Network-Wide Broadcast

Readings:
Bar-Yehuda et al. papers
Kowalski-Pelc
Kushelevits-Mansour
Livadas-Lynch

Next time:
Point-to-point message routing:
Johnson, Maltz: Dynamic Source Routing (DSR)
Perkins, Royer: Ad hoc on-demend distance-vector routing (AODV)
Hu, Johnson: Caching strategies for on-demand routing protocols
Chen, Murphy: Enabling disconnected transitive communication

1 Introduction

Continuing coverage of the problem of broadcasting a message from a known source i_0 to all nodes in a network.
Network is modelled as a connected undirected graph, in which there is an edge between two nodes (they are neighbors) iff they can communicate directly via single-hop broadcast.

Use a strong collision model, which I called Model A: If two or more neighbors of a node transmit, it sounds like silence.
Also introduced Model B, in which this situation can result either in silence, or one of the messages being correctly delivered.

Last time, we covered only the first Bar-Yehuda paper: BGI-1:
1. Upper bound, randomized:
For any ϵ, a randomized protocol achieving bcast in $O((D + log(n/\epsilon)) \log n)$ slots, with probability at least $1 - \epsilon$, where
D = max distance between source and any other node,
n = total number of nodes in the network.
Works for both Model A and Model B.

2. Lower bound, deterministic:
Any deterministic bcast protocol requires $\Omega(n)$ slots, even if diameter is constant.
But (caution) this holds just for Model B—with the weaker collision assumption.
So, they claim that they have shown an exponential gap between randomized and deterministic protocol, in Model A, but this doesn't show it—that has been shown (much more recently) by Kowalski-Pelc.

Today we will continue with the rest of the material, starting with the other Bar-Yehuda paper: BGI-2:
3. Randomized algorithm that allows emulation of any algorithm that is designed for a single-hop radio network with collision detection, in a multi-hop network with no CD, while keeping the costs reasonably bounded.

4. Application of this to leader election.

Then Kowalski-Pelc, briefly:
5. Upper bound, deterministic:
Having pointed out that BGI-1's lower bound proof is incorrect for the stated model A, they prove that, in fact, in the particular counterexample networks used in BGI-1, there is a sublinear (log n) algorithm, for model A.

6. Upper bound, deterministic:
They generalize the algorithm above to get sublinear time algorithms for all graphs of small max distance D.
They prove this in two stages: first, for $D = 2$, and then for any case where $D = o(loglogn)$.
The second of these is technical enough that we will cover the first only. It seems to contain the key ideas.
Open question: Can we get a sublinear time deterministic algorithm for all graphs with sublinear max-distance?

7. Lower bound, deterministic: (This is very hard.)
They construct a new class of graphs, of max distance 2, in which every deterministic bcasting algorithm requires time $\Omega(n^{1/4})$ on these graphs.
When combined with known upper bounds for randomized algorithms (like that of BGI-1), this yields the desired exponential gap between randomized and deterministic.

Kushelevits, Mansour
They focus on the randomized case.
8. Lower bound, randomized.
Then show an $\Omega(Dlog(n/D))$ lower bound on the expected time to complete bcast, for randomized algorithms.

2 Wrap-up on BGI-1

2.1 Randomized broadcast algorithm

Uses Decay subroutine.

I described the synchronization slightly incorrectly in class:
Each node executes Decay t times, but synchronized with other nodes' executions of Decay.
That is, each node groups its slots into (synchronized) batches of k slots (that's the time for one execution of Decay).
After getting a message, waits for the next beginning of a batch of k.
(Last time I was confused about when they synchronize.)

There is also some confusion in my notes, and in class, about two kinds of "truncation".
Now I use "truncation" to express simply cutting off Decay after k rounds.
I use "stopping" vs. "non-stopping" to indicate the distinction between the real algorithm in which each node runs Decay for t rounds and the version where they all run Decay forever.
In any case, I still can't see how to paste together Lemma 2 and 3 to get Theorem 4. Maybe you can help.

2.2 Linear lower bound, for deterministic case

We basically finished the lower bound too, but without pointing out exactly where the error appeared in the proof, that is, where they needed to use Model B (with weaker collision assumptions) rather than Model A.
Errata for BGI-1: from a web page indicated in KP02, by following a pointer to a useful 5-page writeup.

2.2.1 Overview

Where is the error?
It's buried in the proof of Lemma 7, which shows how to get from the Abstract model to the Hitting Game.
This only appeared in the Appendix, and in too little detail to see the mistake.
The proof in BGI-1 is still (apparently) correct, for Model B.

An afterthought: Actually, they say they like Model B better than Model A anyway, since it assumes less about the network.

Q: Does their randomized algorithm work in Model B? Seems so.

2.2.2 Technical details

The change in assumption about collision behavior causes changes in the three models in the paper: the original, the Restricted, and the Abstract.

Definition 1 (p. 16): Change in Property 3:
Node guaranteed to receive a message if exactly one transmits, as before.
But may also receive if more than one transmit.

Definition 2 (p. 18) (Restricted protocols):
Same change here (but implicitly, since this model is defined only by restricting the activity of the source and sink in the general model.

Definition 4 (p. 19) (Abstract protocols):
Change in Property 2: Again, add the possibility that a receiver might receive a message even though more than one nbr transmitted.

They claim that the constructions that reduce to this point are still OK, with these changes.

On the other hand, Lemma 7 works for the revised models, not the original.
Since the proof in BGI-1 didn't include enough details to tell, they fill in more details in the errata note.

Detailed Proof of Lemma 7 (for weaker collision assumptions):
Describes how to use a (worst-case) t-round (slot) bcast strategy for the abstract model to yield a $2t$-move strategy for the Hitting Game.
Each round in the protocol is used to determine two moves of the game.
Referee answers in the game are then used to determine the outcome of the communication round in the protocol.

The problem is that, the way the referee answers determine these outcomes has to be consistent with what is allowed in the Abstract Model.

So, fix some abstract protocol $\mathcal{A}$.
Then (recall), each node's decision of whether to transmit is a function of:
—Its own id,
—Its S-indicator bit (saying if it's in S or not), and
—The global history so far.
This decision is expressed as a predicate π.

Recall: history is a sequence of P_i, each of which is either the id of a unique node that transmitted alone at slot i or $\perp$.
Each round of the abstract protocol is turned into two moves in the game, which consist of two sets—T_i^1 and T_i^0, which I defined last time.
In that order.
Recall T_i^1 = nodes that would send if they are in S, based on history so far; T_i^0 = nodes that would send if they are *not* in S, based on the history so far.

Now consider the Referee's responses, defined as usual:
Win if $M \cap S$ is a singleton, respond x if $M \cap \bar{S} = \{x\}$, otherwise $\perp$.
If the game wins, we declare that abstract protocol $\mathcal{A}$ is also finished.

Otherwise, we need to specify the behavior at the round in question, in $\mathcal{A}$—specifically, what messages are delivered to whom.
The rule they use is:
–If the second referee answer (to the query for T_i^0) is a singleton $\{x\}$, that is, if $T_i^0 \cap \bar{(}S) = \{x\}$, then deliver just the message from x.
Otherwise deliver nothing.

Now the key point: Is this delivery rule consistent with the requirements of the Abstract model?
The answer is no for Model A, but yes for Model B.

Why yes for Model B?
Case 1: We do deliver a message (from x) in $\mathcal{A}$.
Show we are allowed to do this, according to the rules of the abstract protocol model.
The actual transmitters, in step i of $\mathcal{A}$, are $(T_i^1 \cap S) \cup (T_i^0 \cap \bar{S})$, that is, the nodes who are in S and whose rule says they would transmit if they are in S, plus the nodes that are not in S and whose

rule says they would transmit if they are not in S.
Note that this set does contain x, because the second set in the union does.
However, x need not be the only transmitter.
But Model B allows this delivery anyway.

Case 2: We don't deliver any message in $\mathcal{A}$, and the broadcast is not completed.
Show we are allowed to do this, according to the rules of the abstract protocol model.
In this case, it must be that:
–$T_i^1 \cap S$ is not a singleton (because the move for T_i^1 would then have won the game), and
–$T_i^0 \cap \bar{S}$ is not a singleton (because then the message would have been delivered).
So, the union $(T_i^1 \cap S) \cup (T_i^0 \cap \bar{S})$ is also not a singleton. (Think about it.)
Which means Model B allows us to not deliver a message.

