A (Truly) Local Broadcast Layer for Unreliable Radio Networks

Nancy Lynch
MIT CSAIL
Cambridge, MA
lynch@csail.mit.edu

Calvin Newport
Georgetown University
Washington, DC
cnewport@cs.georgetown.edu

ABSTRACT
In this paper, we implement an efficient local broadcast service for the dual graph model, which describes communication in a radio network with both reliable and unreliable links. Our local broadcast service offers probabilistic latency guarantees for: (1) message delivery to all reliable neighbors (i.e., neighbors connected by reliable links), and (2) receiving some message when one or more reliable neighbors are broadcasting. This service significantly simplifies the design and analysis of algorithms for the otherwise challenging dual graph model. To this end, we also note that our solution can be interpreted as an implementation of the abstract MAC layer specification—therefore translating the growing corpus of algorithmic results studied on top of this layer to the dual graph model. At the core of our service is a seed agreement routine which enables nodes in the network to achieve “good enough” coordination to overcome the difficulties of unpredictable link behavior. Because this routine has potential application to other problems in this setting, we capture it with a formal specification—simplifying its reuse in other algorithms. Finally, we note that in a break from much work on distributed radio network algorithms, our problem definitions (including error bounds), implementation, and analysis do not depend on global network parameters such as the network size, a goal which required new analysis techniques. We argue that breaking the dependence of these algorithms on global parameters makes more sense and aligns better with the rise of ubiquitous computing, where devices will be increasingly working locally in an otherwise massive network. Our push for locality, in other words, is a contribution independent of the specific radio network model and problem studied here.

Categories and Subject Descriptors
C.2.1 [Network Architecture and Design]: Wireless Communication; G.2.2 [Discrete Mathematics]: Graph Theory—network problems

Keywords
radio network; local broadcast; abstract MAC layer; dual graph; unreliability

1. INTRODUCTION
In this paper, we implement an efficient local broadcast service in the dual graph radio network model [14, 16, 3, 7, 10, 8], which captures wireless communication over both reliable and unreliable links. In more detail, the dual graph model describes a network with two graphs, one for each type of link. In each round, the network topology used for node communication is a combination of the edges from reliable link graph and some subset of the edges from the unreliable link graph, the latter chosen adversarially. As argued in our earlier studies of this setting, the inclusion of unpredictable links in formal models of radio communication is motivated by the ubiquity of such behavior in real networks (e.g., [23]).

Our local broadcast algorithm yields two types of probabilistic latency guarantees: a fast progress bound, which bounds the time for a node to receive something when one or more of its reliable neighbors (i.e., neighbors connected by reliable links) are transmitting, and a slower but still reasonable acknowledgment bound, which bounds the time for a sender to finish delivering a broadcast message to all of its reliable neighbors. The service we implement operates in an ongoing manner, which makes it suitable for use as an abstraction layer for developing higher-level distributed algorithms for unreliable radio networks. To this end, we note that our algorithm can be interpreted as an implementation of the Abstract MAC Layer specification [13, 15]. It follows that the growing corpus of results designed to run on top of this abstraction (e.g., [9, 20, 6, 12, 11, 5]) can be composed with our implementation, automatically porting these existing solutions to the dual graph model for the first time.

More generally speaking, we note that since 2009 [14] we have waged (algorithmic) battle with the complexities introduced by the presence of unpredictable link behavior in the dual graph model—describing upper and lower bounds for a variety of problems [14, 16, 3, 7, 10, 8]. This paper can be seen as the culmination of this half-decade of effort, in which we integrate what we have learned into a powerful abstraction that renders this realistic but difficult setting tractable to a wider community of algorithm designers.

True Locality. Before proceeding to the details or our results, we must first emphasize an important property of our service: its implementation is “truly local,” by which we mean that its specification, time complexity, and error
bounds are expressed independent of global parameters such as network size. To make this work: we define our correctness and performance properties locally, in terms of individual nodes, rather than globally, in terms of all the nodes; we express our time complexity bounds with respect to local properties, such as bounds on local neighborhood size, not the full network size, $n$; and we capture our error probabilities as generic $\epsilon$ parameters, rather than the common approach of bounding the error in terms of $(1/n^c)$, for some constant $c \geq 1$. Though locality of this type has been well-studied in other network models (e.g., [19, 17]), it is less understood in the wireless setting—a deficiency we believe must be addressed. There are two justifications for this belief.

First, the common practice of seeking “high probability” (i.e., an error probability bound of the form $n^{-c}$) seems unnatural for most deployment scenarios—why should you have to grow your network size to decrease your error probability? Of course, by instead introducing a generic error parameter, $\epsilon$, as we do in this paper, we are not eliminating the possibility of high probability error bounds when useful, as you can simply set $\epsilon = (1/n^c)$ if needed. But we believe that one should try to postpone considering such dependence until it is really necessary. Second, as we see an increasing emphasis on an Internet of Things style ubiquity in the wireless setting [1], global properties such as network size can grow to a massive scale. In studying local algorithms for such scenarios, it is important that we separate time complexity and error guarantees from global parameters and instead reorient them toward the relevant local conditions. This paper provides an example of what is required for a reorientation of this type.

**Results.** Our local broadcast service is parametrized by two time bounds, $t_{ack}$ and $t_{prog}$, and an error bound, $\epsilon$. It assumes a dual graph model with an oblivious link scheduler (i.e., decisions about which unreliable links to include in the topology in each round are made at the beginning of the execution), and a natural geographic constraint that requires all nodes within distance 1 be connected by a reliable edge, and no nodes at distance more than $r \geq 1$ be connected by an unreliable edge. (We typically assume $r$ is constant and therefore omit it in our result summary below. For the sake of completeness, however, we keep $r$ as a parameter in our analysis during all steps leading to these results.) Our service guarantees: (1) for each broadcast of a message $m$ by a node $u$, with probability at least $1 - \epsilon$, every reliable neighbor of $u$ will receive $m$ within $t_{ack}$ rounds; and (2) for a given receiver $v$ and span of $t_{prog}$ rounds, such that at least one reliable neighbor of $v$ is broadcasting throughout the entire span, the probability that $v$ fails to receive at least one message during these rounds is no more than $\epsilon$. We present an algorithm that takes $\epsilon$ as a parameter and implements this service for values of $t_{ack}$ and $t_{prog}$ bounded as follows:

$$t_{prog} = O\left(\log \Delta \log \left(\frac{\log^2 \Delta}{\epsilon}\right)\right)$$

$$t_{ack} = O\left(\Delta \log (\Delta/\epsilon) \log (\log \Delta)\right)$$

where $\Delta$ is an upper bound on node degree. We emphasize that that these results are near optimal, as even in the absence of unreliable links: (1) any progress bound (which reduces to symmetry breaking among an unknown set of nodes) requires logarithmic rounds (e.g., [21]); and (2) any acknowledgement bound requires at least $\Delta$ rounds in the worst case (imagine a receiver $u$ neighboring $\Delta$ broadcasters: $u$ can only receive one message per round, delaying one of these broadcasters by at least $\Delta$ rounds).

