Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2011-045 October 12, 2011

Leader Election Using Loneliness Detection

Mohsen Ghaffari, Nancy Lynch, and Srikanth Sastry

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Leader Election Using Loneliness Detection[*]

Mohsen Ghaffari, Nancy Lynch, and Srikanth Sastry
{ghaffari, lynch, sastry}@csail.mit.edu
CSAIL, MIT
Cambridge, MA

October 12, 2011

Abstract

We consider the problem of leader election (LE) in single-hop radio networks with synchronized time slots for transmitting and receiving messages. We assume that the actual number n of processes is unknown, while the size u of the ID space is known, but is possibly much larger. We consider two types of collision detection: *strong (SCD)*, whereby all processes detect collisions, and *weak (WCD)*, whereby only non-transmitting processes detect collisions.

We introduce *loneliness detection (LD)* as a key subproblem for solving LE in WCD systems. LD informs all processes whether the system contains exactly one process or more than one. We show that LD captures the difference in power between SCD and WCD, by providing an implementation of SCD over WCD and LD. We present two algorithms that solve deterministic and probabilistic LD in WCD systems with time costs of $\mathcal{O}(\log \frac{u}{n})$ and $\mathcal{O}(\min(\log \frac{u}{n}, \frac{\log(1/\epsilon)}{n}))$, respectively, where ϵ is the error probability. We also provide matching lower bounds.

We present two algorithms that solve deterministic and probabilistic LE in SCD systems with time costs of $\mathcal{O}(\log u)$ and $\mathcal{O}(\min(\log u, \log\log n + \log(\frac{1}{\epsilon})))$, respectively, where ϵ is the error probability. We provide matching lower bounds.

1 Introduction

We study the leader election problem in single-hop radio networks with synchronized time slots for transmitting messages, but where messages are subject to collisions. We focus on the time cost of electing a leader and the dependence of this cost on the number n of actual processes in the network as well as on a known, finite ID space I of the processes. We assume that, while n may be unknown, each process knows its own ID and the ID space I. We only restrict the size of the ID space, $u = |I|$, to be at least n; the number of processes in the system may be much smaller than the size of the ID space.

The time cost of leader election depends significantly on the ability of the processes to detect message collisions. The problem has been well studied in single-hop systems which have no collision detection (*e.g.*, [3, 6, 11, 1, 9]) and in systems with *strong collision detection* (SCD) in which all processes can detect message collisions (*e.g.*, [14, 2, 4, 3, 7, 10]). However, the cost of leader election is under-explored in systems with *weak collision detection* (WCD) wherein only the listening processes detect message collisions.

[*] This work partially supported by the NSF under award numbers CCF-0937274, and CCF-0726514, and by AFOSR under award number FA9550-08-1-0159. This work is also partially supported by the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370.

We focus on the time costs of solving deterministic and probabilistic leader election in WCD systems and compare them to the time costs for leader election in SCD systems.

The primary challenge to solving leader election in WCD systems is to distinguish the following two cases: (1) $n > 1$ and all the processes transmit simultaneously resulting in a collision, which remains undetected because no process is listening, (2) $n = 1$ and any message transmitted by the process does not collide, but the successful transmission is undetected because no process is listening. In both cases the transmitting processes receive the same feedback from the WCD system despite different outcomes. Note that these two cases are distinguishable in SCD systems. Hence, *loneliness detection* — determining whether or not there is exactly one process in the system — is a key subproblem of leader election.

Summary of Results. We define the Loneliness Detection (LD) problem and determine the time complexity of solving LD in single-hop wireless networks. We show that LD can be solved deterministically in WCD systems in $\mathcal{O}(\log \frac{u}{n})$ time slots for $n > 1$ and in $\mathcal{O}(\log u)$ time slots for $n = 1$. Interestingly, in the probabilistic case, if $n > 1$, LD can be solved in WCD systems in a constant number of rounds with high probability; however, if $n = 1$, then our algorithm takes $\mathcal{O}(\log u)$ time slots. We demonstrate that these time bounds are tight by presenting matching lower bounds.

We use the aforementioned LD solution to implement an SCD channel on top of a WCD system. This allows us to deploy SCD-based protocols on WCD systems. We explore such SCD-based protocols for LE and present upper and lower bounds for both deterministic and randomized LE in SCD systems. First, we present a deterministic LE protocol that terminates in at most $\mathcal{O}(\log u)$ time slots and show a matching lower bound. For probabilistic LE, we interleave Nakano and Olariu's algorithm from [10] with our deterministic algorithm to solve the problem in $\mathcal{O}(\min(\log u, \log\log n + \log(\frac{1}{\epsilon}))$ rounds with termination probability at least $1 - \epsilon$ (where $1 < \epsilon < 0$). We present a lower bound of $\Omega(\min(\log(\frac{u}{n}), \log(\frac{1}{\epsilon})))$ for probabilistic LE on SCD systems with termination probability at least $1 - \epsilon$. Note that the lower and upper bounds match when $\epsilon = \mathcal{O}(\frac{1}{n})$ and $u \gg n$. Subsequently, we demonstrate that the same upper and lower bounds hold for LE in WCD systems as well.

Organization. Section 2 presents the formal definitions and the system model assumptions used in the remainder of this article. Section 3 specifies the deterministic and probabilistic leader election problem as well as a third variant which is used to demonstrate lower bounds. Subsequently, we introduce the problem of loneliness detection in Section 4 where we provide formal specifications for deterministic and probabilistic variants of the problem. Also, in Section 4, we demonstrate the use of loneliness detection to implement strong collision detection in weak collision detection systems. Next, in Section 5, we provide algorithms to solve deterministic and probabilistic loneliness detection on weak collision detection systems and demonstrate matching lower bounds. In effect, results from Sections 4 and 5 demonstrate tight bounds for achieving strong collision detection on weak collision detection systems. We use these results in Section 6 to provide algorithms and matching lower bounds for solving leader election in strong as well as weak collision detection systems. Finally, we end with a discussion on possible avenues for future research in Section 7.

2 System Models, Definitions, and Notations

Our model considers a finite set of n processes with unique IDs from I — a finite ID space of size u. The set $J \subseteq I$ denotes the set of IDs of the n processes. The processes communicate with each other through a broadcast communication channel. We assume that execution proceeds in rounds and that the rounds are synchronized at all processes. We model this system using the Timed I/O Automata formalism [5].

While Timed I/O Automata adequately model deterministic system behavior, they do not model probabilistic behavior. For the latter, we use Probabilistic Timed I/O Automata from [12] which is described in [13, Chapter 2]. In deterministic Timed I/O Automata described in [5], the effect of an action a on the automaton in a state s is a new state s' which is uniquely determined by a and s. On the other hand, in Probabilistic Timed I/O Automata derived from [12], the effect of an action a on the automaton in state s is determined by a, s, and a discrete probability distribution μ over the set of possible states that the automaton could be in as a result of executing action a (from state s).

2.1 Processes

Each process is modeled as a probabilistic automaton that interacts with a channel automaton and possibly other automata. Each process is assumed to start from a fixed initial state. Processes communicate by broadcasting messages on a wireless channel. Processes send messages from a fixed alphabet $\mathcal{M}$. We assume that $\mathcal{M}$ does not contain special placeholder elements $\bot$ and $\top$, which denote silence and collision, respectively. We assume that time is divided into *rounds*, and processes have synchronized clocks and can detect the start and end of each round. Processes communicate only at round boundaries and the transmissions are contained within a single round. We assume that all processes wake up at time 0, which is the beginning of round 1. While the processes may or may not send a message in any given round, each process receives either a message or silence ($\bot$) or collision notification ($\top$) in every round.

