MIT Libraries | DSpace@MIT

MIT Open Access Articles

Structuring Unreliable Radio Networks

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

Citation: Keren Censor-Hillel, Seth Gilbert, Fabian Kuhn, Nancy Lynch, and Calvin Newport. "Structuring unreliable radio networks." In Proceedings of the 30th annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC '11). ACM, New York, NY, USA, 79-88.

As Published: http://dx.doi.org/10.1145/1993806.1993818

Publisher: Association for Computing Machinery (ACM)

Persistent URL: http://hdl.handle.net/1721.1/73186

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of use: Creative Commons Attribution-Noncommercial-Share Alike 3.0

Massachusetts Institute of Technology

Structuring Unreliable Radio Networks

Keren Censor-Hillel[*]
Computer Science and Artificial Intelligence Lab, MIT
ckeren@csail.mit.edu

Seth Gilbert[†]
Department of Computer Science, National University of Singapore
gilbert@comp.nus.edu.sg

Fabian Kuhn
Faculty of Informatics, University of Lugano
fabian.kuhn@usi.ch

Nancy Lynch[‡]
Computer Science and Artificial Intelligence Lab, MIT
lynch@csail.mit.edu

Calvin Newport[§]
Computer Science and Artificial Intelligence Lab, MIT
cnewport@csail.mit.edu

ABSTRACT

In this paper we study the problem of building a connected dominating set with constant degree (CCDS) in the dual graph radio network model [4, 9, 10]. This model includes two types of links: *reliable*, which always deliver messages, and *unreliable*, which sometimes fail to deliver messages. Real networks compensate for this differing quality by deploying low-layer detection protocols to filter unreliable from reliable links. With this in mind, we begin by presenting an algorithm that solves the CCDS problem in the dual graph model under the assumption that every process u is provided a local *link detector* set consisting of every neighbor connected to u by a reliable link. The algorithm solves the CCDS problem in $O(\frac{\Delta \log^2 n}{b} + \log^3 n)$ rounds, with high probability, where Δ is the maximum degree in the reliable link graph, n is the network size, and b is an upper bound in bits on the message size. The algorithm works by first building a Maximal Independent Set (MIS) in $\log^3 n$ time, and then leveraging the local topology knowledge to efficiently connect nearby MIS processes. A natural follow up question is whether the link detector must be perfectly reliable to solve the CCDS problem. With this in mind, we first describe an algorithm that builds a CCDS in $O(\Delta \text{polylog}(n))$ time under the assumption of $O(1)$ unreliable links included in each link detector set. We then prove this algorithm to be (almost) tight by showing that the possible inclusion of only a single unreliable link in each process's local link detector set is sufficient to require $\Omega(\Delta)$ rounds to solve the CCDS problem, regardless of message size. We conclude by discussing how to apply our algorithm in the setting where the topology of reliable and unreliable links can change over time.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*network problems*

General Terms

Algorithms, Theory

Keywords

unreliable networks, dual graphs, maximal independent set, connected dominating set

1. INTRODUCTION

In this paper we study the problem of constructing a connected dominating set with constant degree (CCDS) in a radio network. The CCDS problem is important in this setting as it provides a routing backbone that can be used to efficiently move information through the network [15, 19]. In more detail, we study this problem in the dual graph network model, which describes static ad hoc radio networks. The dual graph model, previously studied in [4,9,10], includes two types of links: *reliable*, which in the absence of collisions always deliver messages, and *unreliable*, which sometimes fail to deliver messages. This model was inspired by the observation that in real radio network deployments unreliable links are an unavoidable (and much cursed) feature; c.f., [1, 2, 5–7, 16, 18, 20]. To mitigate the difficulties introduced by such links, most modern ad hoc radio network deployments use low-level link detector protocols (e.g., [2, 5–7, 20]) or sometimes even specialized hardware (e.g., [1]) that attempt to isolate reliable from unreliable links. We capture this strategy in our model with the new *link detector* abstraction, which provides each process u, at the beginning of each execution, a set of ids that represent an estimate of which neighbors are reliable (i.e., connected to u by a reliable link).

Using this abstraction, we are able to explore two important questions: (1) How can we leverage the link detection information commonly assumed in practice to build efficient solutions to the CCDS problem? (2) How reliable must these link detectors be

[*]Supported by the Simons Postdoctoral Fellows Program.

[†]Partially supported by NUS (FRC) R-252-000-443-133.

[‡]Supported by AFOSR award number FA9550-08-1-0159, NSF award numbers CCF-0726514, CCF-0937274, and NSF-Purdue-STC award number 0939370-CCF.

[§]Supported by Mobile Mesh Networks (Ford-MIT Alliance Agreement January 2008).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODC'11, June 6–8, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0719-2/11/06 ...$10.00.

for their information to be useful? Our answers potentially extend beyond the realm of theoretical interest and into the realm of practice, where the optimal use of link detection is considered an open problem.

Results.

In this paper, we study the *τ-complete link detector*, $0 \leq \tau \leq n$. A τ-complete link detector, for a given process u, contains the id of every reliable neighbor of u and potentially up to τ additional ids. In other words, τ bounds the number of classification mistakes made by the detector, with 0-complete indicating perfect knowledge of reliable neighbors.[1] Notice, however, that assuming a 0-complete link detector is different than assuming a network with only reliable edges: the completeness of the link detector only describes the quality of knowledge about the network topology, but one still must grapple with the uncertainty caused by the presence of unreliable edges.

As mentioned, practical network deployments seek to accurately filter reliable from unreliable links; i.e., implement a 0-complete link detector [2, 5–7, 20]. With this in mind, in Section 5 we describe a randomized upper bound that uses a 0-complete link detector to construct a CCDS. In more detail, the algorithm constructs a CCDS in $O(\frac{\Delta \log^2 n}{b} + \log^3 n)$ rounds, with high probability, where Δ is the maximum degree in the reliable link graph, b is an upper bound in bits on the message size, and n is the network size. For reasonably large messages ($b = \Omega(\Delta)$), this algorithm terminates in polylogarithmic time. The algorithm works by first building a Maximal Independent Set (MIS) in $O(\log^3 n)$ rounds (the algorithm for which is presented separately, in Section 4), and then leveraging the link detector information to execute a novel *path finding* procedure to identify paths to nearby MIS processes.

A natural follow-up question is whether such accuracy in our link detector is necessary. In other words, can we find efficient solutions to the CCDS problem for some $\tau > 0$? To answer this question, we start by describing, in Section 6, an algorithm that solves the CCDS problem in $O(\Delta \text{polylog}(n))$ rounds, given a τ-complete detector for any $\tau = O(1)$. We then prove in Section 7 that this bound is (almost) tight by showing that with a 1-complete link detector, every algorithm that solves the CCDS problem requires $\Omega(\Delta)$ rounds, regardless of message size. This bound not only defines a separation with the classic radio network model, which assumes only reliable links, but also defines a separation with the $\tau = 0$ case. Concurrent work has identified a CCDS algorithm for the classic model that uses no topology knowledge and requires only $O(\text{polylog}(n))$ rounds [17].

We conclude by discussing, in Section 8, how to apply our algorithm in the setting where the topology of reliable and unreliable links can change over time.

Related Work.

The dual graph model was introduced in [4], where it was called the dynamic fault model, and then later studied in [9, 10] under its current name. These papers show, among other results, that the canonical problem of multihop broadcast is strictly harder in the presence of unreliable links. There are some similarities between the dual graph model and the quasi-unit disk graph model [13], which includes a gray zone distance at which two nodes in a radio network may or may not have a link. Unlike the dual graph model, however, the quasi-unit disk graph model features uncertainty only in the definition of the topology; once the links have been decided, they behave reliably.

The CCDS problem, along with related coordination problems, have been extensively studied in general graph models (see [12] for a good overview). In the context of radio networks without unreliable links (what we call the *classic radio network model*), [19] describes an $O(n)$ time CCDS algorithm, and [15] describes an $O(\log^2 n)$ time algorithm. The latter algorithm, however, requires that processes know their multihop local neighborhoods so they can construct collision-free broadcast schedules. In our model, for a process to learn its $(h+1)$-hop neighborhood (of reliable links) would require $\Omega(\Delta^h)$ time, and even then the broadcast schedules constructed in [15] could be thwarted by unreliable links causing collisions. As with our paper, both [19] and [15] assume synchronous starts (i.e., processes start during the same round). Concurrent work has identified a $O(\text{polylog}(n))$-time CCDS solution in the classic radio network model *without synchronous starts* [17].