**Discussion.** A core difficulty in solving local broadcast in the dual graph model is the presence of the unreliable links, which are included or excluded in the network topology according to an arbitrary link schedule (see the model section below for details). To understand this difficulty, recall that the standard strategy for broadcast in a radio network is to cycle through a fixed schedule of geometrically decreasing broadcast probabilities [2]. The intuition for this fixed schedule approach is that for each receiver, one of these probabilities will be well-suited for the local contention among its nearby broadcasters. In the dual graph model, however, there is no fixed amount of local contention: the link schedule can effectively change this amount at each receiver at each round by changing the edges included in the network topology. It is possible, for example, that the link schedule was constructed with the intent of thwarting this fixed schedule strategy by including many links (i.e., increasing contention) when the schedule selects high probabilities, and excluding many links (i.e., decreasing contention) when the schedule selects low probabilities.

To overcome this difficulty, we use as a starting point the general strategy we originally identified in [10]: permute the broadcast probability schedule after the execution begins (and therefore, after the link schedule has already been generated) to regain independence from the topology. The challenge in permuting a broadcast probability schedule, however, is coordinating the nodes sufficiently that they can apply the same permutation. This creates a chicken and egg problem: to share information among processes requires that we solve broadcast, but we are sharing this information to help solve broadcast. Our solution is to instead solve a form of loose coordination we call seed agreement. This problem assumes each participant generates a seed (for our purposes, this seed will be random bits for use in generating a probability permutation), and then attempts to convince nearby nodes to commit to its seed. A solution to this problem must guarantee that every node commits to some nearby seed (perhaps its own), and, crucially, that there are not too many unique seeds committed in any node’s neighborhood (in this paper, we achieve a bound on the order of $\log (1/\epsilon)$, for error probability $\epsilon$). If nodes subsequently use these seeds to permute their broadcast probability schedules (thereby gaining independence from the link schedule), we are assured that there are not too many different schedules in a given neighborhood. An extra conflict resolution mechanism is then introduced to our broadcast strategy to help resolve contention among these competing schedules.

We note that the seed agreement subroutine provides a general strategy for taming adversarial link scheduling, and is therefore of independent interest. Accordingly, in this
paper we provide a standalone formal specification for the problem that is satisfied by our algorithm. This simplifies the process of subsequently integrating our seed agreement solution into other algorithms.

Finally, we note that to solve these problems in the absence of global parameters such as $n$ requires the introduction of new and non-trivial proof techniques, also of independent interest for the general pursuit of true locality in radio network algorithms. For example, in analyzing our seed agreement subroutine, we could not simply assume that certain key parameters involving local contention hold for the whole network, as this would require a dependence on $n$. We instead established a region of goodness surrounding the target node in our analysis. Using a careful induction on algorithm steps, we showed that although the guaranteed radius of this region must contract as time advances (due to the influence of nodes outside the region for whom we make no assumptions), this contraction is slow enough that our target node safely completes its computation in time. Similarly, in analyzing the local broadcast routines that leverage the seed agreement bits, we had to leverage a new notion of “high probability” that is defined with respect to node degree, not network size (this is the source of the log$(\Delta/\epsilon)$ factors in the $t_{slack}$ bound described above).

**Related Work.** The dual graph model of unreliable radio communication was introduced by Clementi et al. [4], and subsequently given the name “dual graph” by Kuhn et al. [14]. It has since been well-studied [14, 16, 3, 7, 10, 22].

2. THE DUAL GRAPH MODEL

We use a radio network model based on dual graphs, which describes randomized algorithms executing in a synchronous multihop radio network with both reliable and unreliable links. The model describes the network topology with a dual graph $(G, G')$, where $G = (V, E)$, $G' = (V', E')$, and $E \subseteq E'$, where $E$ describes reliable links and $E' \setminus E$ describes unreliable links. We use $n$ to denote $|V|$, the number of vertices in the graphs. For $u \in V$, we write $N_G(u)$ ($N_G'(u)$) to denote $u$’s immediate neighbors in $G$ ($G'$), not including $u$ itself. We assume two degree bounds: $\Delta$, an upper bound on $|N_G(u)|$, $|N_G'(u)|$, and $\Delta'$, an upper bound on $|N_G'(u) \setminus \{u\}|$, defined over every $u$.

An embedding of a (finite) set $V$ of graph vertices in the Euclidean plane is simply a mapping $emb : V \to \mathbb{R}^2$; this provides a pair of $(x, y)$ coordinates for each vertex $V$. If $emb$ is an embedding of the vertices $V$ of a dual graph $(G, G')$ and $r$ is a real number, $r \geq 1$, then we say that $(G, G')$ is $r$-geographic with respect to $emb$ provided that, for every $u, v \in V, u \neq v$, the following conditions hold (where $d$ represents Euclidean distance):

1. If $d(emb(u), emb(v))) \leq 1$ then $\{u, v\} \in E$.
2. If $d(emb(u), emb(v))) > r$, then $\{u, v\} \notin E'$.

In other words, nearby vertices must be neighbors in $G$, and distant vertices cannot even be neighbors in $G'$, but vertices in the grey zone represented by the intermediate distances in $(1, r)$ might or might not be neighbors in $G$ or $G'$. We say that $(G, G')$ is $r$-geographic provided that there exists an embedding $emb$ of the vertex set $V$ such that $(G, G')$ is $r$-geographic with respect to $emb$. We sometimes also say that $(G, G')$ is geographic provided that there exists a real $r \geq 1$ such that $(G, G')$ is $r$-geographic.

The dual graphs we consider in this paper are assumed $r$-geographic, for some particular $r$, which we fix for the rest of the paper. Moving forward, fix an space $I$. An algorithm is an injective mapping $proc(i)$ from $I$ to some set of processes, which are some type of probabilistic timed automata that model wireless devices. Thus, $proc(i)$ denotes the process with id $i$. We assume that each process knows (e.g., has in a special component of its initial state) its own id, and also knows the quantities $\Delta$, and $\Delta'$. Notice, we do not assume processes know $n$ (as is typical in such models) as we seek problem definitions and solutions that operate independently of the network size. A process assignment for a local graph $(G, G')$ and id space $I$ is an injective mapping $id(i)$ from $V$ to $I$, that assigns a different id to each graph vertex. The two mappings, $proc$ and $id$, together serve to assign a distinct process to each graph vertex. That is, $proc(id(u))$ is the process assigned to graph vertex $u$. To simplify terminology, we often write node $u$ to indicate $proc(id(u))$ or
process $i$ to indicate $\text{proc}(i)$. We assume that processes do not know the $\text{id}(i)$ mapping in advance.

