Structuring Unreliable Radio Networks*

Keren Censor-Hillel† Seth Gilbert‡ Fabian Kuhn§ Nancy Lynch¶ Calvin Newport‖

August 25, 2013

Abstract

In this paper we study the problem of building a constant-degree connected dominating set (CCDS), a network structure that can be used as a communication backbone, in the dual graph radio network model [5, 11–13]. 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 with 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.

*A preliminary version of this paper appeared in the Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC), pages 79-88, 2011.

†Technion, ckeren@cs.technion.ac.il. Shalon Fellow. Part of this work was done while the author was at MIT and supported by the Simons Postdoctoral Fellows Program.

‡National University of Singapore, seth.gilbert@comp.nus.edu.sg. This work is partially supported by the Singapore MOE AcRF-2 grant MOE2011-T2-2-042.

§University of Freiburg, Germany, kuhn@cs.uni-freiburg.de.

¶MIT CSAIL, lynch@csail.mit.edu. Supported by AFOSR Award Numbers FA9550-13-1-0042 and FA9550-08-1-0159, and NSF Award Numbers CCF-0726514, CCF-AF-0937274, 0939370-CCF, and CCF-1217506.

‖Georgetown University, cnewport@cs.georgetown.edu. Supported by Mobile Mesh Networks (Ford-MIT Alliance Agreement January 2008).

1 Introduction

In this paper, we study the problem of constructing a *constant degree connected dominating set* (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 [19, 22].

We study this problem in the *dual graph* network model, which describes static ad hoc radio networks. The dual graph model, previously studied in [5, 11–13], 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 a simple observation: in real radio network deployments, unreliable links are an unavoidable (and much cursed) feature; c.f., [2, 3, 6, 7, 9, 20, 21, 23]. To mitigate the difficulties introduced by such links, most modern ad hoc radio network deployments attempt to isolate unreliable links by using low-level link detector protocols (e.g., [3, 6, 7, 9, 23]), or sometimes even specialized hardware (e.g., [2]). We capture this strategy in our model with a new *link detector* formalism, which provides each process u, at the beginning of each execution, with a set of ids that represent an estimate of which neighbors are reliable, i.e., are connected to u by a reliable link.

Using this formalism, we explore two important questions: (1) How can we leverage link detection information, commonly assumed in practice, to build efficient solutions to the CCDS problem? (2) How reliable must these link detectors be for their information to be useful? Our answers potentially extend beyond the realm of theoretical interest and into the realm of practice, where real networks rely on link detector information and can make use of structures such as a CCDS.

1.1 Results.

In this paper, we study *τ-complete link detectors*, $0 \leq \tau \leq n$. A τ-complete link detector, for a given process u, contains the identifier 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.

As previously mentioned, practical network deployments seek to accurately filter reliable from unreliable links. That is, they try to implement a 0-complete link detector [3, 6, 7, 9, 23]. With this in mind, in Section 5 we describe a randomized algorithm that uses a 0-complete link detector to construct a CCDS. The algorithm runs for $O(\frac{\Delta \log^2 n}{b} + \log^3 n)$ rounds, with high probability, where Δ is the maximum degree in the reliable link graph, b is an upper bound on the message size, measured in bits, 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 leverages 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 even 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 respect to the $\tau = 0$ case, but also defines a separation with respect to the classical radio network model, which assumes only reliable links. Previous work has identified a CCDS algorithm for the classical model that uses no topology knowledge and uses only $O(\text{polylog}(n))$ rounds [1].

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, and, in Section 9, how to modify our MIS algorithm to work in the classical radio model, yielding a solution that is comparable with the best known MIS solutions.

1.2 Discussion.

A 0-complete link detector seems like a quite powerful tool. It is worth pausing to discuss what capabilities such a link detector does and does not provide.

First, we observe that a 0-complete link detector gives us signficant power to do things that are impossible otherwise. Since it provides each process with the set of neighbors that are reliable, it enables an algorithm to build structures that solely rely on reliable neighbors and reliable links. For example, using a link detector, we can build a CCDS that is connected via reliable links. Without a link detector, this would be impossible: any link that we believe to be reliable could later turn out to be unreliable.

Second, we consider the algorithmic challenges of using a 0-complete link detector. It is important to note that assuming a 0-complete link detector is very different from assuming a network with only reliable edges: the link detector provides knowledge about the network topology; however, one still must grapple with the uncertainty caused by the presence of unreliable edges.

Consider the following example. Assume we have n processes, each with one neighbor (which we refer to as its partner); each process wants to send a message to its partner. In a system with only reliable links, it is simple to accomplish this task. For example, each process i can broadcast its message with probability $1/2$ until it succeeds. Even if the partner process of i also wants to send a message to i, process i succeeds with probability $1/4$ in each round. Now, assume that there are unreliable links as well: each process has an unreliable link to some (unknown) subset of (other) processes in the system. In this case, the broadcast problem is significantly harder, even with a 0-complete link detector. Each process knows that it has a reliable link to its partner. However, there is also an indeterminate amount of contention caused by other unreliable neighbors. A process can choose to broadcast with probability $1/n$, and thus avoid all contention—but now the broadcast takes $\Theta(n)$ time to complete. Alternatively, a process can choose to broadcast with probability $1/2$, in which case the unreliable links may cause too much contention, resulting in even slower message delivery. Thus, even with a 0-complete link detector, the existence of unreliable links leads to significant complications.

1.3 Related Work.

The dual graph model was introduced in [5], where it was called the dynamic fault model, and then later studied in [11–13] 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 [16], 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, has been extensively studied in general graph models (see [15] for a good overview). In the context of radio networks with only reliable links (i.e., what we call the *classical radio network model*), [22] describes an $O(n)$ time CCDS algorithm, and [19] 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, where Δ is the maximum degree of the network; even then the broadcast schedules constructed in [19] could be thwarted by unreliable links causing collisions. As with our paper, both [22] and [19] assume synchronous starts (i.e., processes start during the same round). Concurrent work has identified an $O(\text{polylog}(n))$-time CCDS solution in the classical radio network model *without synchronous starts* [1].

