Broadcast and Associative Operations on Fat-Trees*

G. Bilardi† B. Codenotti‡ G. Del Corso§ C. Pinotti¶
G. Resta‡

Abstract

This paper analyzes the cost of performing broadcast, product and prefix computation on the ideal fat-tree, a model proposed here to capture distance and bandwidth properties common to a variety of fat-tree networks. Algorithms are developed and analyzed in terms of the capacity of channels at different levels of the fat-tree. Non trivial lower bounds are derived establishing the optimality of our algorithms for a wide range of channel capacities.

1 Introduction

A number of networks have been introduced in the literature and referred to as *fat-trees*, e.g., the concentrator fat-tree, the pruned-butterfly fat-tree, and the sorting fat-tree. Loosely speaking, a fat-tree is a tree whose leaves act as input/output terminals, whose internal nodes are subnetworks with switching capability, and whose edges are channels of appropriate capacity. Proposed fat-trees differ in node structure and channel capacities.

Fat-trees have interesting *universality* properties in VLSI and form the basis for a number of universal routers [Lei85, GL89, LMR88, BB95, Gre94, BCDM94] and universal circuits [BB93]. The CM-5 parallel supercomputer uses a fat-tree as its interconnection pattern [LAD+92].

In spite of its wide use, the term fat-tree has not been defined precisely. A definition useful for the study of layout area has been proposed in [BB94].

*This research was partially supported by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (Project GEPPCOM).

†DEI, Università di Padova, Padova (Italy) and EECS, University of Illinois at Chicago, Chicago, (Illinois).

‡Istituto di Matematica Computazionale del CNR, Pisa (Italy).

§Dipartimento di Matematica, Università degli Studi di Milano, Milano (Italy).

¶Istituto di Elaborazione dell'Informazione del CNR, Pisa (Italy).

In the present paper (Section 2), we propose the *ideal* fat-tree model, which captures broad distance and bandwidth constraints of fat-tree networks and provides a basis to design algorithms portable among different fat-trees. We show how a fat-tree network can simulate an ideal fat-tree.

We study two classes of algorithms for ideal fat-trees. In the first class, the communication network is not pipelined, in the sense that it will not accept new messages while others are already in transit. This constraint leads to a natural decomposition of the computation as as sequence of rounds, which makes algorithms easier to design and analyze. Perhaps surprisingly, in many cases this constraint does not prevent achieving optimal performance. In a second, unrestricted class, the network accepts messages at any time.

Sections 3 and 4 respectively deal with upper and lower bounds to broadcast time on the ideal fat-tree, with and without pipelining. The derived bounds are in the form of recurrence relations, expressed in terms of a function $w(n)$, defining the capacity of the channels at the roots of subtrees with n leaves. We solve such recurrences for a number of significant shapes of $w(n)$, obtaining the results reported in Table 1.

Capacity	Lower Bounds		Upper Bounds	
$w(N)$	Unrestricted	No pipelining	Unrestricted	No pipelining
$O(1)$	$\Omega\left(\frac{\log^2 N}{\log\log N}\right)$	$\Omega(\log^2 N)$	$O\left(\frac{\log^2 N}{\log\log N}\right)$	$O(\log^2 N)$
$\log^{1+O(1)} N$	$\Omega\left(\frac{\log^2 N}{\log\log N}\right)$		$O\left(\frac{\log^2 N}{\log\log N}\right)$	
$2^{\log^\beta N}$, $\beta < 1$	$\Omega(\log^{2-\beta} N)$		$O(\log^{2-\beta} N)$	
$N^{1/\log\log N}$	$\Omega(\log N \log\log N)$		$O(\log N \log\log N)$	
N^β, $\beta \leq 1$	$\Omega(\log N)$	$\Omega(\log N \log\log N)$	$O(\log N \log\log N)$	

Table 1: Time bounds for broadcasting in an ideal fat-tree.