An execution of an algorithm in a given dual graph network topology $(G, G')$ proceeds in synchronous rounds $1, 2, \ldots$. In each round $t$, each node decides whether to transmit a message or receive, based on its process definition; this might involve a random choice. The communication topology in round $t$ consists of the edges in $E$ plus an arbitrary subset of the edges in $E' \setminus E$. This subset, which can change from round to round, is determined by an adversary that we call a link scheduler (see below). Once the topology is fixed for a given round, we use the following standard collision rules to determine communication: node $u$ receives a message $m$ from node $v$ in round $t$, if and only if: (1) $u$ is receiving; (2) $v$ is transmitting $m$; and (3) $v$ is the only node transmitting among the neighbors of $u$ in the communication topology chosen by the link scheduler for round $t$. If node $u$ does not receive a message, then we assume that it receives a special “null” indicator $1$: that is, we assume no collision detection.

We now formalize the notion of a link scheduler: the entity responsible for resolving the non-determinism concerning which edges from $E' \setminus E$ are added to the topology in each round. Formally, we define a link scheduler to be a sequence $G = G_1, G_2, G_3, \ldots$, where each $G_i$ (also denoted $G[i]$) is the graph used for the communication topology in round $t$. We assume each $G_i$ is allowable given our above model definition. We assume the link scheduler for a given execution is specified at the beginning of the execution. Notice, this definition implies oblivious behavior concerning the network dynamism, as all decisions on the topology are made at the beginning of an execution.

The other relevant source of non-determinism in our model is the environment which we use to provide inputs and receive outputs as required by a specific problem (when relevant). For example, in solving local broadcast, an environment provides the messages to broadcast, whereas for a problem like consensus, it provides the initial values. The details of what defines a well-formed environment is specified on a problem-by-problem basis. As with the scheduler, when analyzing an execution we first fix the environment for the execution. Though it is possible to conceive of an environment as a probabilistic entity, for the sake of simplicity, the environments we consider in this paper are all deterministic (i.e., once you fix an environment for an execution, all non-determinism regarding inputs is resolved). To formally model the interaction with an environment, we break down the synchronous steps each process takes within a round as follows: first all processes receive inputs (if any) from the environment, next all processes that decide to transmit do so, they then all receive, and finally, they generate outputs (if any) which are processed by the environment to end the round.

We call the combination of a dual graph, process assignment, link scheduler, and environment a configuration. Notice, a configuration resolves all relevant model and problem nondeterminism. It follows that a configuration combined with a probabilistic algorithm defines a distribution over possible executions (which we sometimes call the execution tree). When we say an algorithm satisfies a certain property with a given probability, we mean that for all allowable configurations, in the execution tree that results from combining the algorithm with the configuration, the property is satisfied with that probability.

3. **SEED AGREEMENT**

The seed agreement problem provides a loose form of coordination: each participating node $u$ generates a seed $s$ from some known seed domain $S$, and then eventually commits to a seed generated by a node in its neighborhood (perhaps its own). The safety goal is to bound the number of unique seeds committed in any given neighborhood by a sufficiently small factor $\delta$, while the liveness goal is to do so in a minimum of rounds. In this section, we provide a dual graph algorithm that yields a bound $\delta$ that is roughly $O(\log(\frac{1}{\epsilon}))$, and that operates within time that is polynomial in $\log(\Delta)$ and $\log(\frac{1}{\epsilon})$, with (provided) error probability $\epsilon$.

In Section 4, we use seed agreement as a crucial subroutine in our local broadcast service implementation. It is potentially useful, however, to any number of problems in the dual graph model (and elsewhere), so we take our time here to first provide a careful formal specification, which we then satisfy with a new dual graph algorithm. The analysis of our algorithm was rendered particularly tricky by our goal of avoiding dependence on global parameters such as $n$, and provides some of the main technical contributions of this paper. Due the long length of this analysis, we defer the details to the full version of this paper [18]. The strategy deployed in this section is to bound the rate at which a region of “goodness” (i.e., sufficiently bounded contention) surrounding our target node contracts as the node races toward termination.

3.1 **The Seed Agreement Problem**

Fix a finite seed domain $S$. We specify the problem as $\text{Seed}(\delta, \epsilon)$, where $\delta$ is a positive integer representing the seed partition bound, and $\epsilon$ is a small nonnegative real representing an error probability. This specification describes correctness for a system based on some arbitrary system configuration, running according to our execution definition. The specification has no inputs. Its outputs are of the form $\text{decide}(j, s)_u$, where $j \in I$, $s \in S$, and $u \in V$. This represents a decision by the node at graph vertex $i$ to commit to the seed $s$ proposed by the node with $id j$ (in the following, we call $j$ the owner of seed $s$). We begin with two basic non-probabilistic conditions on the outputs; these must hold in every execution:

1. **Well-formedness:** In every execution, for each vertex $u$, exactly one $\text{decide}(*, *)_u$ occurs.

2. **Consistency:** In every execution, for each pair of vertices $u_1, u_2$, if $\text{decide}(j, s_1)_{u_1}$ and $\text{decide}(j, s_2)_{u_2}$ both occur, then $s_1 = s_2$.

That is, if outputs contain the same owners then they also contain the same seeds. The two remaining conditions are probabilistic. To talk about probabilities of events, we must first specify the probability distribution. As noted in Section 2, the combination of the system configuration fixed above and a given seed agreement algorithm defines a distribution on executions. We state our remaining properties in terms of this distribution. In more detail, we start by considering an agreement property. Let $B_{u, \delta}$ be the event
3. Agreement: For each vertex \( u \), \( Pr(\text{Byz}(u, \delta)) \geq 1 - \epsilon \).

Note that we state Condition 3 for each vertex \( u \) separately, rather than in terms of all vertices, as in [10]. This change is needed for expressing costs in terms of local parameters.

We now express independence of the choices of seed values corresponding to different owners. An owner mapping \( M_o \) is a mapping from \( V \) to \( I \), that is, an assignment of an (owner) \( i \) to each vertex. A seed mapping \( M_s \) is a mapping from \( V \) to \( S \), that is, an assignment of a seed to each vertex. We say that a seed mapping \( M_s \) is consistent with an owner mapping \( M_o \) provided that, if two vertices have the same owner, then also have the same seed. That is, if \( M_s(u) = M_s(v) \) then \( M_s(u) = M_s(v) \). Let \( \text{Own}_{M_o} \) be the event in the probabilistic execution that \( M_o \) is the owner mapping.

