Tolerating Faults in Synchronization Networks

Sandeep N. Bhatt[*] *Fan R. K. Chung*[†]
F. Thomson Leighton[‡] *Arnold L. Rosenberg*[§]

Abstract. A ***synchronization network*** (*SN*) consists of processing elements (PEs) at the leaves of a complete binary tree, with routing switches at interior nodes. We study the problem of rendering an SN tolerant to PE failures, by adding queues to its edges. We obtain the following results. In the ***worst-case***, an N-PE SN whose edges have queues of capacity $O(\log\log N)$ can tolerate the failure of a positive fraction of its PEs, no matter how the failed PEs are distributed; furthermore, this capacity requirement cannot be lowered by more than a small constant factor. In the *expected-case*, with probability exceeding $1 - N^{-\Omega(1)}$, an N-PE SN whose edges have queues of capacity $O(\log\log\log N)$ can tolerate the failure of a positive fraction of its PEs; we do not know if this capacity requirement can be lowered. We also present an algorithm which, given an SN with queues of capacity C, ***salvages*** a maximum number of fault-free PEs; the running time is a low-degree polynomial in N even when C is as large as $\log(N/\log N)$.

1 Introduction

A synchronization network (*SN*) consists of processing elements (PEs) that communicate through a complete binary tree interconnection network. The leaves of the tree hold the PEs of the SN (whence N is typically a power of 2); the nonleaf nodes route messages and perform simple combining, broadcast, and accumulation. An SN can be a useful ***auxiliary network*** when adjoined to a processor network having a richer topology (say, a mesh or hypercube). A variety of computations [4] yield to the simple, fast combining within trees; these are exploited in [3] and [10], where SNs are adjoined to data-processing machines for speedy searching, selection, and combining; most recently, in [5] and [6], SNs have been adjoined to MIMD hypercubes.

[*]Dept. Computer Science, Yale University, New Haven, CT 06520 USA.
[†]Bell Communications Research, 435 South St., Morristown, NJ 07960 USA.
[‡]Dept. of Mathematics and Lab. for Computer Science, MIT, Cambridge, MA 02139 USA.
[§]Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003 USA.

This study is motivated by the vulnerability of aggressive VLSI designs to fabrication defects which almost certainly disable some fraction of the PEs. (Wires and communication nodes, being considerably smaller than PEs, are commensurately less vulnerable to both defects and faults [11].) This has led researchers to seek strategies for retaining a large proportion of the computational power of a processor network even after it has been crippled by PE failures; studies have focused on the hypercube [7], the de Bruijn graph and the butterfly [2], the mesh [12], [9], and the tree [1]. This paper considers the problem of rendering an SN tolerant to PE failures. Although this problem is nominally subsumed by the study of fault tolerance in trees [1], the special structure and mode of use of SNs opens avenues to fault tolerance that are significantly — in fact, exponentially — more efficient than analogous techniques for general tree machines. We achieve the desired tolerance to faults by adding small-capacity queues to the edges of the SN. The quality of our solution is gauged in terms of the capacities of the queues.

Our study considers three possible demands on the design of a fault-tolerant N-PE SN $\mathcal{N}$, delimited by the following scenarios. In the *worst-case* scenario (Section 2), we insist that the fault-tolerant version of $\mathcal{N}$ be able to survive the failure of a positive fraction α of its PEs, no matter how the failures are distributed. We show that if the edges of $\mathcal{N}$ are equipped with queues of capacity[1] $\log^{(2)} N + c(\alpha)$, where $c(\alpha)$ is a constant depending only on the fraction α, then $\mathcal{N}$ can tolerate the failure of any αN of its PEs (Theorem 2.1). Moreover, we show that, this queue-capacity is necessary, to within a small constant factor; i.e., there is a set of αN PEs whose failure render an N-PE SN inoperative unless the edge-queues have capacity $\Omega(\log^{(2)} N)$ (Theorem 2.2).

In the *expected-case* scenario (Section 3), we insist that a set of up to αN PE failures be tolerated ***with very high probability,*** where the probability is determined by considering the relative frequencies of various failure patterns — assuming processors fail independently with fixed probability. We show that, with probability exceeding $1 - N^{-\Omega(1)}$, the SN, equipped with queues of capacity $O(\log^{(3)} N)$, can tolerate the failure of a positive fraction $\alpha < .7$ of its PEs (Theorem 3.1). It remains an inviting challenge to determine whether or not a smaller queue-capacity suffices.

In the *salvaging* scenario (Section 4), we are given a version of the SN already equipped with queues of some given capacity C. Our task is to determine how to configure $\mathcal{N}$ for a given pattern of failed PEs, in a way that salvages a maximum number of fault-free PEs. The salvage algorithm we present works in an amount of time that grows with the queue-capacity C, but is a low-degree polynomial in N even when C is as large as $\log(N/\log N)$ (Theorem 4.1).