The resulting running time ranges from $O(\log N \log\log N)$ (for channel capacity proportional to subtree size) to $O(\log^2 N/\log\log N)$ (for constant channel capacity). We observe that pipelining is crucial to achieve optimal performance only for rather small capacities, whereas it plays no significant role for larger capacities (say, $w(n) \geq \log n$). A comparison between upper and lower bounds shows that our algorithms are optimal for a wide range of channel capacities. A $\log\log N$ gap remains for large capacities, in the unrestricted case, pointing at interesting aspects of the problem that require further investigation.

Associative product and prefix computations are the subject of Section 5. We show that these operations are related to broadcast in various ways, affording an extension both of the algorithmic techniques and of the lower bound results developed for broadcast.

Concluding remarks are in Section 6.

As broadcast is a basic primitive for parallel computation, it has been widely studied in many different models. Among the most similar to the ideal fat-tree are a number of models based on bandwidth/latency parameters such as the postal model [BK92], LogP [CK+93, KSSS93], BSP [Va90, BHPPS96], and the ring of processors of [RSS94] where messages can cross k links in time $O(\log k)$ (modeling the use of cascaded drivers to accelerate message transmission along MOS lines). Clearly, broadcast algorithms for most models resemble each other to some extent, since they are all based on replicating and distributing the message to be broadcast. The strategies differ in making different tradeoff between replication and distribution. In particular, while all the models cited in this paragraph are invariant under an arbitrary permutation of the processors, the ideal fat-tree is faster when communication occurs between processors that belong to a small subtree, which leads to different broadcasting strategies.

2 The Ideal Fat-Tree

We introduce the *ideal* fat-tree, the model of computation used in the rest of this paper.

For N a power of two, an ideal fat-tree is a network of N *processors* $P_0, P_1, \ldots, P_{N-1}$, placed from left to right at the leaves of a complete binary tree. We say that p is the identity of P_p.

The internal nodes of the tree represent *switching modules* forming a routing network among the processors. Except for the one at the root, each module is connected to its parent by two unidirectional channels (in opposite directions). Both such channels have *capacity* $w(n)$, where $n \in \{1, 2, \ldots, N/2\}$ is the number of leaves of the subtree rooted at that module node (Fig. 1 illustrates the structure of an ideal fat-tree).

We assume *global synchronization* among the nodes of the fat-tree. In one period of the global clock, a processor can execute one local operation, submit one message to the network, and receive one message from the network.

A *message* submitted to the network moves along the shortest path between its source and its destination in the tree, traversing one channel per period. Thus, the message arrives $2h$ steps after it is sent, where

Figure 1: An ideal fat-tree with $N = 16$ and $w(n) = \log n + 1$. Black circles represent processors. Grey circles represent the switching modules.

$h \in \{1, 2, \ldots, \log N\}$ is the height of the lowest common ancestor of source and destination.

An algorithm for an ideal fat-tree can be specified by a program for the generic processor, with a distinguished read-only variable interpreted as the processor identity. An algorithm is defined to be *legal* if, on any appropriate input, at any step, the number of messages traversing any channel does not exceed the capacity of that channel.

We can see that our model is ideal in the sense that routing time is assumed to be the minimum compatible with capacity and distance constraints. We propose the ideal fat-tree as a potentially portable abstraction for the class of fat-tree networks. (For a model different from ours, but similar in spirit, see [LM88].) Indeed, it can be shown that a specific fat-tree with good routing properties can simulate an ideal fat-tree with moderate slowdown. Technically, we have (see [Lei85] for the definition of the load factor λ):

Theorem 1 *Let F be a fat-tree network of N leaves with capacity function $w(n)$, such that a set of messages of load factor λ can be routed on line, in time $O(\lambda s(N) + d(N))$. Then, F can simulate an ideal fat-tree with the same capacity function $w(n)$ with slowdown $O(s(N) + d(N))$.*

As an example, for the randomized routing of constant-size messages on the butterfly fat-tree, $s(N) = 1$ and $d(N) = \log N$ [LMRR94]; thus, the slowdown to simulate an ideal fat-tree according to the preceding theorem would be $O(\log N)$. For for the deterministic routing of $O(\log N)$-size

messages on the sorting fat-tree, $s(N) = \log N$ and $d(N) = \log^2 N$ [BB95], yielding a slowdown of $O(\log^2 N)$.