Condition 4 says that the probability of each consistent seed mapping is just what it would be if the seed mapping were determined in the following way: All processes first choose a seed from \( S \), uniformly at random. Then after every process chooses a seed owner, it also adopts the associated seed value.

3.2 A Seed Agreement Algorithm

We now describe our seed agreement algorithm, \( \text{SeedAlg} \), which takes as its sole parameter, an error bound, \( \epsilon \). We will show, in Theorem 3.1, that this algorithm implements \( \text{Seed}(\delta, \epsilon) \) for values of \( \delta \) and \( \epsilon \) that depend on \( \epsilon_1 \). Its main strategy is to hold aggressive local leader elections that yield bounded safety violations (i.e., multiple nearby leaders). In the following description, we assume for simplicity that \( \Delta \) is a power of 2. We also use a “sufficiently large” constant \( \epsilon_4 \) for the phase length.

Algorithm \( \text{SeedAlg}(\epsilon_1) \), for process \( i \) at graph vertex \( u \), where \( 0 < \epsilon_1 \leq \frac{1}{2} \):

The algorithm uses \( \log \Delta \) phases, each consisting of \( \epsilon_4 \log^2 \left( \frac{1}{\epsilon_1} \right) \) rounds.

Process \( i \) maintains a local variable containing its “initial seed” in \( S \), which it chooses uniformly at random from the seed domain \( S \). It also keeps track of its status in \{“active”, “leader”, “inactive”\}, and the current phase number and round number.

Now we describe process \( i \)’s behavior in any particular phase \( h \in \{1, \ldots, \log \Delta \} \). If \( \text{status} = \text{active} \) at the beginning of phase \( h \), then process \( i \) becomes a leader; i.e., sets \( \text{status} = \text{leader} \), with probability \( 2^{-\log \Delta - h+1} \). Thus, it uses probabilities: \( \frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2}, \frac{1}{2} \), as it progresses through the phases.

If process \( i \) becomes a leader at the start of phase \( h \), it immediately outputs \( \text{decide}(i, s) \), where \( s \) is its initial seed. Then, during the remaining rounds of the phase, process \( i \) broadcasts \( (i, s) \) with probability \( \frac{1}{\log (\frac{1}{\epsilon_1})} \) in each round. At the end of the phase, it becomes inactive.

If process \( i \) is active but does not become a leader at the start of phase \( h \), then it just listens for the entire phase. If it receives a message containing a pair \( (j, s) \), then it immediately outputs \( \text{decide}(j, s) \) and becomes inactive. If it receives no messages during this phase, then it remains active.

If process \( i \) completes all phases and is still active, then it outputs \( \text{decide}(i, s) \), where \( s \) is its initial seed.

3.3 Correctness of \( \text{SeedAlg} \)

The analysis of \( \text{SeedAlg} \) contained in the full version [18] culminates with the main theorem below. For the following, recall that \( r \) is the value used in defining the \( r \)-geographic property assumed of our dual graph (Section 2), and \( \Delta \) is the maximum node degree in \( G \).

**Theorem 3.1.** \( \text{SeedAlg}(\epsilon_1) \) satisfies the \( \text{Seed}(\delta, \epsilon) \) specification, where:

- \( \delta = O(r^2 \log (\frac{1}{\epsilon_1})) \), and
- \( \epsilon = O(r^4 \log^4(\Delta)(\epsilon_1)^{2^2}) \), for a constant \( c \), \( 0 < c < 1 \).

The algorithm takes \( O((\log \Delta) \log^2 (\frac{1}{\epsilon_1})) \) rounds.

4. LOCAL BROADCAST

We now define our local broadcast service, then describe and analyze an efficient solution which uses the \( \text{SeedAlg} \) algorithm from Section 3 as a key subroutine.

4.1 The Local Broadcast Problem

The local broadcast problem described here requires nodes to implement an ongoing probabilistic local communication service with timing and reliability guarantees. In more detail, we call the problem \( \text{LB}(t_{\text{ack}}, t_{\text{prog}}, \epsilon) \), where \( t_{\text{ack}} \geq t_{\text{prog}} \geq 1 \) are integer round bounds, and \( \epsilon \) is a small real representing an upper bound on the error probability. To define the problem, we must first fix the underlying dual graph network in which it is being solved: \( (G = (V, E), G' = (V', E')) \).

We then define a set \( M_u \) of possible messages for each \( u \in V \). For simplicity, we assume these sets are pairwise disjoint. Let \( M = \bigcup_u M_u \) be the set of all possible messages. Every node \( u \in V \) has a \( \text{bcast}(m)_u \) input and \( \text{ack}(m)_u \) output, for each \( m \in M_u \). Node \( u \) also has a \( \text{recv}(m)_u \) output for each \( m' \in M \).

We now restrict the behavior of the environments we consider for this problem. In more detail, we assume that (1) the environment generates each input at a given node \( u \) at most once per execution (i.e., each message it passes a node to broadcast is unique), and (2) if the environment generates a \( \text{bcast}(m)_u \) input at \( u \) at some round \( r \), it must then wait until \( u \) subsequently generates a \( \text{ack}(m)_u \) output (if ever) before it can generate another \( \text{bcast} \) input at \( u \). To simplify analysis, we restrict our attention to deterministic environments. Therefore, we can assume the environment
is modeled as synchronous deterministic automaton that receives the nodes’ \textit{ack} outputs as input, and generates their \textit{bcast} inputs as its output.

In every execution, the following must always hold: deterministic and probabilistic constraints on the nodes’ out-

tively broadcasting during this round. The problem places

in a given round if there is some message that 
next acknowledged a subsequent \textit{ack}(m) output. Similarly, we say \textit{u} is active in a given round if there is some message that \textit{u} is active-

casting during this round. The problem places deterministic and probabilistic constraints on the nodes’ out-

utput behavior. We begin with the deterministic constraints.

In every execution, the following must always hold:

1. \textit{Timely Acknowledgement}. If node \textit{u} receives a \textit{bcast}(m) input in round \textit{r}, it will generate a single corresponding \textit{ack}(m) output in the interval \textit{r} to \textit{r} + \textit{t}_{\text{ack}}. These are the only \textit{ack} outputs that \textit{u} generates.

2. \textit{Validity}. If node \textit{u} performs a \textit{recv}(m) output in some round \textit{r}, then there exists some \textit{v} \in \mathcal{N}_G(\textit{u}) such that \textit{v} is actively broadcasting \textit{m} in round \textit{r}.