The MIS problem, which we use as a step in our construction of a CCDS, was studied in the classical radio network model without synchronous starts in [14], which provides a $O(\log^6 n)$-time solution. This

was later improved in [18] to $O(\log^2 n)$. The MIS algorithm presented in the main body of this paper uses $O(\log^3 n)$ rounds, and it assumes synchronous starts and a 0-complete link detector. In Section 9, however, we describe a minor variation to the algorithm that has the same running time in the classical radio network model, without synchronous starts or any topology information. This algorithm is a factor of $O(\log n)$ slower than the result of [18], 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 $E \subseteq E'$. Informally, graph G describes the reliable edges, while graph G' describes all possible edges, both reliable and unreliable. 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 the presentation of time complexity bounds later in the paper, we assume that $\Delta = \omega(\log n)$ (for smaller values of Δ, even simple solutions work well).

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 that all close nodes are neighbors in G, i.e., for all $u, v \in V$ where $dist(u, v) \leq 1$, $(u, v) \in E$. We also assume that all neighbors in G' are not too far apart, i.e., there exists a *constant* distance $d \geq 1$, such that 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') begins with the adversary assigning a different process to each node in V. The processes have no advance knowledge of this assignment. Specifying an execution in this manner formally captures the fact that a process has no knowledge of its position in the network topology.

We 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 the process assigned to node v. Throughout this paper, we refer to processes in two ways: (1) we use the notation *process* u, for some $u \in V$, to refer to the process assigned to node u in the execution in question; (2) we 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 processes starting in the first round. At the beginning of each round, r, each process v decides whether or not to send a message. Next, the adversary chooses a *reach set* of edges that consists of E and some subset, potentially empty, of edges in $E' \setminus E$. This set describes the links that will behave reliably in this round. Let $B_{v,r}$ be the set of nodes whose corresponding processes 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 process v broadcasts in r then it receives only its own message. If process v does not broadcast and $|B_{v,r}| = 1$, then it receives the message sent by the single broadcaster in $B_{v,r}$. Otherwise, if $|B_{v,r}| = 0$ or $|B_{v,r}| > 1$, and v does not broadcast, then process v receives $\perp$; that is, we assume no collision detection.

In the following we sometimes use the notation $[i]$, for positive integer i, to indicate the set $\{1, ..., i\}$. 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 error probability that is sufficiently small.

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* formalism to capture the functionality of these services.

In more detail, this formalism provides each process u a link detector set $L_u \subseteq [n]$. The set L_u is provided to process u before the execution begins. The set L_u 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 $W_u \subseteq \{id(w) : w \in N_{G'}(u) \setminus N_G(u)\}$, 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 now 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: $id(u) \in L_v$ and $id(v) \in L_u$ (that is, u and v are in each others' link detector sets). Recall, link detectors, as defined above, are not required to be symmetric (in real networks, link quality is not necessarily symmetric). The graph H is the subset of links that happen to be categorized as reliable by the detector at both endpoints. Notice, for a τ-complete link detector, for any value of τ, G is a subgraph of H and H is a subgraph of G'. For $\tau = 0$, $H = G$.

Maximal Independent Set. A maximal independent set (MIS) algorithm requires every process to eventually output 0 or 1, where 1 indicates that the process is in the MIS, and 0 indicates that 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 requires every process to eventually output 0 or 1, where 1 indicates that the process is in the CCDS, and 0 indicates that it is not. The algorithm has an associated constant denoted by δ (to be explained below), independent of the size of the input graph. We say an execution of a CCDS algorithm *solves 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]** for every process u, no more than δ neighbors of u in G' output 1. Notice, by requiring the connectivity property to hold in the stricter sub-graph H, and the constant-bounded property to hold in the more general G', we are making the definition more difficult for the algorithm to match, resulting in a structure that will be more useful in practice.

Note that the definitions depend on all three graphs G, G', and H and they therefore differ from the standard definition of an MIS or a CCDS for graph G. In particular, maximality in the MIS definition and connectivity in the CCDS definition are slightly weaker than in the standard definition where they would both be defined w.r.t. graph the edge set E (instead of E_H). Note however that the model assumptions force

Algorithm 1 Maximal Independent Set Algorithm (for node u)

$M_u \leftarrow \emptyset$
for $e \in \{1, \ldots, \ell_E\}$ **do**
  $active \leftarrow (M_u = \emptyset)$
  for $c \in \{1, \ldots, \lceil \log n \rceil\}$ **do**
    for $r \in \{1, \ldots, \ell_P\}$ **do**
      if $active$ **then**
        transmit with probability $2^{c-1}/n$
        if message from G-neighbor received **then** $active \leftarrow$ **false end if**
      end if
    end for
  end for
  if $active$ **then** $M_u \leftarrow \{u\}$ **end if**
  for $r \in \{1, \ldots, \ell_P\}$ **do**
    if $active$ **then**
      transmit with probability $1/2$
    else
      if message from G-neighbor v received **then** $M_u \leftarrow M_u \cup \{v\}$ **end if**
    end if
  end for
end for
Node u outputs 1 (joins the MIS) if and only if $M_u = \{u\}$

us to use these slightly weaker conditions as an algorithm cannot distinguish between an edge in $e \in E$ and an edge $e' \in E_H$ if the adversary decides to always include e' in the reach set (or to include e' in the reach set for long enough). On the other hand, if $\tau = 0$, then $H = G$, and the definitions become closer to standard (they still differ in that there are unreliable links in the network, and the constant bounded property will therefore still be defined with respect to all types of links).

4 Maximal Independent Set Algorithm

In this section, we present an algorithm that solves the MIS problem in $O(\log^3 n)$ rounds, w.h.p. We assume that the processes have access to a 0-complete link detector and that the message size is $b = \Omega(\log n)$ 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 knowledge about the topology, it does not eliminate the negative impact of unreliable edges.

4.1 Algorithm Description

We now give an overview of the algorithm. Pseudo-code for the algorithm is given in Algorithm 1. Throughout an execution, each process u discards messages received from processes not in u's link detector set. Therefore, in the following description, when we say that a process *receives a message*, we mean to imply that it is a message sent from a neighbor in E. By using the link detector in this manner, we ensure that every node either joins the MIS or has a reliable neighbor (guaranteed by 0-complete link detector) in the MIS. (Without the link detector, the protocol would still output an independent set; however the maximality would not be satisfied.)