We develop algorithms and derive lower bounds for a fat-tree with a general capacity function $w(n)$, simply assuming two natural constraints:

1. $w(2n) \geq w(n)$, that is, the outgoing bandwidth of a subtree does not decreases with its size;

2. $w(2n) \leq 2w(n)$, that is, the bandwidth toward the parent does not exceed the total bandwidth toward the children.

Typical $w(\cdot)$ functions are $w(n) = \Theta(n^{1/2})$, for area-universal trees, and $w(n) = \Theta(n^{2/3})$, for volume-universal trees.

We shall often make use of the following *pseudo-inverse* of $w(n)$:

$$w^{-1}(c) = \min\{n : w(n) \geq c\}. \tag{1}$$

Clearly, $w^{-1}(c)$ represents the number of leaves of the smallest subtree whose root capacity is at least c.

3 A Broadcast Algorithm

Simple diameter considerations show that any broadcast algorithm on an ideal fat-tree takes time $\Omega(\log N)$. An obvious algorithm using only one unit of capacity per channel easily achieves time $O(\log^2 N)$. The goal of this section is to design algorithms that improve on the $O(\log^2 N)$ performance by taking advantage of larger channel capacity. We also investigate the further potential offered by pipelining.

3.1 The Algorithm

We present a broadcast algorithm for fat-trees of size N with running time ranging from $O(\log N \log\log N)$ (for $w(n) = n$) to $O(\log^2 N/\log\log N)$ (for $w(n) = 1$). The algorithm maximizes parallelization while avoiding congestion. Roughly speaking, a certain number of copies of the message is generated and then distributed so that each subtree of a suitable size holds a copy; then the algorithm is recursively applied in each subtree.

More precisely, algorithm *Broadcast* (Algorithm 1) relies on a recursive procedure, *Sparse-Broadcast* (Algorithm 2). The goal of Sparse-Broadcast, called on a tree of size n, is to generate $r = \max(w(n), 2)$ uniformly spread copies of the message.

This is accomplished in two phases: in the first phase Sparse-Broadcast calls itself on a subtree of size $m = w^{-1}(\sqrt{r})$ [line 6], thus generating $\sqrt{r}$ copies of the message. This is the smallest subtree (hence the one with the minimum distance among the processors) whose root capacity is at least $\sqrt{r}$ (hence affording these copies to be distributed without congestion) [lines 7,8]. In the second phase, each of the $\sqrt{r}$ copies produced in the first phase is used to generates other $\sqrt{r}$ copies, by means of $\sqrt{r}$ simultaneous calls to Sparse-Broadcast on $\sqrt{r}$ subtrees of size m [lines 9,10]. Finally, the resulting r copies are distributed uniformly [lines 11,14].

Algorithm Broadcast first calls procedure Sparse-Broadcast on the same tree [line 3], which generates $r = \max(w(n), 2)$ copies of the message to be broadcast, one in each subtree of size n/r. Then Broadcast is recursively called in parallel [lines 4,5] on each of these subtrees.

3.2 Analysis

Let $S(N)$ denote the time needed to perform a Sparse-Broadcast on a tree of N leaves by Algorithm 2. We have:

$$S(N) \leq 2S(m) + 4\log N = 2S(w^{-1}(w^{1/2}(N))) + 4\log N\,, \qquad (2)$$

where the term $2S(m)$ accounts for the two calls on subtrees of size m [lines 6 and 10], and the term $4\log N$ accounts for the time to spread the copies [lines 7-8 and 11-15].