OK, so we can translate the responses back into allowable message deliveries in $\mathcal{A}$.
Now, note that, if a run of $\mathcal{A}$ completes within t rounds, the corresponding sequence of $2t$ Hitting Game moves will win.
Thus, if $\mathcal{A}$ always finishes in t rounds, the corresponding sequence of Hitting Game moves always (that is, for any S) finishes within $2t$ moves.
QED Lemma 7

3 BGI-2

This paper presents an efficient randomized emulation algorithm, to emulate any algorithm for a single-hop network with collision detection on a multi-hop network with no collision detection. Each step of the single-hop network is emulated by $O((D + log(\frac{n}{\epsilon}))log\Delta)$ rounds of the multi-hop network, and succeeds with probability $\geq 1 - \epsilon$.
They also show how to emulate polynomially many steps while maintaining a probability of failure $\leq \epsilon$.

Corollary: By emulating Willard's leader election algorithm, they show an efficient randomized algorithm for leader election in a multi-hop network.

3.1 Introduction

They aim to define different models for wireless networks, and show that, in a sense, they are all equivalent.

The first model presented is the "single-hop" model. This is the model used by a single segment of an Ethernet network. It has the following characteristics:

1. All processors share one channel.
2. Synchronous rounds (slots).
3. In each round:
   - If 1 sends, all receive it.
   - If 0 send, no one receives anything (of course).

- If ¿1 send, all receive special collision notification c (Collision Detection (CD))

Next, they talk about the "multi-hop" model. This is similar to the networks we've been discussing in this class. The essential characteristics are:

1. Arbitrary connected graph, as opposed to complete graph from single-hop.
2. Synchronous rounds.
3. In each round:
   - Each processor can choose to be either a transmitter or receiver.
   - A processor receives only if it's a receiver.
   - If it's a receiver, then:
     - If 1 nbr sends, it receives the msg.
     - If ¿1 nbr sends, it may receive one of the msgs, detect the collision, or receive nothing. Note that this is the same as Model 2 presented in BGI-1 and the errata. Receivers cannot distinguish between the case where nothing was sent, and the case where a collision simply resulted in silence.
4. Don't require uids.

When designing protocols for multihop networks, it can be difficult to overcome the lack of CD and the challenges of unknown topology. They suggest designing algorithms for the Ethernet model, and then emulating them to get protocols for multihop. This may not be perfectly efficient, since the emulation adds some overhead, but it may be simpler than trying to design a distributed algorithm. Depending on the efficiency requirements of your application, this methodology may well be perfectly acceptable.

The randomized emulation presented here runs in $O((D + log\frac{n}{\epsilon})log\Delta)$ slots. Notice that this is the time needed to emulate one round, and is essentially the same as the bound on the broadcast time from BGI-1. The emulation scheme uses the two primitives from BGI-1: *Decay* and *Broadcast*.

Recall that *Decay* allows each processor to receive, with probability ¿ 1/2, a message sent by one of its neighbors, regardless of the number of neighbors that want to send it a message.

Broadcast makes use of *Decay*, and behaves exactly as described in BGI-1. The only difference here is that multiple sources may be transmitting at once. The bound given applies to the probability that everyone receives SOME message.

3.2 The Emulation of a Single Round

The emulation procedure trys to emulate one round of the Ethernet model in multiple rounds of the multi-hop model. The emulation proceeds in three phases:

1. Propagation: All initiators choose random tags and *Broadcast* the message they wish to send. Everyone remembers the first message they receive (initiators always hear their own message first)

2. Detection: The goal of this phase is to implement an abstract CD mechanism. This phase attempts to detect when there is more than 1 initiator. If there is more than one initiator, there must be two different tags in the system, since each message was associated with a randomly chosen tag. Moreover, somewhere, two different tags must appear at neighboring nodes. Basically, the nodes then compare tags with their neighbors. They do this by systematically considering successive bits of the tags. Each check takes $2log\Delta$ rounds. For the i-th bit of the tag, all nodes which have a 1 for that bit become transmitters, using the *Decay* primitive. All other nodes become receivers. If any node actually receives a message (ie, functioned as a receiver when some other node was transmitting), then the two nodes have different tags, and a collision occured. Any node receiving a message sets a flag indicating that a collision was detected.
3. Notification: This phase informs all nodes if a collision occured. Every node that detected a collision in the previous phase sends a *Broadcast* of some standard collision message using failure probability $\frac{\epsilon}{3}$. Any node receiving this message knows a collision occurred. All nodes receive this message with high probability.

3.2.1 Analysis

Now that we've seen the algorithm for emulation, let's try to analyze it and validate BGI's claims.

Lemma 1: Assuming that the propagation phase succeeded (everyone received at least one message):

1. If there was a single initiator, then after the detection phase, everyone has the right message and no conflicts are detected.
2. If there is more than one initiator, then with probability $\geq 1 - \frac{\epsilon}{3}$, at the end of the detection phase, some vertex has discovered the conflict.

Proof: (1) is obvious. If only one person tried to send, everyone will hear it correctly. This is the result from BGI-1.

(2) follows simply. Since the network is connected, we know that there are two adjacent nodes which heard different messages, which means that they have different tag. For every bit i, the two tags differ with probability 1/2. This means that one of the nodes will detect it with probability 1/2 at round i. In other words, *someone* detects the collision at round i with probability 1/4. Thus, the probability that a collision occurred but was not detected over the k rounds is $\leq (1 - \frac{1}{4})^k$. k is defined to be $\lceil 2.5log\frac{\epsilon}{3} \rceil$, which leads to the probability of a collision not being detected being $\leq \frac{\epsilon}{3}$.

Lemma 2: The entire protocol requires $(2 + o(1))B_\epsilon$ rounds to execute, where B_ϵ is defined to be $O((D + log(\frac{n}{\epsilon}))log\Delta)$. This simplifies to simply $O(B_\epsilon)$.

Proof: The Propagation and Notification phases simply consist of running the *Broadcast* primitive, which executes in $O(B_\epsilon)$ time as shown in BGI-1. The Detection phase consists of k iterations of *Decay*, where each iteration takes 2 log Δ rounds to complete. Since $k = \lceil 2.5log\frac{\epsilon}{3} \rceil$, the total time for the algorithm to complete is dominated by $O(B_\epsilon)$, and the Lemma holds.

Now we combine Lemmas 1 and 2 to get **Theorem 1**: If there is a single initiator, then with probability $\geq 1 - \frac{\epsilon}{3}$, everyone receives its message. If, however, there are more initiators, then with probability $\geq 1 - \epsilon$, everyone detects a conflict.

Proof: For any number of initiators, the Propagation phase succeeds in disseminating the message everywhere with probability $\geq 1 - \frac{\epsilon}{3}$, because it uses the *Broadcast* primitive. With more initiators, the Detection phase succeeds with probability $\geq 1 - \frac{\epsilon}{3}$. The last possible point of failure is the *Broadcast* in the Notification phase. Since this failure is bounded by $\frac{\epsilon}{3}$, the theorem holds.

Based on Theorem 1, this algorithm provides successful emulation. In other words, if possible, it delivers the message. Otherwise, it detects a collision. Of course, this is all with some high probability, not guaranteed.

3.3 A side issue: Implementing a CD mechanism in arbitrary multi-hop networks

A side issue: BGI present a method for implementing a CD mechanism for a multi-hop network directly, without turning it into a single-hop network first. The algorithm they use is as follows:

1. Any node wishing to transmit selects a random tag.
2. All transmitters perform *Decay(m,tag)* k times.
3. Any node not transmitting listens, and detects conflict if it hears 2 msgs with different tags.
4. Receivers remember the first message they hear.

This can be shown to guarantee collision detection at the receiver with high probability. (But they don't actually show it...)

This method is for having receivers detect collisions. Most models assume that the *transmitters* are detecting the collisions. Essentially, they do two rounds. The first round uses their method as described above. They then have all receivers who detect collision broadcast a message. BGI claim that in the second round, the original transmitters either receive an explicit collision report, or else detect a collision. This isn't really obvious, since it might be possible not to hear anything. If two neighbors of the original transmitter both send collision reports, these could collide. Remember that the original definition mentions that a collision might not be distinguishable from no transmissions at all. BGI don't address this, and it makes me somewhat nervous, seeing as this was basically the error from the first paper....