A process i sends a message m at the start of round r through the action $send(m, r)_i$, which is an output from process i, and receives a message m' by the end of round r through the action $receive(m', r)_i$, which is an input to process i. If a process i does not send a message in round r, then, for convenience, we state and assume that process i executes action $send(\bot, r)_i$. Also, if a process does not send a message in round r, then it is assumed to be listening in round r. If a process i does not receive a message in round r, then the process either receives silence through action $receive(\bot, r)_i$ or receives a collision notification through action $receive(\top, r)_i$. For the remainder of this paper, we assume that in every execution, for each process i and each round r, exactly one event of the form $send(*, r)_i$ and exactly one event of the form $receive(*, r)_i$ occur.

In addition, processes may interact with other services and the "external world" through other input and output actions.

2.2 Wireless Channels

A wireless channel, modeled as an I/O automaton and denoted *channel automaton*, is a broadcast medium in which at most one process can successfully send a message in any round. Recall that processes communicate by sending and receiving messages on a wireless channel. The properties of wireless channels described here hold assuming that processes send their messages only at the start of each round. If processes send messages only at the start of each round, then a wireless channel is guaranteed to provide feedback to all the processes by the end of each round, and the feedback provided to each process is a function of the set of messages sent in that round.

A channel automaton interacts with process automata through send actions, which are inputs to a channel, and receive actions, which are outputs from a channel. A process i sends a message m in round r through action $send(m, r)_i$ and receives a message m in round r through action $receive(m, r)_i$. For each process i and each round r, if i does not send a message in round r, it performs action $send(\bot, r)_i$ as input to the channel, and if the channel does not provide any feedback to process i in round r, then the channel performs action $receive(\bot, r)_i$ as input to process i. If message collision occurs in round r, then the channel may provide a collision feedback to a process i through action $receive(\top, r)_i$.

2.2.1 Well-formed Behavior

Recall that we assume that processes send their messages only at the start of each round, and the channel guarantees that these processes receive messages and feedback from the channel by the end of each round. We formalize this notion through the definition of *well-formedness* in the input and output events of the channel and the times at which they occur. Let the pair (e, τ) denote infinite sequence e of $send()$ and $receive()$ events and the associated sequence τ of increasing times $t_0, t_1, \ldots$ at which these events occur, and let $t_0 = 0$. The pair (e, τ) is said to be well-formed if (1) $\lim_{r \to \infty} = \infty$; (2) for every time t not in the sequence τ, no $send()$ or $receive()$ events occur at time t; (3) for each process i, some $send()_i$ event occurs at time t_0; and (4) for each $r \in \mathbb{N}^+$, and for each process i, some $receive()_i$ event occurs at time t_r, and this is followed by some $send()_i$ event at time t_r. Each time instance t_r in τ marks the end of round r and the start of round $r + 1$.

The process automata that preserve the well-formedness behavior of the system are said to be a well-formed environment. We assume that in every admissible execution[1] of the environment composed with the channel, the channel preserves the well-formedness behavior; that is, if the environment is well-formed, then for each aforementioned admissible execution α, the event sequence and time sequence (e, τ) associated with α is also well-formed.

2.2.2 Time-bound Functions for Deterministic and Probabilistic Channels

Channel automata may be deterministic or probabilistic. In the case of deterministic channels, let α be an arbitrary admissible execution of a channel composed with a well-formed environment consisting of n processes. Suppose that, in α, the sends and receives occurs at times $t_0, t_1, t_2, \ldots$. We assume that a known *time-bound function* $B : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{R}^{\geq 0}$ maps each round r to an upper bound $B(n, r)$ on the real time at which the round ends; B is monotonically non-decreasing with respect to r. Thus, in α, for each $r \in \mathbb{N}^+$, $t_r \leq B(n, r)$. Any deterministic channel that satisfies a time-bound function B is said to be a *B-time-bounded channel*.

In the case of probabilistic channels, the real-time duration of a round is not fixed *a priori*: we assume that every round terminates in finite time with probability 1. Moreover, we assume a probabilistic upper bound on the time for each round to terminate, given by a time-bound function $\beta : \mathbb{N}^+ \times \mathbb{N}^+ \times (0, 1] \to \mathbb{R}^{\geq 0}$. Informally, for each $n \in \mathbb{N}^+$, for each $r \in \mathbb{N}^+$ and for each $\epsilon \in (0, 1]$, $\beta(n, r, \epsilon)$ is an upper bound on the real time by which round r terminates with probability at least $1 - \epsilon$ in a system consisting of n processes. The function β is monotonically nondecreasing with respect to r and monotonically nonincreasing with respect to ϵ. More precisely, for each well-formed probabilistic environment E consisting of n processes, consider the probability distribution on admissible executions of the probabilistic channel with E. For each $r \in \mathbb{N}^+$, let t_r be a random variable that denotes the time at which round r terminates. Then the probability that $t_r \leq \beta(n, r, \epsilon)$ is at least $1 - \epsilon$. Any probabilistic channel whose round duration is upper bounded by a function β is said to be a *β-time-bounded channel*.

2.2.3 Channel Behavior

The behavior of a channel in a round r is determined by the set of transmitting processes in round r. A process i is said to be a *transmitting process* in round r if there exists an $m_i \in \mathcal{M}$ such that the event $send(m_i, r)_i$ occurs. Note that, by our assumptions, if a process i is not a transmitting process in round r, then the event $send(\perp, r)_i$ occurs. Let T denote the set of transmitting processes in round r.

[1] An *admissible execution* is one in which time advances to infinity.

If no process transmits a message in round r, and so $|T|$ is 0, then for each process i in the system, the event $receive(\bot, r)_i$ occurs; that is, all the processes receive silence in round r. If exactly one process j transmits a message (say) m in round r, and so $|T|$ is 1, then for each process i in the system, the event $receive(m, r)_i$ occurs; that is, all the processes receive m in round r.

If two or more processes send messages in a given round, and so $|T| > 1$, then we say that the round experiences a message collision. The responses given by a channel in the event of a message collision are determined by the collision detection ability of the wireless channel. We consider two types of channels with different collision detection properties.

Weak Collision Detection (WCD) Channels. In weak collision detection (WCD) channels, in the case of a collision in a round r, every transmitting process receives its own message, and every process that is listening receives a collision notification in that round. That is, if $|T| > 1$, then for each process i in T for which event $send(m_i, r)_i$ occurs, the event $receive(m_i, r)_i$ occurs, and for each process j not in T, the event $receive(\top, r)_j$ occurs.

Strong Collision Detection (SCD) Channels. In strong collision detection (SCD) channels, if a message collision occurs in a given round, then all the processes receive a collision notification in that round. That is, if $|T| > 1$, then for each process i, the event $receive(\top, r)_i$ occurs.

2.2.4 SCD is stronger than WCD

Systems with SCD are at least as powerful as systems with WCD because SCD provides more information than WCD to the transmitting processes in a given round. We formalize this notion through a collection of automata which uses a (deterministic) SCD channel and satisfies the properties of a (deterministic) WCD channel. If the SCD channel is B-time-bounded, then the WCD channel is B-time-bounded as well. We do not consider probabilistic SCD channels here.

