative to the number of processors in the subtree, the processors will not be densely packed. On the other hand, if we choose the channel capacities too small, the processors will indeed be densely packed, but we will not take maximal advantage of the available communication bandwidth. Consequently, we choose the capacity of a given channel to be proportional to the perimeter of the square region the processors rooted at that channel would occupy if there were no wires, that is $\Theta(\sqrt{n})$ if the subtree has n processors. It turns out that the additional area required by the wires in the channels does not greatly increase the area beyond that required by the processors alone: the area for n processors is $\Theta(n \lg^2 n)$, the same as that required by Leighton's tree of meshes graph [8].

Following this intuition, an area-universal fat-tree can be constructed by giving each leaf channel a constant capacity, and then growing the channel capacities by $\sqrt{2}$ at each level as we go up the tree, rounding off to integer capacities. Another scheme that avoids rounding is to double the channel capacities every two levels, as is done in the fat-tree of Figure 1. Either of these methods yields a $\Theta(n \lg^2 n)$-area layout for n processors, and a root capacity of $\Theta(\sqrt{n})$. Volume-universal fat-trees can be constructed in a similar fashion by picking a growth rate of $\sqrt[3]{4}$, or equivalently, by quadrupling the capacity every three levels. The volume of an n-processor fat-tree constructed by these methods is $\Theta(n \lg^{3/2} n)$, and the root capacity is $\Theta(n^{2/3})$, as can be shown with the arguments in [10] or [12].

Even without a good routing algorithm for fat-trees, it is possible to prove a simple universality property. The theorem is presented for area-universal fat-trees—a similar theorem holds for volume-universal ones.

> **Theorem 8** *Let R be a interconnection network occupying a square of area n such that all connections are point-to-point between processors with no intervening switches. Then an area-universal fat-tree of area $O(n \lg^2 n)$ can simulate every step of network R with at most $O(\lg^2 n)$ switching delay.*

Proof. We use an area-universal fat-tree such as that shown in Figure 1, where the channel capacity to leaves is 4 and the root capacity is $4\sqrt{n}$. Network R lies in a square with side length $\sqrt{n}$. Each processor of R is mapped to the corresponding processor of the fat-tree in the natural geometric fashion. This mapping satisfies the property that the capacity of any channel of the fat-tree is at least as great as the perimeter of the corresponding region

of the layout of network R. Therefore, any communication step performed by R induces at most a load factor of 1 on the fat-tree and thus can be routed in one delivery cycle. Since each delivery cycle requires only $O(\lg^2 n)$ bit times, the theorem follows. ■

This universality result is weak in several ways. For example, the fat-tree network occupies slightly more area than the simulated network R. It seems reasonable to compare networks of exactly equal cost. Another weakness in the result is that it forbids networks with intermediate switches that buffer messages for several time steps. We could model switches as processors, but in some contexts, processors might be considerably more expensive than switches. Since we can directly route messages sets with large load factors using *RANDOM*, we can prove a stronger universality result that addresses these concerns. The next theorem, a generalization of that in [12], is presented for volume-universal fat-trees. One can prove an analogous theorem for area-universal fat-trees.

> **Theorem 9** *Let FT be a volume-universal fat-tree of volume v, and let R be an arbitrary routing network also of volume v on a set of $n = O(v/\lg^{3/2} v)$ processors. Then the processors of R can be mapped to processors of FT such that any message set M that can be delivered in time t by R can be delivered by FT in time $O((t + \lg\lg n)\lg^3 n)$ with probability $1 - O(1/n)$.*

Proof. The proof parallels that of [12]. The reader is referred to that paper for details. The routing network R of volume v is mapped to FT in such a way that any message set M that can be delivered in time t by R puts a load factor of at most $O(t \lg(n/v^{2/3}))$ on FT. By Theorem 7, the message set M can be delivered by *RANDOM* in $O(t \lg(n/v^{2/3}) + \lg n \lg\lg n)$ delivery cycles with high probability. Since each delivery cycle takes at most $O(\lg^2 n)$ time, the result follows. ■