Our results derive from a new notion of graph embedding which is presented in Subsection 1.1. Our lower bounds are among the first based solely on the congestion of embeddings.

[1]All logarithms are to the base 2. The iterated logarithm $\log^{(k)}$ is defined by: $\log^{(1)} N = \log N$; $log^{(k+1)} N = \log\log^{(k)} N$.

Before turning to the formal development, it is worth asking whether or not the added queues degrade computation on the SN enough to negate the benefits of the SN. We believe that the answer is generally no; we justify this belief in terms of one specific application. In [5], [6], SNs are used as a mechanism for synchronizing a MIMD array of microprocessors. It is claimed there that an appropriately designed SN will accumulate and distribute the messages that lead to synchronization in time roughly commensurate with a single instruction cycle of the underlying processor network. In contrast, synchronization using network interconnections takes time proportional to the diameter of the network, with possibly nontrivial delay at each PE. Hence, even if our queues slow the operation of an SN down from a single instruction cycle-time to roughly $\log^{(2)} N$ such cycle-times, the network with the adjoined SN will still synchronize much faster than will the underlying network by itself. Moreover, one could probably design a way to bypass the queues unless they are needed.

1.1 The Formal Setting

Terminology. We label each node of the *height-n complete binary tree* $\mathcal{T}_n$ with a distinct binary word of length at most n, so that each node x is adjacent to its successors $x0$ and $x1$. For each $\ell \in \{0, 1, \ldots, n\}$, the 2^ℓ words/nodes of length ℓ form *level* $n - \ell$ of $\mathcal{T}_n$. The unique node at level n is the *root* of $\mathcal{T}_n$, and the 2^n nodes at level 0 are the *leaves* of $\mathcal{T}_n$. We say that node x is a *(proper) ancestor* of node y, or, equivalently, that node y is a *(proper) descendant* of node x, iff the string x is a (proper) prefix of the string y. For each node x of $\mathcal{T}_n$, the *subtree of* $\mathcal{T}_n$ *rooted at* x is the induced subgraph of $\mathcal{T}_n$ on the nodes $\{xy \; : \; 0 \leq |y| \leq n - |x|\}$. Finally, a *forest* is a nonempty set of complete binary trees.

Red-Green Graph Embeddings. Let us be given a complete binary tree $\mathcal{H}_n$ (the host tree) whose 2^n leaves have each been colored either red or green. Say that the fraction $0 < G \leq 1$ leaves have been colored green. The tree $\mathcal{T}_n$ represents the ideal SN we want to implement. Its green leaves represent functional PEs, and its red leaves represent faulty PEs. Let us further be given a complete binary tree $\mathcal{G}_k$ (the guest tree), where $k \leq \log G2^n$. $\mathcal{T}_k$ represents the actual SN we want to "salvage" from the fault-laden SN $\mathcal{T}_n$.

A *red-green embedding* (*RG-embedding*, for short) $\langle \mathbf{a}, \mathbf{r} \rangle$ of $\mathcal{G}_k$ in $\mathcal{H}_n$ has two components: (i) an assignment $\mathbf{a}$ of the nodes of $\mathcal{G}_k$ to nodes of $\mathcal{H}_n$ that maps each leaf of $\mathcal{G}_k$ to a unique *green* leaf of $\mathcal{H}_n$, and such that each ancestor x of a node y in $\mathcal{G}_k$ is mapped to an ancestor $\mathbf{a}(x)$ of node $\mathbf{a}(y)$ in $\mathcal{H}_n$[2], and (ii) a routing function $\mathbf{r}$ that assigns to each edge (x, y) of $\mathcal{G}_k$ a path in $\mathcal{H}_n$ that connects nodes $\mathbf{a}(x)$ and $\mathbf{a}(y)$.

[2]We term this latter property of RG-embeddings *progressiveness*.As in the ideal SN, messages follow an up-then-down path in a "progressively" salvaged SN.

By extension, an RG-embedding of a *forest* of trees in $\mathcal{H}_n$ is a set of RG-embeddings of the trees in the forest, whose leaf assignments are node disjoint. This node disjointness guarantees that if any subset of the trees in the forest are grown and combined into a single tree, an RG-embedding of that single tree can use the leaf assignments of the forest.