Before we describe the automata, recall that channels interact with other automata using $send$ and $receive$ actions. In order to distinguish the actions of an SCD channel from those of the automaton that implements a WCD channel, we rename the $send$ and $receive$ actions associated with the automaton implementing the SCD channel as $sendSCD$ and $receiveSCD$ actions respectively. The automata are illustrated in Figure 1.

Pseudocode Notation. We use the following notations for the pseudocode presented in this document. When an action at a process is triggered by an event e in the system, we denote the trigger with the notation "**upon event** e" in the pseudocode. Similarly, when the automaton is waiting for the occurrence of an event e to proceed, we denote it with the notation "**wait until event** e". Instances in which an algorithm performs an action a are denoted "**perform** a".

We also use the following additional notations to bind values to certain variables. Consider the statements "upon event $e(x, y)$" and "wait until event $e(x, y)$". In both cases, if (say) x is undefined and y is defined at the point in the code where the statements occur, then the semantics of the code is to wait until any event of the form $e(*, y)$, and when an event (say) $e(x', y)$ occurs, bind the value of x' to the variable x.

Next we describe a collection of automata A_i, one for each process i (collectively denoted A), that interact with an SCD channel through $sendSCD$ and $receiveSCD$ actions and interact with the processes in the system through $send$ and $receive$ actions. The pseudocode is given in Algorithm 1. The automata satisfies the properties of a WCD channel with respect to the $send$ and $receive$ actions.

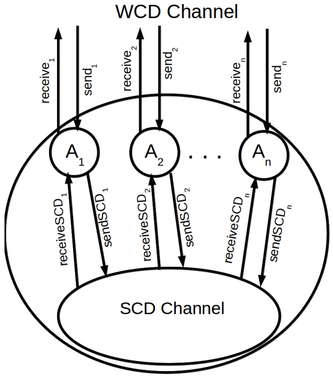

Figure 1: The automata collection that implement WCD channel on top of an SCD channel

For each process i, the automaton A_i behaves as follows. When event $send(m, r)_i$ occurs, the automaton executes the action $sendSCD(m, r)_i$. When event $receiveSCD(m', r)_i$ occurs, if $m \neq \perp$ (process i sent a message in the current round), then the automaton executes $receive(m, r)$; otherwise the automaton executes $receive(m', r)$.

Algorithm 1 Implementing WCD on a system with an SCD channel.

```
Variables:
    $m \in \mathcal{M} \cup \{\perp\}$
    $m' \in \mathcal{M} \cup \{\top, \perp\}$
    $r \in \mathbb{N}^+$
Actions for each process $i$:
for $r$ := 1 to $\infty$
    upon event $send(m, r)_i$
    perform $sendSCD(m, r)_i$
    wait until event $receiveSCD(m', r)_i$
    if ($m \neq \perp$) then perform $receive(m, r)_i$
    else perform $receive(m', r)_i$
endfor
```

Theorem 1. *Automata A, which is described in Algorithm 1, when used with a B-time-bounded SCD channel, satisfies the properties of a B-time-bounded WCD channel.*

Proof. We show that A satisfies the requirements of a WCD channel by showing three properties: (1) If no process sends a message in round r, then all the processes receive silence ($\perp$) in round r; (2) if exactly one process sends a message (say) m in round r, then all the processes receive m in the round r; and (3) if two or more processes send messages in round r, then all the listening processes receive a collision notification ($\top$) in round r whereas each transmitting process receives the message it sent. We demonstrate each of the above three properties.

Property 1. If no process transmits a message in round r, then for each process i in the system, the event $send(\perp, r)_i$ occurs, and from the description of A, event $sendSCD(\perp, r)_i$ occurs. From the properties of SCD channels we know that event $receiveSCD(\perp, r)_i$ occurs for each process i, and from the description of A we know that the event $receive(\perp, r)_i$ occurs; that is, all processes receive silence in round r.

Property 2. If exactly one process j transmits a message (say) m in round r, then event $send(m, r)_j$ occurs and for every other process i in the system, the event $send(\perp, r)_i$ occurs. From the description of A, we know that event $sendSCD(m, r)_j$ occurs for process j and event $sendSCD(\perp, r)_i$ occurs for every other process i. From the properties of SCD channels we know that event $receiveSCD(m, r)_i$ occurs for every process i in the system, and from the description of A we know that the event $receive(m, r)_i$ occurs for every process i; that is, all processes receive the message m in round r.

Property 3. If a set T of two or more processes transmit messages in round r, then event $send(m_j, r)_j$ occurs for every process $j \in T$ where $m_j \in \mathcal{M}$ and event $send(\perp, r)_i$ occurs for every process $i \notin T$. From the description of A we know that for every process $j \in T$ the event $sendSCD(m_j, r)_j$ occurs, and for every process $i \notin T$, the event $sendSCD(\perp, r)_i$ occurs. From the properties of SCD channels we know that event $receiveSCD(\top, r)_i$ occurs for every process i in the system, and from the description of A we know that the event $receive(\top, r)_i$ occurs for every process $i \notin T$, and for every process $j \in T$, the event $receive(m_j, r)_j$ occurs; that is, each process i not in T receives a collision notification, and each process i in T receives its own message.

Thus, we have shown that A implements a WCD channel from an SCD channel. To complete the proof, we have to verify that the time-bound function B (given for the SCD channel) is also a time-bound function for the WCD channel. From the pseudocode in Algorithm 1, we know that the action $sendSCD(*, r)_i$ immediately follows the action of the form $send(*, r)_i$ and the action $receive(*, r)_i$ immediately follows the action of the form $receiveSCD(*, r)_i$. In other words, if round r of the SCD channel ends at time t, then the round r of the WCD channel ends at time t as well. Thus, if the time-bound function of the SCD channel is B, then the time-bound function of the WCD channel is B as well. □

3 The Leader Election Problem

In this section, we specify and describe two variants of the leader election problem: deterministic and probabilistic. We also provide an overview of prior work in solving the leader election problem in wireless systems.

3.1 Specification

Briefly, leader election (LE) is a problem in which all the processes in the system elect some process as the leader unanimously. A solution to LE is an automaton that has an output action $leader(l)_i$ for each process i and no input actions. The automaton eventually outputs $leader(l_i)_i$ (exactly once) at each process i, where l_i is the ID of some process in the system, and for every pair of processes i and j, $l_i = l_j$. The process l_i is the *leader*.

More precisely, the LE problem is specified by two sets of properties of automaton executions: safety and liveness properties. We consider two variants of the problem, deterministic and probabilistic, which differ only in their liveness properties.

Safety Properties. We consider two safety properties. First, in any execution, for every process $i \in J$, at most one $leader()_i$ event occurs. Second, in any execution, for every pair of processes $i, j \in J$, if events $leader(l_i)_i$ and $leader(l_j)_j$ occur, then $l_i = l_j$ and $l_i \in J$.

Deterministic Liveness Property. The deterministic liveness property states that in any admissible execution, for every process $i \in J$ some event of the form $leader(*)_i$ occurs.

Probabilistic Liveness Property. The probabilistic liveness property states that in the space defined by all admissible executions, for every process $i \in J$, some event of the form $leader(*)_i$ occurs with probability 1.

Automata that solve *deterministic leader election* satisfy the safety properties and the deterministic liveness property, whereas automata that solve *probabilistic leader election* satisfy the safety properties and the probabilistic liveness property. Note that an automaton that solves deterministic leader election solves probabilistic leader election as well.