Each process u maintains a set M_u of MIS process ids in u's 1-hop neighborhood (initially empty). The execution is divided into *epochs*, each one consisting of *phases* of several rounds. The exact description is as follows. There are $\ell_E = \Theta(\log n)$ epochs, indexed by $1, \ldots, \ell_E$. 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 reliable neighbor. Only active processes participate in the epoch.

Each 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, then process u is knocked out: it sets its status to inactive and does no further broadcasting during this epoch. (At the beginning of the next epoch, it again decides whether to be active, as described above.) 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 by outputting 1, adding its own id to M_u, and broadcasting 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 in the announcement phase adds u to its MIS set M_v.

4.2 Correctness Proof

As with the MIS solutions presented in [14, 18], which are analyzed in the standard radio model where $G = G'$, we begin by covering the plane with an overlay of disks of radius $1/2$, arranged on a hexagonal lattice to minimize overlap. We index the occupied disks: $D_1, D_2, \ldots$. (Notice, because our graph is connected, no more than n disks are required to cover all occupied portions of the plane.) Also following [14, 18], we use the notation E_i^h to reference a disk of radius h centered at disk D_i. We introduce the new notation I^h to reference the maximum number of overlay disks that can intersect a disk of radius h. The following fact concerning this overlay, also used in [14, 18], 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. In other words, p describes the broadcast probability of a single process, while P describes the sum of the broadcast probabilities of all processes in a given disk.

Before continuing to our proof, we conclude this discussion of preliminaries by defining a pair of standard probability facts:

Fact 4.2.
1. *For any $p \leq 1/2$: $(1 - p) \geq (1/4)^p$.*
2. *For any $p > 0$: $(1 - p) < e^{-p}$.*

We now begin our argument with an important lemma that bounds the broadcast probability in a disk.

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. We need to bound the probability that some disk ends up with a broadcast probability greater than 1. Notice, if any disks end up with this large probability, then there must be some disk that is *first* to breach this threshold. With this in mind, fix some disk D_i. We will 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 the fact that broadcast probabilities double in each competition 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, thus 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 process 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 processes 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. This fact ensures that the output set is independent.

Lemma 4.4. *(Independence) With high probability, there are no G-neighbors u and v such that both u and v output* 1.

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 the property in Lemma 4.3 holds in this epoch (this happens with high probability). 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 process is u. In each round of this phase, process u broadcasts 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 to 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 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 level of contention ensured 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 its new status to u. Over $\Theta(\log n)$ epochs, therefore, u will either output 1 or 0, with high probability.

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

Proof. Assume Lemma 4.3 holds in all $\ell_E = \Theta(\log n)$ epochs. Fix some process u that starts some epoch active. During this epoch, either u is knocked out or it survives and joins the MIS. If it joins the MIS we are done, so we will consider the case where it is knocked out. Let v be the process that knocks out u. Because v knocked out u, we know it broadcasts at least once during this epoch. We will now bound the probability that this broadcast is received by *all* processes in v's 1-hop neighborhood in G. Let D_i be the disk containing v. Let p_1 be the probability that v broadcasts alone in $E_i^{d+1.5}$ during the round in which it broadcasts. (Notice this range is large enough to include v's 1-hop neighbors in G and these neighbors' 1-hop neighbors in G', which are potential sources of contention.) Using Fact 4.2 in a similar manner to the previous lemma, we can bound $p_1 \geq \frac{1}{4}^{\gamma'}$, where $\gamma' = I^{d+1.5}$.

Assume that this event occurs, i.e., v is the only process broadcasting $E_i^{d+1.5}$. In this case, its message must be received by every process in its 1-hop neighborhood in G. With these processes now knocked out, v will continue on, uncontested, to join the MIS in this epoch.

Now let us return to u. Using the same argument applied in Lemma 4.4 to show that some process u would have its message received by a G-neighbor during the final competition phase, w.h.p., we observe that the process u in this proof will receive an announcement message from v during the announcement phase, w.h.p. Let us call this high probability p_2. On receiving this message, u outputs 0.

We have shown, therefore, that in each epoch, u outputs 0 or 1 with probability at least $p_3 = p_1 p_2 = O(1)$. Therefore, the probability that u fails to output 0 or 1 at the end of all $\ell_E = \Theta(\log n)$ epochs is no more than $(1 - p_3)^{\ell_E}$. By Fact 4.2 this is less than $e^{-p_3 \ell_E}$. For sufficiently large constant factors in our definition of ℓ_E, 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 $O(n)$ processes, and a union bound with the probability that Lemma 4.3 holds in all $\Theta(\log n)$ epochs. □

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

Proof. By definition, the algorithm takes $O(\log^3 n)$ 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 process 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 Constant-Bounded Connected Dominating Set 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. (Recall that Δ is the maximum number of G-neighbors that a process has.) This algorithm uses the MIS algorithm from Section 4 as a subroutine. As in that previous section, we assume that $b = \Omega(\log n)$ and specifically that $b \geq c \log n$ for an appropriate constant c. 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 processes within 3 hops in G with a path consisting of CCDS processes. 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 processes. In more detail, the simple approach would be to have each MIS process give each of its neighbors a chance to explore whether it is on a path to a nearby MIS process. This would require, however, $O(\Delta)$ explorations. This is too slow given that there are only $O(1)$ nearby MIS processes 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 process to ensure that an MIS process only gives a neighbor a chance to explore if that neighbor is on the path to a nearby MIS process that has not yet been discovered. This reduces the number of explorations from $O(\Delta)$ to $O(1)$. The $O(\frac{\Delta \log^2 n}{b})$ term in the running time describes the time needed for an MIS process 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 neighboring paths 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 two subroutines before moving on to the main algorithm. The first subroutine, bounded-broadcast, is used to broadcast a message to a process's 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 non-MIS neighbors wanting to send it a message. The subroutine delivers at least one message to each MIS process.