Costs of an Embedding. An RG-embedding $\langle \mathbf{a}, \mathbf{r} \rangle$ of $\mathcal{G}_k$ in $\mathcal{H}_n$ has two costs. The first is the *harvest* of the embedding, which equals the ratio $2^{k-n}/G$ of the number of leaves of $\mathcal{G}_k$ to the number of green leaves of $\mathcal{H}_n$. The second is *congestion*, which equals the maximum, over all edges (x, y) of $\mathcal{H}_n$, of the number of routing paths that cross edge (x, y)[3].

Within this formal setting, the specific problems we study are stated below.

CONGESTION-HARVEST TRADEOFF PROBLEMS. *Determine, as a function of the size parameter n of $\mathcal{H}_n$, the fraction $0 < G \leq 1$ of green leaves, and the desired harvest fraction*[4] $0 < H \leq 1/2$, *the smallest congestion $C = C(n, G, H)$ for which there is a congestion-C RG-embedding of $\mathcal{G}_{\lfloor \log HG2^n \rfloor}$ in $\mathcal{H}_n$:*
(a) *for the* worst-case *pattern of red and green leaves in $\mathcal{H}_n$;*
(b) *in the* expected case, *i.e., with probability $\geq 1 - 2^{-\Omega(n)}$, when leaves of $\mathcal{H}_n$ are colored red and green independently with probability p.*

HARVEST-MAXIMIZATION PROBLEM. *Given an integer $C \leq n$, find the largest k for which there is an RG-embedding of $\mathcal{G}_k$ in $\mathcal{H}_n$ with congestion $\leq C$.*

2 Optimizing Worst-Case Congestion

This section derives upper and lower bounds on the minimum congestion $C_{\min}$ that we must suffer in order to harvest a fraction H of the green leaves in the worst case. As we will see, the bounds are tight to within constant factors.

2.1 An Upper Bound

We formulate and analyze a "greedy" RG-embedding algorithm, in order to obtain an upper bound on the quantity $C_{\min}$. The algorithm proceeds level-by-level, bottom-up from level 0 to level n in $\mathcal{H}_n$. As each node x is encountered, the algorithm assigns node x a label $\lambda(x)$ which is the length-$(n+1)$ binary representation of a nonnegative integer. For such a label $\lambda(x) = \lambda_n(x)\lambda_{n-1}(x) \cdots \lambda_0(x)$, where

[3]The *dilation* measure [8] is not relevant here because of the assumed speed of the ideal SN relative to the speed of each individual PE [5].

[4]We cannot aim at harvests exceeding $1/2$, because the number of harvested leaves must be a power of 2, but G could be such that the largest power of 2 not exceeding $G2^n$ may be close to $G2^{n-1}$.

- $I(\lambda(x)) = \sum_{i=0}^{n} \lambda_i(x)2^i$, i.e., the integer value of the binary string $\lambda(x)$;

- $S(\lambda(x)) = \{i \mid \lambda_i(x) \neq 0\}$, i.e., the set of nonzero bit-positions in label $\lambda(x)$;

- $W(\lambda(x)) = |S(\lambda(x))|$, i.e., the *weight*, or, number of nonzero bit-positions in label $\lambda(x)$.

The intended interpretation of the labels is that if node x of $\mathcal{H}_n$ receives label $\lambda(x)$, then there is an RG-embedding of the forest $\{\mathcal{T}_k : k \in S(\lambda(x))\}$ in the subtree of $\mathcal{H}_n$ rooted at x. This interpretation and the progressiveness of RG-embeddings imply that $\lambda_k(x) = 0$ for all $k >$ level(x).

The Labelling/Embedding Procedure. Let C be the maximum congestion allowed in any RG-embedding. We exploit the fact that all labels have length $n+1$ by specifying each label $\lambda(x)$ implicitly, via the integer $I(\lambda(x))$.

Algorithm Worst-Case:

```
Step 0. {Label nodes on level 0 of $\mathcal{H}_n$}
    Assign each leaf $x$ a label $I(\lambda(x)) = \begin{cases} 1 & \text{if } x \text{ is green} \\ 0 & \text{if } x \text{ is red} \end{cases}$
Step $\ell > 0$. {Label nodes on level $\ell$ of $\mathcal{H}_n$}
    Substep $\ell.a$ {Assign the string label}
        Assign each level-$\ell$ node $x$ a label $I(\lambda(x)) = I(\lambda(x0)) + I(\lambda(x1))$
    Substep $\ell.b$ {Combine small embedded trees}
        if there was a chain of carries from bit-positions $k-i, k-i+1, \ldots k-1$
        of $\lambda(x0)$ and $\lambda(x1)$ into bit-position $k$ of $\lambda(x)$
        then embed the roots of copies of $\mathcal{T}_{k-i+1}, \ldots, \mathcal{T}_k$ in node $x$, and
        route edges from those roots to the roots of two copies each of $\mathcal{T}_{k-i}, \ldots, \mathcal{T}_{k-1}$
        that are embedded in proper descendants of $x$
    Substep $\ell.c$ {Honor the congestion bound $C$}
        for $k = 0$ to $\lceil \log I(\lambda(x)) - C \rceil$
        if $W(\lambda(x)) > C$ then $\lambda_k(x) \leftarrow 0$
```