The MIS problem, which we use as a step in our construction of a CCDS, was studied in the classic radio network model without synchronous starts in [11], which provides a $O(\log^6 n)$ time solution. This was later improved in [14] to $O(\log^2 n)$. The MIS algorithm presented in the main body of this paper requires $O(\log^3 n)$ rounds, and it assumes synchronous starts and a 0-complete link detector. In the full version of this paper, however, we describe a minor variation to the algorithm that works in the same running time in the classic radio network model, without synchronous starts or any topology information. This algorithm is a factor of $O(\log n)$ slower than the result of [14], but trades this decreased speed for increased simplicity in both the algorithm description and proof structure.

2. MODEL

Fix some $n > 2$. We define a network (G, G') to consist of two undirected graphs, $G = (V, E)$ and $G' = (V, E')$, where V is a set of n wireless nodes and $E \subseteq E'$. We assume G is connected. For each $u \in V$, we use the notation $N_G(u)$ to describe the neighbors of u in E, and the notation $N_{G'}(u)$ to describe the neighbors of u in E'. Let Δ be the maximum size of N_G over all nodes and Δ' be the maximum size of $N_{G'}$ over all nodes. To simplify notation we assume in this paper that $\Delta = \omega(\log n)$. We assume that each node in V is embedded in a two-dimensional plane, and use $dist(u, v)$ to denote the distance between nodes u and v in the plane. We assume there exists a *constant* distance $d \geq 1 = O(1)$, such that for all $u, v \in V$ where $dist(u, v) \leq 1$, $(u, v) \in E$, and for all $(u', v') \in E'$, $dist(u', v') \leq d$. Notice, this is a generalization of the unit disk graph model that now captures the (potentially) large *gray zone* of unpredictable connectivity observed in real wireless networks.

We next define an algorithm $\mathcal{A}$ to be a collection of n processes. An execution of an algorithm $\mathcal{A}$ on network (G, G') first fixes some bijection $proc$ from processes of $\mathcal{A}$ to V. This bijection represents the assignment of processes to graph nodes. We assume an adversary controls the definition of $proc$. We also assume that each process in $\mathcal{A}$ has a unique identifier from the range 1 to n. We use the notation $id(v)$, $v \in V$, with respect to an execution, to indicate the unique identifier of $proc(v)$. For simplicity, throughout this paper we sometimes use the notation *process* u, for some $u \in V$, to refer to $proc(u)$ in the execution in question. We also sometimes use the notation *process* i, for some $i \in [n]$, to refer to the process with id i.

An execution proceeds in synchronous rounds, $1, 2, \ldots$, with all nodes starting in the first round. At the beginning of each round,

[1] Notice, these detectors never misclassify reliable neighbors as unreliable. In practice, we suspect such misclassifications would not affect our algorithms' correctness, provided that the correctly classified reliable edges still describe a connected graph. We omit this variant for the sake of conciseness.

r, every node v decides whether or not to send a message, as indicated by its process, $proc(v)$. Next, the adversary chooses a *reach set* of edges that consists of E and some subset, potentially empty, of edges in E' but not E. This set describes the links that will behave reliably in this round.[2] Let $B_{v,r}$ be the set of nodes that broadcast in round r and are connected to v by an edge in the reach set for this round. The messages received by v depend on the size of $B_{v,r}$. If node v broadcasts in r then it receives only its own message. If node v does not broadcast and $|B_{v,r}| = 1$, then it receives the message sent by the single broadcaster in $B_{v,r}$. Otherwise, it receives $\perp$; i.e., we assume no collision detection.

We sometimes use the notation $[i]$, for positive integer i, to indicate the sequence $\{1, ..., i\}$. Furthermore, we use the notation w.h.p. (i.e., *with high probability*) to indicate a probability at least $1 - \frac{1}{n^c}$, for some positive constant c. For simplicity we omit the specific constants used in our proofs, and assume only that they are large enough such that the union bounds applied to our various w.h.p. results produce a final probability that is also at least $1 - \frac{1}{n}$.

Link Detectors.

As described in the introduction, real wireless network deployments compensate for unreliability by using low-level protocols and special hardware to differentiate reliable from unreliable links. Because these link detection strategies often make use of information not described in our network model (e.g., properties of the received physical layer signal) we introduce the *link detector* abstraction to capture the functionality of these services.

In more detail, this abstraction provides each process $proc(u)$ a link detector set $L_u \subseteq [n]$. This set, fixed through the entire execution, is an estimate of which neighbors are connected to u by a reliable link. In this paper we study the **τ-complete link detector**, $0 \leq \tau \leq n$. In more detail, we say a link detector set L_u is **τ-complete** if and only if $L_u = \{id(v) : v \in N_G(u)\} \cup W_u$, where $W_u \subseteq \{id(w) : w \in V \setminus N_G(u)\}$, and $|W_u| \leq \tau$. That is, the detector contains the id of every neighbor of u connected by a reliable link, plus up to an additional τ additional neighbors. This makes τ a bound on the number of links that are mistakenly classified as reliable.

3. PROBLEM DEFINITIONS

We define the maximal independent set and constant-bounded connected dominating set problems. In both definitions, we reference the graph $H = (V, E_H)$, defined with respect to a specific execution, where V is the vertex set from G and G', and E_H is the edge set consisting of every edge (u, v) such that: $u \in L_v$ and $v \in L_u$ (that is, u and v are in each other's link detector set). Notice, for a τ-complete link detector, for any value of τ, G is a subgraph of H, and for $\tau = 0$, $H = G$.

Maximal Independent Set.

A maximal independent set (MIS) algorithm has every process eventually output a 0 or a 1, where a 1 indicates the process is in the MIS, and a 0 indicates it is not. We say an execution of an MIS algorithm has *solved the MIS problem* by round r, if and only if the following three conditions hold: (1) **[Termination]** every process outputs 0 or 1 by the end of round r; (2) **[Independence]** if processes u and v both output 1, then $(u, v) \notin E$; and (3) **[Maximality]** if process u outputs 0, then there exists a process v such that v outputs 1 and $(u, v) \in E_H$.

Constant-Bounded Connected Dominating Set.

A constant-bounded connected dominating set (CCDS) algorithm has every process eventually output a 0 or a 1, where a 1 indicates the process is in the CCDS, and a 0 indicates it is not. We say an execution of a CCDS algorithm has *solved the CCDS problem* by round r, if and only if the following four conditions hold: (1) **[Termination]** every process outputs 0 or 1 by the end of round r; (2) **[Connectivity]** the set of processes that output 1 is connected in H; (3) **[Domination]** if a process u outputs 0, then there exists a process v such that v outputs 1 and $(u, v) \in E_H$; and (4) **[Constant-Bounded]** there exists a constant δ, such that for every process u, no more than δ neighbors of u in G' output 1.

4. MIS ALGORITHM

In this section, we present an algorithm that solves the MIS problem in $O(\log^3 n)$ rounds, w.h.p. We assume the processes have access to a 0-complete link detector and that the message size b is $\Omega(\log n)$ bits. In Section 5, we use this algorithm as a subroutine in our solution to the CCDS problem. Recall that having a 0-complete link detector is not equivalent to having a network model with only reliable edges. The completeness of the link detector describes the quality of information about the topology; it does not eliminate the negative impact of unreliable edges. In particular, as highlighted by the proofs in this section and the next, one of the main difficulties presented by unreliable edges is that their unpredictable behavior can thwart standard contention-reduction techniques, such as the exponential increase of broadcasting probability.

Algorithm Description.

The algorithm has processes discard messages received from a process not in its link detector set. Therefore, in the following description, when we say that a process *receives a message*, we imply that it is a message sent from a neighbor in E. Each process u maintains a set M_u of MIS process ids (initially empty). The execution is divided into $\ell_E = \Theta(\log n)$ groups of consecutive rounds that we call *epochs*, and which we index $1, \ldots, \ell_E$. At the beginning of each epoch i, each process u declares itself *active* if and only if its MIS set M_u does not include its own id or the id of a process in its link detector set. Only active processes will participate in the epoch.