3.4 Emulating an Entire Algorithm

All they do here is choose ϵ cleverly. They let ϵ' be the probability of error for a single round. This number is then equal to $\frac{\epsilon}{t}$, where t is a known upper bound on the number of rounds. Thus, for t rounds with probability of error ϵ', the probability that the algorithm succeeds is $(1-\epsilon')^t > 1 - t\epsilon'$.

If they don't know an upper bound, they can adapt by decreasing epsilon sufficiently at each step, so that the total error probability stays bounded. The analysis here is on page 8/9 of the BGI-2 paper, together with Theorem 2 which pulls the whole thing together. Basically, Theorem 2 bounds the time taken to emulate any Ethernet algorithm.

3.5 Applications and Conclusions

As an example, they assert that Willard's Ethernet algorithm for leader election can be run in a multi-hop network. Willard's algorithm uses collision detection, and elects a leader in $O(\log \log n)$ rounds. Thus, BGI's emulation scheme leads to a $O(B_\epsilon \log \text{lon } n)$ algorithm for leader election on multi-hop networks without CD mechanisms.

Limitation of the approach: It requires global synchronization for every round of the emulated algorithm. This is a consequence of the slotted notion of time used the algorithms. This won't be efficient for distributed algorithms with a "local" flavor, (e.g., the topology control algorithm from Li and Halpern). If global synchronization is not practical, this emulation scheme will no longer be a reasonable approach.

4 Kowalski, Pelc

After noting the error in BGI-1, they decided that they in fact like Model A of that paper—collisions result in no info (guaranteed).
Enough to invest a lot of time developing new algorithms and lower bounds for the same model.

Results:
1. They show that the BGI-1 result is in fact false in model A, by giving an algorithm that bcasts in log(n) time on all of the BGI-1 graphs.
2. They generalize this to sublinear time for all graphs of small diameter.
3. They construct a new class of graphs, of diameter 4, such that every deterministic bcasting algorithm requires time $\Omega(n^(1/4))$ on these graphs.
4. Combining this with the randomized algorithm from BGI-1 does yield an exponential gap between deterministic and randomized algorithms for bcast, in model A—they are now considered the first to obtain this.

4.1 Introduction

Same as Model A in BGI-1:
Steps (slots, rounds)
In every step, each node decides to be either a transmitter or a receiver (or inactive).
Node receives a message iff it is a receiver and exactly one of its nbrs transmits.
Deterministic

Nodes have uids, which they assume to be from the set $\{1, ..., r\}$.
The nodes know r.
r is polynomial in the actual number n of nodes.
Nodes know their own ids and the ids of their neighbors.

For lower bounds, assume n itself is known.

Related work: Nice discussion
BGI papers, Kushelevits-Mansour.
KP results also invalidate another lower bound, by Hwang

Other papers that studied the problem in deterministic setting where nodes know their own ids but not the ids of their neighbors; yields linear upper and lower bounds.
Some papers that assume the entire topology is known.

4.2 Logarithmic broadcasting in BGI networks

They discuss the BGI-1 error.
Here, they describe the flaw in a different way from the errata note: They say that the lower bound works for oblivious algorithms, where the nodes can't change their decisions about whether to transmit based on past history.
But they also say that the proof is correct in the other communication model, model B, presumably without the obliviousness restriction.
The errata note explains why the oblivious restriction is OK...

Then they show the result itself is wrong, by exhibiting an $O(logn)$ algorithm (for the specific graphs in $\mathcal{C}_n$.

Main idea of algorithm:
Simulate collision detection for some nodes.
For these graphs, it's enough to simulate collision detection at the source only.
For other networks, they will need to do this for other nodes as well.

Uses subroutine:
$ECHO(i, A)$, where A is a set of nodes at layer L_1:
1. Every node in A transmits its id.
2. Every node in $A \cup \{i\}$ transmits its id.

This leads to 3 possible effects at the source:
Case 1: Message received in step 1 and none in step 2:
Then, because of the very strong collision assumptions, the source knows that A contains exactly one element, and moreover, it knows who it is.

Case 2: Message received in step 2 and none in step 1:
Then the message must come from i. The source knows that $A = \emptyset$.

Case 3: No message received at either round.
Then A has at least 2 nodes.

Thus, $ECHO$ allows the source to determine whether A has 0, 1, or > 1 members.

KP will use $ECHO$ to help select one node in the set S in a BGI network.
Once such a node is selected, it transmits alone, thus completing the bcast.

Assume for convenience that r is a power of 2.

$Binary - Selection - Broadcast$
Step 0: Source transmits the source message and the smallest id i of any of its neighbors in L_1.

Step 1: Node i (alone) transmits the source message and its degree.
If $degree(i) = 2$, the sink receives the message and bcast is completed.
Otherwise, $degree(i) = 1$, which means $i \in \bar{S}$.
Source sets range $R = \{x, ..., y\}$, where $x = 1, y = r/2$.

Step 2: Source sends R.

Steps 3 and 4:
$ECHO(i, R \cap S)$
If case 1 occurs, bcast is done.
If case 2 occurs, source sets $x := y + 1, y := y + (y - x + 1)/2$
If case 3 occurs, source keeps x unchanged, sets $y : (y + x - 1)/2$

Step 5: Repeat from Step 2

Theorem 3.1: Completes bcasting in $O(logn)$.

Proof: Successively halves the range, so it takes $O(log(b))$ to home in on a unique element of S. That's also $O(log(n))$, since b is poly in n.

4.3 Sublinear Broadcasting in networks with $D = 2$

They show how to bcast in networks with $D = 2$, in time $O(n^(2/3)logn)$.
They also handle networks where D is $o(loglogn)$, getting an $o(n)$ bound, but we will cover just the simpler case.

4.3.1 Some results from earlier papers

In these results, $G = (V, E)$, where $V \subseteq \{1, \ldots, r\}$.

Theorem 4.1:
Assume $A \subseteq V$, and all nodes in A have same message m.
Then there is a protocol with time bound $O(min(r, dlog(r/d)))$, after which message m is known to all nodes in $V - A$ having at least 1 and at most d neighbors in A.

Theorem 4.2:
Suppose every node in V has a (possibly different) message.
Then there is a protocol with time bound $O(min(r, d^2log(r)))$, after which every node of degree $\leq d$ has learned the messages of all of its neighbors.

These were proved nonconstructively (recall Komlos-Greenberg for an example of a nonconstructive existence theorem for an algorithm).
Constructive counterparts exist, with only small complexity increases.
This paper assumes the nonconstructive results (with the better bounds), but the constructive ones would yield similar results.

4.3.2 The Algorithm

Nodes are source 0, L_1, L_2.
Assume constants d_1, d_2, to be determined later.
Source maintains set DIS of "discovered" nodes—those that it knows have received its message.

Part 0:
Step 0: Source sends (m, L_1). Sets $DIS := \{0\} \cup L_1$

Step 1: The node u with the smallest label in L_1 sends $(m, N_u \cap L_2)$—the message and the set of its neighbors in L_2.
At this point, the source knows that m has been delivered to $N_u \cap L_2$, so it sets $DIS := DIS \cup (N_u \cap L_2)$.

Step 2: Source sends DIS.

Part 1: (Now it starts getting interesting...)
Using an algorithm like Binary-Selection-Broadcast, the source selects a single node $v \in L_1$ for which the number of undiscovered nbrs in L_2, $N_v \cap L_2$, is maximum, adds all nodes in $N_v \cap L_2$ to DIS, and transmits DIS.

How does this work?
The source first does binary search to figure out the maximum number of neighbors:
Starting from $x = r/2$, bcast DIS and a request to hear from nodes having at least x neighbors in $L_2 - DIS$.
Home in on the maximum x in $logr$ steps, which is $O(logn)$.
Then, after the source knows the exact number x of neighbors it's looking for, it bcasts x and does binary search for a single node v having $|N_v \cap L_2| = x$.
Then the source can add all of $N_v \cap L_2$ to DIS.
Repeat this until no nodes are left in L_1 with $> d_1$ undiscovered neighbors in L_2.
Let $R \subseteq L_1$ be the remaining nodes in L_1; thus, every node in R has $\leq d_1$ neighbors in L_2.

So at this point, DIS includes 0, all of L_1, and "most" of L_2: the only nodes in $L_2 - DIS$ are N_v for some $v \in R$ (and such that $|N_v - DIS| \leq d_1$).
All the nodes in L_2 that are now in DIS become silent for the rest of the protocol.
We must deal with the remaining, undiscovered nodes $U \subseteq L_2$.