The time $T(N)$ of the full broadcast (Algorithm 1) satisfies

$$T(N) = S(N) + T\left(\frac{N}{w(N)}\right)\,. \qquad (3)$$

`Broadcast(first, n)`
{Send copy originally at `first` to leaves `first` $+ 1, \ldots,$ `first` $+$ `n` $- 1$}

```
1.   if n > 1 then
2.       r = max{w(n), 2};
3.       Sparse-Broadcast(first, n)
4.       for 0 ≤ i ≤ r − 1 pardo
5.           Broadcast(first + i n/r, n/r);
6.   endif
```

Algorithm 1: Broadcast.

`Sparse-Broadcast(first, n)`
{Send copy originally at leaf `first` to $\max\{w(\mathtt{n}), 2\}$ equally spaced leaves}

```
1.   r = max{w(n), 2};
2.   if r = 2 then
3.       P[first] → P[first + n/2];
4.   if r > 2 then
5.       m = w^{-1}(√r);
6.       Sparse-Broadcast(first, m)
7.       for 0 ≤ i ≤ √r − 1 pardo
8.           P[first + i m/√r] → P[first + i n/√r];
9.       for 0 ≤ i ≤ √r − 1 pardo
10.          Sparse-Broadcast(first + i n/√r, m);
11.      for 0 ≤ i ≤ √r − 1 pardo
12.          for 0 ≤ j ≤ √r − 1 pardo
13.              P[(first + i n/√r) + j m/√r] → P[(first + i n/√r) + j n/r];
14.      endfor
15.  endif
```

Algorithm 2: Sparse Broadcast. (Notation: $P[k] \rightarrow P[h]$ means that processor k sends the message to processor h.)

In the full paper, we solve the above equation for some interesting cases of function $w(n)$, with the results reported in Table 1 of the Introduction. We also provide a detailed discussion motivating the non obvious choice $m = w^{-1}(\sqrt{r})$ in Line 5 of procedure Sparse-Broadcast.

3.3 Pipelining

The performance of the algorithm described above can be improved by pipelining. After the execution of the sparse broadcast routine, we have $w(n)$ copies of messages. Note that, using pipelining, each message can be replicated $O(\log N)$ times in time $O(\log N)$. In fact, if a processor sends k messages in pipeline, the first message will reach its destination at most after $\log N$ time steps, and the last one after $\log N + k - 1$ time steps.

Thus, spending an additional $O(\log N)$ time, we can call recursively the broadcast algorithm on subtrees of size $N/(w(N)\log N)$ instead of

$N/w(N)$. The corresponding recurrence is

$$T'(N) = S(N) + T'\left(\frac{N}{w(N)\log N}\right) + \gamma \log N\,, \tag{4}$$

where γ is a small constant. Again, see Table 1 for solutions in specific cases.

4 Lower Bounds for Broadcast

In this section, we establish lower bounds for the running time $T(N)$ of broadcasting algorithms on ideal fat-trees with N leaves. A first bound applies to any algorithm, even with pipelining. The pipelined algorithm of Subsection 3.3 attains this lower bound for $w(N) \leq N^{1/\log\log N}$. A stronger lower bound is also derived for a restricted class of algorithms. Algorithm 1 of Subsection 3.1 belongs to this class and attains the lower bound for capacities $w(N) \geq \log N$.

4.1 A General Lower Bound

The next theorem provides a lower bound in the form of a recurrence relation.

Theorem 2 *The running time of any broadcast algorithm on an ideal fat-tree satifies*

$$T(N) \geq \log(N/2) + T\left(\frac{N/w(N/2)}{\log(2N/w(N))}\right).$$

Proof. The proof is in the Appendix. □

The lower bound of Theorem 2 matches the performance of our algorithm, for capacities not exceeding $N^{1/\log\log N}$. If $w(N) > N^{1/\log\log N}$, the trivial lower bound $\Omega(\log N)$ is obtained.

4.2 A Lower Bound for the Round Model (no Pipelining)

The constraint that the network can not be pipelined can be conveniently translated into the following rules.

Round–Model Rules:

- The algorithm consists of a sequence of *rounds*.
- During a round a processor can send or receive at most one message.