In more detail, the epoch is divided into $\lceil \log n \rceil$ *competition phases* each of length $\ell_P = \Theta(\log n)$, followed by a single *announcement phase* of the same length. During the first competition phase, in each round, each active process broadcasts a contender message, labeled with its id, with probability $1/n$. If an active process u receives a contender message from another process, process u is knocked out: it sets its status to non-active and does no further broadcasting during this epoch. At each successive competition phase, the remaining active processes double their broadcast probabilities. In the second competition phase they broadcast with $2/n$, in the third $4/n$, and so on, up to probability $1/2$ in the final competition phase.

An active process u that makes it through all $\lceil \log n \rceil$ competition phases without being knocked out adds itself to the MIS set. It outputs 1, adds its own id to M_u, and broadcasts an MIS message labeled with its id, with probability $1/2$, in every round of the announcement phase. Every process v that receives an MIS message from a process u adds u to its MIS set M_v.

[2]This behavior might seem to constrain the adversary, as it requires reliability to be symmetric. In practice, however, this restriction has no noticeable effect. In more detail, the only way to learn about the reliability of a directed link (u, v) is for $proc(u)$ to broadcast and $proc(v)$ to receive (as broadcasters receive only their own messages). Therefore, even if the adversary could specify differing reliability on (u, v) versus (v, u), only the reliability of one of these directions could be assessed in any given round.

Correctness Proof.

As with the MIS solutions presented in [11,14], which are proved for the standard radio model where $G = G'$, we begin by covering the plane with an overlay of disks of radius $1/2$, arranged on an hexagonal lattice to minimize overlap. We index the disks: $D_1, \ldots$. (Notice, because our graph is connected, no more than n disks are required to cover all occupied portions of the plane.) Also following [11, 14], we use the notation E_i^r to reference the disk of radius r centered at disk D_i. We introduce the new notation I^r to reference the maximum number of overlay disks that can intersect a disk of radius r. The following fact concerning this overlay, also used in [11, 14], will prove useful:

FACT 4.1. *For any $c = O(1)$: $I^c = O(1)$.*

In the following, let $P_i(r) = \sum_{u \in A_i(r)} p(u, r)$, where $A_i(r)$ is the set of active processes in D_i at the beginning of the epoch that contains round r, and $p(u, r)$ is the broadcast probability of process u in round r. The following standard probability facts will prove useful:

FACT 4.2. *For any $p \leq 1/2$ it holds that $(1 - p) \geq (1/4)^p$, and for any $p > 0$ it holds that $(1 - p) < e^{-p}$.*

We continue with an important lemma that bounds the broadcast probability in the network.

LEMMA 4.3. *Fix some epoch. During every round r of this epoch, and every disk index i: $P_i(r) \leq 1$, w.h.p.*

PROOF. Fix some disk D_i. We begin by bounding the probability that D_i is the first disk to have its broadcast sum (P_i) exceed 1. For P_i to exceed 1, there must be some round r, such that r is the first round in which D_i's broadcast probability is greater than 1. Round r must be the first round of a competition phase, as these are the only rounds in which processes increase their broadcast probabilities. Furthermore, r cannot be the first round of the first competition phase, as the broadcast sum of the first phase can never exceed 1, as it has each process broadcasting with probability $1/n$. Combining these observations with fact that broadcast probabilities double between each phase, it follows: there exists a full competition phase before r, such that during every round r' of this preceding phase: $1/2 \leq P_i(r') \leq 1$. Furthermore, by assumption, r was the first round in which *any* disk exceeds a broadcast sum of 1, so we also know that for all disks $j \neq i$, during every round r' of this preceding competition phase, $P_j(r') \leq 1$.

We will now use these two observations to prove that there is a high probability that a single process in D_i broadcasts alone among nearby disks, and therefore knocks out all other active processes in D_i: reducing its broadcast probability to $1/2$ for the remainder of the epoch. To start, fix any round r' of the phase preceding r. Let p_1 be the probability of a single process broadcasting in D_i during this round. Using Fact 4.2 and our our bounds on disk broadcast sums from above, we can bound p_1 as follows: First, note that $p_1 = \sum_{u \in A_i(r')} \left(p(u, r') \prod_{v \in A_i(r'), v \neq u} (1 - p(v, r')) \right)$, which is greater than or equal to

$$\sum_{u \in A_i(r')} \left(p(u, r') \prod_{v \in A_i(r'), v \neq u} \frac{1}{4}^{p(v, r')} \right) \geq \frac{1}{2} \cdot \frac{1}{4}.$$

Next, let D_j be a disk that contains a G' neighbor of a node in D_i, and let probability p_2 be the probability that no process in D_j broadcasts in r'. By the same approach used above, we can bound $p_2 = \prod_{u \in A_j(r')} (1 - p(u, r'))$, which we know is greater than or equal to: $\prod_{u \in A_j(r')} \frac{1}{4}^{p(u, r')} \geq \frac{1}{4}$. Let $\gamma = I^{d+1/2}$ describe the total number of disks potentially containing G' neighbors of nodes in D_i (recall that $d = O(1)$ is the maximum distance at which a G' edge exists), and let p_3 be the probability that a single process in D_i broadcasts in r', and this message is received by all processes in D_i (an event we call an *uncontested* broadcast). We know: $p_3 \geq p_1 p_2^{\gamma} = \frac{1}{2} \cdot \frac{1}{4}^{(\gamma+1)} = (\frac{1}{4})^{\gamma+1.5}$. (Notice, by Fact 4.1, $\gamma = O(1)$, therefore p_3 is also constant.)

To conclude the proof, we note that the probability that we *fail* to achieve an uncontested broadcasts in D_i in all ℓ_P rounds of this phase is no more than $(1 - p_3)^{\ell_P}$. By Fact 4.2 this is less than $e^{-p_3 \ell_P}$. For sufficiently large constant factors in our definition of ℓ_P, this evaluates to $\frac{1}{n^c}$, with a sufficiently large constant c that we retain high probability even after we perform a union bound over all $O(n)$ occupied disks. The result, is that w.h.p no disk is the first to exceed 1 during this epoch. □

The following lemma leverages the observation that if the broadcast probability in the system is low (as established by Lemma 4.3), then a process about to enter the MIS will have a good probability of both knocking out its G neighbors and announcing to them its new status, during the $\Theta(\log n)$ round final competition phase and subsequent announcement phase.

LEMMA 4.4. *(Independence) For every pair of nodes u and v, $(u, v) \in E$, it is not the case that both output 1, w.h.p.*

PROOF. Fix any epoch in which neither u nor v has yet output 1. Such an epoch must exist in any execution where u and v proceed to both output 1. Assume that Lemma 4.3 holds in this epoch. Under this assumption, we will show that with high probability, either neither process joins the MIS in this epoch, or one process joins and the other outputs 0. Assume that at least one process makes it through the final competition phase (otherwise, we are done). Without loss of generality, assume this is u. Process u broadcasts in this phase with probability $p_1 = 1/2$. Let D_i be the disk containing u, and let p_2 be the probability that no process other than u in a disk intersecting $E_i^{d+1.5}$ broadcasts in r. (This is sufficient to ensure that v would receive any message sent by u, as $E_i^{d+1.5}$ contains all G' neighbors of v.) Under the assumption that Lemma 4.3 holds, we can use Fact 4.2 in a similar manner as in Lemma 4.3 to bound $p_2 \geq \frac{1}{4}^{\gamma'}$, where $\gamma' = I^{d+1.5}$. Let p_3 be the probability that v receives a message from u during this final competition phase, and is therefore knocked out and does not join the MIS. We combine p_1 and p_2 to yield $p_3 \geq \frac{1}{2} \cdot \frac{1}{4}^{\gamma'}$. (By Fact 4.1, $\gamma' = O(1)$, therefore p_3 is also constant.) We note that u fails to knock v in all ℓ_P rounds of the phase with probability no more than $(1 - p_3)^{\ell_P}$. By Fact 4.2 this is less than $e^{-p_3 \ell_P}$. For sufficiently large constant factors in our definition of ℓ_P, this evaluates to $\frac{1}{n^c}$, for any constant c. We can use the same argument to show that u fails deliver its MIS message to v during the subsequent announcement phase with a similarly low probability. For sufficiently large constant factors in our definition of ℓ_P, these probabilities are small enough to retain high probability even after we perform a union bound over all $O(n^2)$ pairs of processes and all $O(\log n)$ epochs, combined with a union bound establishing that Lemma 4.3 holds in each epoch. □

This next lemma, whose proof is deferred to the full version of this paper, leverages the observation that a process u, in each epoch, either joins the MIS or is knocked out by a G neighbor v. If the latter occurs, due to the low amount of contention provided by Lemma 4.3, v has a constant probability of knocking out *all* of its G neighbors, and then continuing uncontested to join the MIS and announce this to u. Over $\Theta(\log n)$ epochs, therefore, u will either output 1 or 0, w.h.p.