At this point, the source and all nodes in L_1 know the set DIS.

Part 2:
Now use the algorithm from Theorem 4.1, with $d = d_2$, where $V = L_1 \cup U$, and where $A = L_1$.
This gets the message to every node in U having $\leq d_2$ neighbors in L_1.
Let $X \subseteq U =$ all the newly-informed nodes in L_2.
Thus, every node in $U - X$ has $> d_2$ neighbors in L_1.

So at this point, the only nodes who haven't received the source message are those in $U - X$.

Q: How big can $U - X$ be?

Claim: $|U - X| \leq nd_1/d_2$:
We have a bipartite graph connecting L_1 and $U - X$. Of course, $|L_1| \leq n$.
Each node in $U - X$ is connected to $> d_2$ of the nodes in L_1; hence, the number of edges is $> |U - X|d_2$. On the other hand, each node in L_1 has $\leq d_1$ neighbors in $U - X$.
(Tricky here—using the fact that the only nodes in L_1 that are actually connected to anything in U are those in R.)
so the number of edges is $\leq d_1 n$. Thus, $|U - X|d_2 \leq d_1 n$, which implies that $|U - X| \leq nd_1/d_2$, as claimed.

Part 3:
Next use the algorithm from Theorem 4.2, with $d = d_1$, where $V = L_1 \cup X$, to let all of $L_1 \cup X$ broadcast their own ids.
This gets all these ids to every node in L_1 that is connected to $\leq d_1$ nodes in X, and thus, to every node in L_1 that is connected to $\leq d_1$ nodes in U (since $X \subseteq U$).
In particular, every node in L_1 that was not chosen in Part 1 has now learned the ids of its neighbors in X—those who got the message in Part 2.
So it can deduce which of its neighbors are in $U - X$, and so still must get the message.

Part 4:
Let $Z \subseteq L_1$ be the nodes in L_1 that weren't chosen in Part 1, and that have some still-uninformed neighbors in L_2.
Nodes in Z know who they are, as above.
Now the source uses the binary-search strategy again to isolate one node $z \in Z$, who then transmits (alone) the source message plus the set of its neighbors in L_2; the source adds these to DIS.
The source repeats this selection of nodes in Z, each time being sure to pick a node that still has uninformed neighbors, until there are no more such nodes in Z.

Key counting argument: How many selections can be performed?
Answer: No more than the number of undiscovered nodes left in $U - X$, since at least one such node gets informed at each selection of a node z.
That's at most $|U - X| \leq nd_1/d_2$, as shown above.

We have to be careful here. The source must be doing the bookkeeping to ensure that each selection in fact ensures that at least one new node in $U - X$ is discovered each time. It has to inform the nodes in L_1 so they can "drop out" of the algorithm if they have no more L_2 nodes left to inform.
The bookkeeping seems similar to before (in Part 1).

4.3.3 The Theorem

Theorem 4.4: This completes broadcasting (that seems clear), and does so within time $O(n^(2/3)logn)$.

Proof:
Part 0 takes 3 steps.

Part 1:
$O(logr)$ for each selected L_1 node.
There are at most n/d_1 selected nodes, since each one must yield at least d_1 new discovered L_2

nodes.
So the total time here is $O((n/d_1 logr))$.

Part 2: By Theorem 4.1, this is $O(d_2 logr)$ (overestimate).
Part 3: By Theorem 4.2, this is $O(d_1^2 logr)$.
Part 4: Each selection takes $O(logr)$, and there are at most nd_1/d_2 selections.
Therefore, the entire phase takes $O(nd_1/d_2 log(r))$.

Now, they fix values of $d_1 = n^(1/3)$ and $d_2 = n^(2/3)$, and all these bounds stay within $O(n^(2/3)logr) = O(n^(2/3)logn)$, as needed.
QED Theorem 4.4

Very cute algorithm and analysis.

4.3.4 Extension to arbitrary networks of radius $o(log(log(n)))$

I think we've already seen the important ideas; we'll skip this one.

Here they use multiple layers $L_1, L_2, \ldots$ instead of just 2.
Algorithm works in phases $1, 2, \ldots$, where each layer L_k informs layer L_{k+1}.

4.4 Lower bound

We will have to skip this. It's quite complex. I'll try to summarize a few key ideas from the proof.
They give a lower bound—not as high as the claimed (but incorrect) lower bound in BGI-1, just $\Omega(n^(1/4))$.
They also use a sequence of classes of 2-layer networks $\mathcal{C}_n$, but now L_2 contains about $n^(1/4)$ nodes, and each node in L_2 is adjacent to only one in L_1.

The specific networks that exhibit the lower bound are constructed by an adversary, based on the behavior of a particular algorithm Π.

Notation:
$V = \{0\} \cup L_1 \cup L_2$
$N_v = neighbors of v$

Idea of the construction:
Step by step, based on steps of the given algorithm Π.
However, the construction doesn't assume that it has the entire execution up to some point in order to construct the next step; rather, it assumes partial information about the execution, in the form of an "abstract history" for each particular node v.
An abstract history for v contains:
A neighborhood of v, and a sequence of messages received by v so far.

However, at a given point in the execution, not all the neighborhoods are yet determined, so there may be several abstract histories to consider.
The next step is generated by an "abstract action function", which says what node v would do for

each possible abstract history it might have seen.
This generates new possible abstract histories for the next step.

At the same time, the construction determines neighborhoods for some nodes in L_2.
Thus, the neighborhoods get specified along the way, which prunes down the set of possible abstract histories.

At the end of the construction, all the neighborhoods have been determined, and the abstract histories pruned down until there is only one possibility for each node at each point—then we have a completely-specified actual history.

The strategy for constructing the abstract histories and neighborhoods is designed to prevent some nodes in L_2 from getting any message for a long time.
In particular, as long as the nbhd of a node of L_2 isn't determined, it hasn't received the source message.

The construction is similar in spirit to the BGI-1 lower bound.
You have already seen a kind of "abstract action function" in that proof, where I pointed out the use of a predicate $\pi(id, S - indicator, history)$ that allowed us to talk about what a node would do in various circumstances.

4.5 Conclusion

Summarize results.

Open problem: Is there a deterministic algorithm running in sublinear time on all networks with sublinear radius?

5 Kushilevitz-Mansour

5.1 Introduction

Continuing work on the study of randomized broadcast protocols.

KM claim that the BGI-1 paper's randomized algorithm has an expected time bound of $O(D \log N + \log^2 N)$. Does this match up with the actual bound in that paper? BGI-1 don't analyze expected time, just high-probability of termination within a certain time. In fact, a nonzero probability subset of their executions don't terminate, so the expected time would in fact be infinite! Anyway, the BGI-1 bound is stated as $O((D + \log(n/\epsilon)) \log n)$, which is only of the form KM describe in case ϵ is regarded as a constant.

Alon et al. worked on proving the BGI-1 algorithm's optimality: they showed that for $D = 3$, there is an $\Omega(\log^2 N)$ lower bound. This work actually gives a lower bound for any broadcast algorithm—not just an expected bound.

The Kushilevitz-Mansour paper deals with non-constant-diameter networks and randomized algorithms, and proves a lower bound on expected time. It shows that for any *randomized* broadcast protocol for radio networks there exists an ordering of the N nodes in which the expected time to broadcast a message is $\Omega(D \log \frac{N}{D})$.

So, if $D \leq N^{1-\epsilon}$ this yields an $\Omega(D \log N)$ lower bound. We need a little calculation to show this: It suffices to show that, for any k_1, exists k_2 such that $k_1 \log(N/D) \geq k_2 \log(N)$.
Or (exponentiating): for any k_1, exists k_2 such that $(N/D)^{k_1} \geq N^{k_2}$.
Since we know $D \leq N^{1-\epsilon}$, it's enough to show that $(N/N^{1-\epsilon})^{k_1} \geq N^{k_2}$; in other words, $N^{\epsilon * k_1} \geq N^{k_2}$. Clearly, taking $k_2 = \epsilon * k_1$ is enough for this.

$\Omega(D \log N)$ gives the first term of the BGI-1 upper bound. Together with Alon et al, this starts to approach a proof of the tightness of the Bar-Yehuda algorithm for all N and $D \leq N^{1-\epsilon}$.