5.1 The Bounded-Broadcast Subroutine

The subroutine ***bounded-broadcast***(δ, m), when called by a process u with message m, attempts to deliver m to u's G neighbors. The parameter δ is a bound on the number of other nearby processes (within distance $d + 1$) that are also trying to perform a bounded-broadcast.

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

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.*

Proof. Let p_1 be the probability that u broadcasts alone among all processes within distance $d+1$ during some round of the subroutine. (Notice, if this event occurs, u succeeds in delivering its message to its G-neighbors.) We can bound this probability as: $p_1 \geq 1/\delta(1-1/\delta)^\delta \geq 1/4\delta$. Using Fact 4.2, we note that the probability that u fails to succeed in broadcasting alone in all $\ell_{BB}(\delta)$ rounds of the subroutine with probability no greater than $(1-p_1)^{\ell_{BB}(\delta)} \leq e^{-p_1 \ell_{BB}(\delta)}$. For any constant c, we can use sufficiently large constant factors in our definition of $\ell_{BB}(\delta)$ to ensure this probability is no greater than $\frac{1}{n^c}$, as needed. □

5.2 The Directed-Decay Subroutine

The ***directed-decay***$(\langle m_1, m_2, ... \rangle)$ 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—each covered processes attempts to send at most one message to each 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. For a given MIS process u, we use the notation *covered set of* u to refer to the set of neighboring processes with a message to send to u. The subroutine attempts to ensure that every MIS node u receives at least one message from its covered set, if it is not empty.

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 $1/2$. If a covered process has multiple messages to send, it simulates a unique covered process for each message. 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. (Since no process has more than a constant number of neighbors in the MIS, 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(δ_{DD}, m), where $\delta_{DD} = I^{d+1}$, to send its neighbors a *stop* order, m, labeled with its id. On receiving a stop order from its message's destination, the (perhaps simulated) covered process that originally sent the message 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. By the end of the subroutine, with high probability, every MIS node u receives at least one message from its covered set, if it is not empty.*

Proof. To simplify notation, round n up to the nearest power of 2, and number the phases of directed decay from 1 to $\log n$. For concision, we will drop the term "simulated" when discussing the behavior of simulated processes, and treat each like an independent process.

In each Phase i, active processes broadcast with probability $\frac{1}{2^{\log n - i + 1}}$. We say a given MIS process's covered set is *completed* if all processes in the set have received a stop order from the MIS process.

We proceed by induction on the phase number. The inductive hypothesis is as follows: by the first round of Phase i, all MIS processes with covered sets larger than $2^{\log n - i + 1}$ have completed their set. The base case ($i = 1$) follows trivially: no process has more than n neighbors in G.

To prove that the inductive step holds, assume the hypothesis holds for some Phase $i \in [1, \log n - 1]$: we will prove that it holds for Phase $i+1$, w.h.p. Let u be an MIS process that has not completed its covered set by the end of Phase i, and has a covered set S, s.t., $2^{\log n - i} \leq |S| \leq 2^{\log n - i + 1}$. During this phase, processes in S broadcast with probability $p_1 = \frac{1}{2^{\log n - i}}$. Let p_2 be the probability that a single process in S broadcasts in some round of this phase. Using the same techniques as in Lemma 4.3, combined with our bounds on the size of S, we can bound this probability as: $p_2 \geq \sum_{v \in S} p_1 \prod_{w \in S, w \neq v} (1 - p_1) \geq \frac{1}{8}$.

Assume that some process $v \in S$ broadcasts alone. We now calculate the probability that no other G'-neighbor of u broadcasts during this round—therefore allowing u to receive v's message. Only MIS processes within distance $d + 1$ of u can have covered subsets that include processes within range of u (recall that $d = O(1)$ is the maximum distance at which a G' edge exists). By Corollary 4.7, there exists at most $I^{d+1} = O(1)$ MIS processes within distance $d + 1$ of u. By our inductive hypothesis, the sum of broadcast probabilities in each of the covered subsets associated with these MIS processes is also bounded by a constant.

Therefore, for each such subset, the probability that no process broadcasts in a given round is also constant. Combining, we have a constant number of potentially interfering covered subsets, each with a constant probability of not interfering. The probability of no interference is therefore constant. Let p_3 denote this constant probability of no interference, and p_4 denote the probability that u receives a message. We can bound $p_4 \geq p_2 p_3 = O(1)$. Using Fact 4.2, we note that the probability that u fails to receive a message in all ℓ_{DD} rounds of this phase is no more than $(1 - p_4)^{\ell_{DD}} \leq e^{-p_4 \ell_{DD}}$. For any constant c, we can use sufficiently large constant factors in our definition of ℓ_{DD} to ensure that this probability evaluates to no more than $\frac{1}{n^c}$,

At this stage, u has received a message from its covered set. The inductive hypothesis, however, requires that u also now successfully delivers a stop order to its covered set, using bounded-broadcast(δ_{DD}). By Lemma 5.1, u succeeds in delivering its stop order so long as no more than δ_{DD} other processes within G' range of u's covered set are executing the subroutine concurrently. As argued above, only MIS processes within distance $d + 1$ of u are potentially within interference range of u's covered set. For $\delta_{DD} \geq I^{d+1} = O(1)$, bounded-broadcast($\delta_{DD}$) successfully delivers the start order w.h.p. By a series of $O(n)$ union bounds, we show that every MIS process completes its covered set during the appropriate phase of the subroutine, w.h.p.

The induction establishes that at the beginning of the final phase (Phase $\log n$), all covered sets of size greater than 2 have completed. For the small covered sets that remain in the final phase, the same argument used above for the earlier stages of the induction holds. For each such covered set, in every round, there is a constant probability of delivering a message to their destination. Over all ℓ_{DD} rounds of this phase, this event occurs with high probability. Using a union bound, we combine this result with the high probability that bounded-broadcast(δ_{DD}), called at the end of this final phase, successfully delivers the stop orders. □

5.3 The Main CCDS Algorithm

Having described our subroutines, we continue by describing the main CCDS algorithm (which relies on these subroutines). As already described, the algorithm consists of two phases: first, it builds an MIS, and then it connects nearby MIS processes.

