On Randomized Broadcasting and Gossiping in Radio Networks

Ding Liu* Manoj Prabhakaran*

{dingliu,mp}@cs.princeton.edu

Abstract

This paper has two parts. In the first part we give an alternative (and much simpler) proof for the best known lower bound of $\Omega(D \log(N/D))$ for randomized broadcasting in radio networks with unknown topology. In the second part we give an $O(N \log^3 N)$-time randomized algorithm for gossiping in such radio networks. This is an improvement over the fastest previously known algorithm that works in time $O(N \log^4 N)$.

1 Introduction

We consider two classical problems of distributing information in communication networks: *broadcasting* and *gossiping*. In broadcasting, the goal is to distribute a message from a distinguished source node to all other nodes in the network. In gossiping each node in the network holds a message, and the goal is to distribute each message to all nodes in the network. In both problems we want to use as less time as possible to finish the task.

We are interested in broadcasting and gossiping in *radio networks*. The radio network is an abstraction of communication networks with minimal assumptions and features and it can model many situations. Communication in radio networks and variants thereof have been widely studied for a long time [6, 16, 1, 4, 19, 18, 8, 9, 10, 12, 11, 17].

A radio network is modeled as a directed graph where nodes represent distributed processors and each edge represents a relation of one node being in the range of another. However the processors have no knowledge of the network topology. A node u can receive a message in a time step if and only if *exactly* one of its neighbors is transmitting in that step. If two or more nodes in the neighborhood of u transmit then a *collision* occurs and none of the messages is received by u. Furthermore u cannot distinguish such collisions from the situation where none of its neighbors is transmitting. Also, a node does not know whether its transmissions were successful or not. The model is described in detail in the next section. See [20, 15, 3, 4, 9] for a discussion on this model and related models.

This model is suited for channels with high noise and unreliable *collision detection*, which is the case in some applications where it is difficult to distinguish a collision from the background noise. Further the topology of the network is considered unknown to

*Department of Computer Science, Princeton University, Princeton, NJ, 08544

the processors, which makes it suitable to model mobile networks or networks with dynamically configured topology or unreliable links. [1] Thus algorithms for radio networks have potentially wide applications. On the other hand one is also interested in lower bounds in this model, because along with efficient algorithms in less stricter models (for instance see [14, 13]) it demonstrates the importance of the various components in the communication model.

In this work we explore the power of randomization in broadcasting and gossiping in radio networks. Chlebus [7] surveys the state of the art in randomized protocols for radio networks.

Previous results. For deterministic broadcasting in unknown radio networks, the best currently known upper bound is due to Chrobak, Gąsieniec and Rytter [10]. Their algorithm runs in time $O(N \log^2 N)$ in a network with N nodes (throughout this paper log refers to logarithm to the base 2 and ln refers to the natural logarithm). Their algorithm is non-constructive in the sense that they show the existence of such algorithms without explicitly constructing one. Recently Indyk [17] gave a constructive solution with similar bounds. The best known lower bound is a simple $\Omega(N \log N)$ bound, due to Bruschi and del Pinto [5], and independently Chlebus *et al* [8]. For randomized broadcasting we often assume that the diameter of the network, D, is also known to the algorithm. Bar-Yehuda, Goldreich and Itai [4] gave a randomized algorithm that achieves broadcast in expected time $O(D \log N + \log^2 N)$. Kushilevitz and Mansour [18] established a lower bound of $\Omega(D \log (N/D))$. This lower bound matches the upper bound for $\epsilon \log N \leq D \leq N^{1-\epsilon}$ for all $\epsilon > 0$. In this work we give an alternative (and simpler) proof for the same lower bound.

For deterministic gossiping in radio networks with unknown topology, Chrobak, Gąsieniec and Rytter [10] presented the first sub-quadratic algorithm whose running time is $O(N^{3/2} \log^2 N)$. Again their algorithm is non-constructive and a constructive solution with similar time bound was recently provided by Indyk [17]. Chrobak, Gąsieniec and Rytter also gave a randomized $O(N \log^4 N)$-time algorithm for gossiping in radio networks with unknown topology [11]. In this work we give a randomized algorithm running in time $O(N \log^3 N)$.

Our results. In section 3 we give an alternative proof for the $\Omega(D \log (N/D))$ lower bound on randomized broadcasting algorithms. Specifically we show that any randomized algorithm that completes radio broadcasting with $\Omega(1)$ probability must have expected running time $\Omega(D \log (N/D))$. Our proof is essentially different from, and much simpler than the previous one [18], which is complicated and involves a reduction from the general case to a special "uniform" case. We hope that our proof will give a fresh view of the lower bounds for the problem, and may help in bridging the gap between the known upper and lower bounds.

In section 4 we give a randomized $O(N \log^3 N)$-time algorithm for gossiping in radio networks with unknown topology. Our basic algorithm is Monte Carlo type and it has the following characteristics: for any $0 < \epsilon < 1$, in time $O(N \log^2 N \log (N/\epsilon))$ it completes gossiping with probability at least $1 - \epsilon$. This easily yields a Las Vegas algorithm with expected running time $O(N \log^3 N)$. Our algorithm follows the one in [11], and the essential difference is that we replace a deterministic procedure (called

[1] For gossiping to be meaningful, we require that the underlying network is strongly connected. For broadcasting to be meaningful, we require that all nodes are reachable from the source. These are the only assumptions on the network topology.