5.2 The Randomized Model

Assume an undirected graph $G = (V, E)$ with a distinguished node 0, the source. It could be directed, and the results won't change. Undirected is the usual case.
n = number of nodes.
D = diameter of the network = maximum distance in the graph (number of hops) between node 0 and any other node.

Many radio networks are mobile, with changing topology. Therefore, we want to assume as little as possible about the layout of the nodes. This paper assumes the nodes know only the size of the network and the diameter. In real networks, the diameter might be changing and nodes might only know some upper bound for D: this makes the lower bound stronger.

Slotted, time synchronized.

This paper assumes that receivers cannot distinguish a collision from "nothing sent", as in Model A of the deterministic algorithms.

Nodes may NOT know anything about the network topology. If they do, the lower bound may be broken (it is still an open question). But the networks constructed in the proof can actually complete broadcast in $O(D + \log^2 n)$ time, if they know the complete network topology ahead of time.

Nodes can be unique or uniform, and may use their unique IDs in an algorithm. The lower bound holds in either case, but the proofs are a bit different. I will cover the uniform transmitter case in detail. The paper covers half of the non-uniform case, which is just a bit more complex.

5.3 Uniform Processors

Π is a broadcast protocol, successful when 1 processor transmits
$t = 2^\ell$ is the number of participants
ℓ is chosen uniformly from the range $1 \leq \ell \leq \log\ n$
$E(T_\ell^\Pi)$ is the expected number of rounds until success in Π

Lemma 1: $E_\ell[E(T_\ell^\Pi)] = \Omega(\log n)$

Lemma 1 is a subproof on the way to $\Omega(D \log \frac{N}{D})$ proof. It says that if there are n processor in a clique (completely connected) and t processors wish to transmit ($2 \leq t \leq n$), the time until exactly ONE (the first one) of the t processors transmits is $E_\ell[E(T_\ell^\Pi)] = \Omega(\log n)$. Because the nodes know nothing about the network topology, t is unknown, so the lower bound is based on n.

Here, the inner expectation is just the ordinary expectation of the random variable T_ℓ, for the fixed ℓ. The outer expectation assumes that ℓ itself is chosen randomly, uniformly, from the range $[1, \log n]$.

We only deal with first success, therefore processors can act at all times as if all previous rounds were unsuccessful. Because unsuccessful rounds mean you hear only silence (Model A), you can decide ahead of time on a transmission schedule. Each of 2^ℓ processors chooses whether to transmit in a round s.

Assume WLOG that n is a power of 2.

Define $p_{s,\ell}$ = probability of failure in rounds $1, ..., s-1$ and success in round s, assuming 2^ℓ participants.

Equation (1): $\sum_{s=1}^{\infty} p_{s,\ell} = 1$ Sure, the sum of the probabilities is 1. That is, no matter what ℓ is, eventually a successful round will happen.

This will be useful later: $p_{s,\ell} \leq Pr(success\ in\ round\ s)$. (Eqn 2) The probability of success in round s is smaller than the probability of that *and* all rounds $1, ..., s-1$ failing.

Now they calculate a lower bound for $E(T_\ell)$ (for specific ℓ): Just write out the sum. Then break the sum up at $s = \lambda \log(n)$ (for a parameter λ not yet specified). For the second term in the sum, then just (under)estimate s by $\lambda \log(n)$. For the first term, they play around a bit. Then stuff cancels easily, yielding simply $\lambda \log n - \alpha(\ell)$, where $\alpha(\ell) = \Sigma_{s=1}^{\lambda \log(n)-1}(\lambda \log(n) - s) * p_{s,\ell}$ That gets us to the bottom of p. 68. (Eqn 3)

Now we want to start solving $E_\ell[E(T_\ell^\Pi)]$ using Eqn 3, but we will get stuck and do a side proof.
$E_\ell[E(T_\ell^\Pi)] = \Sigma_{\ell=1}^{\log(n)} Pr(\ell) * E(T_\ell^\Pi)$
$= \Sigma_{\ell=1}^{\log(n)}(1/\log n) * (\lambda \log n - \alpha(\ell))$
$= \Sigma_{\ell=1}^{\log(n)}(\lambda - (\alpha(\ell)/\log n))$
$= \lambda \log n - (\Sigma_{\ell=1}^{\log(n)} \alpha(\ell)/\log n)$

Now, so far they have just been considering a fixed ℓ. At this point we are stuck and need to switch to taking expectations over all ℓ (see the sum over ℓ above), chosen randomly.

Now we need $\Sigma_{\ell=1}^{\log(n)} \alpha(\ell)$.
$= \Sigma_{\ell=1}^{\log(n)} \Sigma_{s=1}^{\lambda \log(n)-1}(\lambda \log(n) - s) * p_{s,\ell}$
$\leq \Sigma_{s=1}^{\lambda \log(n)-1}(\lambda \log(n) - s) * \Sigma_{\ell=1}^{\log(n)} Pr(success\ in\ round\ s \mid 2^\ell\ participants)$

As a helpful lemma, they prove: Claim 2: For any s, $\Sigma_{\ell=1}^{\log(n)} Pr(success\ in\ rounds \mid 2^\ell\ participants) < 2$. (Eqn 4)

This is saying, whatever strategy Π is used, if the processes make all their choices in advance, then not all numbers of participants can yield good probabilities of success.

Proof of Claim 2: Fix s.
Define $q(s) = Pr(trying\ in\ round\ s) = \Sigma_{history\ h} Pr(h) * Pr(trying\ in\ round\ s\ after\ history\ h)$.
Note that q(s) doesn't depend on ℓ, and is the same for each processor.

So $Pr(success\ at\ round\ s \mid 2^\ell\ participants) = 2^\ell * q(s) * (1 - q(s))^{2^\ell - 1}$.
That is, the number of choices of successful transmitter times probability it transmits times probability the others don't.

Then summing this up for all ℓ, we get the obvious summation. Pull out the term q(s).
Then there's a slightly cryptic inequality—this is based on noticing that the two sums are both sums of powers of p = (1-q(s)), only the first one repeats p^1 once, p^3 twice, p^7 four times, etc., whereas the second one includes all the consecutive powers.
OR! do this step by variable substitution? $j = 2^\ell - 1$.
Then they give the closed for for the geometric progression (powers of p). QED Claim 2

Now we go back to our summation of $\alpha(\ell)$ using Eqn 4:
$\Sigma_{\ell=1}^{\log(n)}\alpha(\ell)$
$\leq \Sigma_{s=1}^{\lambda\log(n)-1}(\lambda\log(n)-s) * \Sigma_{\ell=1}^{\log(n)} Pr(\textit{success in round } s \mid 2^\ell \textit{ participants})$
$\leq 2 * \Sigma_{s=1}^{\lambda\log(n)-1}(\lambda\log(n)-s)$
$\leq \lambda^2\log^2 n$ (Eqn 5)

Now back to the original equation using Eqn 5
$E_\ell[E(T_\ell^\Pi)] = \lambda\log n - (\Sigma_{\ell=1}^{\log(n)}\alpha(\ell)/\log n)$
$= \lambda\log n - (\lambda^2\log^2 n/\log n)$
$= (\lambda-\lambda^2)\log n$

So the result is just $(\lambda-\lambda^2)\log(n)$. Provided we choose lambda to maximize this expression, namely, to be 1/2, we get this expectation to be at least 1/4 log(n). QED Claim 1

5.4 Non-Uniform Processors

Lemma 3: Looks like same statement as Lemma 1, but for the case of non-uniform processes.

Proof of Lemma 3: Suppose we have a collection of non-uniform processes $P_1, ..., P_n$. That means each process has a probability distribution of schedules that it uses to decide when to transmit, as before. But now, each process may have a different distribution.

Based on $P_1, ..., P_n$, we define a new, single program for a uniform set of processes $Q_1, ..., Q_L$: Simply put, each Q process chooses at random a number i in $[1, n]$ and simulates process P_i. Cute. Of course, it is possible that two Q_j's will choose the same i, but we will choose L to be small enough so that this will be very unlikely.

Claim 4: Says that, if the Q_j's do choose different i's, then the probability distribution of the schedules chosen by the Q_j's is the same as that of a RANDOMLY chosen set of processes P_i.

This isn't hard to believe.