6 Greedy strategies

It is natural to wonder whether a simple greedy strategy of sending all undelivered messages on each delivery cycle, and letting them battle their ways through the switches, might be as effective as *RANDOM*, which we have shown to work well on every message set. As a practical matter, a greedy strategy may be a good choice, but it seems difficult to obtain tight bounds on the running time of greedy strategies. In fact, we show in this section that no naive greedy strategy works as well as *RANDOM* in terms of asymptotic running times. For simplicity, we restrict our proof to deterministic strategies and comment later on the extension to probabilistic ones. Specifically, we show that for a wide class of deterministic greedy strategies, there exist n-processor fat-trees and message sets with load factor λ such that $\Omega(\lambda \lg n)$ delivery cycles are required. Thus, if λ is asymptotically larger than $\lg\lg n$, the greedy strategy is worse than *RANDOM*, which essentially guarantees $O(\lambda + \lg n \lg\lg n)$ delivery cycles for any set of messages. The lower-bound proof for greedy routing is based on an idea due to F. M. Maley [13].

Figure 5 shows the greedy algorithm. The code for *GREEDY* does not completely specify the behavior of message routing on a fat-tree because the switches have a choice as to which messages to drop when there is congestion. (The processors also have this choice, but we shall think of them as being switches as well.) In the analysis of *RANDOM*, we

```
1    while M ≠ ∅ do
2        send M
3        M ← M − {messages delivered}
4    endwhile
```

Figure 5: The algorithm *GREEDY* for delivering a message set M. This algorithm repeatedly sends all undelivered messages. The performance is highly dependent on the behavior of the switches.

presumed that all messages in a channel are lost if the channel is congested. To completely specify the behavior of *GREEDY*, we must define the behavior of switches when channels are congested.

The lower bound for *GREEDY* covers a wide range of switch behaviors. Specifically, we assume the switches have the two properties below.

1. Each switch is greedy in that it only drops messages if a channel is congested, and then only the minimum number necessary.

2. Each switch is *oblivious* in that decisions on which messages to drop are not based on any knowledge of the message set other than the presence or absence of messages on the switch's input lines.

We define the switches of a fat-tree to be *admissible* if they have these two properties. The conditions are satisfied, for example, by switches that drop excess messages at random, or by switches that favor one input channel over another. An admissible switch can even base its decisions on previous decisions, but it cannot predict the future or make decisions based on knowing what (or how many) messages it or other switches have dropped. (The definition of oblivious in property 2 can be weakened to include an even wider range of switch behaviors without substantially affecting our results.)

At this point, we restrict attention to deterministic greedy strategies and present the lower-bound theorem for *GREEDY* operating on an area-universal fat-tree. The theorem can be extended to a variety of other fat-trees. A discussion of the extension to probabilistic greedy strategies follows the proof of the theorem.

> **Theorem 10** *Consider an n-processor area-universal fat-tree with deterministic admissible switches whose channel capacities are 1 nearest the processors and double at every other level going up the tree. Then there exist message sets with load factor λ for which GREEDY requires $\Omega(\lambda \lg n)$ delivery cycles.*

Proof. For any $\lambda \geq 12$, we will construct a "bad" message set M_n on the n processors of the fat-tree by induction on the subtrees. The message set M_n will consist entirely of messages to be routed out of the root and will satisfy the following three properties:

1. The message set M_n has load factor at most λ.

2. The root channel of the fat-tree is full for the first $\frac{1}{3}\lambda$ delivery cycles.

3. A total of at least $\frac{1}{6}\lambda + \frac{1}{12}\lambda \lg n$ delivery cycles[3] are required to deliver all the messages in M_n.