LEMMA 4.5. *(Termination) By the end of the last epoch, every process has outputted* 0 *or* 1*, w.h.p.*

THEOREM 4.6. *Using* 0*-complete link detectors, our MIS algorithm generates an execution that solves the MIS problem in* $O(\log^3 n)$ *rounds, w.h.p.*

PROOF. By definition, the algorithm requires $O(\log^3 n)$ rounds: $O(\log n)$ epochs each consisting of $O(\log n)$ phases each of length $O(\log n)$. To satisfy termination, we note that by Lemma 4.5, every process outputs 0 or 1 by the end of the algorithm, w.h.p. To satisfy independence, we note that by Lemma 4.4, no two processes who are neighbors in E both output 1, w.h.p. And finally, to satisfy maximality, we note that by the definition of the algorithm, a process does not output 0 unless it receives an MIS message from a neighbor in E, and any process that sends an MIS message, outputs 1. To achieve our final high probability we simply use a union bound to combine the two high probability results from above. □

This corollary about the density of the resulting MIS follows from the definition of independence which allows no more than a single MIS node in any disk.

COROLLARY 4.7. *Fix an execution in which the MIS algorithm solves the MIS problem. For any process* u *and distance* r*, there are no more than* I^r *MIS processes within distance* r *of* u*.*

5. CCDS ALGORITHM

In this section, we present an algorithm that solves the CCDS problem in $O(\frac{\Delta \log^2 n}{b} + \log^3 n)$ rounds, w.h.p., where b is the bound on message size in bits. This algorithm uses the MIS algorithm from Section 4 as a subroutine. As in that previous section, we assume that $b = \Omega(\log n)$. Without loss of generality, we also assume that $b = O(\Delta \log n)$ (as our algorithm never sends messages of any larger size). Finally, we assume that processes are provided a 0-complete link detector.

At a high-level, the algorithm proceeds in two phases. First, it has processes build an MIS, placing each MIS node in the CCDS. Next, it connects all MIS nodes within 3 hops in G with a path consisting of CCDS nodes. Standard techniques show that the resulting structure satisfies the definition of a CCDS. The core technical novelty of the algorithm is its efficient method for discovering and connecting nearby MIS nodes. In more detail, the simple approach would be to have each MIS node give each of its neighbors a chance to explore whether it is on a path to a nearby MIS node. This would require, however, $O(\Delta)$ explorations. This is too slow given that there are only $O(1)$ nearby MIS nodes to be discovered (a property that follows from Corollary 4.7, which bounds the density of our MIS). The algorithm presented here, by contrast, makes use of a *banned list* data structure to ensure that an MIS node gives a neighbor a chance to explore only if that neighbor is on the path to a nearby MIS node that has not yet been discovered. This reduces the required number of explorations from $O(\Delta)$ to $O(1)$. The $O(\frac{\Delta \log^2 n}{b})$ term in the time bound expression describes the time required to for an MIS node to communicate its banned list to its neighbors. For large message size (i.e., large b), this is fast, and the time to build the MIS and explore dominates the time complexity. For small message size, however, this banned list communication time dominates the time complexity and yields an algorithm no faster than the simple approach of giving each neighbor a chance to explore.

For clarity, we start by presenting and proving the correctness of the subroutines before moving on to the main algorithm.

Subroutine Descriptions.

We start by describing the two subroutines used by our CCDS algorithm. The first subroutine, *bounded-broadcast*, is used by a process to broadcast a message to its G neighbors, given a known bound on contention for this message. The second subroutine, *directed-decay*, assumes an MIS and that each MIS process has a subset of its covered neighbors wanting to send it a message. The subroutine efficiently delivers at least one message to each MIS process.

bounded-broadcast(δ, m): This subroutine, when called by a process u with message m, attempts to deliver m to u's G neighbors.

The subroutine works as follows: A process calling bounded-broadcast(δ, m) broadcasts m with probability $1/2$ for $\ell_{BB}(\delta) = \Theta(2^\delta \log n)$ consecutive rounds.

The following lemma, whose proof is deferred to the full version of this paper, states the property that the above subroutine guarantees.

LEMMA 5.1. *Assume process* u *calls bounded-broadcast*(δ, m)*, and that during every round of the subroutine, no more than* δ *other processes within distance* $d + 1$ *of* u *are running the subroutine concurrently. It follows that* u *delivers* m *to all of its* G *neighbors, w.h.p.*

directed-decay$(\langle m_1, m_2, ... \rangle)$: This subroutine assumes that the processes have already solved the MIS problem. We will use the terminology *covered processes* to describe the processes that are not in the MIS. It also assumes that all processes call the subroutine during the same round. Covered processes pass the subroutine a vector containing the messages they want to attempt to send—at most one message per neighboring MIS process. We assume each message is labeled with the id of its destination. All other processes pass an empty vector to the subroutine. The subroutine attempts to ensure that for every covered process v with a message to send to MIS neighbor u, u will receive at least one message from one of its neighbors with a message to send to u.

The subroutine works as follows: The subroutine divides time into $\lceil \log n \rceil$ *phases* of length $\ell_{DD} = \Theta(\log n)$, each associated with an exponentially increasing broadcast probability, starting with $1/n$ and ending with $1/2$. Every covered process with a message to send simulates a unique covered process for each of its messages. Initially all simulated covered processes are *active.* If a simulated covered process with a message starts a phase active, it broadcasts its message with the corresponding probability during every round of the phase. If a process has multiple simulated processes broadcast during the same round, it combines the messages. (No process has more than a constant number of neighbors in the MIS, therefore these messages are of size $O(\log n)$ bits, matching our assumption that $b = \Omega(\log n)$.) At the end of each phase, every MIS process that received a message during the phase runs bounded-broadcast$(d + 2, m)$ to send its neighbors a *stop order*, m, labeled with its id. On receiving a stop order from its message's destination, a simulated covered process sets its status to *inactive* for the remainder of the subroutine.

LEMMA 5.2. *Assume that in some round after the processes have solved the MIS problem, they run the directed-decay subroutine. It follows that by the end of the subroutine, for every covered process* v *that has a message to send to MIS neighbor* u*,* u *will receive at least one message intended for it from a neighboring covered process, w.h.p.*

Main Algorithm Description.

Having described our subroutines we continue by describing the main CCDS algorithm that makes use of these subroutines. Our algorithm begins with the processes executing the MIS algorithm from Section 4. We assume every process not in the MIS knows the ids of the MIS processes that neighbor it in G (the algorithm in Section 4 provides this information). After building the MIS, the algorithm adds every MIS process to the CCDS, then attempts to discover, and add to CCDS, a constant-length path between every pair of MIS processes that are within 3 hops in G.

At a high-level, this path-finding procedure works as follows: Each MIS process u maintains a *banned list*, initially set to contain its id and the id of the processes in its link detector set (i.e., its neighbors in G). Throughout the path-finding procedure, process u will add to its banned list B_u the MIS processes that it discovers as well as the G neighbors of these discovered processes. When a given MIS process asks its neighbors to a nominate a nearby process to explore (i.e., to see if its connected to an MIS), it uses this banned list to prevent exploration of processes that lead to already discovered MIS processes. In other words, an MIS process will ask processes to report any neighbors that are *not* already in its banned list.