- All messages in a given round leave their source at the beginning of the round.
- A round ends when all the messages sent have reached their destination.

The time of a round is twice the *round height*, the maximum height in the tree reached by any message during that round. The time of an algorithm is the sum of the times of its rounds. It is easy to see that, when $w(N) \geq \log N$, Algorithm 1 of the previous section can be cast into the round model without increasing its running time.

In the round model, we have the following lower bound.

Theorem 3 *In the round model, the running time $T(N)$ of any broadcast algorithm satisfies*

$$T(N) > (1/2) \log N \sum_{i=0}^{\log \log N - 1} \frac{2^i}{\log(w(2^{2^i}) + 1)} .$$

Proof. The proof is in the Appendix. □

It is straightforward to derive the following corollary.

Corollary 4 *In the round model, the running time $T(N)$ of any broadcast algorithm satisfies*

$$T(N) = \Omega\left(\frac{\log^2 N}{\log w(N)}\right).$$

Corollary 5 *In the round model, for a fat-tree with $w(N) = N^\beta$, with $\beta \leq 1$, the running time $T(N)$ of any broadcast algorithm satisfies*

$$T(N) = \Omega(\log N \log \log N).$$

Our results on the complexity of broadcast are summarized in Table 1.

5 Associative Product and Prefix Computations

In this section, we consider two basic associative computations, which are intimately related to the broadcast operation, i.e. prefix and associative product in a semigroup.

Let S be a semigroup with the $\star$ operation, and let $a_0, a_1, \ldots, a_{N-1}$ be elements of S.

- For the prefix computation, we assume that

  (i) initially processor P_i stores the value a_i, $i = 0, 1, \ldots, N-1$;

  (ii) at the end of the computation, P_i contains $a_0 \star a_1 \star \cdots \star a_i$, $i = 0, 1, \ldots, N-1$.

- For the product, we assume that initially there is the same data distribution as above, and at the end of the computation P_0 stores $a_0 \star a_1 \star \cdots \star a_{N-1}$.

We have the following relations among the above computations and broadcast:

1. Product is part of prefix.
2. Assuming the existence of an identity element e, broadcast is a subcase of prefix when $a_0 = a$ (the value to broadcast) and $a_i = e$ for $i \geq 1$.
3. The interplay between product and broadcast is less obvious. It emerges by viewing any computation for broadcast as a sort of "reverse" of the product computation. More precisely, at each step of a product computation a node P_k computes, say, $A_i \star A_j$, where A_i and A_j are partial products computed at nodes i and j, respectively. In the corresponding broadcast computation, processor P_k sends the data to processors P_i and P_j. On the other hand, we can associate to any broadcast algorithm a computation graph, which keeps track, for all the nodes, of the node which sent it the data first. This computation graph is a tree, which, if visited from the leaves, provides a computation graph for the product. Note that the corresponding product algorithm requires, in general, commutativity. The above observations are summarized in Figure 2.

By taking take advantage of these properties we establish the following results (details in the full paper):

Theorem 6 *In the ideal fat-tree, prefix and product have the same asymptotic complexity as broadcast. In particular the best lower bound is exactly the same for the three problems.*

6 Conclusions

In this paper, we have designed efficient algorithms for broadcast and other related computations on idealized fat-trees, which are optimal for a wide

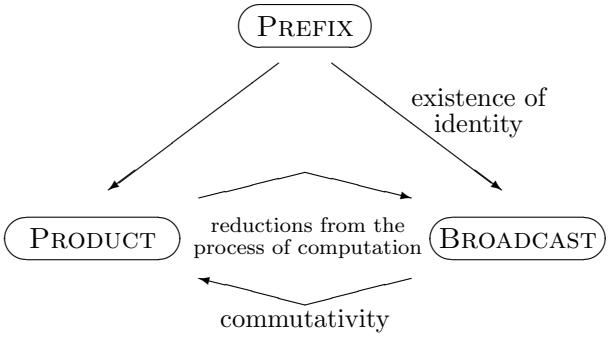

Figure 2: Relations between Broadcast, Product and Prefix.

range of the capacity of the interconnection network. We conjecture that optimality holds even outside this range, although a different lower bound argument might be needed. We have also presented a lower bound technique that applies to algorithms restricted to work in rounds. It would be interesting to establish that the above restriction is not necessary. Further work to be done include analyzing basic linear algebra computations, e.g. matrix-vector multiplication, using the primitive operations described in this paper.