Our algorithm begins with the processes executing the MIS algorithm from Section 4. We assume every process not in the MIS knows which of its G-neighbors are in the MIS. (Notice that 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* B_u, initially set to contain its own 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 adds to its banned list B_u the MIS processes that it discovers as well as the G-neighbors of these discovered MIS processes. When a given MIS process asks its neighbors to nominate a nearby process to explore (i.e., to see if it is connected to an MIS process or the neighbor of an MIS process), it uses this banned list to prevent exploration of processes that lead to already discovered MIS processes. In other words, an MIS process asks 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, an MIS process u transmits 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, MIS 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, first in order to find out if it is in the MIS or if it is a neighbor of a process in the MIS, and then second to 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 the 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, there are never more than Δ new ids added to B_u in a search epoch, and hence 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 hears from one process in N_u, if the set is non-empty. The fixed length of this process is the 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 initiates 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 to run $O(1)$ search epochs is bounded by $O(\frac{\Delta \log^2 n}{b} + \log^2 n)$. Combined this provides our final running time of $O(\frac{\Delta \log^2 n}{b} + \log^3 n)$ rounds.

Following is our main theorem, which shows that indeed we obtain a CCDS.

Theorem 5.3. *Using* 0*-complete link detectors, our CCDS algorithm 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 Section 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 $O(1)$ 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 that it nominates for u. It follows that u discovers 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 exists 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 eventually discover each other. It follows that by the end of the algorithm w discovers 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.

We have shown that, with high probability, the algorithm constructs a dominating set consisting of all MIS processes, plus a constant-length path between every pair of MIS processes within 3 hops. By a standard argument (see Section 2.6.1 of [10]), this yields a connected dominating set. We are left to show that the dominating set 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 Upper Bound 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 answer the natural follow-up question of whether we can still solve the problem with an *incomplete* link detector (i.e., a τ-complete link detector for some $\tau > 0$). In more detail, 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 $\tau = \Theta(1)$. Notice, this represents a clear separation from the algorithm for 0-complete detectors, which can solve the CCDS problem in $O(\text{polylog}(n))$ rounds when the message size is $b = \Omega(\log n)$. In the next section, we will prove that this gap is fundamental by proving a lower bound of $\Omega(\Delta)$ rounds for the problem for any $\tau > 0$.

The algorithm follows the same basic 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. A main difficulty we face now that $\tau > 1$ is that a dominating structure established in H might not provide a sufficient building block for a connected structure. To understand why, recall that in the setting with reliable links (i.e., where $\tau = 0$), it was sufficient for each dominator to find and connect to every dominator within 3 hops. With $\tau > 1$, this strategy no longer guarantees a connected structure. Imagine, for example, a line of nodes u_1, u_2, u_3, u_4, u_5 in G (i.e., each (u_i, u_{i+1}) is an edge in G) that is fully connected in G'. Imagine that u_3 incorrectly contains u_1 in its link detector (which is valid behavior even for $\tau = 1$). Depending on the dominating set algorithm, it is possible that u_1 and u_5 end up in the dominating set, with u_3 believing it is dominated by u_1 (even though, in reality, they are only connected in G'). Now, when the time comes for

u_1 and u_5 to search for other dominators within 3 hops, imagine the G' edges are excluded from the reach set. Neither of these nodes will find each other, and therefore, no nodes between them will be added to the CDS. The resulting structure is not connected.

Our solution to this problem is to deploy a more involved dominating set structure that does not just guarantee every node is covered by a neighbor in its link detector, but that every node is covered by at least one G-neighbor. In our proof, we argue that this coverage property is sufficient to guarantee that a 3-hop search will yield a connected structure. To achieve this property we run an MIS algorithm $\tau + 1$ times. In each instance, nodes disregard nodes not in their link detector sets. Also, a node that joins the MIS in one instance will not participate in future instances. This guarantees that every node either joins the MIS, or ends up dominated by a G-neighbor, in at least one of the $\tau + 1$ instances.

Algorithm Description. The CCDS algorithm proceeds in two stages. The first stage builds a dominating set by running $\tau + 1$ consecutive iterations of an MIS algorithm. The second stage attempts to connect all nearby dominating set nodes with a constant-length path. The dominating nodes and the nodes on the connecting paths all join the CCDS.

We begin with the first phase. We cannot use the MIS solution from Section 5 as it requires $\tau = 0$. Fortunately, because our time complexity target is less ambitious in this setting, we can make due with a slower approach. In particular, we use the SAP local broadcast algorithm of [8] to simulate the broadcast version of the synchronous message passing model (i.e., the version of the message passing model where you must send the same message to all your neighbors in each round). That is, we run one instance of the local broadcast protocol for each simulated round of the message passing model, with nodes broadcasting the message (if any) they want to send in the simulated round. A node receiving a message from the local broadcast protocol will simulate receiving it in the message passing model if and only if it comes from a neighbor in its link detector set.

On top of this simulated message passing model, we run $\tau + 1$ consecutive iterations of Luby's MIS algorithm [17]. It is important to note that this algorithm works in our broadcast variant of the message passing model, as it always has a node broadcast the same message to all its neighbors. To keep these iterations non-overlapping we assigned a fixed group of t_{MIS} rounds to each, where $t_{MIS} = O(\log n)$ is the high probability termination time of Luby's algorithm. A node participates in a given iteration if and only if it did not join the MIS in a previous iteration. Every node that joins the MIS in some iteration also joins the dominating set that is used as a basis for the next phase of the algorithm.

In the second phase of our CCDS algorithm, we must now search for short paths between nearby CCDS nodes, and add the nodes on these paths to our structure. Our goal is to find such a path between all dominators within 3 hops in G. For this second phase, we abandon the message passing model simulation, as it is too slow for our search strategy (which requires dominators to communicate, sequentially, with each of their $O(\Delta)$ neighbors). Instead, we make use of the *bounded broadcast* subroutine of Section 5, which allows a node to successfully broadcast a message to its neighbors in $O(\log n)$ rounds, with high probability, so long as no more than $O(1)$ other nodes within a constant number of hops are also running the subroutine.