We divide the search procedure into *search epochs*. During the first phase of each epoch, process u will transmit its banned list to its neighbors using bounded-broadcast. The time required to do this depends on b: this is the source of the Δ/b term in the final time complexity.

During the second phase, process u asks its reliable neighbors to use directed-decay to nominate one of their reliable neighbors for further exploration (recall, "reliable neighbor" refers to a neighbor connected by a reliable link). The restriction for such a nomination, however, is that the nominated process cannot be in the banned list. Notice, these nominations require that each process knows its set of reliable neighbors: this is where the assumption of 0-complete link detectors proves useful. By the definition of the banned list, any such nominated process must either be an MIS process that u does not know about, or be a neighbor of an MIS process that u does not know about. In both cases, we find a new MIS process within 3 hops if such an MIS process exists.

In the third phase, bounded broadcast is used to talk to the nominated process, find out if it is in the MIS, or if it is a neighbor of a process in the MIS, and then transmit the necessary new information to u to add to its banned list.

This *path finding* process, which ensures that u never explores a path that leads to an MIS process it already knows, is what provides our efficient running time (as long as the message size is large). If we instead had u explore every reliable neighbor, and in turn had these neighbors explore each of *their* neighbors, the running time would be $O(\Delta \text{polylog}(n))$, regardless of the message size.

We continue by describing more details of this path finding procedure: Each MIS process maintains a *banned list* B_u and a *delivered banned list* D_u. B_u is initially set to u's link detector set and D_u is empty. Each non-MIS process v maintains a *replica banned list* B_u^v and a *primary replica banned list* P_u^v, both initially empty, for each MIS process u that neighbors it in G. The algorithm proceeds by dividing groups of consecutive rounds into $\ell_{SE} = O(1)$ *search epochs*. Each search epoch is divided into 3 *search phases*, which we describe below.

Phase 1: Each MIS process u divides $B_u \setminus D_u$ into messages of size $b - \log n$ bits, where b is the maximum message size. It then includes its own id with each message so recipients know its source. Process u sends these messages to its non-MIS neighbors using bounded-broadcast(δ, m), with $\delta = I^{d+1} = O(1)$. Let process v be a non-MIS process that neighbors u in G. This process adds the values received from u to B_u^v. If this is the first search epoch, it also adds these values to P_u^v. At the end of the phase, u sets $D_u = B_u$. The phase is of a fixed length, long enough for the maximum number of calls to bounded-broadcast that might need to be made. As will be clear by the description of subsequent phases, the set $B_u \setminus D_u$ never contains more than Δ ids, therefore we can bound the number of calls by $O(\frac{\Delta \log n}{b})$.

Total Length: $O(\frac{\Delta \log^2 n}{b})$ rounds.

Phase 2: Let N_u be the subset of processes that neighbor MIS process u in G, where each $v \in N_u$ has a neighbor w in its link detector set such that $w \notin B_u^v$. We say w is the neighbor *nominated for* u by v. To do so, the processes run directed-decay to report their nominations to their neighbor MIS processes. With high probability, each MIS process u will hear from one process in N_u, if the set is non-empty. The fixed length of this process is number of rounds required to run directed-decay.

Total Length: $O(\log^2 n)$ rounds.

Phase 3: Let u be an MIS process that heard from a process $v \in N_u$ during the previous phase. Let w be the process nominated for u by v. During this phase, u will initiate an exploration of w. In more detail, using bounded-broadcast, with the same parameters as phase 1, u tells v that it has been selected. Next v uses bounded-broadcast with these same parameters to tell w that it wants to find out more about it. If w is in the MIS, it sends u its neighbor set. If w is not in the MIS, it chooses a neighbor x that is in the MIS, and sends to u the id of x and P_x^w (i.e., x's neighbor set). Finally, v uses bounded-broadcast to pass this information along to u, which adds the new values to its banned set, B_u. Process v and w add themselves to CCDS by outputting 1 if they have not already done so. The fixed length of this phase is set to the number of rounds required for the maximum number of calls that might need to be made to bounded-broadcast, which, as in phase 1, is bounded as $O(\frac{\Delta \log n}{b})$.

Total Length: $O(\frac{\Delta \log^2 n}{b})$ rounds.

The total running time for the MIS algorithm is $O(\log^3 n)$ rounds, and the time required to run $O(1)$ search epochs is $O(max\{\frac{\Delta \log^2 n}{b}, \log^2 n\})$. Combined this provides our final running time of $O(\frac{\Delta \log^2 n}{b} + \log^3 n)$ rounds.

We conclude with our main theorem:

THEOREM 5.3. *Using 0-complete link detectors, our CCDS algorithm generates an execution that solves the CCDS problem in* $O(\frac{\Delta \log^2 n}{b} + \log^3 n)$ *rounds, w.h.p.*

PROOF. Our CCDS algorithm first constructs an MIS using the algorithm presented in 4. It then executes ℓ_{SE} search epochs, each consisting of three phases. The MIS algorithm and the search epoch phases are of fixed length, so the running time of the algorithm follows directly from its definition.

For the remainder of the proof, assume that the MIS algorithm called by the CCDS algorithm solves the MIS problem, and that all $O(n)$ calls to bounded-broadcast and directed-decay during the search epochs satisfy the guarantees of Lemmas 5.1 and 5.2. By a union bound, these assumptions hold w.h.p.

Useful Notation. We begin by defining some useful notation: (a) We say a process v is *covered* by an MIS process u if v and u are neighbors in G. (b) we say an MIS process u has *discovered* an MIS process v ($u \neq v$) if u learned about v and v's G neighbors during phase 3 of a search epoch; (c) we say an MIS process u is *connected* to an MIS process v ($u \neq v$) if there exists a path in the CCDS of length 6 hops or less between u and v in G; and (d) we

define U_u, for MIS process u, to be the set of MIS processes (not including u) that are within 3 hops of u and that are *not* connected to u.

Useful Claims. Our goal will be to show that this set U_u becomes empty by the end of the algorithm. To aid this task, we define the following useful claims:

Claim 1: If MIS process u discovers MIS process v, then by the end of the same search epoch it adds a path of length no more than 3 hops between u and v to the CCDS.

Proof. This claim follows from the definition of the algorithm.

Claim 2: Let u be an MIS process. Assume that during phase 2 of some search epoch at least one process covered by u has a neighbor to nominate to u. It follows that u will discover a new MIS process during this epoch.

Proof. Let u' be the process covered by u assumed by the claim. Assume u' is nominating a neighbor v' for u. By definition of the algorithm, v' is not in the banned set B_u for this epoch. It follows that either v' is an MIS process that has not been discovered by u, or none of the MIS neighbors of v' have been discovered by u. At least one such u' succeeds in its call to directed decay. In either case, u discovers a MIS process during phase 3 of this epoch.

Main Proof Argument. By repeated application of Claim 2, it follows that u will keep discovering processes within 3 hops until its neighbors run out of nominations for u. There are two things to note here: first, banned sets are monotonically increasing, so once a process runs out of nominations it will never again have nominations; second, by Corollary 4.7, we know there are no more than $I^{3d} = O(1)$ MIS processes within 3 hops of u, so if we set $\ell_{SE} = I^{3d}$, we have enough search epochs to reach the point where we run out of nominations.

We will now consider the set U_u of processes that are in the MIS, are within 3 hops of u, but are still undiscovered by u after the point where its neighbors have run out of nominations. Our goal is to show that a constant length path in the CCDS between u and these processes will exist by the end of algorithm. We first note that every process $v \in U_u$ must be exactly 3 hops from u: if some v was within 2 hops, it would have been nominated by its common neighbor with u until discovered. Let u, u', v', v be a 3 hop path from u to some $v \in U_u$. Because we assume that no neighbor of u has nominations for u at this point, v' must be in the banned set B_u—otherwise, u' could nominate it. By the definition of the algorithm, it follows that that u must have previously discovered some MIS process w such that w neighbors v'. This, in turn, puts MIS process w within 2 hops of v, on the path w, v', v. As we argued above, however, any MIS processes within 2 hops will eventually discover each other. It follows that by the end of the algorithm w will discover v.

We now have a path from u to w and from w to v. By claim 1, because each path was from a discovery, each is of length 3 hops or less. We can combine these two paths to get a single path, of length 6 hops or less, from u to v.