References

[BK92] A. Bar-Noy and S. Kipnis. Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems. In *Proceedings of the 4th Annual Symposium on Parallel Algorithms and Architectures*, pages 13–22. ACM, 1992.

[BB93] P. Bay and G. Bilardi. An area-universal VLSI circuit. In *Proceedings of the 1993 Symposium on Integrated Systems*, pages 53–67, March 1993.

[BB94] G. Bilardi and P. Bay. An area lower bound for a class of fat-trees. In *Proceedings of the 1994 European Symposium on Algorithms*, pages 413–423, Utrecht, The Netherlands, Springer-Verlag LNCS 855, 1994.

[BB95] P. Bay and G. Bilardi. Deterministic on-line routing on area-universal networks. *Journal of the ACM*, 42(3):614–640, May 1995.

[BCDM94] G. Bilardi, S. Chauduri, D. Dubashi, and K. Mehlhorn. A lower bound for area-universal graphs. *Information Processing Letters*, 51, 101–105, 1994.

[BHPPS96] G. Bilardi, K.T. Herley, A. Pietracaprina, G. Pucci, and P. Spirakis, BSP vs LogP. In *Proceedings of the 8th Annual Symposium on Parallel Algorithms and Architectures*, pages 25–32. ACM, 1996.

[CK+93] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.* ACM, 1993.

[KSSS93] R. Karp, A. Sahay, E. Santos, K. E. Schauser. Optimal Broadcast and Summation in the LogP Model. In *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures*, June 1993.

[GL89] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In S. Micali, editor, *Randomness and Computation*, pages 345–374. JAI Press, Inc., 1989.

[Gre94] R. I. Greenberg. The fat-pyramid and universal parallel computation independent of wire delay. *IEEE Transactions on Computers*, C-43(12):1358–1364, December 1994.

[LAD+92] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak. The network architecture of the Connection Machine CM-5. In *Proceedings of the* 4^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285, July 1992.

[Lei85] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–900, October 1985.

[LM88] C. E. Leiserson and B. Maggs. Communication-efficient parallel algorithms for distributed random-access machines. *Algorithmica*, 3, 53-77, 1988.

[LMR88] T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *Proceedings of the* 29^{th} *Annual Symposium on the Foundations of Computer Science,* White Plains, New York, October 1988.

[LMRR94] T. Leighton, B. Maggs, A. Ranade and S. Rao. Randomized Routing and Sorting on Fixed-Connection Networks. *Journal of Algorithms*, 17(1):157–205, 1994.

[RSS94] A.L. Rosenberg, V. Scarano and R.K. Sitaraman. The Reconfigurable Ring of Processors: Fine-grained Tree-structured Computations. *6th IEEE Symposium on Parallel and Distributed Processing*, 1994.

[Va90] L. G. Valiant. A Bridging Model for Parallel Computation. *Communication of the ACM*, 33:103–111, 1990.

Appendix. Proofs of Theorems 2 and 3

Proof of Theorem 2. For $n \leq N/2$, consider a subtree $\mathcal{T}$ of n leaves and assume that, at a given time t, no node of $\mathcal{T}$ contains any copy of the message to be broadcast.

At time $t + \log n$ no message has yet reached any leaves of $\mathcal{T}$, because any copy must enter the root after time t and it will take $\log n$ steps before it can reach a leaf. We will argue next that at least one subtree of $\mathcal{T}$ of suitable size is still empty.

