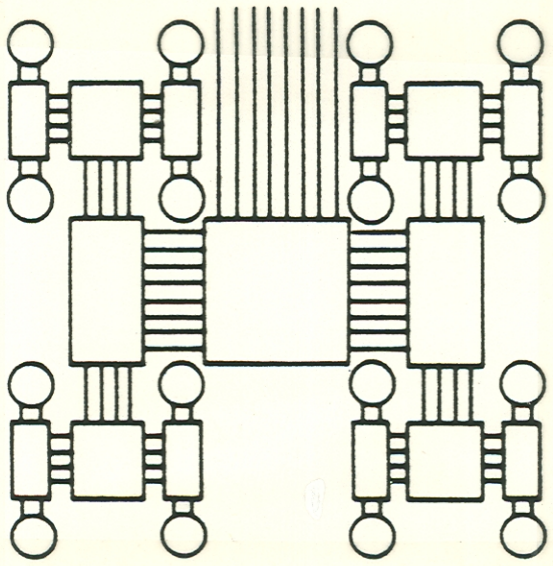
LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-307

RANDOMIZED ROUTING ON FAT-TREES

RONALD I. GREENBERG
CHARLES E. LEISERSON



MAY 1986

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Randomized Routing on Fat-Trees

Ronald I. Greenberg and Charles E. Leiserson

April 1986

A preliminary version of this paper appeared in the 26th IEEE Foundations of Computer Science conference, October 21–23, 1985 in Portland, Oregon.

This research was supported in part by the Defense Advanced Research Projects Agency under Contract N00014–80–C–0622. 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.

Key words and phrases: Fat-trees, interconnection networks, parallel supercomputing, randomization, randomized routing, routing networks, universality, VLSI theory.

Randomized Routing on Fat-Trees

Ronald I. Greenberg
Charles E. Leiserson

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

April 28, 1986

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 where switch settings can be determined in advance. In a VLSI-like model where hardware cost is equated with physical volume, the routing algorithm demonstrates that fat-trees are universal routing networks in the sense 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 for a given volume of hardware, a fat-tree is nearly the best routing network that can be built. This universality result had been proved only for *off-line* simulations [10], where switch settings can be determined 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. 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.

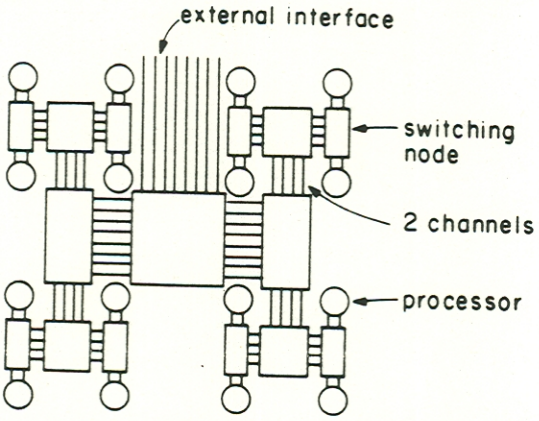


Figure 1: The organization of a fat-tree. Processors are located at the leaves, and the internal nodes contain concentrator switches. The capacities of channels increase as we go up the tree.

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 interconnection which is "skinny all over," each channel of a fat-tree 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 capacities of the channels in a fat-tree determine how much hardware is required to build it, where we measure hardware in terms of three-dimensional volume. The greater the capacities of the channels, the greater the communication potential, and also, the greater the physical volume of an implementation of the network. The randomized routing algorithm that will be presented in this paper can be used to show that a fat-tree with properly chosen channel capacities is a universal network for a given volume of circuitry.

The problem that a routing algorithm for a volume-universal network must face is "pin-boundedness"—the bandwidth limitation imposed by surfaces of three-dimensional regions—a constraint that makes some communication patterns among a set of processors harder than others. To illustrate this point, consider a three-dimensional region of volume v containing an n-processor routing network, and consider a plane that cuts through the region perpendicular to the longest dimension and which divides the set of processors in half. Suppose each processor sends a message to a processor on the other side of the cut. Since the cross-sectional area of the cut has area $O(v^{2/3})$, the time required by the network to deliver all the messages is $\Omega(n/v^{2/3})$. If the processors fill the region with substantial density (i.e., $v = O(n^{3/2-\epsilon})$), where $\epsilon > 0$), the time required to deliver the messages is polynomial in n. In contrast, the communication pattern in which each processor communicates with its nearest neighbor in the region, as in a three-dimensional systolic array, can be accomplished in constant time.