Recall that if we fix some configuration and an algorithm, the combination yields a well-defined probability distribution (equiv., execution tree) over executions of the algorithm in this configuration. To aid our probabilistic constraint definitions, we first introduce some notation for discussing an execution tree determined by a configuration. When considering any execution from such a tree, we can partition time into phases of length \textit{t}_{\text{prog}} starting in the first round. We number these 1, 2, .... We use the terminology phase \textit{i} prefix to describe a finite execution prefix that includes every round up to the beginning of phase \textit{i} (i.e., it does not include the first round of phase \textit{i} but does include the last round—if any—before phase \textit{i} begins). For a given execution tree, phase \textit{i} prefix \textit{a} in this tree, and node \textit{u}, let \textit{A}^\textit{i}\textit{u} describe the set of \textit{t}_{\text{prog}}-round extensions of \textit{a} in which there is a \textit{G}-neighbor of \textit{u} that is active throughout every round of phase \textit{i}, and let \textit{B}^\textit{i}\textit{u} describe the set of \textit{t}_{\text{prog}}-round extensions where \textit{u} receives at least one message \textit{m} \in \mathcal{M}_G from a node \textit{v} in a round \textit{r} such that \textit{v} is actively broadcasting \textit{m} in \textit{r}.

We now define two probabilistic constraints that must hold for every configuration:

1. \textit{Reliability}. For every configuration, node \textit{u}, and \textit{r}-round execution prefix such that \textit{u} receives a \textit{bcast}(m) input at the beginning of round \textit{r}: the probability that every \textit{v} \in \mathcal{N}_G(\textit{u}) generates a \textit{recv}(m) output before \textit{u}’s corresponding \textit{ack}(m) output, is at least \(1 - \epsilon\).

(Notice: this property leverages the timely acknowledgment property which tells us that in every extension of this prefix, \textit{u} generates an \textit{ack}(m) output within \textit{t}_{\text{ack}} rounds.)

2. \textit{Progress}. For every configuration, node \textit{u}, phase \textit{i}, and phase \textit{i} prefix \textit{a} in the resulting execution tree: \(\Pr(B^i_u \mid A^i_u) \geq 1 - \epsilon\).

\subsection{4.2 Constants}

Below is a summary of the constants used in the algorithm description and analysis that follow. In the following, we assume \(\Delta\) is a power of two.

- \(\epsilon_1\) is the notation used in the below algorithm description to describe the desired error probability.
- \(\epsilon'\) is the maximum error probability bound that guarantees, given the constraints of Theorem 3.1, that the SeedAlg(\(\epsilon'\)) algorithm satisfies the Seed(\(\delta, \epsilon\)) spec for an \(\epsilon \leq \epsilon_1/2\).

\textit{Note:} given the relationship between SeedAlg and Seed’s error bounds, as established in Theorem 3.1, \(\epsilon' = \Theta((\frac{\epsilon_1}{\epsilon})^{1/(2c^2)})\), where \(c < 1\), is the constant provided by Theorem 3.1. Because \(c < 1\), we can rewrite the bound as \(\Theta((\frac{1}{\epsilon_1 \log^2 \Delta})^{\gamma/r^2})\), for some constant \(\gamma > 1\).
- \(\epsilon_2 = \min\{\epsilon', \epsilon_1\}\).
- \(\epsilon_2\) is a constant used in our analysis of successful receptions of messages.
- \(\epsilon_1\) is a constant we use in defining the length of a phase in the algorithm (see \textit{T}_{\text{prog}} below).
- \(\text{T}_{\text{prog}} = (\frac{\epsilon_1 \cdot r^2 \cdot \log (\frac{1}{\epsilon_1}) \cdot \log (\Delta) \log (\Delta)}{2 \epsilon_2 T_{\text{prog}} (1 - \epsilon_1/2) \log (\Delta)})\).
- \(\text{T}_{\text{ack}} = \frac{\epsilon_2 T_{\text{prog}} (1 - \epsilon_1/2) \log (\Delta)}{2 \epsilon_2 \epsilon_1 \log_2 (\Delta) \log (\Delta)}\).
- \(\text{T}_{\text{prog}} = (\frac{\epsilon_1 \cdot r^2 \cdot \log (\frac{1}{\epsilon_1}) \cdot \log (\Delta) \log (\Delta)}{2 \epsilon_2 T_{\text{prog}} (1 - \epsilon_1/2) \log (\Delta)})\).
- \(\text{T}_{\text{ack}} = \frac{\epsilon_2 T_{\text{prog}} (1 - \epsilon_1/2) \log (\Delta)}{2 \epsilon_2 \epsilon_1 \log_2 (\Delta) \log (\Delta)}\).

\subsection{4.3 A Local Broadcast Algorithm}

We now describe \textit{LBAlg}: our solution to the local broadcast problem. This description makes use of several constants described in the previous section.

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

Node \textit{u} partitions rounds into \textit{phases} of length \textit{T}_{\text{r}} + \textit{T}_{\text{prog}} rounds. We label these phases 1, 2, 3, .... During each phase, \textit{u} can be in one of two states:

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}

\begin{algorithm} \textbf{LBAlg}(\epsilon_1), for process \textit{i} at vertex \textit{u} for some real \(\epsilon_1, 0 < \epsilon_1 \leq \frac{1}{2}\).
\end{algorithm}
receiving or sending. Node $u$ begins the execution in the receiving state. After receiving a \textit{bcast}($m$) input, $u$ will spend the next $T_{\text{ack}}$ full phases in the sending state (if it receives the \textit{bcast} input in the middle of a phase, it waits until the beginning of the next phase to switch to the sending state). At the end of the last round of the last of these $T_{\text{ack}}$ phases, node $u$ generates an \textit{ack}($m$) output, and then returns to the receiving state.

We now define what happens during these phases. At the beginning of each phase, regardless of $u$'s state, it executes \textit{SeedAlg}($s$) as a subroutine, using the seed set $S_u = \{0, 1\}^n$; i.e., the set describing every bit sequence of length $\kappa$. Let $s_u^{(j)}$ be the seed that node $u$ commits in the beginning of phase $j$. We call the rounds spent at the beginning of a phase running \textit{SeedAlg} the preamble of the phase, and the remaining rounds the body of the phase.

Node $u$'s behavior during the body of a given phase $j$ depends on its state. If it is in the receiving state, it simply receives during each of these rounds. If during one of these rounds, node $u$ receives a message $m'$ that it has not yet previously received, it generates a \textit{recv}($m'$) output.