Let $R : \mathbb{N}^+ \rightarrow \mathbb{N}^+$ be a function that maps n — the number of processes in the system — to a nonnegative integer. A solution to deterministic LE in which all the $leader()$ events occur within $R(n)$ rounds is said to be *R-round-bounded.*

Let $\rho : \mathbb{N}^+ \times (0, 1] \rightarrow \mathbb{N}^+$ be a function that maps n — the number of processes in the system and ϵ — the failure probability — to a nonnegative integer; let ρ be monotonically nonincreasing with respect to ϵ. A solution A_{LE} to probabilistic LE is said to be *ρ-round-bounded* if the following condition holds. In the probability space defined by all admissible executions of A_{LE} in a system consisting of n processes, for every $\epsilon \in (0, 1]$, the probability that all the $leader()$ events at all the processes occur within $\rho(n, \epsilon)$ rounds is at least $1 - \epsilon$.

3.2 Prior Work on Leader Election Using Collision Detection

There has been a significant body of work exploring the solvability and complexity of LE and the related problem of local broadcast in wireless systems with collision. A significant portion of the results focus on LE in SCD systems. In this section, we first discuss prior work related to SCD systems and then discuss existing results for LE in WCD systems.

Deterministic LE in SCD Systems. For deterministic LE in single-hop networks, there exist matching time bounds $\mathcal{O}(\log n)$ from [2, 4] and $\Omega(\log n)$ from [3] where n is the number of processes in the system and n is known. For the scenarios where n, the number of processes in the system, is unknown, the best known deterministic upper bound on the time complexity of LE in SCD systems is $\mathcal{O}(n)$ in [7] for arbitrary multi-hop networks; however, the result in [7] assumes that an upper bound u of n is known and is in $\mathcal{O}(n)$. As a special case, the same upper bound holds for single-hop networks as well.

To our knowledge, for deterministic leader election in single-hop SCD systems where n is unknown and u is known, but u is unconstrained by n except that $u > n$, better upper and lower bounds than from [7] and [3], respectively, are not known.

Probabilistic LE in SCD Systems. In the seminal paper [14], Willard established time bounds for probabilistic leader election in single-hop wireless networks with SCD. Willard presented an algorithm that solves leader election in expected time $\mathcal{O}(1)$, $\mathcal{O}(\log \log u)$, and $\log \log n + o(\log \log n)$ in the cases where n, the number of processes in the system is known, where n is unknown, but u — an upper bound on n — is known, and where both n and u are unknown, respectively. For the case where n and u are unknown, the results in [10] provided an improved algorithm with running time $\log \log n + o(\log \log n) + \mathcal{O}(\log(\frac{1}{\epsilon}))$ with probability of termination exceeding $1-\epsilon$. Although a lower bound of $\Omega(\log \log n)$ for this problem has been presented in [10], the lower bound applies only for “uniform algorithms” in which all the processes transmit with the same probability in each round (although the probability can vary from one round to another).

LE in WCD Systems. To our knowledge, the problem of leader election seems to be relatively underexplored in the context of WCD. The best known timing bounds for deterministic leader election in single-hop wireless networks with WCD are $\Theta(\log n)$ where n is the number of processes in the system and is known; the time bound is based on the results for broadcast in multi-hop wireless networks in [11].

4 The Loneliness Detection Problem

In this section, we specify and describe the loneliness detection problem. Also, we demonstrate that loneliness detection is the discriminator between SCD and WCD systems by implementing an SCD channel on top of a WCD channel augmented with an automaton that solves loneliness detection.

4.1 Specification

Briefly, loneliness detection (LD) informs all the processes in the system whether or not there exists exactly one process in the system. A solution to LD is an automaton that has an output action $alone()$ and no input actions. In some round r, for each process i, the loneliness detector outputs $alone(a_{LD}, r)_i$ where a_{LD} is Boolean. The value of a_{LD} is $true$ iff there is exactly one process in the system. Note that we assume that the loneliness detector outputs its $alone()$ events at all processes in the same round; in this sense, the loneliness detector is synchronous.

More precisely, the LE problem is specified by two sets of properties of automaton executions: safety and liveness properties. We consider two variants of the problem, deterministic and probabilistic, which differ only in their liveness properties.

Safety Properties. We consider three safety properties. First, in any execution, (a) if an $alone(true, *)_*$ event occurs, then there is exactly one process in the system; that is, $n = 1$, and (b) if an $alone(false, *)_*$ event occurs, then the system contains more than one process; that is, $n > 1$. Second, in any execution, for every process i, at most one $alone()_i$ event occurs. Third, in any execution, if two events $alone(*, r)_i$ and $alone(*, r')_j$ occur for some pair of processes i and j, then $r = r'$.

Deterministic Liveness Property. The deterministic liveness property states that in any admissible execution, for each process i some $alone()_i$ event occurs.

Probabilistic Liveness Property. The probabilistic liveness property states that in the space defined by all admissible executions, for every process i some $alone()_i$ event occurs with probability 1.

Automata that solve *deterministic loneliness detection* satisfy the safety properties and the deterministic liveness property, whereas automata that solve *probabilistic loneliness detection* satisfy the safety properties and the probabilistic liveness property. Note that an automaton that solves deterministic loneliness detection solves probabilistic loneliness detection as well.

Let $R : \mathbb{N}^+ \rightarrow \mathbb{N}^+$ be a function that maps n — the number of processes in the system — to a nonnegative integer. A solution to deterministic LD in which all the $alone()$ events occur within $R(n)$ rounds is said to be *R-round-bounded.*

Let $\rho : \mathbb{N}^+ \times (0, 1] \rightarrow \mathbb{N}^+$ be a function that maps n — the number of processes in the system and ϵ — failure probability — to a nonnegative integer; let ρ be monotonically nonincreasing with respect to ϵ. A solution A_{LD} to probabilistic LD is said to be *ρ-round-bounded* if the following condition holds. In the probability space defined by all admissible executions of A_{LD} in a system consisting of n processes, for every $\epsilon \in (0, 1]$, the probability that all the $alone()$ events at all the processes occur within $\rho(n, \epsilon)$ rounds is at least $1 - \epsilon$.

4.2 Loneliness Detection and Collision Detection

We show that Loneliness Detection (LD) is, in a sense, exactly the difference between SCD and WCD. Specifically, we present two results: (1) we show that LD is trivially solvable in SCD systems, and (2) we present an algorithm that, given a solution to LD on a WCD system, implements an SCD system.

4.2.1 LD in SCD systems

The following trivial algorithm, which we call *single-broadcast-LD*, solves LD in SCD systems and requires just one round of communication. Each process i in the system sends a default message m at the beginning of round 1 and waits until the end of round 1. If a collision notification is received at the end of round 1, then the algorithm returns $alone(false, 1)_i$, otherwise it returns $alone(true, 1)_i$.

Theorem 2. *The single-broadcast-LD algorithm implements LD in and SCD system.*

Proof. If there is exactly one process in the system, then that process i does not receive a collision notification, and hence returns $alone(true, 1)_i$. On the other hand, if there is more than one process in the system, then the messages of all the processes collide and each process i receives a collision notification in round 1. Therefore, each process i returns $alone(false, 1)_i$. □

Thus, we have shown that LD can be solved on SCD systems using one round of communication. Next, we show how, using the LD service, we can implement an SCD channel on top of a WCD system. The implementation of LD on WCD systems is discussed later, in Section 5.