LtdBroadcast) in that algorithm by a new randomized procedure, thereby answering a question posed in [11].

Finally in section 5 we note that a linear time deterministic *gossiping* algorithm exists for symmetric networks, which is asymptotically optimal. Previously it was known that a linear time deterministic *broadcasting* algorithm exists for symmetric networks [8]. But the similar result for gossiping seems to have been unobserved before.

2 Preliminaries

A radio network [4, 9] is modeled as a directed graph $G(V, E)$ where $|V| = N$. Sometimes the diameter of the graph D is also known. [2] The nodes of the graph represent processors in the network and they are assigned different identifiers from the set $\{1, 2, \ldots, N\}$. A directed edge from node u to node v means that u can send messages to v, and we say that v is an *out-neighbor* of u and u is an *in-neighbor* of v. Time is divided into discrete time steps. All nodes have access to a global clock and work synchronously. A node v receives the message from its in-neighbor u in a step if and only if u is the only in-neighbor of v that is transmitting in that time step.

An algorithm is a distributed protocol that for each node v and each time step t, specifies the action (including Transmit or Receive) of v at step t, possibly based on all past messages received by v. The model allows *unlimited computational power*, and all computations are carried out between time steps; but a node can do only one Transmit or Receive operation in a time step. If v transmits at time step t the algorithm also specifies the message (the message is allowed to be arbitrarily long).

A broadcasting algorithm is an algorithm initiated by a distinguished node called the *source* that holds a message. The aim of the algorithm is to distribute this message to each node in the network. The running time of a Monte Carlo broadcasting algorithm is the smallest t such that for any connected network topology,[3] and for any assignment of identifiers to the nodes, the probability of completing the broadcast no later than at step t is $\Omega(1)$. For a Las Vegas algorithm, the running time is defined as the expected time of completion. A gossiping algorithm solves the problem in which each node holds a message and the aim is to deliver all the messages to all the nodes. The running time for a gossiping algorithm is defined similar to that of broadcasting algorithm.

Both of our upper and lower bounds are based on some observations about a special family of networks denoted by $\mathcal{D}_m$ which we define next. Here m could be any positive integer.

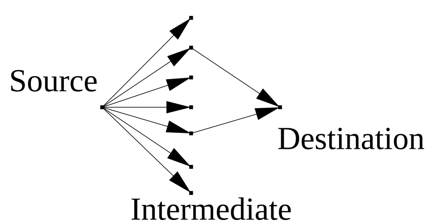

Figure 1: *The graph family $\mathcal{D}_m$.*

[2] This is the case in our lower bound.

[3] By *connected* we mean each node is reachable from the source.

Definition 2.1 *The network family $\mathcal{D}_m$ is the set of all networks of the following type. There are totally $m+2$ nodes: one source, one destination and m intermediate nodes (Fig. 1). There are edges from the source to all intermediate nodes, and there are edges from some intermediate nodes (at least one) to the destination.*

We consider broadcasting on family $\mathcal{D}_m$. Initially only the source holds the message and it transmits first. After that every intermediate node gets the message. Now the goal is to let the destination get the message as early as possible. It is easy to show that any error-free deterministic algorithm for $\mathcal{D}_m$ must use at least m steps in the worst case, using an adversarial argument to construct an instance of $\mathcal{D}_m$ to beat any algorithm running for fewer steps. On the other hand, there exists a randomized broadcasting algorithm that runs in $O(\log m)$ time. We give one below; a similar algorithm appears in [4].

The algorithm is executed in parallel by all the nodes. The procedure names are subscripted by an identifier referring to the node. We use the variable v for this purpose.

Procedure DecayingBroadcast$_v$ (m)
for $i \leftarrow 1$ to $\log m + 1$ **do**
  with probability $1/2^{i-1}$ Transmit

DecayingBroadcast runs in $O(\log m)$ time and achieves a constant success probability.[4] To see this, note that if there are k edges connecting intermediate nodes to the destination, then in the i-th round the success probability is $k/2^{i-1}(1-1/2^{i-1})^{k-1}$. So in round $i = \lceil \log k \rceil + 1$ this probability is lower bounded by a constant, at least 1/8. In the next section we shall see that the running time of this algorithm is asymptotically optimal.

3 The $\Omega(D \log (N/D))$ Lower Bound For Randomized Radio Broadcast: A Simple Proof

In [18] an $\Omega(D \log (N/D))$ lower bound for randomized (Las Vegas type) broadcasting algorithm was established. Here we give an alternative proof of the same result. We think our proof is interesting not only because it is much simpler than the previous one, but also it uses a completely different approach. We hope that our proof will open a new line of attack for proving lower bounds on similar problems.

We state the theorem for directed networks first, but see the remark at the end of this section. Below, the algorithm is allowed to know N and D.

Theorem 3.1 *For any Monte Carlo broadcasting algorithm (i.e., randomized broadcasting algorithm that errs with probability $1-\Omega(1)$) there is a network with N nodes and diameter D, in which the algorithm takes $\Omega(D \log (N/D))$ expected time.*

The proof of Theorem 3.1 is based on the following well-known Yao's minimax principle ([21], Theorem 3).