For the base case we consider a subtree with 1 processor, that is, a leaf connected to a channel of capacity 1. The bad message set M_1 consists of λ messages to be sent from the single processor. The properties are satisfied since the root channel is congested throughout the first $\frac{1}{3}\lambda$ delivery cycles, and at least $\frac{1}{6}\lambda$ delivery cycles are needed to deliver all the messages.

We next show that we can construct the bad message set M_n assuming that we can construct a bad message set for a subtree of $n/4$ processors. The construction uses an adversary argument. First, we specify the pattern of inputs seen by the root switch of the fat-tree during certain delivery cycles, without giving any indication of how that input pattern can be achieved. Then since we have given enough information to determine the behavior of the root switch during these cycles, the root switch must announce which messages it passes through to its output. Finally, we give a construction for a message set that achieves the input pattern we called for in the first step. We take advantage of the announced behavior of the root switch in order to ensure that the message set also satisfies properties 1, 2, and 3.

We begin by calling for the input channels of the root switch of the fat-tree to be full for t delivery cycles, where t is $\frac{1}{3}\lambda$. If this is achieved, the total number of messages removed from the fat-tree during the first t delivery cycles is $m = \frac{1}{3}\lambda\sqrt{n}$, since the root capacity is $\sqrt{n}$ and the root switch is greedy. Also, as mentioned before, the specified input pattern determines the behavior of the root switch because the switch is oblivious.

The behavior of the root switch determines how many of the m messages removed from the fat-tree by delivery cycle t come from each of the four subtrees shown in Figure 6. At least one of these subtrees provides no more than $m/4$ of the messages. We choose one such subtree and refer to it as the *unfavored* subtree. The other subtrees are referred to as the *favored* subtrees.

Having determined the unfavored subtree given the conditions specified so far, we can complete the construction of M_n. The unfavored subtree contains a copy of the bad message set $M_{n/4}$ for that subtree. Each of the other three subtrees contains $\frac{1}{6}\lambda\sqrt{n}$ messages evenly divided among the processors in the subtree. Now we must prove that M_n meets all of our requirements.

First, we show that M_n is consistent with the input pattern specified for the root switch. To show that the input channels of the root switch of the fat-tree are full through the first t delivery cycles, it suffices to prove that the root channels of the four subtrees are full through this time. The root channel of the unfavored subtree is full by the induction hypothesis (property 2). The root channel of each favored subtree is also full for the first t delivery cycles, since its messages are evenly distributed, its switches are greedy, and t times its root capacity does not exceed the number of messages emanating from it.

We now prove that properties 1, 2, and 3 hold for M_n. The load factor in the favored subtrees is less than λ by construction. The load factor is at most λ in the unfavored subtree by the induction hypothesis (property 1), so the number of messages in the unfavored

[3]Without loss of generality, we assume henceforth that $\frac{1}{12}\lambda$ is integral, since we could otherwise use $\lfloor \frac{1}{12}\lambda \rfloor$ with only a constant factor change.