A volume-universal network should be able to simulate the communication of any (bounded-degree) network of a given volume with at most polylogarithmic degradation in time, much as traditional universal networks can simulate the communication of any network of a given number of processors with at most polylogarithmic degradation. The

routing algorithm that we will give to show that fat-trees are volume-universal networks cannot use traditional randomization techniques such as the one proposed by Valiant in his seminal paper [15], however. Using Valiant's technique for routing permutations on a hypercube, each message is sent to a random intermediate destination, and from there, to its true destination. Using the analysis above, the expected time for *every* permutation, including simple nearest-neighber communication, is $\Omega(n/v^{2/3})$ because we expect $n/2$ messages to cross the cut when they are sent to their intermediate destinations. Thus, if Valiant's technique is used, the simulation of routing networks whose processors densely fill a given volume causes polynomial degradation in time.

This paper presents and analyzes a randomized algorithm for routing on fat-trees which shows that fat-trees can efficiently simulate any routing network of a comparable volume. We present a measure of congestion for a set of messages, called the *load factor*, which is a lower bound on the time to route the messages on a fat-tree. We show that if a set of messages has load factor λ, our routing algorithm can route them 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 [10].

The analysis in terms of load factor 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. Furthermore, we make no assumptions about the statistical distribution of messages, except insofar as they affect the load factor. Our routing algorithm also differs from others in the literature in the way randomization is used. Unlike the algorithms of Valiant [15], Valiant and Brebner [16], Aleliunas [2], Upfal [14] and Pippenger [12], for example, it does not randomize with respect to paths taken by messages. Instead, for each delivery cycle, each undelivered message randomly chooses whether to be sent.

The remainder of this paper is organized as follows. Section 2 further describes the fat-tree network, defines the load factor, and discusses universality. 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 gives an existential lower bound for the naive greedy approach to routing messages which shows that the greedy strategy is inferior to the randomized algorithm for worst case inputs. Section 6 contains a variety of results that follow from the randomized routing algorithm. It shows how the universality result of [10] can be extended to on-line simulations, and it includes a modification of the routing algorithm which achieves better bounds when each channel has capacity $\Omega(\lg n)$. Finally, Section 7 contains some concluding remarks.

2 Fat-trees

This section describes an implementation of a fat-tree routing network, and it shows how to choose the channel capacities for volume-universal and area-universal fat-trees. We precisely define the *load factor* of a set of messages on a general network, which is a lower

bound on the time required to deliver the messages. We give a proof that fat-trees satisfy a simple universality property, and we indicate why a good message-routing algorithm based on load factor can strengthen this result.

The implementation of fat-trees described here follows that of [10]. 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. (Section 6 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 that must cross S. The *load factor* of M on S is

$$\lambda(M, S) = \frac{\text{load}(M, S)}{\text{cap}(S)} ,$$

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 simple lower bound on the time required to deliver all messages in the set. For fat-trees, the load factor of a set of messages is determined by the cuts on the channels alone.

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. ∎

The randomized routing algorithm for fat-trees presented in Section 3 can deliver a set M of messages in $O(\lambda(M) + \lg n \lg\lg n)$ delivery cycles with high probability. In fact, the running time is asymptotically even less for message sets with small load factors.

We are particularly interested in the application of the routing results to universal fat-trees. In order for a fat-tree to be universal for area, the channel capacities must be picked properly. One way is to give each leaf channel a constant capacity, and then grow the channel capacities by $\sqrt{2}$ at each level as we go up the tree, rounding off to integer capacities where needed. Another scheme that avoids rounding is to double the channel capacities every two levels, as is shown in 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 $\Theta(n^{2/3})$.

Even without a good routing algorithm for fat-trees, it is possible to prove a simple universality property.

Lemma 2 *Let R be an interconnection network 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 assume without loss of generality, that network R lies in a square with side length $\sqrt{n}$. The layout of the fat-tree has $\sqrt{n}$ processors on a side, and thus each processor of R can be 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 proportional to 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, which takes $O(\lg^2 n)$ time. ∎

This universality result can be strengthened if a good routing algorithm for fat-trees is known. For example, it seems natural to consider networks with intermediate switches that might buffer messages for several time steps. Given a set of messages that are delivered over time on such a network, the load factor induced on channels of the fat-tree is typically greater than 1. We could model switches as processors, but we would like to prove a stronger universality theorem without disrupting the abstraction of processors connected to a routing network. Thus, we must have a routing algorithm that can directly route messages sets with large load factors. Section 3 presents a general routing algorithm for fat-trees that routes messages quickly with high probability. Section 6 uses this routing algorithm to show that a fat-tree of a given volume with n processors can simulate any other n-processor network of the same volume with only polylogarithmic degradation in time.