4.2.2 SCD on WCD Systems Using LD

We now present an algorithm that implements an SCD channel on top of a WCD channel augmented with an LD service. In order to distinguish the actions of a WCD channel from the automaton that implements an SCD channel, we rename the $send$ and $receive$ actions associated with a WCD channel as $sendWCD$ and $receiveWCD$ actions, respectively. The implementation is described in Algorithm 2.

Algorithm 2 Implementing SCD on a system with a WCD channel augmented with an LD service.

```
The algorithm executes two tasks, Init and Communicate, concurrently at each process $i$.
Variables:
    $m \in \mathcal{M} \cup \{\bot\}$
    $m' \in \mathcal{M} \cup \{\top, \bot\}$
    $p2msg \in \{$"ack"$, \bot, \top\}$
    $r_{LD}, r_s \in \mathbb{N}^+$
Task Init:
    Wait until event $alone(a_{LD}, r_{LD})_i$ from the LD service; halt
Task Communicate:
    loop forever
        /* Note that round $r_s$ for the SCD channel starts here and ends when $receive(*, r_s)_i$ is performed */
        upon $send(m, r_s)_i$ wait for Task Init to terminate
        Transmit Round:
            perform $sendWCD(m, r_{LD} + 2r_s - 1)_i$
            wait until $receiveWCD(m', r_{LD} + 2r_s - 1)_i$
        Ack Round:
            if ($m = \bot$ and $m' \in \mathcal{M}$) then perform $sendWCD(\text{"ack"}, r_{LD} + 2r_s)_i$
            else perform $sendWCD(\bot, r_{LD} + 2r_s)_i$
            wait until $receiveWCD(p2msg, r_{LD} + 2r_s)_i$
        if ($m' = \bot$) then perform $receive(\bot, r_s)_i$
        else if ($a_{LD} = true$) then perform $receive(m, r_s)_i$
        else if ($p2msg \neq \bot$) then perform $receive(m', r_s)_i$
        else perform $receive(\top, r_s)_i$
    end loop
```

The algorithm consists of two concurrent tasks: Init and Communicate. In the Init task each process i waits for the output $alone(a_{LD}, r_{LD})_i$ from the LD service. The Communicate task consists of two rounds, Transmit and Ack, which are executed for every round r_s of the SCD channel. Each of the rounds, Transmit and Ack, uses one round of the underlying WCD channel. Let T denote the set of processes that transmit some message m_i in round r_s of the SCD channel via $send(m_i, r_s)_i$ for each process $i \in T$. In the Transmit round, each process $i \in T$ executes $sendWCD(m_i, r_{LD}{+}2r_s{-}1)_i$. All other processes listen to the channel via $send(\bot, r_{LD} + 2r_s - 1)_i$. At the end of the Transmit round, each process $i \in T$ receives its own message via the event $receiveWCD(m', r_{LD}{+}2r_s{-}1)_i$ where $m' = m_i$. Each listening process j receives one of the following: silence ($\bot$), some message, or collision notification ($\top$) via $receiveWCD(m', r_{LD} + 2r_s - 1)_j$. At the end of the Transmit round, m' contains the response from the channel in the Transmit round.

In the Ack round, each process $i \in T$ listens to the channel via the event $sendWCD(\bot, r_{LD} + 2r_s)_i$. Each process $j \in J \setminus T$ sends an ack via the event $sendWCD(\text{“}ack\text{”}, r_{LD} + 2r_s)_j$ iff j received a message in the Transmit round; otherwise j listens to the channel via the event $sendWCD(\bot, r_{LD} + 2r_s)_j$. At the end of the Ack round, each process receives either silence ($\bot$), “ack”, or collision notification ($\top$) via $receiveWCD(p2msg, r_{LD} + 2r_s)_*$. Thus, at the end of the Ack round, $p2msg$ contains the response from the channel in the Ack round.

If $p2msg$ is “ack” or $\top$, then the transmission in the Transmit round was successful, and m', which was received by the process at the end of the transmit round, is that transmitted message. If a_{LD} is $true$, then there is only one process in the system, and so the transmission in the Transmit round was successful. However, if a_{LD} is $false$ and $p2msg$ is $\bot$, then there must have been a collision.

Although the Init and Communicate tasks are executed concurrently, the Communicate task waits for the Init task to terminate before proceeding to sending and receiving messages on the WCD channel. This ensures that the output of the loneliness detector is available to each process before the latter sends messages on the WCD channel in the Communicate Task. The availability of the loneliness detector output allows processes to detect sender-side collisions.

Lemma 3. *Algorithm 2 satisfies the properties of an SCD channel.*

Proof. The properties of an SCD channel are the following: (1) if no process sends a message in a round (say) r_s, then all the processes receive silence ($\bot$) in round r_s; (2) if exactly one process sends a message (say) m in round r_s, then all the processes receive m in round r_s; and (3) if two or more processes send messages in round r_s, then all the processes receive a collision notification ($\top$) in round r_s. Thus, the proof involves demonstrating the aforementioned three properties. We consider each of them in turn.

Let T denote the set of transmitting processes in round r_s of the SCD channel.

Case 1. $T = \emptyset$. For every process i, $m = \bot$ in the Communicate Task and so i performs $sendWCD(\bot, r_{LD}{+}2r_s{+}1)_i$; from the properties of a WCD channel we know that event $receiveWCD(\bot, r_{LD}{+}2r_s{-}1)_i$ occurs. Consequently, each process i executes $receive(\bot, r_s)_i$ after the Ack round in Algorithm 2. Thus, if no process sends a message in round r_s, then all processes receive silence in round r_s of the SCD channel.

Case 2. $|T| = 1$. Let j be the unique ID such that $T = \{j\}$. Thus, $m_j \in \mathcal{M}$ and for every other process $i \in J \setminus T$, m_i is $\bot$. Therefore, from the properties of a WCD channel, we know that for every process i, m'_i is m_j after the Transmit round. Since $m_j \in \mathcal{M}$, every process $i \in J \setminus T$, executes $sendWCD(\text{“}ack\text{”}, r_{LD} + 2r_s)_i$ in the Ack round. We have two subcases to consider: (a) $J \setminus T = \emptyset$, and (b) $J \setminus T \neq \emptyset$.

Case 2(a). $J \setminus T = \emptyset$. Therefore, $J = T = \{j\}$; that is, j is the only process in the system. Therefore, in the output $alone(a_{LD})_j$ at process j, $a_{LD} = true$. Consequently, after the Ack round, the event $receive(m_j, r_s)_j$ occurs.

Case 2(b). $J \setminus T \neq \emptyset$. Therefore, there exists at least one other process (say) i' in the system, and in the output $alone(a_{LD})_j$ at process j, $a_{LD} = false$. From the algorithm, we know that some event $sendWCD(\text{“}ack\text{”}, r_{LD} + 2r_s)_{i'}$ occurs for $i' \in J \setminus T$ in the Ack round, and from the properties of a WCD channel we know that for every process i (including j) $p2msg_i$ is not $\perp$. Since, at each process i, $m'_i \neq \perp$, $a_{LD} = false$, and $p2msg_i \neq \perp$, process i executes $receive(m'_i, r_s)$. But we already know that $m'_i = m_j$ for every process i. That is, if exactly one process sends a message in round r_s, then all processes receive that message in round r_s.