Theorem 2.1 *Let the 2^n leaves of $\mathcal{H}_n$ be colored red and green, in any way, with $G2^n$ green leaves for some $0 < G \leq 1$. For any rational $0 < H \leq 1/2$, Algorithm* **Worst-Case** *finds an RG-embedding of $\mathcal{G}_{\lfloor \log HG2^n \rfloor}$ in $\mathcal{T}_n$, with congestion $C \leq$* $\log n - \log((1-H)G) + 1$.

Proof. It is clear that, for each node x of $\mathcal{H}_n$, Algorithm **Worst-Case** RG-embeds $\mathcal{G}_{\lfloor \log I(\lambda(x)) \rfloor}$ in the subtree of $\mathcal{H}_n$ rooted at node x; hence, overall, the Algorithm RG-embeds the tree $\mathcal{G}_{\lfloor \log I(\lambda(r)) \rfloor}$ in $\mathcal{H}_n$, where r is the root of $\mathcal{H}_n$.

We need only verify that the salvaged tree is big enough when C is as big as the bound in the statement of the Theorem.

Note first that Algorithm **Worst-Case** never requires us to abandon any green leaves as we work up from level 0 through level $C-1$ of $\mathcal{H}_n$, because the high-order bit of $I(\lambda(x))$ can be no greater than the level of x in $\mathcal{H}_n$. To see what happens above this level, focus on a specific node x at level $\ell \geq C$ of $\mathcal{H}_n$. The congestion bound may require us, in Substep $\ell.c$ of the Algorithm, to abandon one bit in each position $k \leq \ell - C$ of $\lambda(x)$. This is equivalent to abandoning one green-leafed copy of each tree $\mathcal{G}_k$ with $k \leq \ell - C$; however, at most one tree of each size is abandoned, because any two trees of the same size would have been coalesced (by embedding a new root) at this step, if not earlier. It follows that, when the Algorithm processes node x, it abandons no more than $\sum_{i=0}^{\ell-C} 2^i < 2^{\ell-C+1}$ green leaves; hence, at the entire level ℓ, strictly fewer than $2^{\ell-C+1}2^{n-\ell} = 2^{n-C+1}$ previously unabandoned green leaves are abandoned. Thus, the entire salvage procedure abandons fewer than $(n+1-C)2^{n-C+1}$ green leaves due to congestion. Since there are $G2^n$ green leaves in all, we see that more than $(G-(n+1-C)2^{1-C})2^n$ green leaves are *not* abandoned due to congestion. Now, at the end of the Algorithm, we may have to abandon almost half of these unabandoned green leaves — because the green leaves we finally use in the RG-embedding must be a power of 2 in number. The Algorithm will have succeeded in salvaging the desired fraction of green leaves as long as the number of salvaged green leaves, which is no fewer than $2^{\lfloor \log(G-(n+1-C)2^{1-C})2^n \rfloor}$, is at least as large $2^{\lfloor \log HG2^n \rfloor}$, the number of green leaves we want to salvage.

Simple estimates show that, if we allow our RG-embedding to have congestion $C = \log n - \log((1-H)G) + 1$, then we shall have accomplished this task. □

2.2 A Lower Bound

Theorem 2.2 *Let G and H be rationals with $0 < G < 1$ and $0 < H \leq 1/2$. For each n, there exists a coloring of the leaves of $\mathcal{H}_n$ with the colors red and green, with the fraction $G2^n$ leaves colored green, such that every RG-embedding of some $\mathcal{G}_m$ in $\mathcal{H}_n$, where $2^m \geq HG2^n$, has congestion $C > (const)\log n$.*

Proof. Let us be given an algorithm, call it Algorithm **A**, that solves our Congestion-Harvest Tradeoff Problem in the worst-case scenario, while incurring minimal congestion. By Section 2.1, we know that the congestion incurred by Algorithm **A** for *any* coloring of the leaves of $\mathcal{H}_n$, in particular for the advertised malicious coloring, is $C \leq (const)\log n$.

The coloring of $\mathcal{H}_n$ that we claim defies efficient salvage is described as follows (L is a parameter we fix later, and n is a multiple of L): for each level $\ell = 0, L, 2L, \ldots, n$, proceed left to right along the level-ℓ nodes of $\mathcal{T}_n$, coloring red all of the leaves in the subtree rooted at every 2^L-th node encountered. All other leaves are colored red.