3 The routing algorithm

This section gives a randomized algorithm for routing a set M of messages. 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 cylces, 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 idea is sufficiently important to merit 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 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 $\lambda(M)$, each iteration of the **while** loop in *TRY-GUESS* halves the load factor $\lambda(U)$ of the set U of

```
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 ← λ_guess^2
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 $1 - O(1/n)$.

undelivered messages with high probability, 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 $\lg \lambda_{guess}$ for λ_{guess} small, and as λ_{guess} for λ_{guess} 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 *RANDOM*

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 the number of delivery cycles given by Figure 4. This may be summarized as $O(\lambda(M) + \lg n \lg\lg n)$ delivery cycles for all message sets.

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 more general, but weaker, 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 3

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

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

Lemma 4 *Let $X \leq b$ be a random variable with expectation μ. Then for any $w < \mu$, we have*

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

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 5 *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 3 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

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

The next lemma analyzes 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 one parameter of the capacities which will enable us choose a suitable upper bound for p.

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

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

For a 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 $r \leq 4e \lg n$. For universal fat-trees, 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 $\Omega(\lg \lg n)$. The remaining analysis 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 6 *Let M be a set of messages on a fat-tree which 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 5. The probability that at least one of the channels is congested is, therefore, much 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 7 *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 of the proof 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 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

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

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 (1) 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. Using Lemma 6 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 4, the probability that a given cycle is bad is at most $1 - 1/(8r-1) < 1 - 1/8r$. (Although this bound is sufficiently strong to prove our theoretical results, it is weak because the probability that a message is delivered in a given cycle is not independent from the probabilities for other messages, and thus we must rely on the bound given by Lemma 4. In practice, one would anticipate that the dependencies 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.)

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 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 will be 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

$$\Pr\left\{\leq \tfrac{1}{2}|C| \text{ messages delivered}\right\} \ \leq \ \binom{z}{(1-q)z}\left(1 - \frac{1}{8r}\right)^{(1-q)z}$$

$$\leq \left(q^q(1-q)^{1-q}\right)^{-z}\left(1-\frac{1}{8r}\right)^{(1-q)z}$$
$$\leq 2^{-z/12r},$$

if we choose $q = 1/e^4 r \ln r$, as the reader may verify. 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 8 *For any message set M on an n-processor fat-tree, the probability is at least $1 - O(1/n)$ that RANDOM will deliver 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 7 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,

$$\sum_{0 \leq i \leq 1+\lg\lg\lambda(M)} O(\lg n \lg(\lambda(M))^{2^{1-i}}) = \sum_{0 \leq i \leq 1+\lg\lg\lambda(M)} O(2^{1-i} \lg n \lg(\lambda(M)))$$
$$= O(\lg n \lg \lambda(M)),$$

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 $\lambda(M) \leq (k_2/k_1) \lg n \lg\lg n$, 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 which guarantees to deliver all the messages in finite time with expected number of delivery cycles 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 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, and in fact, we can show 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 the probabilistic case. 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. This lower-bound result is based on an idea originally due to M. Maley [11].

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 could presume that all messages in a channel were lost if the channel was 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.

```
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.

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.)

It is also important to realize that the lower bound proof for the greedy strategy which we will present does not apply to every possible choice of channel capacities in the fat-tree. Our result is strong in the sense that it provides a lower bound on the time required just to route messages from the leaves out the root, but it does not apply to certain types of fat-trees. For example, on a fat-tree in which channel capacities double at every level, there is never any congestion in routing from the leaves to the root, so a greedy strategy is guaranteed to finish in λ delivery cycles. Similarly, a fat-tree in which all channel capacities are the same will also require only λ delivery cycles. The lower bound does apply to a wide variety of fat-trees which exhibit a substantial degree of uniform and nonextreme growth. For the sake of simplicity, we shall consider fat-trees like the one in Figure 1 in which the channel capacities double at every other level. As discussed earlier, these fat-trees are universal. We also assume that the number of levels $(\lg n)$ is even and that the capacities of the channels nearest the processors are 1. We refer to such a fat-tree as a *standard* fat-tree.

We are now ready to state the lower-bound theorem for *GREEDY*. At this point, we restrict attention to deterministic strategies.