Lemma 3.2 (Yao's minimax principle [21]) *Let $0 < \lambda < 1/2$. Let $\mathcal{P}$ be a probability distribution over the set of inputs. Let $\mathcal{A}$ denote the set of all deterministic algorithms*

[4] "success" means the destination receives the message.

that err with probability at most 2λ over $\mathcal{P}$. For $A \in \mathcal{A}$ let $C(A, \mathcal{P})$ denote the expected running time of A over $\mathcal{P}$. Let $\mathcal{R}$ be the set of randomized algorithms that err with probability at most λ for any input, and let $E(R, I)$ denote the expected running time of R on input I. Then, for all $\mathcal{P}$ and all $R \in \mathcal{R}$,

$$\min_{A \in \mathcal{A}} C(A, \mathcal{P}) \leq 2 \max_{I} E(R, I)$$

The power of this principle is that it reduces the task of proving randomized lower bound to that of proving deterministic lower bound. In order to prove Theorem 3.1, we pick a probability distribution over a suitable family of networks with N nodes and diameter D, such that any deterministic algorithm that errs with probability $1 - \Omega(1)$ for this distribution has expected running time $\Omega(D \log (N/D))$. This implies the lowerbound for randomized algorithms with error probability $\frac{1}{2} - \Omega(1)$; but since any Monte Carlo algorithm– i.e., a randomized algorithm with error probability $1 - \Omega(1)$, can be repeated a constant number of times to achieve an error probability of $\frac{1}{2} - \Omega(1)$, the lower bound extends to all Monte Carlo algorithms. This is the approach used in our lower bound argument. On the other hand, the proof of [18] is based on direct analysis of randomized algorithms.

We first give two lemmas that together establish a lower bound for deterministic algorithms that finish broadcasting on network family $\mathcal{D}_m$ with $\Omega(1)$ probability.

Lemma 3.3 *There exists a probability distribution $\mathcal{P}$ over $\mathcal{D}_m$ such that the probability for the destination to get the message in any* one *deterministic step is $O(1/\log m)$.*

Proof: We partition $\mathcal{D}_m$ into m subfamilies $\bigcup_{k=1}^{m} \mathcal{D}_m^k$, where $\mathcal{D}_m^k$ is the set of networks with exactly k intermediate nodes connected to the destination. We pick $\mathcal{P}$ in two steps. First for each k ($1 \leq k \leq m$) we assign weight $c/(k \log m)$ to subfamily $\mathcal{D}_m^k$, where c is a normalization factor such that the weights add up to 1. When m is large c is close to $\ln 2$. Then for each $\mathcal{D}_m^k$ we evenly distribute its total weight to all $\binom{m}{k}$ networks belonging to it. In other words, each network with exactly k intermediate nodes connected to the destination is assigned weight $c/(k\binom{m}{k} \log m)$.

Now look at any deterministic step and suppose that j ($1 \leq j \leq m$) intermediate nodes are transmitting in that step. Over the distribution $\mathcal{P}$ we picked, the success probability is easily seen to be

$$c \cdot \sum_{k=1}^{m} \frac{j\binom{m-j}{k-1}}{k\binom{m}{k} \log m} = \frac{c}{\log m} \cdot \frac{j}{m} \cdot \sum_{k=1}^{m-j+1} \frac{\binom{m-j}{k-1}}{\binom{m-1}{k-1}}$$

Note that we have used the fact $k\binom{m}{k} = m\binom{m-1}{k-1}$. In order to show that this probability is $O(1/\log m)$ we only need to show that for *any* j,

$$\sum_{k=1}^{m-j+1} \frac{\binom{m-j}{k-1}}{\binom{m-1}{k-1}} = O(m/j) \tag{1}$$

When $j = 1$ it is obviously true. So we assume that $j > 1$. Let $A_k = \binom{m-j}{k-1}/\binom{m-1}{k-1}$; then for $k \geq 1$, $A_{k+1}/A_k = (m - j - k + 1)/(m - k) < (m - j + 1)/m$. Also $A_1 = 1$, so the left side of equation 1 is bounded by the sum of an infinite geometric series with initial value 1 and decreasing ratio $(m - j + 1)/m$. This series sums to $m/(j - 1) \leq 2m/j$. $\Box$

We define the running time of a deterministic algorithm *on a network* as the time taken till broadcasting succeeds when the algorithm runs on that network. Thus if the algorithm succeeds before it terminates (as the algorithm may not realize that it succeeded), by our definition it will have a shorter running time than the actual running time. In the following lemma it is this definition of running time that is used[5].

Lemma 3.4 *Over the probability distribution $\mathcal{P}$ in Lemma 3.3, the expected running time of any deterministic broadcasting algorithm for $\mathcal{D}_m$ that errs with probability $1 - \Omega(1)$ is $\Omega(\log m)$.*

Proof: Let $p = \Omega(1)$ be the success probability of the algorithm over $\mathcal{P}$. Let t be the smallest number such that the algorithm succeeds for $p/2$ fraction (according to $\mathcal{P}$) of the inputs in at most t steps. Thus for at least $p/2$ fraction, the algorithm runs for at least t steps. So the expected running time is at least $tp/2$. But by Lemma 3.3, to succeed in $p/2$ fraction of $\mathcal{P}$ needs $t = \Omega(\log(m)p/2) = \Omega(\log m)$ steps. $\square$

Remark: By Yao's minimax principle Lemma 3.4 implies a $\Omega(\log m)$ lower bound for Monte Carlo algorithms for $\mathcal{D}_m$, and thus DecayingBroadcast given in section 2 (with an initial transmission by the source) is asymptotically optimal.