We enforce this contention property with a token-based approach. That is, each node in the dominating set is given a token. The token must be explicitly passed from node to node, such that only one node owns a token at any given time. The token can never travel more than 3 hops in H. And finally, a node cannot run bounded broadcast unless it possesses the token. If a node is required to run a bounded broadcast on behalf of multiple tokens at the same time, it can combine the messages into one larger message. Because a node can never possess more than a constant number of tokens, this does not effect the asymptotic message size bound. It is straightforward to see that we can call bounded broadcast with sufficiently large constant factor to ensure it works with high probability under these constraints.

We divide the search phase into several steps. In the following, let M be the set of nodes in the domi-

nating set. Assume we divide rounds into groups of size $t_{BB} = O(\log n)$, where t_{BB} is the time complexity of bounded broadcast. Call each such group a *bounded broadcast round*. And finally, assume that nodes always ignore neighbors not in their link detector set.

The first step has each $u \in M$ pass its token to each of its neighbors, one by one. When each neighbor v receives the token, it calls bounded broadcast to announce its id and the fact it neighbors u. It then returns the token. This requires 3 bounded broadcast rounds for each neighbor. If u does not receive its token back from some v, it recognizes that the link to v is unreliable. In this case, it acts as if v is not in its link detector set (it ignores v for the remainder of the execution) and it respawns the token so it can continue. All nodes in M wait $3(\Delta + \tau)$ bounded broadcast rounds before proceeding to the next step—to ensure all nodes have finished this first step. Notice, at the end of the first step, the neighbors of each $u \in M$ have collective knowledge of all nodes in M within 3 hops of u in G.

In the second step, we add short paths between nodes within 2 hops in M. Each $u \in M$ has a set D_u, initially empty, to record the nearby nodes in M to which it has already added a path in the CCDS. As before, u will pass the token to each neighbor v, along with a copy of D_u (which will never contain more than a constant number of ids). On receiving the token, if v directly neighbors any $w \neq u$ in M (which it knows because it would have received the token during the first step), and $w \notin D_u$, node v joins the CCDS and reports w to u when it returns the token. Node u will then add w to D_u. As in step 1, if it fails to receive a reply from a neighbor it commits to ignoring that neighbor moving forward and respawns the token. This step again requires $O(\Delta)$ bounded broadcast rounds to complete. (Starting with this step, it is not important that all nodes are synchronized, so a node $u \in M$ can move on to step 3 as soon as it finishes with all of its neighbors in step 2.)

In the third step, we add short paths between nodes that are 3 hops apart in M. This step requires more care. We begin by having each $u \in M$ once again pass the token, along with the current value of D_u, to each neighbor, one by one. When neighbor v receives the token, it responds by reporting $M_v = \{w \in M \setminus D_u \mid w$ is 2 hops from $v\}$. For each $w \in M_v$, it also includes a set $Q_v(w)$ of up to $2\tau+1$ neighbors of v that reported in step 1 that they neighbored w. If v knows of less than $2\tau + 1$ such nodes, it returns them all, if it knows more, it returns just the first $2\tau + 1$ by some arbitrary ordering. Notice, this message still contains only a constant number of ids. As before, if v fails to reply, u ignores v and respawns the token.

After u has gone through all of its neighbors, it now knows of all 3-hop neighbors from M that it must try to connect to. It also has some information about potential paths to these neighbors. We have node u deal with each such w separately. Fix one such w. Node u starts by simulating a reliable communication channel with w. To send a message to w, u will attach the message to a token, and pass the token and message hop-by-hop on some path it learned about to w. When w receives the message, it replies by bouncing the token back on the same path. If node u does not get a response back after 6 bounded broadcast rounds, or it gets back a failure notice from a hop farther along the path that failed to receive a reply, it knows there is an unreliable link somewhere on the path and moves on the next path—using a selection criteria we define below.

In particular, consider the set S of u's neighbors that claim to be on a 3-hop path to w. Node u starts with the first $v \in S$, by some arbitrary ordering. It will then try a path for each $Q_v(w)$, until one works, or all fail. If at some point it fails to hear back from v, or if v reports a failure for all next hops in $Q_v(w)$, u will move on to the next node in S. We note that u's simulation will eventually stabilize. Let u, v, v', w be the 3-hop path in G we assume exists. If the simulation keeps failing, u will eventually move on to v. If v knows of $\leq 2\tau + 1$, 2-hop paths to w, then $v' \in Q_v(w)$, and if the simulation keeps failing, u will eventually select u, v, v', w. Otherwise, if v knows of more such paths, it is possible that $v' \notin Q_v(w)$. In this case, however, we note that there must be *some* $v'' \in Q_v(w)$ that is connected to both v and w in G. To see why, note that $|Q_v(w)| = 2\tau + 1$ in this case, and at most τ of these nodes can fail to be connected to v in G, and up to at most τ can fail to be connected to w in G.

Our simulated channel from u to w, therefore, can have up to $O(\Delta)$ bounded broadcast rounds of delay

beyond its normal constant round delay for each round trip. We can now discuss how u and w coordinate using this reliable channel. To find a path between u and w to add to the structure, we have u propose to w one such path it knows about. Then u uses its token to request the first hop join while w does the same for the second hop. Nodes u and w then report back to each other whether they were successful in their requests. If both are successful, then they are done—a path between them has been added to the structure. If at least one of the nodes failed to hear back, u adds the non-replying node to the list of nodes being ignored, and proposes a new path. Notice, both u and w can each have at most τ neighbors that potentially join but fail to report this back due to a faulty link. Therefore, a path will be founded between the two nodes with at most 2τ extra nodes joining the CCDS. This whole step still takes only $O(\Delta)$ bounded broadcast rounds.

Theorem 6.1. *Using τ-complete link detectors, for any $\tau = O(1)$, the CCDS algorithm described above solves the CCDS problem in $O(\Delta \log^2 n)$ rounds, w.h.p.*

Proof. We begin with the time complexity. Each simulated round of the message passing model takes $t_{SAP} = O(\Delta' \log n)$ rounds. We run $\tau + 1 = O(1)$ iterations of an MIS algorithm that each last $t_{MIS} = O(\log n)$ simulated rounds. During the second phase, as specified in the algorithm description, each node in M requires no more than $O(\Delta+\tau) = O(\Delta)$ bounded broadcast rounds. At this point, the total running time is $O(\Delta' \log^2 n + \Delta \log n) = O(\Delta' \log^2 n)$ rounds. We conclude the time complexity analysis by noting that in our geographic model, $\Delta' = \Theta(\Delta)$. To see why, notice that for any node u, the region containing potential G' neighbors can be covered by $O(d^2) = O(1)$ disks of diameter 1 (i.e., as argued in our analysis of the MIS algorithm from Section 5). Each disk is fully connected in G, and therefore has at most Δ nodes. It follows that there are no more than $O(\Delta)$ nodes in this region.