This coloring can be viewed as turning $\mathcal{H}_n$ into a complete $(2^L - 1)$-ary tree, all of whose leaves are colored green, providing that we look only at levels whose level-numbers are divisible by L. Therefore, our RG-embedding problem now assumes some of the flavor of the problem of efficiently embedding a complete binary tree into a complete $(2^L - 1)$-ary tree that is only slightly larger (by roughly the factor $1/H$). The results in [8] about a similar problem lead us to expect the large congestion that we now show must occur.

Our coloring of $\mathcal{H}_n$ has left the tree with $(2^L - 1)^{n/L}$ green leaves. The worst-case scenario of our Congestion-Harvest Tradeoff Problem assumes that the number of green leaves in $\mathcal{H}_n$ is a positive fraction $G > 0$ of the total number of leaves, namely, 2^n. Elementary estimates show that this assumption implies that $L \geq \log n - \log^{(2)} n - c(G)$ for some constant $c(G) > 0$ depending only on G. In analyzing our putative Algorithm **A**, we may, therefore, assume that we are dealing with RG-embeddings whose congestions satisfy $C < \frac{1}{3}L$ (or else, we have nothing to prove).[5]

For each $\ell \in \{0, 1, \ldots, n/L\}$, let $M(\ell)$ denote the number of leaves in the largest green-leafed tree that can be RG-embedded at a level-ℓL node of $\mathcal{H}_n$, according to Algorithm **A**. Even if there were no bound on congestion, the similarity of the colored version of $\mathcal{H}_n$ and a complete $(2^L - 1)$-ary tree would guarantee that $M(\ell + 1) \leq (2^L - 1)M(\ell)$ for $0 \leq \ell < n/L$.

In order to appreciate the effect of the bound C on congestion, note that, if the embedding of Algorithm **A** has congestion $\leq C$, then each number $M(\ell)$ must be representable as the sum of no more than C powers of 2; in other words, the binary representation of $M(\ell)$ can have weight no greater than C.[6] It follows in particular that $M(1) \leq 2^L - 2^{L-C}$. Starting from this upper bound on $M(1)$, we derive a sequence of upper bounds on the other numbers $M(\ell)$. It is clerically convenient to number the bit-positions in the shortest binary representation of $M(\ell)$ from left to right, i.e., high order to low order. Moreover, we need only restrict attention to bit-positions $1, 2, \ldots, L$, as will become clear in the course of the argument.

Focus on a specific $\ell \in \{0, 1, \ldots, n/L - 1\}$ and on its associated $M(\ell)$. Note the effect of proceeding from $M(\ell)$ to $M(\ell + 1)$, $M(\ell + 2)$, and so on. When we multiply $M(\ell)$ by $2^L - 1$ (thereby going up L levels in $\mathcal{H}_n$), the multiplication affects only the rightmost 1 in bit-positions $1, 2, \ldots, L$; say this rightmost 1 appears in bit-position k. The effect of the multiplication, in the presence of our bound C on the congestion of the RG-embedding, is as follows. Our assumption that the rightmost 1 appears in bit-position k means that the high-order L bit-positions end with a string $100 \cdots 0$, of length $\min(C - k + 1, L - k) + 1$. The multiplication has the effect of replacing this string with the like-length string

[5]Our assumption that $C < \frac{1}{3}L$ simplifies the clerical details of the upcoming argument.

[6]This is true because we focus on *progressive* RG-embeddings. If we allowed arbitrary RG-embeddings, then $M(\ell)$ would be the *algebraic* sum — i.e., the sum/difference — of at most C powers of 2. The added generality of arbitrary RG-embeddings would influence only constant factors in our bounds.

$011\cdots1$. Note that the resulting bit-string satisfies two conditions: (a) it has weight no greater than C, and (b) its rightmost 1 appears in bit-position $\leq L$.

If we continue the process of multiplying by 2^L-1 and "pruning" to maintain the bound on congestion, we find that, every so often — we shall estimate how often — a 1 that originated in a bit-positions $k \in \{1,2,\ldots,C\}$ in the binary representation of $M(1)$, together with every 1 that is "spawned" by it in subsequent multiplications and "prunings," ceases to exist. When such an *annihilation* occurs, we have lost at least the fraction 2^{-k+1} of the green leaves we could conceivably have been salvaging at that point.

The basis for our lower bound on C resides in our ability to bound how many multiplications and "prunings" have to take place before a 1 that began in bit-position k is annihilated. Since each such annihilation loses us a significant fraction of the green leaves, we wish to maximize the stretches of time between annihilations — by having $M(1)$ assume its maximum possible value, namely, $M(1) = 2^L - 2^{L-C}$, and by "pruning" as few leaves as possible after each multiplication. A corollary of this strategy is that we always strive to have the bit-string in positions $1,2,\ldots,L$ of our expression for the values $M(\ell)$ have maximum possible weight, namely C.