On the other hand, if $u$ is in the sending state for this phase, during each of the body rounds, it does the following:

1. Node $u$ consumes $\lceil \log \left( \frac{\log \left( \frac{2^r \log \left( \frac{1}{\epsilon} \right)}{\Delta} \right)}{\log \Delta} \right) \rceil$ new bits from its seed $s_u^{(j)}$. If all of these bits are 0 (which occurs with probability $a \cdot \frac{1}{\log \left( \frac{2^r \log \left( \frac{1}{\epsilon} \right)}{\Delta} \right)}$ for some $a \in [1, 2]$) it sets its status to participant, otherwise it sets its status to non-participant.

2. If $u$ is a non-participant, it receives.

3. If $u$ is a participant, it next consumes $\log \log \Delta$ new bits from $s_u^{(j)}$. Let $b$ be the value in $\lceil \log \Delta \rceil$ specified by these bits. The node then uses an independent (with respect to the other processes) local source of randomness (i.e., not bits from $s_u^{(j)}$), to generate $b$ bits with uniform randomness. If all $b$ bits are 0 (which occurs with probability $2^{-b}$), $u$ broadcasts its message.

As with the receiving state, if during any of these body rounds, node $u$ receives a message $m'$ that it has not yet previously received, it generates a \textit{recv}($m'$) output.

The above algorithm divides rounds into phases and then runs a seed agreement algorithm at the beginning of each phase to synchronize shared random bits for the remainder of the phase. Notice that there is nothing fundamental about this frequency of seed agreements. In some settings, it might make sense to run the agreement protocol less frequently, and generate seeds of sufficient length to satisfy the demands of multiple phases. Such modifications do not change our worst-case time bounds but might improve an average case cost or practical performance.

### 4.4 Correctness of \textit{LBAlg}

Our goal here is to analyze \textit{LBAlg} to prove the following theorem:

\textbf{Theorem 4.1.} \textit{LBAlg}($\epsilon_1$) solves the \textit{LB}($t_{\text{ack}}, t_{\text{prog}}, \epsilon_1$) problem for parameters:

- $t_{\text{prog}} = T_x + T_{\text{prog}} = O \left( r^2 \log \Delta \log \left( \frac{\log \log \Delta}{\epsilon_1} \right) \right)$
- $t_{\text{ack}} = (T_{\text{ack}} + 1)(T_x + T_{\text{prog}}) = O \left( r^2 \log \Delta \log \left( \frac{\log \log \Delta}{\epsilon_1} \right) \frac{\log \log \Delta}{\epsilon_1} \right)$

Below is the core lemma on which we build our proof of Theorem 4.1. This lemma bounds the behavior of \textit{LBAlg} within the scope of a single phase. To do so, we first introduce some useful notation. For a given phase $i$ of an execution, let $B_i$ be the set of nodes that are in sending status during phase $i$, and $R_i = N_G(B_i)$ be the set of nodes that neighbor $B_i$ in $G$. Notice, because sending status is fixed for the duration of a given phase, both $B_i$ and $R_i$ are determined at the beginning of phase $i$ and cannot change during the phase. Also recall from the model definitions that $\Delta'$ bounds the maximum degree in $G'$. Using this notation, we specify and prove the following key probabilistic behavior:

\textbf{Lemma 4.2.} Fix some phase $i$ and an execution prefix through the $(j - 1)^{th}$ body round of this phase, for some $j \in \{2, ..., T_{\text{prog}}\}$. Fix nodes $u$ and $v$, where $u \in R_i$ and $v \in N_G(u) \cap B_i$. Assume the call to \textit{SeedAlg} at the beginning of phase $i$ in this prefix satisfies $B_{u, \delta}$. Let $p_u$ be the probability that $u$ receives some message in the $j^{th}$ round, and let $p_{u,v}$ be the probability that $u$ receives a message from $v$ in this round. It follows that:

- $p_u \geq \frac{r^2 \log \left( \frac{2^r}{\Delta'} \right) \log \Delta}{\epsilon_1}$
- $p_{u,v} \geq p_u / \Delta'$

\textbf{Proof.} Fix some $u, v, t$ and a prefix, as specified by the lemma statement. (Notice, we know that a node $v$ satisfying the constraints of the statement exists due to the assumption that $u \in R_i$, which implies that $|N_G(u) \cap B_i| > 0$.) Let $S = \{S_1, S_2, ..., S_k\}$ be a minimum-sized partition of the nodes in $N_G(u) \cap B_i$ such that all nodes in $S_j$ committed to the same seed in the beginning of this phase. Given the lemma assumption that the preamble of this phase satisfies $B_{u, \delta}$, it follows: $k \leq \delta$. Finally, let $S_v$ be the set from $S$ that contains $v$.

We now analyze the next broadcast round. In this round, nodes in $B_i$ use their seeds to decide whether or not to become a participant. In particular, they become a participant with probability $\frac{1}{r^2 \log \left( \frac{2^r}{\Delta'} \right)} = c/\delta$, for some constant $c > 0$, using bits from their seeds to resolve the random choice. For each $S_j \in S$, all nodes in $S_j$ make the same decision in each round because they are using bits from the same seed. Let $p_{u,v}$ be the probability that set $S_v$ decides to be a participant, and all other sets in $S$ decide to be non-participants. To bound $p_{u,v}$, we apply Lemmata B.17 and B.18 from the analysis of seed agreement in the full version of this paper [18] to obtain the uniformity and independence properties needed to prove:

\textit{...}
\[ p^c = \frac{(c/\delta)(1 - (c/\delta))^{k-1}}{\alpha} \]

\[ \geq \frac{(c/\delta)(1/4)^{\ell}}{\alpha} \]

Assume this event—that only \( S_v \) decides to participate from among the sets in \( S \)—occurs. It follows that only nodes in \( S_v \) can potentially broadcast in this round. Let \( \ell \) be the number of links from nodes in \( S_v \) to \( u \) included in the network topology for the round by the link scheduler included in our configuration definition. Because \( S_v \) contains the \( G \)-neighbor \( v \) (by definition), we know that \( v \) is connected to \( u \) and that therefore \( \ell > 0 \).

The next thing that happens in this round is that the nodes in \( S_v \) use more random bits from their shared seed to choose a value uniformly from \([\log \Delta]\). If \( \ell = 1 \), we define the \textit{correct} choice of value from \([\log \Delta]\) to be 1. If \( \ell > 1 \), we define the \textit{correct} choice to be \([\log \ell]\). By Lemma B.17 (from [18]), we know the nodes in \( S_v \) will choose a value from this set with uniform probability. The probability they choose a correct value with respect to \( \ell \) is therefore at least \( 1/\log \Delta \).