Case 3. $|T| > 1$. Note that $|T| > 1 \Rightarrow |J| > 1$. Therefore, the $alone()$ event for each process i is $alone(false, r_{LD})_i$ in the Init Task; that is $a_{LD} = false$ at all processes i. For each process $j \in T$, $m_j \in \mathcal{M}$, and for each process $i \notin T$, $m_i = \perp$. Therefore, in the Transmit round, there exist at least two processes x and y such that events $sendWCD(m_x, r_{LD} + 2r_s - 1)_x$ and $sendWCD(m_y, r_{LD} + 2r_s - 1)_y$ occur for some $m_x, m_y \in \mathcal{M}$. Since two or more processes send a message, there is a message collision. From the properties of a WCD channel, we know that for every process $j \in T$, m'_j is m_j and for every process $i \notin T$, m'_i is $\top$ (and m_i is $\perp$). Therefore, in the Ack round, for each process i event $sendWCD(\perp, r_{LD} + 2r_s)$ occurs, and consequently, $p2msg_i$ is $\perp$. Given that $a_{LD} = false$, and $m'_i \neq \perp$ at every process i, each process i executes $receive(\top, r_s)_i$. That is, if two or more processes send a message in round r_s, then all processes receive collision notification ($\top$) round r_s.

Cases 1, 2, and 3, show that Algorithm 2 satisfies the properties of an SCD channel. □

Lemma 4. *If the WCD channel is B_{WCD}-time-bounded and the LD service is R_{LD}-round-bounded, then the SCD channel implementation in Algorithm 2 is B_{SCD}-time-bounded, where $B_{SCD}(n, r) = B_{WCD}(n, R_{LD}(n) + 2r)$.*

Proof. Recall that a time-bound function maps the number (n) of processes in the system and each round number (r) to an upper bound on the real time at which that round ends. Let B_{WCD} denote a time-bound function for the underlying WCD channel. From the pseudocode we know that the deterministic LD service outputs the $alone()$ events in (some) round r_{LD} of the WCD channel, and subsequently, for each process i in the system, between every pair of events $send(*, r)_i$ and $receive(*, r)_i$, there are exactly two $sendWCD()$ events of the form $sendWCD(*, r_{LD} + 2r - 1)_i$ and $sendWCD(*, r_{LD} + 2r)_i$. However, from the R_{LD}-round-boundedness assumption, we know that $r_{LD} \leq R_{LD}(n)$ where n is the number of processes in the system.

Therefore, $B_{SCD}(n, 1)$ is given by $B_{WCD}(n, R_{LD}(n) + 2)$ and $B_{SCD}(n, r)$ for each round r is given by $B_{WCD}(n, R_{LD}(n) + 2r)$. □

Theorem 5. *Algorithm 2 implements a deterministic SCD channel on top of a WCD system with a deterministic LD service. If the WCD channel is B_{WCD}-time-bounded and the LD service is R_{LD}-round-bounded, then the SCD channel implementation is B_{SCD}-time-bounded, where $B_{SCD}(n, r) = B_{WCD}(n, R_{LD}(n) + 2r)$. Also, if each round of the WCD channel lasts 1 time unit, then each round r of the SCD channel implementation in Algorithm 2 terminates at time $R_{LD}(n) + 2r$.*

Proof. Follows from Lemmas 3 and 4. □

Algorithm 2 implements a probabilistic SCD channel if the underlying LD service is probabilistic. Namely, consider executions of Algorithm 2 where the underlying LD service is probabilistic and ρ_{LD}-bounded. We demonstrate the following.

Theorem 6. *Algorithm 2 implements probabilistic SCD on top of a WCD channel augmented with a probabilistic LD service. If the WCD channel is B_{WCD}-time-bounded and the probabilistic LD service is ρ_{LD}-round-bounded, then the probabilistic SCD channel implementation is β_{SCD}-time-bounded where $\beta_{SCD}(n, r, \epsilon) = B_{WCD}(n, \rho_{LD}(n, \epsilon) + 2r)$.*

Proof. The proof of correctness follows, in part, from Lemma 3 which shows that Algorithm 2 satisfies the properties of strong collision detection. After the first round of the probabilistic SCD channel has terminated, all future rounds require exactly two rounds of the underlying WCD channel. Since all $alone()$ events occur with probability 1, every SCD round terminates in finite time with probability 1. The remainder of this proof verifies that the probabilistic SCD channel in a system consisting of n processes is β_{SCD}-time-bounded.

Let $\epsilon \in (0, 1]$ denote an arbitrary, but fixed value. By assumption, $B_{WCD}(n, r)$ is an upper bound on the real time at which round r of the underlying WCD channel (in a system consisting of n processes) ends, and $\rho_{LD}(n, \epsilon)$ is an upper bound on the number of rounds within which some $alone()$ event occurs with probability at least $1 - \epsilon$. Note that the first message sent on the SCD channel is transmitted on the WCD channel only after all the $alone()$ events have occurred. Also, for each process i in the system, a $receive(*, 1)_i$ event occurs exactly two rounds after an $alone()_i$ event. From the LD specifications, we know that all the $alone()$ events in the system occur in a single WCD round, and from the specifications of a WCD channel, we know that all the $receive(*, 1)$ events also occur in the same WCD round. Since all $alone()$ events occur within $\rho_{LD}(n, \epsilon)$ WCD rounds with probability at least $1 - \epsilon$, we conclude that all the $receive(*, 1)$ events occur by real time $B_{WCD}(n, \rho_{LD}(n, \epsilon) + 2)$ with probability at least $1 - \epsilon$. In other words, the first round of the SCD channel terminates by real time $B_{WCD}(n, \rho_{LD}(n, \epsilon) + 2)$ with probability at least $1 - \epsilon$.

After the first round of the probabilistic SCD channel has terminated, all future rounds require exactly two rounds of the underlying WCD channel. Therefore, each SCD-channel round r terminates by real time $B_{WCD}(n, \rho_{LD}(n, \epsilon) + 2r)$ with probability at least $1 - \epsilon$. Thus, we have shown that the SCD channel is β_{SCD}-time-bounded where $\beta_{SCD}(n, r, \epsilon) = B_{WCD}(n, \rho_{LD}(n, \epsilon) + 2r)$. □

Corollary 7. *Algorithm 2 implements a β_{SCD}-time-bounded probabilistic SCD on top of a B_{WCD}-time-bounded WCD channel and a ρ_{LD}-round-bounded probabilistic LD service where $\beta_{SCD}(n, r, \epsilon) = B_{WCD}(n, \rho_{LD}(n, \epsilon) + 2r)$.*

4.3 Loneliness Detection Using Leader Election

Implementing LD on top of a WCD system augmented with a solution to LE is straightforward. Given a solution to LE, solving LD takes just one additional round as follows: First, all the processes elect a leader with the assumed solution to LE. In the next round of the assumed WCD system, (1) every process that is not the leader transmits, outputs $false$, and then halts; (2) the leader outputs $true$ iff it receives $\perp$ at the end of this round, and outputs $false$ otherwise. The correctness of the reduction is straightforward.

5 Algorithms and Lower Bounds for Loneliness Detection

In this section, we explore algorithms and lower bounds for Loneliness Detection. As shown in Section 4.2.1, LD can be solved in SCD systems in a single round, which gives us a trivial tight bound of $\Theta(1)$. The remainder of this section focuses on LD in WCD systems.