> **Theorem 9** *Consider an n-processor standard fat-tree with deterministic admissible switches. Then there exist message sets with load factor λ on which GREEDY requires $\Omega(\lambda \lg n)$ delivery cycles.*

Proof. The proof is by induction on the height $(\lg n)$ of the fat-tree. In order to make the induction go through, we first strengthen the statement of the theorem as follows:

> **Claim:** Let FT be an n-processor standard fat-tree, possibly embedded within a larger fat-tree, with deterministic admissible switches. Then if

GREEDY is applied to routing messages out the root of FT, there exists, for any $\lambda \geq 12$, a "bad" message set M_n on FT which has the following three properties:

1. The message set M_n has load factor at most λ.
2. If at most $\frac{1}{12}\lambda\sqrt{n}$ messages of M_n are removed from FT, then the root channel is full for each of the delivery cycles during which the messages are removed.
3. At least an *additional* $\frac{1}{24}\lambda \lg n$ delivery cycles are required to deliver all the remaining messages in M_n.

For the base case we consider a tree with 1 processor, that is, one leaf connected to a root channel of capacity 1. Then if we assign λ messages to be sent from the single processor, the root channel will remain congested throughout the removal of $\frac{1}{12}\lambda$ messages, which will certainly leave us with additional messages requiring additional delivery cycles. (Without loss of generality, we assume henceforth that $\frac{1}{12}\lambda$ is integral, since we could otherwise use $\left\lfloor \frac{1}{12}\lambda \right\rfloor$ with only a constant factor change.)

Now we show that the claim is true for a standard fat-tree FT with n processors assuming that it is true for standard fat-trees with $n/4$ processors. We will construct a message set M_n for FT which satisfies properties 1, 2, and 3 by using an adversary argument. We will first partially specify the pattern of inputs seen by the root switch of FT. Then the root switch must indicate what its behavior is under these conditions. Finally, we will use this information to determine a message set M_n which is consistent with the specified input pattern and which satisfies properties 1, 2, and 3.

We begin by specifying that the input channels of the root switch of FT are full for t delivery cycles, where t is $\frac{1}{12}\lambda$ plus the number of delivery cycles required to remove the first $\frac{1}{12}\lambda\sqrt{n}$ messages from FT. Since the input channels are full for t cycles, the behavior of the oblivious switch during these cycles is determined. Since the root capacity is $\sqrt{n}$, the total number of messages removed from FT during the first t delivery cycles is $m = \frac{1}{6}\lambda\sqrt{n}$.

The behavior of the root switch determines how many of the m messages removed from FT 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.

Having determined the unfavored subtree given the conditions specified so far, we can complete the construction of M_n. The unfavored subtree will contain a copy of the bad message set $M_{n/4}$ for that subtree. Each of the other three subtrees will contain $\frac{1}{6}\lambda\sqrt{n}$ messages evenly divided among the processors in the subtree. Now we must show 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, and then we show that it satisfies properties 1, 2, and 3. As a preliminary step, observe that the number of messages provided by the unfavored subtree by delivery cycle

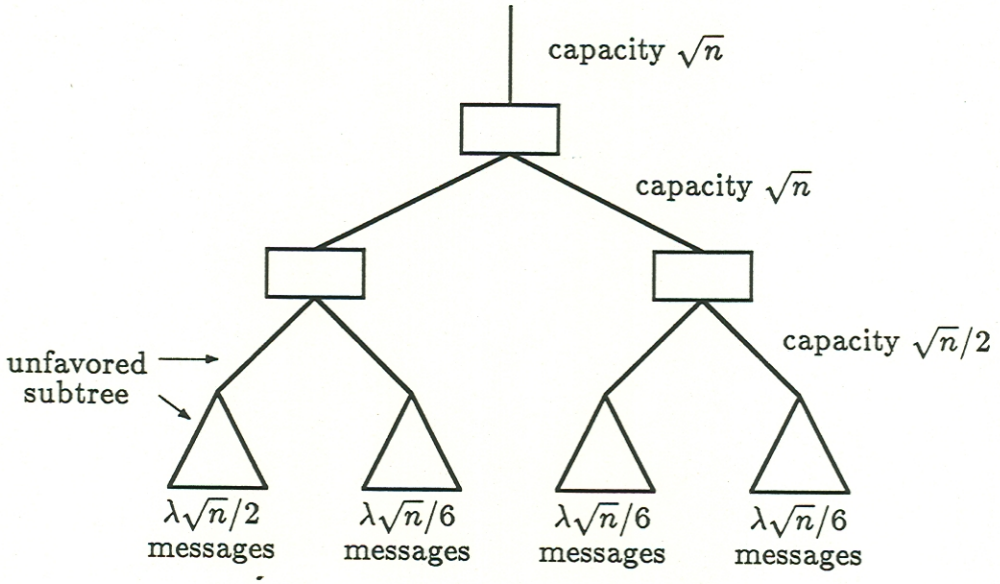


Figure 6: Construction of M_n for the proof of Theorem 9. The subtree from which the fewest number of messages have been delivered by a certain time is loaded with the largest number of messages.

t is at most $m/4 = \frac{1}{12}\lambda\sqrt{n/4}$, which we shall use to invoke the induction hypothesis on the subtree.