Let $d = \log w(n) + \log\log(2n/w(n))$ and observe that, if $w(n) \leq n$, then $d \leq \log n$. Consider now the $2^d = w(n)\log(2n/w(n))$ subtrees of $\mathcal{T}$ of $n/2^d$ leaves. The number of messages that have entered any of these subtrees is at most

$$w(n)[\log n - d] = w(n)[\log(n/w(n)) - \log\log(2n/w(n)],$$

since (i) no replication of copies has taken place within $\mathcal{T}$, (ii) at most $w(n)$ copies can enter at each time unit, and (iii) only those copies entered between $t+1$ and $t + \log n - d$ will have reached a subtree of those being considered, as their root is at distance d from the root of $\mathcal{T}$.

A straightforward comparison shows that the number of subtrees considered exceeds the total number of copies they collectively contain at time $t + \log n$, whence at least one of them is empty at that time. Therefore, we have:

$$T(n) \geq \log n + T(n/2^d) = \log n + T\left(\frac{n/w(n)}{\log(2n/w(n))}\right).$$

The statement of the theorem follows by letting $t = 0$ and $n = N/2$. In fact, when $t = 0$, there is only one copy in the entire tree and therefore there is an empty subtree of size $N/2$.

Proof of Theorem 3. Let $\texttt{round}(1), \texttt{round}(2), ..., \texttt{round}(K)$ be the rounds of the algorithm, with $\texttt{round}(t)$ of round height $\texttt{height(t)}$ and duration 2 $\texttt{height(t)}$. Let $g(h)$ denote the number of rounds with height greater than h. We focus on the sequence of heights $1, 2, 4, \ldots, H$ where $H = 2^{\lceil \log\log N - 1 \rceil}$ is the largest power of two smaller than $\log N$. Since there are $(g(2^i) - g(2^{i+1}))$ rounds with $2^i < \texttt{height} \leq 2^{\texttt{i+1}}$ and each of them takes more than $2 \cdot 2^i$ steps, the total number of steps of the algorithm satisfies the relation

$$T > \sum_{i=0}^{\log H} (g(2^i) - g(2^{i+1}))2^{i+1} \geq \sum_{i=0}^{\log H} 2^i g(2^i). \tag{5}$$

Next, we seek a lower bound to $g(h)$. For $h = 0, 1, \ldots, \log N - 1$, let $s_h(t)$ denote the number of subtrees of height h that, upon completion of $\texttt{round}(t)$, contain at least one copy of the message to be broadcast. For convenience, also let $s_h(0) = 1$, for any h. Let $\texttt{round}(t_h^i)$ be the i-th round, in order of execution, among those of height greater than h. We have:

$$s_h(t_h^{i+1}) \leq (w(2^h) + 1)\ s_h(t_h^i). \tag{6}$$

In fact, during $\texttt{round}(t_h^i + 1), \ldots, \texttt{round}(t_h^{i+1} - 1)$, no message enters (from the root) a subtree of height h. During $\texttt{round}(t_h^{i+1})$, each of the $s_h(t_h^i)$ subtrees of height h that already hold one copy of the message can send out to other subtrees at most $w(2^h)$ copies.

Considering that, upon termination of the broadcast algorithm, all the $N/2^h$ subtrees have a copy of the message, we have $s_h(t_h^{g(h)}) \geq N/2^h$. The latter relation, together with 5 and $s_h(0) = 1$, after simple manipulations yields

$$g(h) \geq \frac{\log N - h}{\log(w(2^h) + 1)}\ . \tag{7}$$

By using Relation 7 in Relation 5, we obtain:

$$T > \sum_{i=0}^{\log H} 2^i \frac{\log N - 2^i}{\log(w(2^{2^i}) + 1)} \geq (1/2) \log N \sum_{i=0}^{\log H - 1} \frac{2^i}{\log(w(2^{2^i}) + 1)}, \tag{8}$$

where, in the last step, for $i < \log H$, we have used $2^i < (1/2) \log N$.