Let us focus on a 1 that originated in bit-position $k \leq C$ in $M(1)$. Once this 1 begins to "move" in the multiply-then-"prune" game — which occurs when it becomes the rightmost 1 in the surviving representation — and until this 1 is annihilated, the configuration of bit-positions $1,2,\ldots,L$ has the form $11\cdots10x$, where (a) bit-positions $1,2,\ldots,k-1$ contain the string $11\cdots1$, (b) bit-position k contains a 0, and (c) bit-positions $k+1,k+2,\ldots,L-1$ contain a bit-string x of weight at most $C-k+1$.

Since the numerical value of the bit-string x in bit-positions $k+1,k+2,\ldots,L-1$ decreases monotonically during the multiply-then-"prune" game, it follows that, once it starts to move, the 1 that began in bit-position k is annihilated after no more than $\sum_{i=0}^{C-k+1}\binom{L-k}{i} < 2\binom{L-k}{C-k+1}$ steps. Therefore, the *total* time it takes to annihilate the 1 that originated in bit-position k, from the very start of the multiply-then-"prune" game — which is the time required to annihilate every 1 that originated in bit-positions $k, k+1, \ldots C$, is bounded above by

$$T(k) < 2\sum_{i=k}^{C}\binom{L-i}{C-i+1} \leq 2\binom{L-k+1}{C-k+1} < 2\left(\frac{e(L-k+1)}{C-k+1}\right)^{C-k+1}.$$

Now, if we play the multiply-then-"prune" game long enough that we annihilate a 1 in some bit-position $k \leq h =_{\text{def}} -\lfloor\log(1-H)\rfloor$, then we shall have lost more than the fraction $1-H$ of the green leaves; hence, we shall have failed in our assigned task of salvaging at least the fraction H of these leaves. Therefore, the depth n of the SN $\mathcal{H}_n$ had better be small enough to preclude our playing the game for this many steps. In other words, we must have

$$\frac{n}{L} \leq (const)\left(\frac{e(L-h+1)}{C-h+1}\right)^{C-h+1}.$$

This inequality implies $C \geq (const)L = \Omega(\log n)$. □

3 Optimizing Expected Congestion

This section derives bounds on the congestion one must incur in order to survive "random" faults. We adopt the model that predominates in the literature by assuming that the PEs of our SNs fail independently, with probability 1/2.[7] It remains an inviting challenge to determine whether or not the upper bound can be lowered.

3.1 An Algorithm with Good Expected Behavior

We show that a modified version of Algorithm **Worst-Case** of Section 2 incurs congestion that is only *triply* logarithmic in the size of the salvaged SN, with extremely high probability, providing that we lower our demands a bit. Specifically, we reduce our demand that we salvage the fraction H of the surviving PEs of our SN to the demand that we salvage only the fraction $0.3H$ of these PEs.

Theorem 3.1 *Let the leaves of* $\mathcal{H}_n$ *be colored red and green, independently, with probability* $1/2$. *For any rational* $0 < H \leq 1/2$, *with probability* $\geq 1 - 2^{-\Omega(n)}$, *a modification of Algorithm* **Worst-Case** *will find an RG-embedding having congestion* $C \leq \log^{(2)} n - \log 0.3(1-H) + 1$ *of some* $\mathcal{G}_m$ *in* $\mathcal{H}_n$, *where* $m \geq n - \lceil \log 10n \rceil + \lfloor \log 2.5n \rfloor$.

The major insight leading to the desired modification of Algorithm **Worst-Case** resides in the following combinatorial fact.

Lemma 3.1 *Let each leaf of* $\mathcal{H}_n$ *be colored red or green, independently, with probability* $1/2$. *Fix any partition of the leaves of* $\mathcal{H}_n$ *into blocks of size* $10n$. *Then, with probability* $\geq 1 - 2^{-\Omega(n)}$, *each block contains at least* $2.5n$ *green leaves.*

Proof. Focus first on a single block of $10n$ leaves.

$$\begin{aligned} Pr(< 2.5n \text{ green leaves}) &= Pr(> 7.5n \text{ red leaves}) \\ &= 2^{-10n} \sum_{k=7.5n+1}^{10n} (\text{number of ways to choose } k \text{ red leaves}) \end{aligned}$$

[7]Changing the probability 1/2 to any fixed p merely changes the constants in our results.