Claim 5 is the main claim of this section; it's a bit tricky. It talks about a particular transformation from a non-uniform protocol $P_1, ..., P_n$ to a uniform one. But this isn't exactly the $Q_1, ..., Q_L$ described above. Rather, it's a modification $Q'_1, ...Q'_L$, which acts on odd steps like the Q protocol above, and on the even steps like the BGI-1 randomized (uniform) protocol (repeated use of Decay?). It's OK to mix two protocols—they are both trying to get the source message sent, and can be mixed freely.

The point of mixing in the BGI protocol is to take care of the (unlikely) cases where two Q_j's happen to choose the same P_i to simulate. If the chosen process happened to act deterministically, then the two Qs would always transmit at the same slots. This could be problematic for bounding the running time. So they use the BGI bound for the running time in this case.

Claim 5 says that the expected running time for the Q' protocol, for a fixed number 2^ℓ of participants, is at most: $2\beta_\ell E(T_\ell) + 8(1-\beta_\ell)\log(n)$, where T_ℓ refers to the expectation in the original P protocol, and β_ℓ is the probability that all the Q's choose different i's.

Thus, the first term handles the cases where all the Q's choose differently, whereas the second term comes from the BGI-1 bound. Specifically, they use the fact that the BGI-1 protocol has expected time at most $4\log(n)$ until the first success. ***This seems plausible, but which result in BGI-1 says this? I don't see it.

They also give a lower bound on the probability β_ℓ—we need the lower bound because what we have to worry about bounding is the second term, in which β_ℓ is subtracted.

...calculations LTTR.

5.5 Main Theorem

So, we have some bounds on the expected time to successful transmission within a one-hop clique, $\Omega(\log n)$ for both the uniform and nonuniform case. We now want a result about multi-hop transmission.

Theorem 7: For any (non-uniform) broadcast protocol, there exists a network with N nodes, max distance D, in which the expected time to complete a broadcast is $\Omega(D \log(N/D))$.

They claim this implies a similar lower bound for the case where the worst-case running time is considered and a small probability of error is allowed (which is the way the properties are formulated in BGI-1). I don't quite see this—if we had an algorithm with the worst-case running time bounded, with high probability, then we don't automatically get an algorithm with a bounded expected running time: the cases where the time isn't bounded could take arbitrarily long. Do they have this backwards?

Notation:
N = total nodes in network
D = network diameter (# hops)
n = N/D = max nodes in a "layer"

Proof: Given n, D. Assume for simplicity that n is a power of 2. Network: D+2 layers:
Layer 0: One node 0, the source.
Layer i, $1 \leq i \leq D$: $n_i = 2^(\ell_i)$, where ℓ_i is chosen randomly, uniformly from the range $[1, ..., \log(n/D)]$.
Layer D+1: All the other nodes, so that the total is N.
Each node connected to all those in preceding and succeeding layers.

They discuss the uniform case only, saying that the non-uniform case can be shown with similar techniques to those in section 4.

Main property:
For all i, and all runs, all the processes in L_i have the same view; every message received at one of these processes is received by all others at the same time. This is because they are all connected to the same potential transmitters. Thus, the broadcast progresses layer-by-layer, and the nodes at each layer contend with each other, but not with nodes in other layers.

There are a few points to note about this layered setup, before we continue with the proof of the main theorem.

- These networks are not physically realizable as stated. The paper mentions that layers are fully connected to the next and previous layers, but not within a layer. If connectivity implies proximity, more than 3 nodes per layer could mean these networks make no physical sense. We think you can add the assumption that nodes within a layer form a clique: broadcasts are randomized, and nodes within a layer interfere with each other at the receivers anyway. Other papers such as BGI and Kowalski-Pelc also seem to have physical realizability problems that may not be so easily argued away.

- Another issue is whether layer $i+1$ can always hear a successful (single) broadcasting node from layer i. The paper assumes this away by requiring all future layers to maintain silence while the protocol progresses. This seems to assume some sort of synchronized broadcast phase, not interrupted by any other chatter.
- Once a broadcast has finished in a layer, can past layers perform extra work to help future layers speed up? This is assumed away by saying that layers know all the sizes of previous layers. As long as layer sizes are independently chosen, this is fine.
- The authors add a funny assumption: when a node at L_i gets the first message from a process at layer $i-1$, it also receives all the other messages it will ever get in the future, from layer $i-1$. But while this is only giving extra info (strengthening the lower bound) this is "predicting the future". This is circular: what node u does at the given point can affect the future, and so can affect what the layer i-1 nodes send in the future. It is also unclear to us where this assumption is used.

Define a random variable t_i: the number of rounds from when the nodes in L_i first get the message until L_i succeeds (someone transmits alone). It suffices to show that, for some choice of the layer sizes, the expected value of a t_i is $\Omega(\log(n))$. The expectation is just over the choices within the algorithm. Recall that we choose the layer sizes ℓ_i randomly. It suffices to show that the expected value of the sum, taken over both the choices in the algorithm and the choices of the layer sizes, is $\Omega(D\log(n))$. (Because if the expected value is high over all the choices of layer sizes, there must be a particular set of layer sizes for which it's high.)

Bottom of p. 72: Here, they simply expand the expectation according to the different choices of sizes for layers 1-(i-1). So far, so good. Next equation, at top of p. 73: Just break up the probability of getting a combination of layer sizes as a product, since the choices are independent. And then observe that we are using the uniform distribution for each layer size, so we pull out the log expression. So now we are left with a summation, over all possible sets of layer sizes, of the expected value of t_i given those layer sizes.

So it remains to lower-bound the expected value of t_i given a particular set of sizes for layers 1,...,i-1. But now they claim they can apply Lemma 1—the problem at one layer is like the single-hop broadcast problem discussed in section 3 (that's for the uniform case, but that is what we are doing here). This yields a lower bound on expectation of at least $c\log(n)$ for some constant c. That's for one set of layer sizes—but when we add these up for all the sets of layer sizes and divide by the log expression, we get back to $c\log(n)$ for the whole sum. Summing over the t_i of every layer yields $\Omega(D\log(n)) = \Omega(D\log(N/D))$.

QED Theorem 7

6 Livadas, Lynch

This is a different sort of paper—just a brief note, giving an idea for a possibly-practical protocol to disseminate a message efficiently in a sensor network.
It arose after a lunchtime discussion with Deborah Estrin and David Culler at a DARPA sensor nets meeting, where they said that they didn't have good practical solutions to this problem.

6.1 Introduction

Stationary sensor net, connected, single source, sending a sequence of numbered messages $m_1, m_2, \ldots$.
We want to get the messages everywhere, reliably.
As in the theory papers.

But unlike in the theory papers, we have to cope with unpredictable occasional message loss (no strong assumptions about collisions, delivery success, etc.)
We assume that message loss, while fairly common, is not overwhelming; thus, we suppose that topology control (e.g., power reduction) and collision-avoidance (e.g., backoff) mechanisms have already been applied, to reduce the percentage of message loss to something manageable.

The main idea is to use a combination of two mechanisms, which run at different time scales:

Normal case:
Flood the messages through the network, very fast.
Nodes notice when they first receive a particular message and
immediately retransmit it.
When no losses occur (the "normal case"), the message gets everywhere, very quickly.

Recovery mode:
In the background, at a more leisurely pace, nodes notice when their neighbors are "behind" them in acquiring messages.
They do this by, periodically (according to a local clock), transmitting their "frontier number", which indicates the largest number such that the node has received all packets through that number.
A node that hears that any neighbor has a frontier number less than his own retransmits.

Consider what happens in a typical run, say with 1 message.
It gets to a lot of the nodes very rapidly, but may stop spreading normally because of some losses or other failures.
Then fairly soon, the nodes on the "boundary" of the region that has received the message discover that their neighbors haven't received it, so they retransmit.
The nodes that previously failed to get the message now receive it for the first time.
They can't tell this situation from the one where they received the message immediately.
So they retransmit immediately.
The message then spreads through new regions of the network very fast.

6.2 Formal Model

The next section just formalizes these simple ideas. LTTR.
Describes the nodes as Timed I/O Automata, writes little programs for their behavior.

You might want to look at the program, top of p. 3, to see the program style.
Basically, the node maintains state that includes the current time, a timer saying when it should next broadcast its frontier, the received messages, etc.
The little code fragments describe what happens to the state in response to arrival of messages, and expiration of timers.

(I think the code is slightly buggy:
Classifies updt as an output in the signature but not in the transition defs.
I think the intention is to send out all messages in the bqueue immediately—but this doesn't seem to be enforced by any stopping conditions on time-passage.