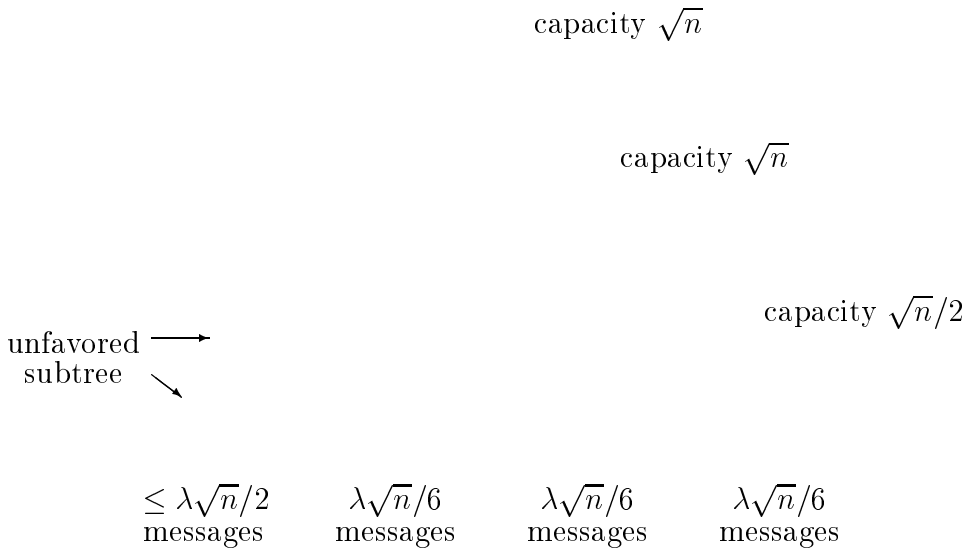

Figure 6: Construction of M_n for the proof of Theorem 10. The subtree from which the fewest number of messages have been delivered by a certain time is loaded with the largest number of messages.

subtree is at most $\frac{1}{2}\lambda\sqrt{n}$, and the total number of messages in M_n is at most

$$\frac{1}{2}\lambda\sqrt{n} + 3 \cdot \frac{1}{6}\lambda\sqrt{n} = \lambda\sqrt{n}.$$

Thus, the load factor of M_n on the fat-tree is at most λ, and property 1 holds. Property 2 is satisfied for M_n because the root switch is greedy. We have already shown that the input channels of the root switch are full through delivery cycle t, so the root channel is certainly full for the required amount of time. Finally, property 3 holds because after running $t = \frac{1}{3}\lambda$ delivery cycles, only $m/4 = \frac{1}{12}\lambda\sqrt{n}$ messages have been removed from the unfavored subtree. If priority had been given to the unfavored subtree, only $\frac{1}{6}\lambda$ delivery cycles would have been required to remove the $m/4$ messages, So by the induction hypothesis (property 3), an additional $\frac{1}{6}\lambda + \frac{1}{12}\lambda\lg(n/4) - \frac{1}{6}\lambda$ cycles are required to empty the unfavored subtree. If we include the original t cycles, the total number of cycles required to deliver all the messages in M_n is at least $\frac{1}{6}\lambda + \frac{1}{12}\lambda\lg n$. ■

When probabilistic admissible switches are permitted, the proof of Theorem 10 can be extended to show that the expected number of delivery cycles is $\Omega(\lambda\lg n)$. The idea is that at least one of the subtrees in Figure 6 must be unfavored with probability at least 1/4. We call one such subtree the *often-unfavored* subtree. The construction of M_n proceeds as before, with the often-unfavored subtrees playing the previous role of the unfavored subtrees. In any particular run of *GREEDY*, we expect 1/4 of the often-unfavored subtrees to be unfavored, so there is a $\Theta(1)$ probability that 1/8 of the often-unfavored subtrees are unfavored (Lemma 3). Thus, the probability is $\Theta(1)$ that $\Omega(\lambda\lg n)$ delivery cycles are required, which means that the expected number of delivery cycles is $\Omega(\lambda\lg n)$.

Although we have shown an unfavorable comparison of *GREEDY* to *RANDOM*, it should be noted that *GREEDY* does achieve the lower bound we proved for routing mes-

```
1    z ← 1
2    while M ≠ ∅ do
3        for each message m ∈ M, choose a random number i_m ∈ {1, 2, ..., z}
4        for i ← 1 to z do
5            send all messages m such that i_m = i
6        endfor
7        z ← 2z
8    endwhile
```

Figure 7: The algorithm *RANDOM′* for routing in a fat-tree with channels of capacity $\Omega(\lg n)$. This algorithm repeatedly doubles a guessed number of delivery cycles, z. For each guess, each message is randomly sent in one of the delivery cycles.

sages out the root. That is, routing of messages out the root or, more generally, up the tree only, can be accomplished by *GREEDY* in $O(\lambda \lg n)$ delivery cycles. This can be seen by observing that the highest congested channel (closest to the root) must drop at least one level every λ delivery cycles. If one could establish an upper bound of λ times a polylogarithmic factor for the overall problem of greedy routing, it would show that *GREEDY* still has merit despite its inferior performance in comparison to *RANDOM*.