This last sum is easily transformed to

$$\sum_{k=0}^{2.5n-1} \binom{10n}{k} 2^{-10n} \leq 2 \binom{10n}{2.5n} 2^{-10n} \leq 2 \left(\frac{10e}{2.5}\right)^{2.5n} 2^{-10n} \leq 2 \left(\frac{\epsilon}{2}\right)^{n}$$

for some $\epsilon < 1$. It follows that

$$Pr(\geq 2.5n \text{ green leaves}) \geq \left(1 - 2\left(\frac{\epsilon}{2}\right)^n\right)^{2^n/10n} \geq \exp(-c_1\epsilon^n/n) \geq 1 - \frac{1}{n2^{c_2 n}}$$

for some constants $c_1, c_2 > 0$. □

Proof of Theorem 3.1. Lemma 3.1 tells us that when we look at the labels assigned by our greedy algorithm to nodes at or above level $\lceil \log 10n \rceil$ of a randomly colored instance of $\mathcal{H}_n$, then, with very high probability, we find every node having a label $\lambda(x)$ for which $I(\lambda(x)) \geq 2.5n$. Let $m = n - \lceil \log 10n \rceil + \lfloor \log 2.5n \rfloor$. If we now abandon all but exactly $2^{\lfloor \log 2.5n \rfloor}$ of the salvaged green leaves at each of these nodes, we can "assemble" a salvaged copy of the 2^m-leaf SN $\mathcal{G}_m$ without incurring any further congestion. □

4 Optimizing Worst-Case Harvest

Algorithm **Worst-Case** of Section 2.1 is guaranteed to be efficient, both in running time — it operates in time $O(2^n)$, which is linear in the size of $\mathcal{H}_n$ — and in harvest — it salvages the fraction H of the green leaf-PEs. But, it is easy to find examples where a nongreedy strategy allows one to salvage a much larger fraction of the green leaves. In particular, when the green leaves are spread out sparsely, any greedy approach abandons many more green leaves than it has to. One finds an analogous deficiency in a "lazy" salvage strategy — one that coalesces small-order trees as late as possible, rather than as early as possible; lazy strategies abandon too many green leaves when the leaves are packed densely, in clumps. It might be of practical interest, therefore, to find a computationally efficient algorithm that salvages optimally many green leaves, while honoring a prespecified limit, C, on congestion. This section presents such an algorithm.

4.1 The Algorithm

Overview. Our salvage algorithm proceeds up $\mathcal{H}_n$, from level 0 to level n, labeling each node x at level ℓ with a *set* $\Lambda(x)$ of length-$(\ell+1)$ integer vectors. Each vector in $\Lambda(x)$ will indicate one possible salvage decision available to x; in particular, for each vector $\langle \nu_0, \nu_1, \ldots, \nu_\ell \rangle$:

- there is an RG-embedding in the subtree of $\mathcal{T}_n$ rooted at x of a green-leafed forest $\mathcal{F}$ containing ν_k disjoint copies of $\mathcal{T}_k$, for $0 \leq k \leq \ell$;

- $\sum_{k=0}^{\ell} \nu_k \leq C$, so that the bound on congestion is always honored.

When we get to the root of $\mathcal{H}_n$ (where $\ell = n$), we select the largest level k for which some vector in the set that labels the root has $\nu_k > 0$. Our harvest, then, is a green-leafed copy of $\mathcal{G}_k$.

The Labelling/Embedding Procedure. We associate each level-ℓ node x of $\mathcal{H}_n$ with a trie (i.e., a digital search tree) of height $\ell + 1$. This trie will store the label-set $\Lambda(x)$ in the obvious way. We now present the details of Algorithm **Optimal-Harvest.**