Assume this event also occurs. At this point, by assumption, only nodes in \( S_v \) are potential broadcasters. Each such node decides to broadcast with the correct probability, \( p^c \), which, as defined above, is within a factor of 2 of \( 1/\ell \). Let us consider the possibilities. We first note that with probability at least \( 1/2 \), \( u \) will decide to receive in this round (broadcast probability \( 1/\ell \), which corresponds to choosing the value 1 from \([\log \Delta]\), is the largest possible broadcast probability).

Assume this event occurs. The probability that exactly one neighbor among the \( \ell \) neighbors connected to \( u \) subsequently decides to broadcast is constant (as there are \( \ell \) neighbors, each deciding to broadcast with probability \( p_u = \Theta(1/\ell) \)). To calculate \( p_u \) we must now combine all three independent probabilities: the \( \Theta(1/\ell) \) probability that \( S_v \) is the only set in \( S \) to participate, the \( 1/\log \Delta \) probability that \( S_v \) nodes choose the correct value, and the \( \Theta(1) \) probability that \( u \) decides to receive and exactly one neighbor of \( u \) in the topology for the round broadcasts. We combine the constants in these asymptotic expression to define a lower bound on the constant \( c_2 \) used in our definition of \( p_u \) from the lemma statement.

Now we step back to consider \( p_{u,v} \). Whereas we just calculated that there is a constant probability that exactly one node from among \( \ell \) nodes decides to broadcast using broadcast probability \( p^c \), we must now ask the probability that a specific node—i.e., \( v \)—is this broadcaster. We can bound this probability as:

\[ p^c(1 - p^c)^{\ell - 1} = \frac{c'}{\ell} \left(1 - \frac{c'}{\ell}\right)^{\ell - 1} \geq \frac{c'}{\ell} \left(\frac{1}{4}\right)^{\ell} \geq \frac{1}{4\Delta}, \]

where \( c' \) is a constant of size at least 1 used in the definition of \( p^c \). By replacing the constant probability for this final step used in the derivation of \( p_u \) above with this new \( \approx 1/\Delta' \) probability, we get the \( p_{u,v} \) bound required by the lemma statement. (As a slight technicality, we omit the \( 1/4 \) in the \( 1/\Delta \) calculation above in this final \( p_{u,v} \) bound, as this can be captured by adjusting the constant \( c_2 \) calculated for \( p_u \) by a factor of 4 to include this extra amount.)

We can now draw on Lemma 4.2 to prove the progress and reliability properties required by the LB specification. We begin with progress:

**Lemma 4.3.** \( LBAlg(\epsilon_1) \) solves the LB\((t_{ack}, t_{prog}, \epsilon_1)\) problem for: \( t_{prog} = T_{s} + T_{prog} \).

\[ p_{fail} = \prod_{j=1}^{T_{prog}} (1 - p_{u,j}) \leq \prod_{j=1}^{T_{prog}} (1 - p_u) < (1/e)^{c_2 \log (1/\epsilon_1)}. \]

It is straightforward to show that for sufficiently large values of constants \( c_1 \) and \( c_2 \), we get \( p_{fail} \leq \epsilon_1/2 \). To conclude the proof, we use a union bound to show that the probability that \( B_{u,\delta} \) does not hold and/or the probability that \( u \) fails to receive a message when this property does hold, is less than \( \epsilon_1 \): providing the needed 1 - \( \epsilon_1 \) probability for \( u \) receiving at least one message in phase \( i \).

We now turn our attention to the reliability property of our local broadcast problem:

**Lemma 4.4.** \( LBAlg(\epsilon_1) \) solves the LB\((t_{ack}, t_{prog}, \epsilon_1)\) problem for: \( t_{ack} = (T_{ack} + 1)(T_{prog} + T_{s}) \).

\[ p_{fail} = \prod_{j=1}^{T_{prog} + T_{s}} (1 - p_{u,j}) \leq \prod_{j=1}^{T_{prog}} (1 - p_u) < (1/e)^{c_2 \log (1/\epsilon_1)}. \]

In the following, we define a body round to be \textit{useful} with respect to \( u \), if it occurs in a phase such that \( B_{u,\delta} \) holds for the preceding \textit{SendAlg} preamble. Let \( p_1 \) be the probability that \( u \) fails to receive a message \( m \) from \( v \) during \( k \) useful rounds in which \( v \) is active with \( m \). Applying Lemma 4.2 to
lower bound the receive probability in each of these rounds, it follows:

\[ p_{1} \leq (1 - p)^{k} < \left(\frac{1}{e}\right)^{k} = \frac{\epsilon_{1}}{2\Delta}. \]

We now investigate the number of phases necessary to ensure that \( v \) experiences at least \( k \) useful rounds with a sufficiently high probability. To do so, we first fix \( q = \lceil \frac{12k}{T_{\text{prog}}\ln(1 + \epsilon_{1}\epsilon_{2})} \rceil \). Consider an experiment where we run \( q \) consecutive phases. Let \( X_{i} \), for \( i \in [q] \), be a random variable that describes the number of useful rounds in phase \( i \) of the experiment. Notice, \( X_{i} \) either takes on the value \( T_{\text{prog}} \) (with probability at least \( 1 - \epsilon_{1}/2 \)) or 0 (else). Let \( Y = X_{1} + X_{2} + \ldots + X_{q} \) be the total number of useful rounds generated by the experiment. It follows:

\[ \mathbb{E}[Y] = q \cdot T_{\text{prog}} \cdot (1 - \epsilon_{1}/2) = 12k. \]

We now apply a Chernoff Bound, to bound the probability that \( Y \) is more than a factor of 2 smaller than its expectation \( \mu = \mathbb{E}[Y] \):

\[ \Pr(Y < (1/2)\mu) < e^{-\frac{\mu - \mathbb{E}[Y]}{\ln(2k)}}. \]

(Notice, in the above we can bound \( e^{-k} \leq e^{-\ln(2\Delta)} = \epsilon_{1}/(2\Delta) \) because \( k \geq \ln(\frac{2\Delta}{\epsilon_{1}/(2\Delta)}) \).

We have now established that with probability at least \( 1 - \epsilon_{1}/(2\Delta) \), \( u \) will experience at least \( 6k > k \) useful rounds in \( q \) phases. We earlier established that if \( u \) experiences at least \( k \) useful rounds during which \( v \) is broadcasting \( m \), then \( u \) receives \( m \) from \( v \) with probability at least \( 1 - \epsilon_{1}/(2\Delta) \). Assume \( u \) broadcasts \( m \) for at least \( q \) consecutive phases. By a union bound, the probability that both events occur with respect to these phases, and \( u \) therefore receives \( v \)'s message \( m \), is greater than \( 1 - \epsilon_{1}/\Delta \).