7 Further results

This section contains three additional results relevant to routing on fat-trees. The first is a simple randomized algorithm for routing on fat-trees in which each channel has at least logarithmic capacity. The second is a new class of fat-trees that have far simpler switches than the ones thus far considered. The final result is an improvement to the off-line routing result of [12].

Larger channel capacities

We can improve the results for on-line routing if each channel c in the fat-tree is sufficiently large, that is if $\mathrm{cap}(c) = \Omega(\lg n)$ Specifically, we can deliver a message set M in $O(\lambda(M))$ delivery cycles with high probability, i. e., we can meet the lower bound to within a constant factor. The better bound is achieved by the algorithm *RANDOM′* shown in Figure 7.

> **Theorem 11** *For any message set M on an n-processor fat-tree with channels of capacity $\Omega(\lg n)$, the probability is at least $1 - O(1/n)$ that RANDOM′ will deliver all the messages of M in $O(\lambda(M))$ delivery cycles, if $\lambda(M)$ is polynomially bounded.*

Proof. Let the lower bound on channel size be $a \lg n$, and let n^k be the polynomial bound on the load factor $\lambda(M)$. We consider only the pass of the algorithm when z first exceeds

$e2^{(k+2)/a}\lambda(M)$. We ignore previous cycles for the analysis of message routing, except to note that the number of delivery cycles they require is $O(\lambda(M))$.

We first consider a single channel c within a single cycle i from among the z delivery cycles in the pass. Since each message has probability $1/z$ of being sent in cycle i, we can apply Lemma 4 with $p = 1/z$ to conclude that the probability that channel c is congested in cycle i is at most

$$\begin{aligned}\left(\frac{e\lambda(M)}{z}\right)^{\mathrm{cap}(c)} &\leq 2^{-\frac{k+2}{a}\mathrm{cap}(c)} \\ &\leq 2^{-(k+2)\lg n} \\ &= \frac{1}{n^{k+2}} \ .\end{aligned}$$

Since there are $O(n)$ channels, the probability that there exists a congested channel in cycle i is $O(1/n^{k+1})$. Finally, since there are $z \leq 2e2^{(k+2)/a}\lambda(M) = O(\lambda(M)) = O(n^k)$ cycles, the probability is $O(1/n)$ that there exists a congested channel in any delivery cycle of the pass. ■

Another universal fat-tree

We have recently discovered a fat-tree design that uses simpler switches than the fat-tree described in Section 1 and in [12]. Figure 8 illustrates the structure of a two-dimensional universal fat-tree of this new type. Each of the switches in this fat-tree can switch messages among four child switches and two parent switches. The area of the fat-tree is $\Theta(n \lg^2 n)$.[4] In three dimensions, we can use switches with eight children and four parents to obtain a fat-tree with volume $\Theta(n \lg^{3/2} n)$.

The new fat-tree design satisfies the universality property of Theorem 9, except that the degradation in time is $O(\lg^4 n)$. The new fat-tree structure removes a factor of $\lg n$ from the time to perform a delivery cycle since the switches have constant depth. The number of delivery cycles needed to route a set M of messages is $O(\lambda(M) \lg^2 n)$, however, which yields $O(\lambda(M) \lg^3 n)$ total time, as compared with $O((\lambda(M) + \lg n \lg\lg n) \lg^2 n)$ for the original fat-tree.

The mechanics of routing on the new fat-tree are somewhat different than on the original. The underlying channel structure for the two fat-trees is the same, but the new fat-tree does not rely on concentrators to make efficient use of the available output wires. Instead, each message sent through the fat-tree randomly chooses which parent to go to next (based on random bits embedded in its address field) until it reaches the apex of its path, and then it takes the unique path downward to its destination. This strategy guarantees that for any given channel through which a message must pass, the message has an equal likelihood of picking any wire in the channel.