To prove the general lower bound of $\Omega(D \log{(N/D)})$ we construct a family $\mathcal{F}_{N,D}$ of networks with $\Theta(N)$ nodes and diameter $\Theta(D)$ and prove the lower bound for this network family. Again we pick a probability distribution $\mathcal{P}^*$ over $\mathcal{F}_{N,D}$ and prove the lower bound on the expected running time (over $\mathcal{P}^*$) of any deterministic algorithm that errs with probability $1 - \Omega(1)$. As discussed earlier, Theorem 3.1 then follows as the result of Yao's minimax principle.

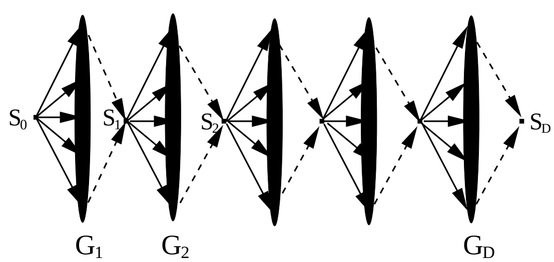

Figure 2: *Structure of the network family $\mathcal{F}_{N,D}$.*

Figure 2 illustrates the structure of $\mathcal{F}_{N,D}$. It consists of D layers G_1 through G_D and each G_i is a network in $\mathcal{D}_m$ where $m = \lfloor N/D \rfloor$. Note that the graph is directed from left to right, and there are no edges within each layer. Every network in $\mathcal{F}_{N,D}$ has $\Theta(N)$ nodes and diameter $\Theta(D)$. The probability distribution $\mathcal{P}^*$ is picked by letting each G_i $(1 \leq i \leq D)$ *independently* comply to the probability distribution $\mathcal{P}$ specified in the proof of Lemma 3.3. That is, $\mathcal{P}^* = \mathcal{P} \otimes \mathcal{P} \otimes \cdots \otimes \mathcal{P}$. The intuition is that on each layer the algorithm is expected to run $\Omega(\log{(N/D)})$ steps and so the total time is $\Omega(D \log{(N/D)})$. Following is the proof of Theorem 3.1.

Proof: It suffices to prove that the expected running time over $\mathcal{P}^*$ of any deterministic algorithm A that errs with probability $1 - \Omega(1)$ is $\Omega(D \log{(N/D)})$.

[5] which makes the lower bound stronger than using the actual running time.

Let t_i $(1 \leq i \leq D)$ be the number of steps A spends on layer G_i (i.e., the duration for which the message does not reach S_i after reaching S_{i-1}). They are random variables. If the message never reaches S_i then t_i is 0. The expected running time of A is $\mathbf{E}(\sum_{i=1}^{D} t_i) = \sum_{i=1}^{D} \mathbf{E}(t_i)$. We would like to use Lemma 3.4 to prove that $\mathbf{E}(t_i) = \Omega(\log(N/D))$. But there are a few issues.

Before the message reaches the layer G_i the nodes could already be running some algorithm, and hence the real running time of the algorithm on that layer is more than t_i. But, note that in our network a node never receives messages from other nodes in its layer or the layers ahead, since there are no such edges. Further we may let the nodes know the entire topology of the previous layers (the arguments below will still hold), and hence the nodes can simulate whatever message they receive before the broadcast message reaches them. Thus it does not help the algorithm do any better, in this distribution of inputs, to start the algorithm before it gets the broadcast message for the first time.

Still we cannot immediately apply Lemma 3.4, because: (i) $\mathbf{E}(t_i)$ is over $\mathcal{P}^*$ but Lemma 3.4 is about $\mathcal{P}$; (ii) we should take into account the possibility that the algorithm can "learn" from *history* and behave differently on networks with identical G_i; in other words, t_i not only depends on the topology of G_i but also those of G_1 through G_{i-1}. Below we take care of both these subtleties.

Fix a layer i and consider the success probability on this layer: this is no less than the overall success probability and hence is $\Omega(1)$. Now we partition the input space of A into finer subspaces such that: (i) within each subspace layers G_1 through G_{i-1} are all fixed; (ii) each subspace has equal weight (for this we may subdivide some instances having the same layers G_1 through G_{i-1} into finer subspaces). Since in $\mathcal{P}^*$, G_i is independent of the earlier layers, for each subspace restricted to G_i, the distribution is $\mathcal{P}$. Within each subspace the algorithm behaves identically on G_i. For each subspace consider the probability of success at the layer G_i, conditional to that subspace (i.e., for fixed G_1 through G_{i-1}). Then, in at least $\Omega(1)$ fraction of the subspaces the algorithm must have $\Omega(1)$ success probability. These subspaces are called *good*. We apply Lemma 3.4 to each *good* subspace to get a $\Omega(\log(N/D))$ lower bound on $\mathbf{E}_{\mathcal{P}}(t_i)$, where the expectation is over the subspace (with distribution $\mathcal{P}$ in G_i). Taking the expectation over distribution $\mathcal{P}^*$ amounts to averaging over all subspaces. Since there are $\Omega(1)$ fraction good subspaces the expectation $\mathbf{E}_{\mathcal{P}^*}(t_i)$ is also $\Omega(\log(N/D))$.