Algorithm Optimal-Harvest:

```
Step 0. {Label nodes on level 0 of $\mathcal{H}_n$}
    Assign each leaf $x$ a label $\Lambda(x) = \begin{cases} \{\langle 1\rangle\} & \text{if } x \text{ is green} \\ \{\langle 0\rangle\} & \text{if } x \text{ is red} \end{cases}$
Step $\ell > 0$. {Label nodes on level $\ell$ of $\mathcal{T}_n$}
    Substep $\ell.a$ {Assemble the vectors }
        Assign each level-$\ell$ node $x$ a label as follows.
        for each pair of length-$\ell$ vectors $\xi \in \Lambda(x0)$ and $\eta \in \Lambda(x1)$, place the
        length-$(\ell+1)$ vector $\zeta$ in $\Lambda(x)$, where
            $\zeta_k = \begin{cases} 0 & \text{if } k = \ell \\ \xi_k + \eta_k & \text{if } k \neq \ell \end{cases}$
    Substep $\ell.b$ {Combine small embedded trees}
        for each vector $\zeta$ of $\Lambda(x)$, for all $0 \leq k < \ell$, if component $\zeta_k$ of $\zeta$ is
        the sum of a nonzero $\xi_k$ (for some $\xi \in \Lambda(x0)$) and a nonzero $\eta_k$ (for
        some $\eta \in \Lambda(x1)$), then add to $\Lambda(x)$ a vector $\zeta'$ that agrees with $\zeta$
        except in positions $k, k+1$; specifically:
            $\zeta'_i = \begin{cases} \zeta_{i+1} + 1 & \text{if } i = k+1 \\ \zeta_i - 2 & \text{if } i = k \\ \zeta_i & \text{otherwise} \end{cases}$
        and embed the root of a copy of $\mathcal{G}_{k+1}$ in $x$, routing edges from that
        root to the roots of two copies of $\mathcal{G}_k$ that are embedded in proper
        descendants of $x$
    Substep $\ell.c$ {Honor the congestion bound $C$}
        for each vector $\xi \in \Lambda(x)$
        if $\sum_{k=0}^{\ell} \xi_k > C$ then replace $\xi$ in $\Lambda(x)$ by all possible vectors $\xi'$ such
        that $\xi'_k \leq \xi_k$ for all $0 \leq k \leq \ell$, and such that $\sum_{k=0}^{\ell} \xi'_k \leq C$.
```

In the full version of the paper we establish the following theorem. The proof is omitted here for lack of space.

Theorem 4.1 *Let the leaves of $\mathcal{H}_n$ be colored red and green, in any way, and let $1 < C \leq n$. Algorithm* **Optimal-Harvest** *finds, in time $O(n^{3C+2}2^n)$, an RG-embedding of some $\mathcal{G}_m$ in $\mathcal{H}_n$, having congestion $\leq C$ and having optimal harvest among embeddings with congestion C.*

Note that for $C = \frac{1}{3}\left(\frac{n}{\log n} - 2\right)$, the time is quadratic in the size of $\mathcal{H}_n$.

ACKNOWLEDGMENTS: The authors thank Don Coppersmith for helpful suggestions. The research of S. N. Bhatt was supported in part by NSF Grant CCR-88-07426, by NSF/DARPA Grant CCR-89-08285, and by Air Force Grant AFOSR-89-0382; the research of F. T. Leighton was supported in part by Air Force Contract OSR-86-0076, DARPA Contract N00014-80-C-0622, Army Contract DAAL-03-86-K-0171, and NSF Presidential Young Investigator Award with matching funds from ATT and IBM; the research of A. L. Rosenberg was supported in part by NSF Grants CCR-88-12567 and CCR-90-13184. A portion of this research was done while S. N. Bhatt, F. T. Leighton, and A. L. Rosenberg were visiting Bell Communications Research.

References

[1] A. Agrawal (1990): Fault-tolerant computing on trees. Typescript, Brown Univ.

[2] F.S. Annexstein (1989): Fault tolerance in hypercube-derivative networks. *1st ACM Symp. on Parallel Algorithms and Architectures*, 179-188.

[3] J.L. Bentley and H.T. Kung (1979): A tree machine for searching problems. *Intl. Conf. on Parallel Processing*, 257-266.

[4] S. Browning (1980): *The Tree Machine: A Highly Concurrent Computing Environment.* Ph.D. Thesis, CalTech.

[5] R.D. Chamberlain (1990): Multiprocessor synchronization network: design description. Tech. Rpt. WUCCRC-90-12, Washington Univ.

[6] R.D. Chamberlain (1991): Matrix multiplication on a hypercube architecture augmented with a synchronization network. Typescript, Washington Univ.

[7] J. Hastad, F.T. Leighton, M. Newman (1989): Fast computation using faulty hypercubes. *21st ACM Symp. on Theory of Computing*, 251-263.

[8] J.-W. Hong, K. Mehlhorn, A.L. Rosenberg (1983): Cost tradeoffs in graph embeddings. *J. ACM 30*, 709-728.

[9] C. Kaklamanis, A.R. Karlin, F.T. Leighton, V. Milenkovic, P. Raghavan, S. Rao, C. Thomborson, A. Tsantilas (1990): Asymptotically tight bounds for computing with faulty arrays of processors. *31st IEEE Symp. on Foundations of Computer Science*, 285-296.

[10] C.E. Leiserson (1979): Systolic priority queues. *1979 CalTech Conf. on VLSI.*

[11] C. Mead and L. Conway (1980): *Introduction to VLSI Systems.* Addison-Wesley.

[12] P. Raghavan (1989): Robust algorithms for packet routing in a mesh. *1st ACM Symp. on Parallel Algorithms and Architectures*, 344-350.