The routing algorithm is a modification of the algorithm *RANDOM′*. We simply surround lines 3–6 with a loop that executes these lines $(k + 1) \lg n$ times, where $|M| = O(n^k)$.

[4] Interestingly, a mesh-of-trees [8] can be directly embedded in this fat-tree. In fact, it can be shown using sorting arguments that a mesh-of-trees is area-universal [9].

Figure 8: Another fat-tree design. The switches in this structure have constant size.

The proof that the algorithm works applies the analysis from Section 4 to individual wires, treating them as channels of capacity 1. Consider a wire w traversed by a message in a p-subset M' of M, and consider the channel c that contains the wire. For any other message in M, the probability is $p/cap(c)$ that the message is directed to wire w when the message set M' is sent. Thus, the probability that w is congested is at most $B(1, \text{load}(M, c), p/\text{cap}(c)) \leq ep\lambda(M)$, and an analogue to Lemma 4 holds because the capacity of w is 1. Lemma 5, which says that the probability is $\frac{1}{2}p$ that a given message of M is delivered when a p-subset of M is sent, also holds if the congestion parameter r is chosen to be $\Theta(\lg n)$.

We can now prove a bound of $O(\lambda(M) \lg^2 n)$ on the number of delivery cycles required by the algorithm to deliver all the messages in M. It suffices to show that with high probability, all the messages in M get routed when the variable z in the algorithm reaches $\Theta(\lambda(M) \lg n)$. When $z \geq r\lambda(M) = \Theta(\lambda(M) \lg n)$, any given message m is sent once during a single pass through lines 3–6, and the probability that the message is not delivered on that pass is at most $\frac{1}{2}$. Thus, the probability that m is not delivered on any of the $(k+1) \lg n$ passes through lines 3–6 is at most $1/n^{k+1}$. Since the number of messages in M is $O(n^k)$, the probability is $O(1/n)$ that a message exists which is not routed by the time z reaches $r\lambda(M)$.

Off-line routing

Our analysis for *RANDOM* has ramifications for the off-line routing problem. We have shown that with high probability, the number of delivery cycles given by Figure 4 suffices to deliver a message set with load factor λ. Consequently, there must exist off-line schedules using only this many delivery cycles, which improves the bound of $O(\lambda \lg n)$ given in [12]. The previous off-line bound was proved by giving a deterministic, polynomial-time construction of a routing schedule that achieves the bound. Whether a deterministic,

polynomial-time algorithm exists that achieves our better bound is an open question.

Perhaps the bound on off-line routing can be further improved (e.g., to $O(\lambda+\lg n)$). The integer programming framework of Raghavan and Thompson [15] is one possible approach which might give a probabilistic construction that achieves this bound. On the other hand, it may be possible to apply more direct combinatorial techniques to yield an improved deterministic bound.

8 Concluding remarks

This paper has studied the problem of routing messages on fat-tree networks. We have obtained good bounds for randomized routing based on the load factor of a set of messages. Our algorithms directly address the problem of message congestion and require no intermediate buffering, unlike many algorithms in the literature. We have shown how to use the routing algorithms to prove that fat-trees are volume-universal networks. This section discusses some directions for future research.

The analysis of the algorithm *RANDOM* gives reasonably tight asymptotic bounds on its performance, but the constant factors in the analysis are large. In practice, smaller constants probably suffice, but it is difficult to simulate the algorithm to determine what constants might be better. Unlike Valiant's algorithm for routing on the hypercube, our algorithm does not have the same probabilistic behavior on all sets of messages, and therefore, the simulation results may be highly correlated with the specific message sets chosen. The search for good constants is thus a multidimensional search in a large space, where each data point represents an expensive simulation.