We do this for each i and the lower bound follows by linearity of expectation. $\Box$

Remark: In [18] the lower bound is stated for *symmetric* (or undirected) networks, but with the additional restriction on the algorithm that a node (other than the source) cannot transmit until it receives a message for the first time. Note that if we make the edges in $\mathcal{F}_{N,D}$ undirected, and impose the above restriction on the algorithm, the proof holds again. Then our result is the same as in [18].

4 The $O(N \log^3 N)$ Randomized Algorithm for Gossiping

In this section we give an $O(N \log^3 N)$-time randomized algorithm for gossiping. The algorithm is an improvement over the recent work in [11], which gives an $O(N \log^4 N)$-time randomized algorithm. Our algorithm uses the same outline as that of the algorithm there, but replaces a *limited-broadcast protocol* used there, by a more efficient randomized protocol.

[11] describes their algorithm in terms of *Distributed Coupon Collection.* There a simple randomized and distributed procedure DistCouponColl is described, in which each node is a *bin* and each message a *coupon.* In a time step, each bin can be opened or left closed; if at some time step exactly one bin is opened all the coupons in that bin are *collected.* There may be many copies of a coupon in the network. The aim of DistCouponColl(s) is to collect all coupons (i.e., at least one copy of each message), and for that each node repeats for s times the following: with probability $1/N$ open itself. Lemma 4.1 proved in [11], tells us how large an s we need for a good probability of collecting all coupons.

Lemma 4.1 [11] *If we have N bins and N coupons and each coupon has at least K copies (each copy belonging to a different bin), then for any constant ε, $0 < \varepsilon < 1$, if we run* DistCouponColl*(s) with $s = (4N/K)\ln(N/\varepsilon)$, with probability at least $1-\varepsilon$ all coupons will be collected.*

The overall algorithm in [11] is as follows: there are $\log N$ stages, and in each stage (with high probability) the number of copies of *each* message in the network is doubled; when stage i begins $K = 2^i$ copies of each message should be present in the network. For this, in stage i the DistCouponColl is performed $(4N/K)\ln(N/\varepsilon)$ times so that at the end of the stage each message would have got *collected.* When a node is opened, it does a limited broadcast to double the number of copies of its coupons. Using a deterministic procedure called LtdBroadcast, this takes time $O(K \log^2 N)$. Thus each stage takes $O(N \log^2 N \log(N/\varepsilon))$ time and has an error probability of ε. Since there are $\log N$ stages, with $\varepsilon = \epsilon/\log N$, the overall error probability is bounded by ϵ and the total time is $O(N \log^3 N \log(N/\epsilon))$.

The New Algorithm

Our new algorithm runs in two phases: in the first phase it does $\log N$ round-robins. After this each message has at least $\log N$ copies in the network, which is necessary for the next phase to have a good success probability. The second phase is identical to the old algorithm, except that the LtdBroadcast is replaced by a new procedure RandLtdBroadcast given here. As we shall see, this allows us to save an asymptotic factor of $\log N$ in the running time.

Recall that in section 2 we give a randomized procedure DecayingBroadcast that finishes broadcasting in $\mathcal{D}_m$ with constant probability. Here we use it to send a broadcast message to a new node with constant probability. In other words, if some nodes have not got the message yet, then there exists one such node v such that at least one in-neighbor of v has the message. Regarding v as the *destination* node (Fig. 1), one round of DecayingBroadcast gives a constant probability of making "progress", that is, sending the message to v. In RandLtdBroadcast we use repeated DecayingBroadcast for carrying out limited broadcast. In fact, if we repeat it $O(N)$ times we get a simple $O(N \log N)$ algorithm for broadcast whose error probability can be easily bounded by a constant by using Markov inequality. But in order to use this as a module in our final algorithm we will need tighter Chernoff-type bounds.

RandLtdBroadcast$_v$ is also executed in parallel by all the nodes. Each node has a local Boolean flag *active*$_v$; initially this variable is set to True for some nodes (the "open" bins) and set to False for all other nodes. Note that if during some round of

RandLtdBroadcast$_v$ node v receives a message, $active_v$ becomes True in the next round.

Procedure RandLtdBroadcast$_v$ (N, K)
for $i \leftarrow 1$ to cK **do** { c is an absolute constant to be determined later}
  Round i:
  if $active_v$ **then**
    DecayingBroadcast$_v$ (N)
  else
    Receive for $\log N + 1$ steps
    if received a new message **then**
      $active_v \leftarrow True$

Analysis of RandLtdBroadcast   Time taken is clearly $O(K \log N)$. We define Boolean random variables X_i, for $1 \leq i \leq cK$, as follows. $X_i = 1$ if a new node receives the message or all nodes have already received the message at round i. Otherwise $X_i = 0$. It is clear that if $\sum_{i=1}^{cK} X_i \geq K$ the limited broadcast has succeeded in getting the message to at least K new nodes. But X_i's are *not independent* of each other, and we cannot directly use Chernoff Bounds. But they have the following property: For all 2^{i-1} settings of $(x_1, \ldots, x_{i-1})$, $Pr(X_i = 1 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \geq 1/8$, as guaranteed by DecayingBroadcast. This allows us to use Chernoff-Bound-like argument to bound the error probability as summarized in the following lemma; a proof is provided in the Appendix.