Section 2.2 has the same title as section 2.1. Should be Env, not Host.)

6.2.1 Performance Analysis

Basically, if the max-distance is D, and there are at most f packet losses, the time to complete the broadcast is at most $Dd + f(update - period)$, where d is the message delay, and $update - period$ is a larger timeout for retransmission.

7 Kushilevitz-Mansour

7.1 Introduction

Continuing work on the study of randomized broadcast protocols.

KM claim that the BGI-1 paper's randomized algorithm has an expected time bound of $O(D \log N + \log^2 N)$. Does this match up with the actual bound in that paper? BGI-1 don't analyze expected time, just high-probability of termination within a certain time. In fact, a nonzero probability subset of their executions don't terminate, so the expected time would in fact be infinite! Anyway, the BGI-1 bound is stated as $O((D + \log(n/\epsilon)) \log n)$, which is only of the form KM describe in case ϵ is regarded as a constant.

Alon et al. worked on proving the BGI-1 algorithm's optimality: they showed that for $D = 3$, there is an $\Omega(\log^2 N)$ lower bound. This work actually gives a lower bound for any broadcast algorithm—not just an expected bound.

The Kushilevitz-Mansour paper deals with non-constant-diameter networks and randomized algorithms, and proves a lower bound on expected time. It shows that for any *randomized* broadcast protocol for radio networks there exists an ordering of the N nodes in which the expected time to broadcast a message is $\Omega(D \log \frac{N}{D})$.

So, if $D \leq N^{1-\epsilon}$ this yields an $\Omega(D \log N)$ lower bound. We need a little calculation to show this:
It suffices to show that, for any k_1, exists k_2 such that $k_1 \log(N/D) \geq k_2 \log(N)$.
Or (exponentiating): for any k_1, exists k_2 such that $(N/D)^{k_1} \geq N^{k_2}$.
Since we know $D \leq N^{1-\epsilon}$, it's enough to show that $(N/N^{1-\epsilon})^{k_1} \geq N^{k_2}$; in other words, $N^{\epsilon * k_1} \geq N^{k_2}$. Clearly, taking $k_2 = \epsilon * k_1$ is enough for this.

$\Omega(D \log N)$ gives the first term of the BGI-1 upper bound. Together with Alon et al, this starts to approach a proof of the tightness of the Bar-Yehuda algorithm for all N and $D \leq N^{1-\epsilon}$.

7.2 The Randomized Model

Assume an undirected graph $G = (V, E)$ with a distinguished node 0, the source. It could be directed, and the results won't change. Undirected is the usual case.

n = number of nodes.
D = diameter of the network = maximum distance in the graph (number of hops) between node 0 and any other node.

Many radio networks are mobile, with changing topology. Therefore, we want to assume as little as possible about the layout of the nodes. This paper assumes the nodes know only the size of the network and the diameter. In real networks, the diameter might be changing and nodes might only know some upper bound for D: this makes the lower bound stronger.

Slotted, time synchronized.

This paper assumes that receivers cannot distinguish a collision from "nothing sent", as in Model A of the deterministic algorithms.

Nodes may NOT know anything about the network topology. If they do, the lower bound may be broken (it is still an open question). But the networks constructed in the proof can actually complete broadcast in $O(D + \log^2 n)$ time, if they know the complete network topology ahead of time.

Nodes can be unique or uniform, and may use their unique IDs in an algorithm. The lower bound holds in either case, but the proofs are a bit different. I will cover the uniform transmitter case in detail. The paper covers half of the non-uniform case, which is just a bit more complex.

7.3 Uniform Processors

Π is a broadcast protocol, successful when 1 processor transmits
$t = 2^\ell$ is the number of participants
ℓ is chosen uniformly from the range $1 \leq \ell \leq \log\ n$
$E(T_\ell^\Pi)$ is the expected number of rounds until success in Π

Lemma 1: $E_\ell[E(T_\ell^\Pi)] = \Omega(\log n)$

Lemma 1 is a subproof on the way to $\Omega(D \log \frac{N}{D})$ proof. It says that if there are n processor in a clique (completely connected) and t processors wish to transmit ($2 \leq t \leq n$), the time until exactly ONE (the first one) of the t processors transmits is $E_\ell[E(T_\ell^\Pi)] = \Omega(\log n)$. Because the nodes know nothing about the network topology, t is unknown, so the lower bound is based on n.

Here, the inner expectation is just the ordinary expectation of the random variable T_ℓ, for the fixed ℓ. The outer expectation assumes that ℓ itself is chosen randomly, uniformly, from the range $[1, \log n]$.

We only deal with first success, therefore processors can act at all times as if all previous rounds were unsuccessful. Because unsuccessful rounds mean you hear only silence (Model A), you can decide ahead of time on a transmission schedule. Each of 2^ℓ processors chooses whether to transmit in a round s.

Assume WLOG that n is a power of 2.

Define $p_{s,\ell}$ = probability of failure in rounds $1, ..., s-1$ and success in round s, assuming 2^ℓ participants.

Equation (1): $\sum_{s=1}^{\infty} p_{s,\ell} = 1$ Sure, the sum of the probabilities is 1. That is, no matter what ℓ is, eventually a successful round will happen.

This will be useful later: $p_{s,\ell} \leq Pr(\textit{success in round } s)$. (Eqn 2) The probability of success in round s is smaller than the probability of that *and* all rounds $1, ..., s-1$ failing.

Now they calculate a lower bound for $E(T_\ell)$ (for specific ℓ): Just write out the sum. Then break the sum up at $s = \lambda \log(n)$ (for a parameter λ not yet specified). For the second term in the sum, then just (under)estimate s by $\lambda \log(n)$. For the first term, they play around a bit. Then stuff cancels easily, yielding simply $\lambda \log n - \alpha(\ell)$, where $\alpha(\ell) = \Sigma_{s=1}^{\lambda \log(n)-1}(\lambda \log(n) - s) * p_{s,\ell}$ That gets us to the bottom of p. 68. (Eqn 3)

Now we want to start solving $E_\ell[E(T_\ell^\Pi)]$ using Eqn 3, but we will get stuck and do a side proof.
$E_\ell[E(T_\ell^\Pi)] = \Sigma_{\ell=1}^{\log(n)} Pr(\ell) * E(T_\ell^\Pi)$
$= \Sigma_{\ell=1}^{\log(n)}(1/\log n) * (\lambda \log n - \alpha(\ell))$
$= \Sigma_{\ell=1}^{\log(n)}(\lambda - (\alpha(\ell)/\log n))$
$= \lambda \log n - (\Sigma_{\ell=1}^{\log(n)}\alpha(\ell)/\log n)$

Now, so far they have just been considering a fixed ℓ. At this point we are stuck and need to switch to taking expectations over all ℓ (see the sum over ℓ above), chosen randomly.

Now we need $\Sigma_{\ell=1}^{\log(n)}\alpha(\ell)$.
$= \Sigma_{\ell=1}^{\log(n)}\Sigma_{s=1}^{\lambda \log(n)-1}(\lambda \log(n) - s) * p_{s,\ell}$
$\leq \Sigma_{s=1}^{\lambda \log(n)-1}(\lambda \log(n) - s) * \Sigma_{\ell=1}^{\log(n)} Pr(\textit{success in round } s \mid 2^\ell \textit{ participants})$

As a helpful lemma, they prove: Claim 2: For any s, $\Sigma_{\ell=1}^{\log(n)} Pr(\textit{success in rounds} \mid 2^\ell \textit{ participants}) < 2$. (Eqn 4)

This is saying, whatever strategy Π is used, if the processes make all their choices in advance, then not all numbers of participants can yield good probabilities of success.

Proof of Claim 2: Fix s.
Define $q(s) = Pr(\textit{trying in round } s) = \Sigma_{\textit{history } h} Pr(h) * Pr(\textit{trying in round } s \textit{ after history } h)$.
Note that q(s) doesn't depend on ℓ, and is the same for each processor.

So $Pr(\textit{success at round } s \mid 2^\ell \textit{ participants}) = 2^\ell * q(s) * (1 - q(s))^{2^\ell - 1}$.
That is, the number of choices of successful transmitter times probability it transmits times probability the others don't.