5.1 Upper Bounds for Loneliness Detection in Weak Collision Detection Systems

We establish upper bounds for LD in WCD systems by presenting two algorithms, one deterministic and the other randomized, that solve the LD problem. The deterministic algorithm solves the deterministic LD problem in $\mathcal{O}(\log \frac{u}{n})$ rounds when $n > 1$ and in $\mathcal{O}(\log(u))$ rounds when $n = 1$, whereas the randomized algorithm solves the probabilistic LD problem in $\mathcal{O}(\frac{\log(1/\epsilon)}{n-1})$ rounds with probability $1 - \epsilon$ where $\epsilon \in (0, 1]$

when $n > 1$. We then combine the deterministic and the randomized algorithms to solve probabilistic LD for all values of n, including $n = 1$. For the lower bound, we show that termination of any solution to the LD problem cannot be guaranteed in fewer than $\log(u) - \log(n-1)$ rounds for $n > 1$ and in fewer than $\log(u)$ rounds for $n = 1$.

The algorithms presented in this section divide the execution into phases. Each phase consists of two rounds of the WCD channel: a Transmit round and an Ack round. If a nonempty set T of processes send a message in the Transmit round, then the processes in $J \setminus T$ transmit in the Ack round to indicate that at least one message was sent in the Transmit round.[2] It is easy to see that for each process i, either sending a message, receiving a message, or receiving a collision notification in the Ack round is a sufficient basis for i to conclude that $n > 1$. In order to ascertain that $n = 1$, the single process waits for an adequate number of phases in which the Ack rounds are silent. We describe the two algorithms in detail next.

5.1.1 Deterministic Algorithm for LD: Bitwise Separation Protocol (BSP).

BSP solves the deterministic LD problem in WCD systems in $\mathcal{O}(\log \frac{u}{n})$ rounds. The algorithm is as follows. Let the ID of each process i be represented as a sequence of bits denoted id_i; since the ID space is of size u, the sequence is $\lceil \log(u) \rceil$ bits long.[3] Starting from the least significant bit, number the bits from 1 to $\lceil \log(u) \rceil$, and let $id_i[k]$ denote the k-th bit of process i's ID. Let $T_k = \{i \in J : id_i[k] = 1\}$. The algorithm proceeds in phases, each phase consisting of a Transmit round and an Ack round. Initially, no process is halted.

In the Transmit round of the k-th phase, exactly the processes in T_k that have not yet halted transmit a message. In the Ack round, if a process $i \in J \setminus T_k$ that has not halted receives either a message or $\top$ in the Transmit round, then i sends an "ack" message; furthermore, processes in T_k do not send a message in the Ack round. If a process $i \in J$ that has not yet halted either sends or receives an "ack" message or receives $\top$ in the Ack round of a given phase k, then i outputs $alone(false, 2k)_i$ and halts.

If $\perp$ is received in all the Ack rounds, then the algorithm terminates at the end of $2\lceil \log u \rceil$ rounds and the lone process (say) j outputs $alone(true, 2\lceil \log u \rceil)_j$. The pseudocode is shown in Algorithm 3.

Algorithm 3 Bitwise Separation Algorithm

```
Let $id[1 \ldots \lceil \log(u) \rceil]$ denote the sequence of bits of a process $i$ where $id[1]$ denotes the least significant bit.
Let $m \in \mathcal{M}$ denote some special message that $i$ sends to signal its presence in the system.
Execution proceeds in phases. Each phase consists of two rounds: a Transmit round and an Ack round.
Process $i$ executes the following:
    for $k$ := 1 to $\lceil \log(u) \rceil$
        Transmit round:
            if $(id[k] = 1)$ then perform $send(m, 2k-1)_i$
            else perform $send(\perp, 2k-1)_i$
            wait until $receive(m', 2k-1)_i$
        Ack round:
            if $(id[k] \neq 1$ and $m' \neq \perp)$ then perform $send(\text{"}ack\text{"}, 2k)_i$
            else perform $send(\perp, 2k)_i$
            wait until $receive(m'', 2k)_i$
            if $(m'' \neq \perp)$ then perform $alone(false, 2k)_i$; halt.
    endfor
    perform $alone(true, 2\lceil \log(u) \rceil)_i$
```

[2] Recall that J is the set of processes comprising the system.

[3] Note that id_i may be represented as a natural number from the set $[0, u-1]$.

In order to show that BSP solves deterministic LD, we demonstrate that BSP satisfies the safety and deterministic liveness properties of LD. For the proof of correctness, we define a *failed phase* as follows: a phase k is said to be a failed phase if in the Transmit round of phase k, all the processes in the system that have not yet halted, perform the same action (either transmit or remain silent). A phase that is not failed is called a *non-failed phase*.

Proposition 8. *In BSP, in any execution, for every process i there exists exactly one $alone()_i$ event.*

Proof. The proof follows from the pseudocode in Algorithm 3 in which every process i eventually outputs an $alone()_i$ event and immediately halts. □

Lemma 9. *In any execution of BSP, for each k, $1 \leq k < \lceil \log u \rceil$, let H denote the set of processes that have not halted by the start of phase k; if phase k is a failed phase, then no process in H halts at phase k.*

Proof. Since phase k is a failed phase, either all the processes in H transmit or all the processes in H remain silent in the Transmit round of phase k. Furthermore, processes not in H remain silent in the Transmit round of phase k. Thus, at the end of the Transmit round, all the processes that are not halted and are listening in that round receive silence. From the pseudocode, we see that in the following Ack round, no process sends an "ack" message, and therefore, no process in H halts by phase k. □

Lemma 10. *In any execution of BSP, for any phase k, let H denote the set of processes have not halted by the start of phase k; if phase k is a non-failed phase, then all the processes in H halt by the end of phase k by outputting events $alone(false, 2k)_*$.*

Proof. Recall that in a non-failed phase, some process in H transmits in the Transmit round whereas some process in H listens in the Transmit round. Therefore, in phase k some process (say) $i \in H$ transmits in the Transmit round whereas some other process (say) $j \in H$ listens in the Transmit round. From the pseudocode, we see that j receives either a message or a collision notification by the end of the Transmit round, and therefore sends an "ack" message in the Ack round. Hence, by the end of the Ack round, all the processes in H receive either an "ack" message or a collision notification; from the pseudocode, we see that each process $i \in H$ halts at the end of such an Ack round and outputs $alone(false, 2k)_i$. □

Let $K = \{k \in \mathbb{N}^+ : (\exists i, j \in J)(id_i[k] = 1 \wedge id_j[k] = 0)\}$, and if $K \neq \emptyset$, then let $\hat{k} = \min(K)$. That is, $\hat{k}$ denotes the smallest bit position in which the IDs of two processes in the system differ.

Lemma 11. *In BSP, if K is nonempty and $\hat{k} = \min(K)$, then in an (arbitrary) execution α of the system, every phase smaller than $\hat{k}$ is failed, no process halts before phase $\hat{k}$, and phase $\hat{k}$ is non-failed.*

Proof. By construction, $\hat{k} \leq \lceil \log u \rceil$; the first $\hat{k} - 1$ bits of the IDs all the processes in the system are identical; and the $\hat{k}$-th bit of some two processes in the system differ. Next, we prove the following claim: for any k, $1 \leq k < \hat{k}$, phase k is a failed phase and no process halts at phase k.

If $\hat{k} = 1$, then the claim is vacuously true. We assume $\hat{k} > 1$.