To show that the input channels of the root switch of FT are full through the first t delivery cycles, it suffices to show 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), since we have shown that the number of messages removed from this subtree by delivery cycle t is sufficiently small. The root channel of each other subtree is also full since it is the source of at most m messages during the first t delivery cycles, and the subtree's message set consists of m messages arranged in such a way as to maintain a full root channel at least until m messages have been delivered.

We now show that the three properties hold for M_n. The load factor is at most λ in each of the subtrees, so 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 FT 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 $\frac{1}{12}\lambda$ delivery cycles, we can still invoke the induction hypothesis to conclude that an additional $\frac{1}{24}\lambda \lg(n/4)$ cycles are required to empty the unfavored subtree. Thus the total number of cycles required to deliver all the messages in M_n is at least $\frac{1}{24}\lambda \lg n$. ∎

When probabilistic admissable switches are permitted, the proof of Theorem 9 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 4). 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 the lower bound we proved for routing messages out the root is achievable. 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*.

6 Further results

This section contains additional results relevant to routing on fat-trees. We first present an improved version of the universality theorem from [10]. Then we give two results on fat-tree routing in special cases.

Universality

The performance of the routing algorithm *RANDOM* allows us to generalize the universality theorem from [10], 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 was off-line. Our results show that the simulation can be carried out in the more interesting on-line context.

> **Theorem 10** *Let FT be a 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 $1 - O(1/n)$.*

Sketch of proof. The proof parallels that of [10]. 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 8, 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. ∎

Off-line routing

Our analysis for *RANDOM* has repercussions for the off-line routing case. Since 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, there must exist off-line schedules using only this many delivery cycles, which improves the bound of $O(\lambda \lg n)$ given in [10]. The previous off-line bound was proved by deterministically constructing a routing schedule that achieves the bound. Our better bound does not yield a deterministic construction of the routing schedule, but it does yield a probabilistic one.

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 [13] 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.

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 5 with $p = 1/z$ to conclude that the probability that channel c is congested in cycle i is at most

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

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 $z \le 2e2^{(k+2)/a}\lambda(M) = O(\lambda(M)) = O(n^k)$ cycles, the probability is $O(1/n)$ that there exists a congested channel in any delivery cycle of the pass. ∎

```
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.

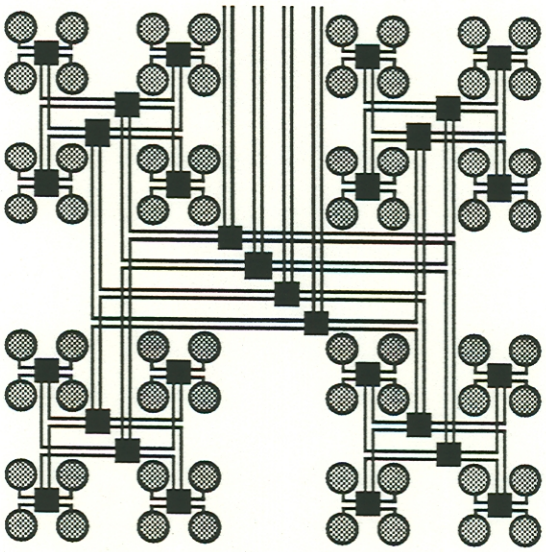


Figure 8: Another fat-tree design. The switches in this structure have constant size.

Another universal network

We have recently discovered a fat-tree design which uses simpler switches than the fat-tree described in Section 1 and [10]. 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 $\Theta(n \lg^2 n)$.[1] 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 10, 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

[1] 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].

number of delivery cycles needed to route a set M of messages is $O(\lambda(M)\lg^2 n)$, however, which yields $\lambda(M)\lg^3 n$ total time, as compared with $(\lambda(M) + \lg n \lg\lg n)\lg^2 n$ 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 $|M| = O(n^k)$.

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 $B(1, \text{load}(M,c), p/\text{cap}(c)) \le ep\lambda(M)$, and an analogue to Lemma 5 holds because the capacity of w is 1. Lemma 6, 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 $z \ge r\lambda(M) = \Theta(\lambda(M)\lg n)$, 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 $r\lambda(M)$.

7 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 $\frac{1}{24}$. Simulations indicate that a greedy algorithm might actually work quite well [6], but we have been unable to provide 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] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, Vol. C–34, No. 10, October 1985.

[11] M. Maley, private communication, October 1984.

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

[13] 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.

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

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

[16] 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.

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