Then summing this up for all ℓ, we get the obvious summation. Pull out the term q(s).
Then there's a slightly cryptic inequality—this is based on noticing that the two sums are both sums of powers of p = (1-q(s)), only the first one repeats p^1 once, p^3 twice, p^7 four times, etc., whereas the second one includes all the consecutive powers.
OR! do this step by variable substitution? $j = 2^\ell - 1$.
Then they give the closed for for the geometric progression (powers of p). QED Claim 2

Now we go back to our summation of $\alpha(\ell)$ using Eqn 4:
$\Sigma_{\ell=1}^{\log(n)}\alpha(\ell)$
$\leq \Sigma_{s=1}^{\lambda \log(n)-1}(\lambda \log(n) - s) * \Sigma_{\ell=1}^{\log(n)} Pr(\textit{success in round } s \mid 2^\ell \textit{ participants})$
$\leq 2 * \Sigma_{s=1}^{\lambda \log(n)-1}(\lambda \log(n) - s)$
$\leq \lambda^2 \log^2 n$ (Eqn 5)

Now back to the original equation using Eqn 5
$E_\ell[E(T_\ell^\Pi)] = \lambda \log n - (\Sigma_{\ell=1}^{\log(n)}\alpha(\ell)/\log n)$
$= \lambda \log n - (\lambda^2 \log^2 n/\log n)$
$= (\lambda - \lambda^2)\log n$

So the result is just $(\lambda - \lambda^2)\log(n)$. Provided we choose lambda to maximize this expression, namely, to be 1/2, we get this expectation to be at least 1/4 log(n). QED Claim 1

7.4 Non-Uniform Processors

Lemma 3: Looks like same statement as Lemma 1, but for the case of non-uniform processes.

Proof of Lemma 3: Suppose we have a collection of non-uniform processes $P_1, ..., P_n$. That means each process has a probability distribution of schedules that it uses to decide when to transmit, as before. But now, each process may have a different distribution.

Based on $P_1, ..., P_n$, we define a new, single program for a uniform set of processes $Q_1, ..., Q_L$: Simply put, each Q process chooses at random a number i in $[1, n]$ and simulates process P_i. Cute. Of course, it is possible that two Q_j's will choose the same i, but we will choose L to be small enough so that this will be very unlikely.

Claim 4: Says that, if the Q_j's do choose different i's, then the probability distribution of the schedules chosen by the Q_j's is the same as that of a RANDOMLY chosen set of processes P_i.

This isn't hard to believe.

Claim 5 is the main claim of this section; it's a bit tricky. It talks about a particular transformation from a non-uniform protocol $P_1, ..., P_n$ to a uniform one. But this isn't exactly the $Q_1, ..., Q_L$ described above. Rather, it's a modification $Q'_1, ...Q'_L$, which acts on odd steps like the Q protocol above, and on the even steps like the BGI-1 randomized (uniform) protocol (repeated use of Decay?). It's OK to mix two protocols—they are both trying to get the source message sent, and can be mixed freely.

The point of mixing in the BGI protocol is to take care of the (unlikely) cases where two Q_j's happen to choose the same P_i to simulate. If the chosen process happened to act deterministically, then the two Qs would always transmit at the same slots. This could be problematic for bounding the running time. So they use the BGI bound for the running time in this case.

Claim 5 says that the expected running time for the Q' protocol, for a fixed number 2^ℓ of participants, is at most: $2\beta_\ell E(T_\ell) + 8(1 - \beta_\ell)\log(n)$, where T_ℓ refers to the expectation in the original P protocol, and β_ℓ is the probability that all the Q's choose different i's.

Thus, the first term handles the cases where all the Q's choose differently, whereas the second term comes from the BGI-1 bound. Specifically, they use the fact that the BGI-1 protocol has expected time at most $4\log(n)$ until the first success. ***This seems plausible, but which result in BGI-1 says this? I don't see it.

They also give a lower bound on the probability β_ℓ—we need the lower bound because what we have to worry about bounding is the second term, in which β_ℓ is subtracted.

...calculations LTTR.

7.5 Main Theorem

So, we have some bounds on the expected time to successful transmission within a one-hop clique, $\Omega(\log n)$ for both the uniform and nonuniform case. We now want a result about multi-hop transmission.

Theorem 7: For any (non-uniform) broadcast protocol, there exists a network with N nodes, max distance D, in which the expected time to complete a broadcast is $\Omega(D\log(N/D))$.

They claim this implies a similar lower bound for the case where the worst-case running time is considered and a small probability of error is allowed (which is the way the properties are

formulated in BGI-1). I don't quite see this—if we had an algorithm with the worst-case running time bounded, with high probability, then we don't automatically get an algorithm with a bounded expected running time: the cases where the time isn't bounded could take arbitrarily long. Do they have this backwards?

Notation:
N = total nodes in network
D = network diameter (# hops)
n = N/D = max nodes in a "layer"

Proof: Given n, D. Assume for simplicity that n is a power of 2. Network: D+2 layers:
Layer 0: One node 0, the source.
Layer i, $1 \leq i \leq D$: $n_i = 2^(\ell_i)$, where ℓ_i is chosen randomly, uniformly from the range $[1, ..., \log(n/D)]$.
Layer D+1: All the other nodes, so that the total is N.
Each node connected to all those in preceding and succeeding layers.

They discuss the uniform case only, saying that the non-uniform case can be shown with similar techniques to those in section 4.

Main property:
For all i, and all runs, all the processes in L_i have the same view; every message received at one of these processes is received by all others at the same time. This is because they are all connected to the same potential transmitters. Thus, the broadcast progresses layer-by-layer, and the nodes at each layer contend with each other, but not with nodes in other layers.

There are a few points to note about this layered setup, before we continue with the proof of the main theorem.

- These networks are not physically realizable as stated. The paper mentions that layers are fully connected to the next and previous layers, but not within a layer. If connectivity implies proximity, more than 3 nodes per layer could mean these networks make no physical sense. We think you can add the assumption that nodes within a layer form a clique: broadcasts are randomized, and nodes within a layer interfere with each other at the receivers anyway. Other papers such as BGI and Kowalski-Pelc also seem to have physical realizability problems that may not be so easily argued away.
- Another issue is whether layer $i + 1$ can always hear a successful (single) broadcasting node from layer i. The paper assumes this away by requiring all future layers to maintain silence while the protocol progresses. This seems to assume some sort of synchronized broadcast phase, not interrupted by any other chatter.
- Once a broadcast has finished in a layer, can past layers perform extra work to help future layers speed up? This is assumed away by saying that layers know all the sizes of previous layers. As long as layer sizes are independently chosen, this is fine.
- The authors add a funny assumption: when a node at L_i gets the first message from a process at layer $i-1$, it also receives all the other messages it will ever get in the future, from layer $i-1$. But while this is only giving extra info (strengthening the lower bound) this is "predicting the future". This is circular: what node u does at the given point can affect the future, and so can affect what the layer i-1 nodes send in the future. It is also unclear to us where this assumption is used.

Define a random variable t_i: the number of rounds from when the nodes in L_i first get the message until L_i succeeds (someone transmits alone). It suffices to show that, for some choice of the layer

sizes, the expected value of a t_i is $\Omega(\log(n))$. The expectation is just over the choices within the algorithm. Recall that we choose the layer sizes ℓ_i randomly. It suffices to show that the expected value of the sum, taken over both the choices in the algorithm and the choices of the layer sizes, is $\Omega(D\log(n))$. (Because if the expected value is high over all the choices of layer sizes, there must be a particular set of layer sizes for which it's high.)

Bottom of p. 72: Here, they simply expand the expectation according to the different choices of sizes for layers 1-(i-1). So far, so good. Next equation, at top of p. 73: Just break up the probability of getting a combination of layer sizes as a product, since the choices are independent. And then observe that we are using the uniform distribution for each layer size, so we pull out the log expression. So now we are left with a summation, over all possible sets of layer sizes, of the expected value of t_i given those layer sizes.

So it remains to lower-bound the expected value of t_i given a particular set of sizes for layers 1,...,i-1. But now they claim they can apply Lemma 1—the problem at one layer is like the single-hop broadcast problem discussed in section 3 (that's for the uniform case, but that is what we are doing here). This yields a lower bound on expectation of at least $c\log(n)$ for some constant c. That's for one set of layer sizes—but when we add these up for all the sets of layer sizes and divide by the log expression, we get back to $c\log(n)$ for the whole sum. Summing over the t_i of every layer yields $\Omega(D\log(n)) = \Omega(D\log(N/D))$.

QED Theorem 7