Assuming our assumptions from the beginning of the proof hold, which occurs w.h.p., we have shown that the algorithm constructs a CCDS consisting of all MIS processes, plus a constant-length path between every pair of MIS processes within 3-hops. By the standard argument (See section 2.6.1 of [8]), this yields a valid CCDS. We are left to show that the CCDS is constant-bounded. To prove this, fix a process u. By Corollary 4.7, there are only a constant number of MIS processes within 1 hop of u in G'. We must also bound, however, the CCDS processes added by connecting nearby MIS processes with a path. Consider every pair of MIS processes (v, w) such that v discovered w and added a path of length 2 or 3 to the CCDS. If v and w are both more than distance $4d$ from u, then neither v, w, nor any process on their connecting path are within 1 hop of u. By Corollary 4.7, there are at most $x = I^{4d} = O(1)$ MIS processes within distance $4d$ of u, and therefore at most $x^2 = O(1)$ pairs of MIS processes, each contributing no more than 4 processes to the CCDS, for a total of no more than $4x^2 = O(1)$ CCDS processes within 1 hop of u, as needed. □

6. CCDS ALGORITHM FOR INCOMPLETE LINK DETECTORS

In the previous section, we described an algorithm that can solve the CCDS problem with a 0-complete link detector. In this section we consider whether we can still solve the problem with an *incomplete* link detector (i.e., a τ-complete link detector for some $\tau > 0$). In particular, we describe an algorithm that solves the problem in $O(\Delta \text{polylog}(n))$ rounds, when combined with a τ-complete link detector for any $\tau = O(1)$.[3] In the next section, we will show the gap between this algorithm and the algorithm for 0-complete detectors is inherent.

The algorithm follows the same strategy as the CCDS algorithm presented in Section 5—i.e., build a dominating structure then connect nearby dominating processes—but differs in its details. To start, notice that the MIS we get in Section 4 guarantees the maximality condition only in H. For $\tau > 0$, however, $H \setminus G$ can be non-empty: potentially resulting in a process that is far from any other MIS process in terms of G edges. Such an event would thwart attempts to connect nearby MIS process—i.e., the strategy used in Section 5—as a lack of a short path in G can prevent communication between two such processes.

To compensate for this unreliability, we instead use a procedure that sequentially executes $\tau + 1$ iterations of the MIS algorithm from Section 4. A process that outputs 1 in any of these iterations does not participate in subsequent iterations. To ensure that the maximality of each iteration is defined for H, we also have processes label their messages with their local link detector sets. A process u *receiving* a message from process v will keep the message if and only if $v \in L_u$ and $u \in L_v$ (i.e., the two processes are connected in H). This procedure provides the following useful properties:

LEMMA 6.1. *Using a τ-complete link detectors, for any $\tau = O(1)$, the above procedure requires $O(\log^3 n)$ rounds, and w.h.p. it satisfies that (a) every process either outputs 1 or has a G neighbor that outputs 1; and (b) there are no more than $O(1)$ processes that output 1 within G' range of any process.*

PROOF. The number of rounds is easily derived: we run the $O(\log^3 n)$ time procedure of Section 4, $\tau + 1 = O(1)$ times. Theorem 4.6, when combined with our above modification to the MIS algorithm that has processes discard messages from non-H neighbors, proves that a single iteration of our modified MIS algorithm satisfies maximality in H. For a process to *never* output 0 in the iterated procedure, it has to receive $\tau + 1$ such MIS messages from an H neighbor, one in each iteration. These must be sent by distinct processes, since a process that outputs 1 in some iteration does not participate in the subsequent iterations. It follows that if a process outputs 0 for the entire iterated procedure, then it has $\tau + 1$ neighbors in H that outputted 1. Since we are using a τ-complete link detector, at most τ of these neighbors can be in $H \setminus G$, implying that this process must have at least G neighbor that outputs 1: providing property (a) of our lemma.

[3]Solving the problem for larger τ remains an open problem, though our intuition is that the problem will become impossible once the τ grows larger than the bound on neighboring CCDS processes allowed by the constant-bounded condition of the CCDS problem.

To prove (b), we can apply the same argument as in Corollary 4.7. In more detail, we know, w.h.p., that each iteration of the MIS has at most one process per disk (in the disk overlay used in Section 4) output 1. Over $\tau + 1$ iterations, therefore, no more $\tau + 1 = O(1)$ processes output 1 in each disk. Finally, because there are at most a constant number of disks within G' range of any process, there are at most a constant number of processes that output 1 within G' of any process. □

Given the structure obtained by our iterated procedure, we can now build a CCDS. As in our previous algorithm, we want each process that outputs 1 in the procedure to connect to all other such processes that output 1 and are within 3 hops in G. Property (a) of Lemma 6.1 promises that this will create a connected dominating set. To satisfy the constant-bounded property of the CCDS definition, we rely on property (b). We are left, therefore, to connect nearby processes that output 1. The CCDS algorithm of Section 5, which uses a banned list approach to make this process more efficient, does not work in this setting.[4] We replace this banned list approach with something much simpler (and slower): each of the processes that output 1 dedicates time for each of its link detector neighbors to announce its id and master, using the bounded broadcast subroutines of Section 5. Call this phase 1. In phase 2, each of these processes gets another turn, this time announcing everything it learned in the previous phase. After these two phases, each process that output 1 knows about every other such process that is within 3 hops in G, and a path in H. (It might also learn about a constant number of such processes connected in H but not G.) This is sufficient to build the CCDS structure. With $O(\Delta)$ link detector neighbors, each requiring $O(\text{polylog}(n))$ rounds for each of their two phases, the total running time is: $O(\Delta\text{polylog}(n))$. We formalize this below:

THEOREM 6.2. *Using τ-complete link detectors, for any $\tau = O(1)$, the CCDS algorithm described above generates an execution that solves the CCDS problem in $O(\Delta polylog(n))$ rounds, w.h.p.*

7. LOWER BOUND

In Section 6, we described an algorithm that solved the CCDS problem in $O(\Delta\text{polylog}(n))$ rounds, given a τ-complete detector, for $\tau > 0$. In this section we show the bound to be nearly tight by proving that even with a 1-complete link detectors, constructing a CCDS requires $\Omega(\Delta)$ rounds. This bound holds regardless of message size. Notice that this represents a clear separation between the algorithms for τ-complete detectors with $\tau > 0$, and 0-complete detectors, which for sufficiently large messages can solve the CCDS problem in $O(\text{polylog}(n))$ rounds. Formally:

THEOREM 7.1. *Let $\mathcal{A}$ be a randomized CCDS algorithm such that $\mathcal{A}$ combined with a 1-complete link detector guarantees, w.h.p., to generate an execution that solves the CCDS problem in $f_1(\Delta, n)$ rounds, where Δ is the maximum degree in G and n is the network size. It follows that $f_1(\Delta, n) = \Omega(\Delta)$.*

Our proof strategy is to reduce an easily boundable game to the CCDS problem. This reduction requires a pair of transformations.

First Transformation.

The first transformation is from a CCDS algorithm to a solution to the *β-double hitting game*, which is defined as follows: There are two players, A and B, represented by the synchronous probabilistic automata $\mathcal{P}_A$ and $\mathcal{P}_B$. At the beginning of the game, an adversary chooses two target values $t_A, t_B \in [\beta]$. It then provides t_B as input to $\mathcal{P}_A$ and t_A as input to $\mathcal{P}_B$. The automata execute in rounds. In each round each automaton can output a guess from $[\beta]$. Notice, however, other than the inputs provided by the adversary at the beginning of the execution, these automata have no communication with each other. That is, their executions unfold independently. The players solve the game when either $\mathcal{P}_A$ outputs t_A or $\mathcal{P}_B$ outputs t_B. We continue with the transformation lemma:

LEMMA 7.2. *Let $\mathcal{A}$ be a CCDS algorithm such that $\mathcal{A}$, combined with a 1-complete link detector, guarantees, w.h.p., to generate an execution that solves the CCDS problem in $f_1(\Delta, n)$ rounds, where Δ is the maximum degree in G and n is the network size. There exists a pair of probabilistic automata $(\mathcal{P}_A, \mathcal{P}_B)$ that solve the β-double hitting game in $f_2(\beta, n) = f_1(\beta, n) + O(1)$ rounds, w.h.p., where β is any positive integer.*

Notice, with this transformation we shift from the world of radio network algorithms to the world of abstract games, where players are represented by probabilistic automata. We maintain n as a parameter in the running time function, however, so we can specify "w.h.p." in a consistent manner.