We next consider the properties of the structure we build. To do so, we note that our key high probability subroutines: MIS, local broadcast, and bounded broadcast, need to succeed for a polynomial number of rounds (or simulated rounds, for the case of the MIS algorithm), in n, for no more than n nodes per round. Therefore, for sufficiently large constant factors, we can assume by a union bound that they always succeed, with high probability. Moving forward with our analysis, assume this holds.

To achieve the constant-degree property, fix some node u. We can apply the same argument used in Section 5 which proves that any given instance of an MIS algorithm can add at most a constant number of u's neighbors to the MIS. Because we run only $\tau + 1 = O(1)$ iterations of an MIS algorithm in the first phase of our algorithm, u ends up with at most a constant number of neighbors in M.

Again leveraging the argument from Section 5, we next consider the nodes added to the CCDS structure when we connect nearby nodes in M. This argument proves that these paths add no more than a constant number of additional nodes to the structure within G' range of u. Unlike Section 5, however, we must finally also consider *failed* paths, where a node joined a path from some $v, w \in M$, but its link to v or w failed so it could not report this fact, leading the dominators to move on to potentially add a different path between themselves . By the definition of the algorithm and our link detector, for 2 hop paths between a pair of nodes in M, this can only happen τ times. For 3 hop paths, the algorithm will have no more than $O(\tau^2)$ failures. In both cases, we are increasing the number of failed paths within range of u by only a constant factor, keeping the total bound on u's neighbors in the CCDS at a constant.

Moving on, we now consider the connectivity property. Because we run $\tau + 1$ iterations of the MIS algorithm, every node either ends up in M or neighbors at least one node in M by G. Let S be the CCDS generated by our algorithm. Our above maximality observation will prove useful in showing that S is connected in H. Assume for contradiction that S is not connected in H. Partition the nodes into components, $C_1, C_2, ...$ such that for each C_i, $S \cap C_i$ is a CCDS for C_i, but for every C_i and C_j, $i \neq j$, $S \cap (C_i \cup C_j)$ is not a CCDS for $C_i \cup C_j$. Furthermore, when defining these components, for each $u \notin S$, make sure it is included in a component that includes a G-neighbor in S (every node has at least one such neighbor in S; if it has multiple neighbors in different components, choose one arbitrarily).

By our contradiction assumption there are at least two such components. Because G *is* connected (by the model assumption), there must be two components C_i and C_j that are connected in G such that they are not connected in H over the nodes in $S \cap (C_i \cup C_j)$. Let (u, v) be an edge in G between C_i and C_j. By definition, either u is in S, or u has a G-neighbor u' in $S \cap C_i$. The same holds with respect to v and some $v' \in S \cap C_j$. Consider the cases for u and v. If both u and v are in S, then C_i and C_j are not separate components—a contradiction. If (w.l.o.g.) only v is in S, then u' is 2 hops from v. In this case, during our CCDS algorithm, u would have heard from v and reported this to u'. During the 2 hop connection step, u' would have added a 2 hop path in H between itself and v (potentially including v, or potentially another node). This path, however, connects the components in H: a contradiction.

Next, assume that neighbor u nor v are in S. It follows that u' and v' are within 3 hops in G. In this case, u would have learned of v' through v, and v would have learned of u' through u. During the 3 hop connection step of our algorithm, u' would have systematically explored paths to v' until successfully adding such a path in H. Again, this connects components in H and generates a contradiction.

To bring together the pieces, we proved that under the high probability assumption that our subroutines worked correctly: (a) no node can have more than a constant number of G'-neighbors in S; and (b) every node neighbors S in G and S is connected in H. Combined, this provides that our algorithm generates a CCDS, with high probability. □

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 detector, constructing a CCDS requires $\Omega(\Delta)$ rounds. This bound holds regardless of message size. 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 solve 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 solve the CCDS problem in $f_1(\Delta, n)$ rounds, where Δ is the maximum degree in G and n is the network size. We can then construct 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.*

Proof. 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.

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 classical 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 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 output 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 [11].) 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

For clarity, this paper considers 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* formalism [4]). 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 from the definition of this algorithm:

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* $r + 2\delta_{CDS}$*.*

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.

9 An MIS Algorithm for Asynchronous Starts and No Topology Knowledge

The model considered in this paper assumes that all processes start during the same round. In this section, we describe a collection of small changes that allows our MIS algorithm from Section 4 to work in a model where processes wake up during different rounds, and have only local knowledge of the round number. This matches the wake up assumptions of previous studies of the MIS problem in the classical radio network model; e.g., [14, 18]. Furthermore, we also show that if deployed in the classical radio network model, our algorithm requires no topology information. It follows that our $O(\log^3 n)$ time solution is directly comparable to the $O(\log^2 n)$ time solution of [18]. We discuss this comparison in this section.

Changes to Algorithm. If processes wake up asynchronously, they cannot start competition phases and epochs at the same time. Therefore the epochs and phases of different processes might not be aligned. It is possible, for example, that when a process wakes up, some neighbors might already have been active for a while and they might already be part of the MIS or close to joining the MIS. A new process should not be able to knock out a neighbor that would soon join the MIS otherwise, and so on. With this in mind, we follow the example of [18], and add an additional *listening* phase of $\Theta(\log^2 n)$ rounds at the beginning of each epoch.[1] During the listening phase, the sending probability of a process is 0. If a process receives a message from a neighbor during the listening phase, it is *knocked out* and starts a new epoch, which begins with a brand new listening phase. The other obvious change we must make is that once a process joins the MIS, it must continue to broadcast and announce this information with probability $1/2$ for the remainder of the execution (e.g., to inform processes that might potentially wake up much later).