Lemma 4.2 *When the network is initialized with a single node as active, in* $O(K \log N)$ *time,* RandLtdBroadcast*(N, K) succeeds in broadcasting the messages in that node to at least K nodes with probability at least* $1 - \exp(-\alpha K)$*, where α can be made arbitrarily large by choosing a sufficiently large c.*

Now we are ready to analyze our final algorithm, RandGossip, shown below.

Procedure RandGossip$_v$ (N, ϵ)
Phase I:
for $i \leftarrow 1$ to $\log N$ **do**
  for $j \leftarrow 1$ to N **do**
    if $j = v$ **then** Transmit
    else Receive
Phase II:
$\varepsilon \leftarrow \epsilon / \log N$
for $i \leftarrow \log \log N$ to $\log N$ **do**
Stage i:
  $K \leftarrow 2^i$
  $s_i \leftarrow (4N/K) \ln(N/\varepsilon)$
  {now do DistCouponColl(s_i)}
  for s_i times **do**
    with probability $1/N$ **do** $active_v \leftarrow$ True
    otherwise $active_v \leftarrow$ False
    RandLtdBroadcast$_v$ (N, K)

Analysis of RANDGOSSIP Phase I takes time $O(N \log N)$. For phase II, each call to RANDLTDBROADCAST takes $O(K \log N)$, and hence each stage takes $O(N \log N \ln(N/\varepsilon))$ time, and the whole phase thus takes $O(N \log^2 N \log(N/\varepsilon))$ time, which dominates the overall running time.

The first phase is deterministic and there is no probability of error. In the second phase, there are two sources of error: the RANDLTDBROADCAST and the DISTCOUPONCOLL. First we analyze the error probability due to RANDLTDBROADCAST in the i-th stage. Recall that $K \geq \log N$. Error probability for each invocation of RANDLTDBROADCAST(N, K) is $\leq \exp(-\alpha K) \leq 1/N^2$, by choosing a sufficiently large c, and for $s_i = O((N/K) \log(N/\varepsilon))$ invocations it is $O(\log(N/\varepsilon)/N)$. Thus (as long as ε is not exponentially small in N) the error probability due to RANDLTDBROADCAST in phase II is comfortably $o(1)$. In fact, by choosing c sufficiently large this error probability can be driven down to any inverse polynomial in N.

Note that for $K = o(\log N)$, $(N/K) \exp(-\alpha K)$ is not $O(1)$. This is the reason for having a separate first phase, so that before phase II starts, the number of copies per message, K, is large enough.

For DISTCOUPONCOLL in each stage, by Lemma 4.1 we bound the error probability by ε. Since we have $\varepsilon = \epsilon/\log N$, and there are less than $\log N$ stages, the error probability due to DISTCOUPONCOLL is bounded by ϵ. This dominates the error probability of the overall algorithm. Thus we have proved the following result.

Theorem 4.3 *For any given constant ϵ, $0 < \epsilon < 1$,* RANDGOSSIP*(N, ϵ) run on an N node radio network completes gossiping in time $O(N \log^2 N \log(N/\epsilon))$ with probability at least $1 - \epsilon$.*

Finally like in [11], this Monte Carlo algorithm can be converted into a Las Vegas algorithm with expected running time $O(N \log^3 N)$.

5 Gossiping In Symmetric Radio Networks

A radio network is *symmetric* if for any two nodes u and v, whenever there is an edge from u to v, there is also an edge from v to u. If the underlying network is symmetric, broadcasting can be done deterministically in linear time [8], which is asymptotically optimal. Here we show that the same is true for gossiping in symmetric networks.

Theorem 5.1 *There exists a deterministic algorithm that finishes gossiping in $5N - 4$ rounds in any unknown symmetric radio networks with N nodes.*

The complete algorithm is in the Appendix and here is a brief description. It has three phases. The algorithm closely resembles the linear time broadcast algorithm in [8]; in fact phase I and phase III of our algorithm together is essentially similar to the broadcast algorithm there. We use an extra phase II in which a "root" node gathers all the messages in the network.

The first phase is a round-robin. After this each node v knows all its immediate neighbours and stores them in a local list N_v. The second and the third phases are Depth-First-Search (DFS for short) in the network, initiated by a pre-determined arbitrary root node (say node 1). Though the nodes have no knowledge on the topology of the network, they can carry out DFS in a distributed way, in linear time, as in [2]. In the DFS of phase II, while the nodes "return" they send all the messages they collected up to their parent in the search. At the end of phase II the root has all the messages. Another round of DFS is used to distribute this message to all the nodes in the network.

6 Acknowledgements

We would like to thank Andy Yao for introducing us to radio networks. Also we thank Amit Chakrabarti for useful discussions.