To conclude, we want to calculate, under the assumption that \( v \) broadcasts \( m \) for at least \( q \) phases, that every \( G \) neighbor of \( v \) succeeds in receiving \( m \). Because there are at most \( \Delta \) such neighbors, and each succeeds with probability at least \( 1 - \epsilon_{1}/\Delta \), a union bound says that every neighbor succeeds with probability at least \( 1 - \epsilon_{1} \), as required by the reliability property.

To satisfy reliability, therefore, it is sufficient for any node \( v \) receiving a \( \text{bcast}(m) \_i \) input to spend at least \( q \) full phases with sending status. Notice, this is exactly what \( \text{LBAlg} \) requires, as by definition it has \( v \) spend the next \( T_{\text{ack}} = q \) full phases after a \( \text{bcast}(m) \_i \) input in sending status. The \( T_{\text{ack}} \) in the lemmas statement is defined to be long enough for \( v \) to wait up to a full phase length until the next phase boundary, plus the rounds required for an additional \( q \) phases.

We now pull together the pieces to prove Theorem 4.1:

**Proof of Theorem 4.1.** The definition of the local broadcast problem had four conditions, two deterministic and two probabilistic. We consider each in turn and argue that \( \text{LBAlg} \) satisfies the conditions for the parameter values specified in the theorem statement.

We first note that timely acknowledgment holds because \( \text{LBAlg} \), by definition, has each node generate an \( \text{ack} \) in response to a \( \text{bcast} \) within a fixed number of rounds that is strictly less than the \( T_{\text{ack}} \) factor from the theorem statement. Similarly, the validity condition holds as \( \text{LBAlg} \), by definition, only has nodes broadcast messages they received in a \( \text{bcast} \) input, and nodes only \( \text{recv} \) messages that they actually received from another node. Moving on to the probabilistic properties, Lemma 4.3 tells us that \( T_{\text{prog}} = T_{s} + T_{\text{prog}} \) rounds, and that \( T_{\text{ack}} = (T_{\text{ack}} + 1)T_{\text{prog}} \) rounds. Notice, \( T_{\text{ack}} \) shows up in \( T_{\text{ack}} \) unchanged from its definition. The definition of \( T_{\text{prog}} \), however, shows up in a form that is simplified as compared to the definition provided for \( T_{s} \) and \( T_{\text{prog}} \) (which contain both \( \epsilon_{1} \) and \( \epsilon_{2} \) facets). To match the bounds in the Theorem statement, therefore, it is sufficient to show that \( T_{s} + T_{\text{prog}} = O(\log\Delta\log^{2}(\frac{r}{\epsilon_{1}}) + r^{2}\log\Delta\log(\frac{1}{\epsilon_{1}})\log(\frac{1}{\epsilon_{2}})) \) can be upper bounded by \( O\left(r^{2}\log\Delta\log\left(\frac{r^{4}\log^{2}\Delta}{\epsilon_{1}\epsilon_{2}}\right)\right) \). We deduce the remainder of this proof to this effort.

By definition, \( \epsilon_{2} \leq \epsilon_{1} \). We can, therefore, substitute the former for the latter in our sum, yielding:

\[ T_{s} + T_{\text{prog}} = O(r^{2}\log\Delta\log^{2}(\frac{1}{\epsilon_{2}})). \]

We need a bound, however, that is expressed with respect to the problem parameter \( \epsilon_{1} \). This requires us to dive deeper into the relationship between \( \epsilon_{2} \) and \( \epsilon_{1} \). There are two cases to consider given our definition above that \( \epsilon_{2} = \min(\epsilon', \epsilon_{1}) \). The first case is that \( \epsilon_{2} = \epsilon_{1} \). If this is true, we can simply replace \( \epsilon_{2} \) with \( \epsilon_{1} \) in our above equation, and the result is clearly upper bounded by \( O\left(r^{2}\log\Delta\log\left(\frac{r^{4}\log^{2}\Delta}{\epsilon_{1}^{2}}\right)\right) \) (which strictly increases the log factor by adding the \( r^{4}\log^{2}\Delta \) term).

The second case is that \( \epsilon_{2} = \epsilon' < \epsilon_{1} \). By definition of \( \epsilon' \), it would then follow that \( \epsilon_{2} = \Theta\left(\frac{\gamma}{\log^{2}(\epsilon')}\right) \), for some constant \( \gamma > 1 \). The properties of \( \gamma \) allows us to simplify (asymptotically speaking) the \( \log^{2}(\frac{1}{\epsilon_{2}}) \) term in our above equation as follows:

\[ \log^{2}(\frac{1}{\epsilon_{2}}) = \log\left(\frac{r^{4}\log^{4}\Delta}{\epsilon_{1}\log^{2}(\epsilon')}\right)^{2} = \left[\frac{\gamma}{r^{2}}\log\left(\frac{r^{4}\log^{4}\Delta}{\epsilon_{1}}\right)\right]^{2} = O(\log^{2}(\frac{r^{4}\log^{2}\Delta}{\epsilon_{1}})). \]

Notice in the above we simply drop the \( (1/r^{2}) \), as it too is bounded to be at least 1, so dropping it simply increases the value of the upper bound. Now to conclude our argument, we simply substitute this upper bound for \( \log^{2}(\frac{1}{\epsilon_{2}}) \) in our above sum to get the desired equation.

5. CONCLUSION

In this paper, we described and analyzed an ongoing local broadcast service for the dual graph model. This service hides the complexities introduced by unpredictable link behaviors and therefore has the potential to significantly simplify the development of distributed algorithms for this challenging setting. As noted in the introduction, our solution can also be adapted to implement the abstract MAC layer specification [13, 14], allowing existing results for this abstraction to translate to the dual graph model (e.g., [9, 20, 6, 12, 11, 5]). Though we leave the details of this adaptation to future work, we note that it would likely be straightforward, with the main effort focused on aligning our local broadcast problem definition—which depends on low level model details, such as rounds and receiving messages—with the higher level of the abstract MAC layer,
which is specified in terms of the timing and ordering of input and output events.

Finally, we emphasize that our commitment to truly local algorithms, which required us to avoid global parameters in problem definitions, algorithm strategies, and analysis, is of standalone interest to those studying radio network algorithms. As we argued at this paper’s opening, a local perspective provides more flexibility to practitioners, and will become increasingly necessary as network sizes grows.

6. ACKNOWLEDGMENTS

This research is supported in part by: Ford Motor Company University Research Program, NSF Award CCF-1320279, NSF Award CCF-0937274, NSF Award CCF-1217506, NSF CCF-0939370, and AFOSR Award Number FA9550-13-1-0042.

7. REFERENCES