PROOF. Our transformation requires that we construct two player automata, $\mathcal{P}_A$ and $\mathcal{P}_B$, given a CCDS algorithm $\mathcal{A}$. Our strategy is to design our player automata to cooperatively simulate an execution of $\mathcal{A}$ running on a dual graph network of size 2β, where G consists of two cliques, each of size β, that are connected by a single link, and G' is fully connected. Call the two cliques in this network A and B. Automata $\mathcal{P}_A$ simulates processes 1 to β assigned to nodes in clique A, and $\mathcal{P}_B$ simulates processes $\beta+1$ to 2β assigned to nodes in clique B. Thus we have 2β processes total, each assigned a unique id from $[2\beta]$, as required by our network model.

In this simulation, we want the two target ids, t_A and t_B from the hitting game to correspond to the ids of the processes assigned to the endpoints of the link connecting the two cliques (which we will call the *bridge*). To do so, we must be careful about how we simulate the 1-complete link detectors used by the broadcast algorithm. In more detail, we have $\mathcal{P}_A$ give each of its simulated processes a link detector set consisting of the set $[\beta]$ and the id $t_B + \beta$, and we have $\mathcal{P}_B$ give its simulated processes the set consisting of $\{\beta + 1, ..., 2\beta\}$ and the id t_A. It follows, that each player is simulating their processes receiving a 1-complete link detector set that is compatible with a process assignment that has process t_A (in clique A) and $t_B + \beta$ (in clique B) as the endpoints of the bridge.

We have each of the two player automata simulate each round of the CCDS algorithm as follows: if two or more simulated processes broadcast, or no simulated process broadcasts, then all processes simulated by the automata receive $\perp$. Notice, here we leverage the fact that we are in the dual graph model. Assume, for example, that t_A and one other process, i, broadcast in clique A. In the classic radio network model, t_A's message would be received by process $t_B + \beta$ because i is not connected to $t_B + \beta$. In the dual graph model, however, the adversary can choose in this round to deliver a message on i's G' edge to $t_B + \beta$, causing a collision with t_A's message.

On the other hand, if only one simulated process broadcasts, then all processes simulated by that automata receive the message, and the automata makes a guess at the end of the round. The guessing works as follows: if process i simulated by $\mathcal{P}_A$ broadcasts alone in

[4] In this setting, with $\tau > 0$, it may be possible, for example, that the banned list of an MIS node includes a neighbor in $H \setminus G$. This neighbor will therefore not be nominated, even though it might be on the path to a nearby MIS process.

a simulated round, A guesses i during this round of the game, and if j simulated by $\mathcal{P}_B$ broadcasts alone, B guesses $j - \beta$.

Finally, if the simulated processes in clique A (resp. B) terminate (i.e., they have all outputted 0 or 1), then $\mathcal{P}_A$ (resp. $\mathcal{P}_B$), halts its simulation and guesses i (resp. $i - \beta$), for each simulated process i from its clique that output 1. Because players can only output one value per round, but multiple simulated processes from a clique might join the CCDS, completing this guessing might require multiple rounds. Due to the constant-bounded property of the CCDS, however, no more than $O(1)$ rounds will be needed to complete this guessing.

To conclude this proof, we must now show that this simulation strategy solves the double hitting game. We first notice that the simulations conducted by $\mathcal{P}_A$ and $\mathcal{P}_B$ will remain valid so long as there is no communication required between the cliques. By our model definition, the only scenario in which a message *must* pass between the cliques is if process t_A or $t_B + \beta$ (i.e., the processes at the endpoints of the bridge) broadcasts alone. In this case, however, the player responsible for the solo broadcaster would guess its target, solving the double hitting game.

We now consider the case where the algorithm terminates without communication between the cliques. Assume that the execution under consideration solves the CCDS problem (an event that occurs, by assumption, w.h.p.). Consider the graph H used in the definition of the CCDS problem. In our simulated network, this graph matches G: i.e., cliques A and B connected by a single bridge link. By the domination and connectivity properties of the CCDS problem, the endpoints of this bridge must be included in the CCDS. The processes corresponding to these endpoints are t_A and $t_B + \beta$. Therefore, when the respective players in the double hitting game output the guesses corresponding to their CCDS processes, they will output their targets, solving the game. □

Second Transformation.

Our next transformation is from the β-double hitting game to the *β-single hitting game*, which is defined the same as double hitting game, except there is now only one player and target. That is, the adversary chooses a value from $[\beta]$, and then the synchronous probabilistic automata $\mathcal{P}_{A,B}$ guesses one value per round until it guesses the target value. In the proof of our main theorem statement, we will show that the single hitting game is easily bounded. Note the reason we require a non-trivial transformation from the double hitting game to the single hitting game is because the exchange of input values at the beginning of the double hitting game, allows for subtle cooperative strategies that prevent us from just using one of the automata $\mathcal{P}_A$ or $\mathcal{P}_B$ as our solution to the single player variant. We detail this transformation with the following lemma:

LEMMA 7.3. *Let $(\mathcal{P}_A, \mathcal{P}_B)$ be a pair of automata that solve the β-double hitting game in $f_2(\beta, n)$ rounds, w.h.p., for any positive integer β. We can construct a probabilistic automata $\mathcal{P}_{A,B}$ that solves the β-single hitting game in $f_3(\beta, n) = f_2(2\beta, n)$ rounds, w.h.p., also for any positive integer β.*

PROOF. We are given a pair of automata $\mathcal{P}_A$ and $\mathcal{P}_B$ that solve the 2β-double hitting game in $f_2(2\beta, n)$ rounds, w.h.p. Unwinding the definition of the problem we get the following: for every pair of targets $t_A, t_B \in [2\beta]$, $\mathcal{P}_A$ and $\mathcal{P}_B$ will solve the double hitting game for these targets in no more than $f_2(2\beta, n)$ rounds, w.h.p.

Let us now unwind even more: if we run $\mathcal{P}_A$ with target t_A and input t_B, and run $\mathcal{P}_B$ with target t_B and input t_A, at least one of these two automata will output their target in $f_2(2\beta, n)$ rounds, w.h.p. To make this argument we must proceed carefully. Recall, we define w.h.p. to be $1 - \frac{1}{n^c}$ for some constant c that is sufficiently large for our needs. In this case, assume it is at least of size 2. Let p_A be the probability that $\mathcal{P}_A$ fails to output t_A in $f_2(2\beta, n)$ rounds given input t_B. And let p_B be the probability that $\mathcal{P}_B$ fails to output t_B in $f_2(2\beta, n)$ rounds given input t_A. Notice, these two probabilities are independent as the player automata execute independently once provided their respective inputs. By our assumption that at least one player succeeds with high probability, we know $p_A p_B \leq \frac{1}{n^c}$. To satisfy this inequality, at least one of these probabilities is no larger than $\frac{1}{n^{c/2}}$. The player automata with this probability therefore solves the game fast, when run with (t_A, t_B), with probability at least $1 - \frac{1}{n^{c/2}}$, which still qualifies as "w.h.p." Call this automata the "winner" for this pair of targets (if both output in the required time with the required probability, default to call automata $\mathcal{P}_A$ as the winner).

With this in mind, we can calculate a $(2\beta \times 2\beta)$-sized table, where each position (x, y) contains either A or B depending on which corresponding automata is the winner for targets $t_A = x$ and $t_B = y$. (Notice, this table is not something constructed by $\mathcal{P}_{A,B}$, it is instead something that can be calculated offline to help construct $\mathcal{P}_{A,B}$.) By a simple counting argument, there must exist either: (a) a column with at least β A's; or (b) a row with a least β B's.