References

[1] Alon, N., Bar-Noy, A., Linial, N., Peleg, D. *A lower bound for radio broadcast*, Journal of Computer and System Sciences 43 (1991), 290–298.

[2] Awerbuch, B. *A New Distributed Depth-First-Search Algorithm*, Information Processing Letters, 20, (1985), 147–150.

[3] Bar-Yehuda, R., Goldreich, O., Itai, A. *Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection*, Distributed Computing 5 (1991), 67–71.

[4] Bar-Yehuda, R., Goldreich, O., Itai, A. *On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization*, Journal of Computer and System Sciences 45 (1992), 104–126.

[5] Bruschi, D., Del Pinto, M. *Lower bounds for the broadcast problem in mobile radio networks*, Distributed Computing 10 (1997), 129–135.

[6] Chlamtac, I., Kutten, S. *On broadcasting in radio networks - problem analysis and protocol design*, IEEE Trans. on Communications 33 (1985), 1240–1246.

[7] Chlebus, B.S. Randomized communication in radio networks, A chapter in *Handbook on Randomized Computing*, eds. Pardalos, P.M., Rajasekaran, S., Reif, J., Rolim, J.D.P., Kluwer Academic Publishers, to be published.

[8] Chlebus, B.S., Gąsieniec, L., Gibbons, A.M., Pelc, A., Rytter, W. *Deterministic broadcasting in unknown radio networks*, Proc. ACM-SIAM SODA (2000), 861–870.

[9] Chlebus, B.S., Gąsieniec, L., Östlin, A., Robson, J.M. *Deterministic radio broadcasting*, Proc. ICALP (2000).

[10] Chrobak, M., Gąsieniec, L., Rytter, W. *Fast broadcasting and gossiping in radio networks*, Proc. IEEE FOCS (2000), 575–581.

[11] Chrobak, M., Gąsieniec, L., Rytter, W. *A randomized algorithm for gossiping in radio networks*, Proceedings of 7th COCOON (2001), 483-492.
(url: http://www.csc.liv.ac.uk/~leszek/papers/cocoon01.ps.gz)

[12] Clementi, A.E.F., Monti, A., Silvestri, R. *Selective families, superimposed codes, and broadcasting in unknown radio networks*, Proc. ACM-SIAM SODA (2001), 709–718.

[13] Diks, K., Kranakis, E., Krizanc, D., Pelc, A. *The impact of knowledge on broadcasting time in radio networks*, Proceedings of 7th ESA (1999), 41–52.

[14] Gaber, I., Mansour, Y. *Broadcast in radio networks*, Proc. ACM-SIAM SODA (1995), 577–585.

[15] Gallager, R. *A perspective on multiaccess channels*, IEEE Trans. Inform. Theory, 31 (1985), 124–142.

[16] Hedetniemi, S.M., Hedetniemi, S.T., Liestman, A.L. *A survey of gossiping and broadcasting in communication networks*, Networks 18(1988), 319-359.

[17] Indyk, P. *Explicit constructions of selectors with applications*, Proc. ACM-SIAM SODA (2002), to appear.

[18] Kushilevitz, E., Mansour, Y. *An* $\Omega(D \log (N/D))$ *lower bound for broadcast in radio networks*, SIAM J. Comput. 27 (1998), 702–712.

[19] Kushilevitz, E., Mansour, Y. *Computation in noisy radio networks*, Proc. ACM-SIAM SODA (1998), 236–243.

[20] Tanenbaum, A.S. *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[21] Yao, A. C-C. *Probabilistic computations: Towards a unified measure of complexity*, Proc. IEEE FOCS (1977), 222–227.

Appendix

A A Probability Lemma

Here we prove the claim we made when analyzing RandLtdBroadcast in section 4.

Lemma A.1 *Suppose we have $n = cK$ Boolean random variables X_i, $1 \leq i \leq n$ satisfying the following property: For all 2^{i-1} settings of $(x_1, \ldots, x_{i-1})$, $Pr(X_i = 1 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \geq \delta$, where δ is an absolute constant.*[6] *We also assume that $c > 1/\delta$. Then we have a Chernoff-Bound-like tail estimation for the random variable $X = \sum_{i=1}^{n} X_i$:*

$$\Pr\left(X = \sum_{i=1}^{n} X_i < K\right) \leq e^{-\alpha K}$$

Here α can be made arbitrarily large by setting c big enough.

Proof: The proof is a modification to the standard proof of the Chernoff Bound, using the given assumptions. Let $\lambda > 0$ be a parameter to be specified later; then $\Pr(X < K) \leq \mathbf{E}(e^{-\lambda(X-K)})$. So we only need to bound the latter. In the following, $(x_1, \ldots, x_n) \in \{0,1\}^n$ represents the values of an instance of the random variables $(X_1, \ldots, X_n)$. For brevity, we denote the event $X_i = x_i$ by simply x_i, and write, for instance, $\Pr(x_i | x_1, \ldots, x_{i-1})$ to mean $\Pr(X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1})$.