Although we have shown that *GREEDY* is asymptotically worse than *RANDOM*, it may be that it is more practical to implement. The logarithmic-factor overhead that we have been able to show is mitigated by a constant factor of $\frac{1}{12}$. Simulations indicate that a greedy algorithm might actually work quite well [6], but we have been unable to prove a good upper bound on its performance. Despite the simplicity of control offered by *GREEDY*, it seems unwise to base the design of a large, parallel supercomputer on unproven conjectures of performance. Thus, a comprehensive analysis of *GREEDY* remains an important open problem.

The idea of using load factors to analyze arbitrary networks is a natural one. We have been successful in analyzing fat-trees using this measure of routing difficulty. It may be possible to analyze other networks in terms of load factor, but some improvement to our techniques seems to be necessary if channel widths are small and the diameter of the network is large. The problem is that a message that passes through many small channels has a high likelihood of conflicting with other messages. One solution might involve buffering messages in intermediate processors or switches.

The high probability results reported in this paper for routing on fat-trees are almost deterministic in the sense that substantial deviation from the expected performance will probably never occur in one's lifetime. On the other hand, from a theoretical point of view, it would be nice to match the results of this paper with truly deterministic algorithms. Most deterministic routing algorithms in the literature are based on sorting, and thus a direct application to fat-trees causes congestion problems, much as does Valiant's routing technique. A deterministic routing algorithm for fat-trees that circumvents these problems

would yield even stronger universality properties than we have shown here.

Acknowledgments

We have benefited tremendously from the helpful discussions and technical assistance of members of the theory of computation group at MIT. Thanks to Ravi Boppana, Thang Bui, Benny Chor, Peter Elias, Oded Goldreich, Johan Hastad, Alex Ishii, Tom Leighton, Bruce Maggs, Miller Maley, Cindy Phillips, Ron Rivest, and Peter Shor.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi, "Sorting in $c \log n$ parallel steps," *Combinatorica*, Vol. 3, No. 1, 1983, pp. 1–19.

[2] R. Aleliunas, "Randomized parallel communication," *Proceedings of the 1st Annual ACM Symposium on Principles of Distributed Computing*, August 1982, pp. 60–72.

[3] V. E. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*, New York: Academic Press, 1965.

[4] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Mathematical Statistics*, Vol. 23, 1952, pp. 493–507.

[5] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, 2nd edition, New York: John Wiley & Sons, 1957.

[6] A. T. Ishii, *Interprocessor Communication Issues in Fat-Tree Architectures*, Bachelor's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1985.

[7] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *JACM*, Vol. 27, No. 4, October 1980, pp. 831–838.

[8] F. T. Leighton, *Complexity Issues in VLSI*, Cambridge, Massachusetts: MIT Press, 1983.

[9] F. T. Leighton and C. E. Leiserson, private communication, May 1985.

[10] F. T. Leighton and A. L. Rosenberg, "Three-dimensional circuit layouts," *SIAM Journal on Computing*, Vol. 15, No. 3, August 1986, pp. 793–813

[11] C. E. Leiserson, *Area-Efficient VLSI Computation*, Cambridge, Massachusetts: MIT Press, 1983.

[12] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, Vol. C–34, No. 10, October 1985.

[13] F. M. Maley, private communication, October 1984.

[14] N. Pippenger, "Parallel communication with limited buffers," *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, IEEE, October 1984, pp. 127–136.

[15] P. Raghavan and C. D. Thompson, "Provably good routing in graphs: regular arrays," *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, May 1985, pp. 79–87.

[16] C. D. Thompson, *A Complexity Theory for VLSI*, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.

[17] E. Upfal, "Efficient schemes for parallel communication," *JACM*, Vol. 31, No. 3, July 1984, pp. 507–517.

[18] L. G. Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, Vol. 11, No. 2, May 1982, pp. 350–361.

[19] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, May 1981, pp. 263–277.

[20] A. Waksman, "A permutation network," *JACM*, Vol. 15, No. 1, January 1968, pp. 159–163.