For the remainder of this construction, assume we find some column y such that this column contains at least β A's. The case for a row with β B's is symmetric. Given this column y, we know that there is a subset $S_y \subset [2\beta]$ of size β, such that if we run $\mathcal{P}_A$ with target $t_A \in S_y$ and input $t_B = y$, it will output the target in $f_3(2\beta, n)$ rounds, w.h.p. (e.g., we can define S_y to be the first β rows in column y that contain A.) Let ψ be bijection from S_y to $[\beta]$.

We now define $\mathcal{P}_{A,B}$ as follows: have the automata simulate $\mathcal{P}_A$ being passed input y. If the simulated $\mathcal{P}_A$ outputs a guess x in a round, and $x \in S_y$, $\mathcal{P}_{A,B}$ outputs $\psi(x)$.

We now argue that $\mathcal{P}_{A,B}$ solves the β-single hitting game. Let $t_{A,B} \in [\beta]$ be the target chosen for $\mathcal{P}_{A,B}$ at the beginning of some execution of the single hitting game. By definition, there exists an $x \in S_y$ such that $\psi(x) = t_{A,B}$. By the definition of our table, we know $\mathcal{P}_A$ will output target $t_A = x$, given input $t_B = y$, in $f_2(2\beta, n)$ rounds, w.h.p. It follows that $\mathcal{P}_{A,B}$ simulating $\mathcal{P}_A$ with this input will therefore output $\psi(x) = t_{A,B}$ in this same time with this same high probability, as needed. □

Main Proof.

We can now pull together these pieces to prove Theorem 7.1:

PROOF (OF THEOREM 7.1). Starting with the CCDS algorithm $\mathcal{A}$ provided by the theorem statement, we apply Lemmas 7.2 and 7.3, to produce a solution to the β-single hitting game that solves the game in $f_3(\beta, n)$ rounds. We next note that the β-single hitting game, which requires a player to identify an arbitrary element from among β elements, requires $\Omega(\beta)$ rounds to solve w.h.p. (We formalize this intuitive probability fact as part of the proof for our lower bound on randomized broadcast, presented in [9].) This yields: $f_3(\beta, n) = \Omega(\beta)$. Finally, substituting the running time functions generated by our transformations, we get: $f_3(\beta, n) = f_2(2\beta, n) = f_1(2\beta, n) + O(1)$. It follows from our bound on f_3 that $f_1(2\beta, n) + O(1) = \Omega(\beta)$. There exists a graph in which $\Delta = 2\beta$, and therefore $f_1(\Delta, n) = \Omega(\Delta)$, as needed. □

8. DYNAMIC LINK DETECTORS

This paper has considered building a CCDS as a one-shot problem: processes are provided a static estimate of their reliable neighbors, formalized as a link detector set, and then attempt to build the desired structure as quickly as possible. In long-lived wireless networks, however, link status is not necessarily stable. It is possible for a link that has behaved reliably for a long period to suddenly degrade into unreliability (this could happen, for example, due to a change in the multipath environment). We can capture this setting with a dynamic definition of link detector as a service that provides a set to each process *at the beginning of every round* (a definition more aligned with the classic *failure detector* abstraction [3]). We say a dynamic link detector *stabilizes* at some round r, if in every execution its output matches the definition of the corresponding static link detector at r and never again changes in future rounds.

Given the efficiency of our CCDS solution (at least, under the assumption of large messages), a simple approach to dealing with changing link detector output is to rerun the CCDS algorithm every $\delta_{CDS} = \Omega(\frac{\Delta \log^2 n}{b} + \log^3 n)$ rounds. Call this the *continuous CCDS algorithm*. We can assume that when we rerun the algorithm, processes wait to change their outputs until the very end of the algorithm, so they can transition from the old CDS to the new CCDS all at once. We say that the continuous CCDS algorithm solves the CCDS problem by some round r, if for any round $r' \geq r$, the output solves the CCDS problem, w.h.p. The following theorem follows directly:

THEOREM 8.1. *In any execution of the continuous CCDS algorithm with a* 0*-complete dynamic link detector that stabilizes by round* r*, the algorithm solves the CCDS problem by round* $r + 2\delta_{CDS}$.

9. FUTURE WORK

This work motivates a collection of related open problems. For example, our CCDS algorithm for the 0-complete link detector setting requires large messages in order to terminate fast. It remains open whether this is fundamental, or if there exist fast solutions for the small message case. It is also interesting to consider whether there exist CCDS algorithms for non-constant τ. Finally, our τ-complete link detector abstraction is only one possible definition from many different approaches to defining this style of service. We leave the exploration of different definitions as additional future work.

In addition, it remains an interesting open question to explore the dynamic case in more detail. For example, we might want to redefine what it means to solve problems like MIS and CCDS, with respect to the current output of the link detector. We might also want to design efficient *repair* protocols that can fix breaks in the structure in a localized fashion, rather than reusing the entire protocol.

10. REFERENCES

[1] M. Abusubaih. A New Approach for Interference Measurement in 802.11 WLANs. In *Proceedings of the International Symposium on Personal Indoor and Mobile Radio Communications*, 2010.

[2] D. Aguayo, J. Bicket, S. Biswas, R. Morris, B. Chambers, and D. De Couto. MIT Roofnet. In *Proceedings of the International Conference on Mobile Computing and Networking*, 2003.

[3] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[4] A. Clementi, A. Monti, and R. Silvestri. Round robin is optimal for fault-tolerant broadcasting on wireless networks. *Journal of Parallel Distributed Computing*, 64:89–96, 2004.

[5] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. *Wireless Networks*, 11(4):419–434, 2005.

[6] D. De Couto, D. Aguayo, B. Chambers, and R. Morris. Performance of Multihop Wireless Networks: Shortest Path is Not Enough. *ACM SIGCOMM Computer Communication Review*, 33(1):83–88, 2003.

[7] K. Kim and K. Shin. On Accurate Measurement of Link Quality in Multi-Hop Wireless Mesh Networks. In *Proceedings of the Annual International Conference on Mobile Computing and Networking*, 2006.

[8] F. Kuhn. *The Price of Locality: Exploring the Complexity of Distributed Coordination Primitives*. PhD thesis, ETH Zurich, 2005.

[9] F. Kuhn, N. Lynch, and C. Newport. Brief Announcement: Hardness of Broadcasting in Wireless Networks with Unreliable Communication. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2009.

[10] F. Kuhn, N. Lynch, C. Newport, R. Oshman, and A. Richa. Broadcasting in Unreliable Radio Networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2010.

[11] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Initializing Newly Deployed Ad Hoc and Sensor Networks. In *Proceedings of the Annual International Conference on Mobile Computing and Networking*, 2004.

[12] F. Kuhn and R. Wattenhofer. Constant-Time Distributed Dominating Set Approximation. *Distributed Computing*, 17(4):303–310, 2005.

[13] F. Kuhn and A. Zollinger. Ad-Hoc Networks Beyond Unit Disk Graphs. In *Proceedings of the Workshop on the Foundations of Mobile Computing*, 2003.

[14] T. Moscibroda and R. Wattenhofer. Maximal independent sets in radio networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2005.

[15] S. Parthasarathy and R. Gandhi. Distributed Algorithms for Coloring and Domination in Wireless Ad Hoc Networks. In *Proceedings of the Conference on the Foundations of Software Technology and Theoretical Computer Science*, 2005.

[16] K. Ramachandran, I. Sheriff, E. Belding, and K. Almeroth. Routing Stability in Static Wireless Mesh Networks. *Passive and Active Network Measurement*, pages 73–82, 2007.

[17] J. Schneider. Personal Communication, ETH Zurich, Jan. 2011.

[18] K. Srinivasan, M. Kazandjieva, S. Agarwal, and P. Levis. The β-Factor: Measuring Wireless Link Burstiness. In *Proceedings of the Conference on Embedded Networked Sensor System*, 2008.

[19] P. Wan, K. Alzoubi, and O. Frieder. Distributed Construction of Connected Dominating Sets in Wireless Ad Hoc Networks. In *Proceedings of the IEEE Conference on Computer Communnications*, 2002.

[20] M. Yarvis, W. Conner, L. Krishnamurthy, J. Chhabra, B. Elliott, and A. Mainwaring. Real-World Experiences with an Interactive Ad Hoc Sensor Network. In *Proceedings of the International Conference of Parallel Processing*, 2002.