We have $\mathbf{E}(e^{-\lambda(X-K)}) = e^{\lambda K}\ \mathbf{E}(e^{-\lambda X})$, and

$$\begin{aligned}
\mathbf{E}(e^{-\lambda X}) &= \sum_{(x_1,\ldots,x_n)} \Pr(x_1, \ldots, x_n)\ e^{-\lambda(x_1+\cdots+x_n)} \\
&= \sum_{(x_1,\ldots,x_n)} \Big(\Pr(x_n | x_1, \ldots, x_{n-1}) \cdots \Pr(x_2|x_1)\Pr(x_1)\ e^{-\lambda(x_1+\cdots+x_n)}\Big) \\
&= \sum_{(x_1,\ldots,x_{n-1})} \Big(\Pr(x_{n-1} | x_1, \ldots, x_{n-2}) \cdots \Pr(x_1)\ e^{-\lambda(x_1+\cdots+x_{n-1})}\Big) \\
&\quad \times \sum_{x_n \in \{0,1\}} \Pr(x_n | x_1, \ldots, x_{n-1})\ e^{-\lambda x_n}
\end{aligned}$$

Writing $p = \Pr(x_n = 1 | x_1, \ldots, x_{n-1})\ e^{-\lambda}$, and using $p \geq \delta$,

$$\sum_{x_n \in \{0,1\}} \Pr(x_n | x_1, \ldots, x_{n-1})\ e^{-\lambda x_n} = (1-p)\ e^0 + p\ e^{-\lambda} \leq (1-\delta) + \delta e^{-\lambda}$$

Then,

$$\begin{aligned}
\mathbf{E}(e^{-\lambda(X-K)}) &\leq e^{\lambda K}[(1-\delta) + \delta e^{-\lambda}] \\
&\quad \times \sum_{(x_1,\ldots,x_{n-1})} \Big(\Pr(x_{n-1} | x_1, \ldots, x_{n-2}) \cdots \Pr(x_1) e^{-\lambda(x_1+\cdots+x_{n-1})}\Big)
\end{aligned}$$

[6] For RandLtdBroadcast it is 1/8.

Similarly we can bound the factor involving $x_1, \ldots, x_{n-1}$ above, and continue the simplification until we get

$$\Pr(X < K) \le e^{\lambda K}[(1-\delta) + \delta e^{-\lambda}]^n$$

Now we set $\lambda = \ln \frac{\delta(c-1)}{1-\delta}$. Since $c > 1/\delta$, $\lambda > 0$. Plugging in this value of λ above, we have $\Pr(X < K) \le e^{-\alpha K}$ where,

$$\alpha = c \ln \left(\frac{1 - \frac{1}{c}}{1-\delta} \right) - \ln \left(\frac{\delta(c-1)}{1-\delta} \right)$$

It is easy to see that α can be made arbitrarily large by choosing c big enough. □

B The Linear Time Gossiping Algorithm

The first phase of the algorithm is a simple round-robin, which allows each node to build a list of all its neighbours:

ROUNDROBIN$_v$
for $i \leftarrow 1$ to N **do**
  if $i = v$ **then**
    Transmit (v)
  else
    Receive
    if received (u) **then**
      add u to N_v {N_v is the list of neighbours}

The next two phases are very similar to each other– each phase being essentially a distributed DFS in the network.

DFS$_v$:
$M_v \leftarrow \{m_v\}$
for $2N-2$ time steps **do** {time step is either a Transmit or a Receive }
  Receive {in the first time-step root node assumes it receives the CALL token}
  if received (CALL, t, s) and $t \in N_v$ **then**
    remove t from N_v {this indicates that t has been visited in the DFS}
  if received (TOKEN,$\bar{m}, p, v$) **then**
    if TOKEN = CALL **then**
      PID $\leftarrow p$ {the parent in the DFS tree}
    else {TOKEN = RETURN}
      append $\bar{m}$ to M_v
    if N_v empty **then**
      Transmit (RETURN, M_v, v, PID)
    else
      remove a node u from N_v
      Transmit (CALL, M_v, v, u)

The basic DFS algorithm itself is attributed to Awerbuch [2] in [8]. The algorithm consists of passing a "token" around. The token message is of the form (TOKEN,msg,p,v) whereby the token is passed from node p to node v. There are two kinds of token messages: CALL and RETURN corresponding to the call and return in DFS. The CALL token also serves the purpose of announcing that p has been visited by the DFS. When a neighbour of p sees this message it will remove p from its neighbour-list. This ensures that a node is never passed the CALL token twice. At the first time-step the root node ($v = 1$ say) assumes that it received a CALL token.

We put together these subroutines into our algorithm for gossiping in symmetric networks.

Note that before the DFS can start N_v has to be available at all nodes v. A first phase of RoundRobin ensures this. Phase II is used to gather all the messages in the network, at one node. For this the DFS as shown in the figure is run. Each node returns to its parent a list of messages of all its descendent nodes (the message in the CALL tokens is not necessary). At the end of it, the root node receives all the messages.

In phase III again the DFS is run, but this time it is used to distribute the message list gathered by the root to all the nodes. So when passing the CALL token, the message received from the parent is passed along (and there is no need for the messages with the RETURN tokens.)

The first phase has N time-steps. The second and the third phase each has $2N - 2$ time-steps, because for each edge of the DFS tree there are two token transitions, one for each direction. So gossiping in symmetric networks is finished in $5N - 4$ rounds.