Proof Modifications. We provide slightly modified versions of Lemmas 4.3 and 4.4 to compensate for asynchronous starts, and, in the case where $G = G'$, no topology information. As before, we will only show individual high probability bounds and assume that we choose all constants large enough so that we can do all the necessary union bounds to get an overall high probability bound. This is possible because the number of considered time slots, processes, and disks is at most polynomial in n. For these modified proofs, we also assume that there are global round numbers starting when the first process wakes up. The global round numbers are only used to refer to specific time slots in the analysis, and they are unknown to the processes themselves.

Lemma 9.1. *In every round r, for every disk D_i: $P_i(r) < 2$, w.h.p.*

Proof. We proceed by induction on r. Clearly, $P_i(1) = 0$ and thus the lemma holds for $r = 1$. Suppose that the lemma holds up to round $r - 1$ and for contradiction, assume that $P_i(r) \geq 2$. Consider the interval $[r - \ell_P, r - 1]$ of $\ell_P = \Theta(\log n)$ rounds preceding round r. For every process u in D_i that has a positive broadcast probability p_u in round r, either $p_u = 1/n$ and u started to compete during the considered ℓ_P rounds or the broadcast probability of u is at least $p_u/2$ throughout the considered interval. Hence, for all $r' \in [r - \ell_P, r - 1]$, we have $P_i(r') \geq (2 - n \cdot 1/n)/2 = 1/2$. By the induction hypothesis, we also have $P_i(r') < 2$ for all $r' \in [r - \ell_P, r - 1]$. By also applying the induction hypothesis for all the nearby circles, because $P_i(r') \in [1/2, 2)$ for all such r', the probability of exactly one process sending in D_i, and no other process sending among all the G'-neighbors of processes in D_i in round r' is a constant bounded away from 0. The details of this argument are analogous to the one used in the proof of Lemma 4.3. Therefore, if we choose $\ell_P \geq c \log n$ for a large enough constant c, during the interval $[r - \ell_P, r - 1]$, some process in D_i succeeds in knocking out all the other processes in D_i, in which case we clearly get $P_i(r) < 2$. □

Lemma 9.2. *(Independence) For every pair of processes u and v, $(u, v) \in E$, it is not the case that both output 1, w.h.p.*

Proof. The argument is analogous to the one used in the proof of Lemma 4.4. W.l.o.g., assume that u decides to join the MIS at time r_u and that v decides to join the MIS at some time $r_v \leq r_u$. Let us first assume that $r_u - r_v \geq \ell_P = \Theta(\log n)$. In that case, process u is in the announcement phase for at least ℓ_P rounds while v is awake and still competing. Because by Lemma 9.1, $P_i(r) \leq 2$ for all i and r, the probability that u knocks out v is a constant bounded away from 0 in each of these ℓ_P rounds. If $r_u - r_v < \ell_P$, there are ℓ_P rounds prior to r_v in which both compete with broadcast probabilities at least $1/4$. By again applying Lemma 9.1 for the ℓ_P rounds and nearby circles, in each of these rounds, one of the two processes knocks out the other one with constant probability bounded away from 0. In both cases, this implies that if $\ell_P \geq c \log n$ for a

[1] It would be sufficient to only add the listening phase to the first epoch, but since each epoch requires $\Theta(\log^2 n)$ rounds anyway, for simplicity, we can afford to add the listening phase to each epoch.

large enough constant c, w.h.p. either u or v is knocked out before both of them start the announcement phase. □

Lemma 9.3. *(Termination) If processes have* 0*-complete link detectors, or if* $G = G'$*, each process outputs* 0 *or* 1 *after starting at most* $O(\log n)$ *epochs, w.h.p.*

Proof. In both cases assumed the lemma statement, a process u can only be knocked out by a G-neighbor v. In a given epoch, therefore, either u joins the MIS (at which point, we are done) or it is knocked out by G-neighbor v. We can apply the same argument as in Lemma 4.5 (applied now with Lemma 9.1 instead of Lemma 4.3), to show that with constant probability: v's messages knocks all of v's G-neighbors back to a listening phase, allowing v to join the MIS, at which point, also with constant probability, u receives v's MIS announcement and therefore outputs 0. Hence, the probability that u does not output 0 or 1 after starting $O(\log n)$ epochs is sufficiently small to provided the needed high probability. □

To account for asynchronous starts, we must slightly modify our notation such that saying an algorithm solves the MIS problem in $f(n)$ rounds, means that every process outputs 0 or 1 within $f(n)$ rounds of waking up. With this in mind, combining the argument from our original Theorem 4.6 with our modified lemmas, we get our final theorem statement:

Theorem 9.4. *Assuming asynchronous starts and either* 0*-complete link detectors or* $G = G'$*, our modified MIS algorithm solves the MIS problem in* $O(\log^3 n)$ *rounds, w.h.p.*

Comparison to Existing MIS Algorithms. By the above theorem, our modified MIS algorithm works in the classical radio network model (i.e., $G = G'$) with asynchronous starts. It guarantees each process to output 0 or 1 within $O(\log^3 n)$ rounds of waking up. The algorithm is inspired by the $O(\log^2 n)$ round solution from [18]. We differ from [18] in that we trade speed for simplicity. That is, by running a factor of $O(\log n)$ slower, we can eliminate a phase from the structure used in [18], simplifying the algorithm and proof structure.

10 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 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 formalism 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.

Acknowledgements: The authors thank the anonymous reviewers for useful comments and suggestions that helped improve the presentation of this paper.

References

[1] Personal Communication with Johannes Schneider, ETH Zurich, Jan. 2011.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] M. Ghaffari, B. Haeupler, N. Lynch, and C. Newport. Bounds on Contention Management in Radio Networks. In *Proceedings of the International Symposium on Distributed Computing*, 2012.

[9] 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.

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

[11] 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.

[12] F. Kuhn, N. Lynch, and C. Newport. The Abstract MAC Layer. *Distributed Computing*, 24(3-4):187–206, 2011.

[13] 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.

[14] 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.

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

[16] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad hoc Networks Beyond Unit Disk Graphs. *Wireless Networks*, 14(5):715–729, 2008.

[17] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.