The proof of the claim is by strong induction on k. For the inductive hypothesis, we fix k, where $1 \leq k < \hat{k}$, and for each k', $1 \leq k' < k$, we assume that phase k' is a failed phase and no process halts at phase k'. We show that the inductive hypothesis implies that phase k is a failed phase and no process halts at phase k.

We consider two cases: (1) $k = 1$, and (2) $k > 1$. Let $k = 1$. Since the first bit of the process IDs of all the processes are the same, all the processes either transmit or all the processes remain silent in the first

Transmit round. Thus, by definition, phase 1 is a failed phase. From Lemma 9 we know that no process halts at the end of phase 1.

In the case where $1 < k < \hat{k}$, the k-th bits of the IDs of all the processes are identical. From the inductive hypothesis we know that no process halts prior to phase k. Therefore, in phase k, either all the processes transmit or all the processes remain silent in the Transmit round. Thus, by definition, phase k is a failed phase, and from Lemma 9 we know that no process halts at the end of phase k.

It follows by strong induction that for any k, $1 \leq k < \hat{k}$, phase k is a failed phase and no process halts at phase k.

Therefore, at the start of phase $\hat{k}$, no process is halted. By definition, there exists some pair of processes i and j such that $id_i[\hat{k}] = 1$ and $id_j[\hat{k}] = 0$. Therefore, in the Transmit round of phase $\hat{k}$, process i transmits a message whereas process j listens in the Transmit round. That is, phase $\hat{k}$ is a non-failed phase. □

Lemma 12. *In BSP, if an $alone(true, *)_*$ event occurs in an execution α, then there is exactly one process in the system; that is, $n = 1$.*

Proof. Consider any process i. Assume that the event $alone(true, r)_i$ occurs in α. From the pseudocode in Algorithm 3, we see that event $alone(true, r)_i$ occurs only if $r = 2\lceil \log u \rceil$. For the purposes of contradiction, we assume that $n > 1$.

Consider the set K defined above. Since $n > 1$ and all processes have unique IDs, we know that $K \neq \emptyset$. Let $\hat{k} = \min(K)$. We know that $\hat{k} \leq \lceil \log u \rceil$.

From Lemma 11 we know that phases 1 through $\hat{k} - 1$ are failed phases, no process halts in phases 1 through $\hat{k} - 1$, and phase $\hat{k}$ is a non-failed phase. Applying Lemma 10, we see that all the processes halt in round $2\hat{k}$ and output events $alone(false, 2\hat{k})_*$. However, given that event $alone(true, r)_i$ occurs in α, this contradicts Proposition 8 which states that for each process i exactly one $alone()_i$ event occurs. Therefore, our assumption that $n > 1$ is false. □

Lemma 13. *In BSP, if an $alone(false, *)_*$ event occurs in an execution α, then the system contains more than one process, that is, $n > 1$.*

Proof. Assume that for some process i, event $alone(false, *)_i$ occurs in α. By the pseudocode, this event occurs in an Ack round, say round $2r$. This implies that i sends or receives an "*ack*" message or receives a collision notification in round $2r$. This implies that some message is received by a listening process in the immediately-preceding Transmit round, which in turn implies that there are at least two processes in the system. □

Lemma 14. *In BSP, in any execution, if two events $alone(*, r)_i$ and $alone(*, r')_j$ occur for some pair of processes i and j, then $r = r'$.*

Proof. We fix processes i and j, $i \neq j$, and rounds r and r', such that events $alone(*, r)_i$ and $alone(*, r')_j$ occur. Since $n > 1$, Lemma 13 implies that events $alone(false, r)_i$ and $alone(false, r')_j$ occur.

From the pseudocode, we know that that in round r, $m_i^{''} \neq \perp$. That is, i either sends or receives an "*ack*" message or receives a collision notification in round r. From the properties of the WCD channel we know that all the other processes either send or receive an ack message or receive a collision notification in round r; therefore, event $alone(false, r)_j$ occurs as well. Applying Proposition 8, we conclude that $alone(*, r')_j$ and $alone(false, r)_j$ are the same event. Thus, the lemma is proved. □

Recall that u denotes the size of the ID space for processes in the system, and n denotes the number of processes in the system; naturally $n \leq u$.

Lemma 15. *In BSP, in any execution α, if $n = 1$, then for each process i, some $alone()_i$ event occurs in round $2\lceil\log(u)\rceil$, and if $n > 1$, then for each process i, some $alone()_i$ event occurs in at most $2(\lceil\log u\rceil - \lceil\log n\rceil + 1)$ rounds.*

Proof. Suppose that $n = 1$ and the lone process in the system is i. From Proposition 8 we know that at most one $alone()_i$ event occurs; from the pseudocode, we know that at least one $alone()_i$ event occurs; that is, exactly one $alone()_i$ event occurs. Applying the contrapositive of Lemma 13, we see that the lone $alone()_i$ event cannot be an $alone(false, *)_i$ event; hence, the an $alone(true, *)_i$ event occurs. From the pseudocode, we see that event $alone(true, 2\lceil\log(u)\rceil)_i$ occurs in round $2\lceil\log(u)\rceil$.

Alternatively, suppose that $n > 1$. Consider the set K and $\hat{k} = \min(K)$ described earlier. Since representing unique IDs among n processes requires $\lceil\log n\rceil$ bits, $|K| \geq \lceil\log n\rceil$. Since each process ID in the system is of length $\lceil\log u\rceil$ bits, we infer that $\hat{k} \leq \lceil\log u\rceil - \lceil\log n\rceil + 1$. From Lemma 11 we know that phases 1 through $\hat{k} - 1$ are failed phases, no process halts at phases 1 through $\hat{k} - 1$, and phase $\hat{k}$ is a non-failed phase. Applying Lemma 10, we see that all the processes halt in round $2\hat{k}$ and output events $alone(false, 2\hat{k})_*$. Since, $\hat{k} \leq \lceil\log u\rceil - \lceil\log n\rceil + 1$, we know that if $n > 1$, BSP for each process i, some $alone()_i$ event occurs in at most $2(\lceil\log u\rceil - \lceil\log n\rceil + 1)$ rounds. □

Theorem 16. *BSP solves the deterministic LD problem and is R_{LD}-round-bounded where $R_{LD}(1) = 2\lceil\log(u)\rceil$ and for all n, where $1 < n \leq u$, $R_{LD}(n) = 2(\lceil\log u\rceil - \lceil\log n\rceil + 1)$.*

Proof. The safety properties are proved in Proposition 8, Lemmas 12, 13 and 14, and the deterministic liveness property is proved in Lemma 15. Also, from Lemma 15 we see that for each process i, some $alone()_i$ event occurs in round $2\lceil\log(u)\rceil$ if $n = 1$ and in at most $2(\lceil\log u\rceil - \lceil\log n\rceil + 1)$ rounds if $n > 1$. That is, BSP is R_{LD}-round-bounded where $R_{LD}(1) = 2\lceil\log(u)\rceil$ and for n where $1 < n \leq u$, $R_{LD}(n) = 2(\lceil\log u\rceil - \lceil\log n\rceil + 1)$. □

Corollary 17. *If the underlying physical system is a WCD system in which the duration of each round is 1 real-time unit, then BSP solves the deterministic LD problem in at most $2(\lceil\log u\rceil - \lceil\log n\rceil + 1)$ real-time units if $n > 1$ and in $2\lceil\log(u)\rceil$ real-time units if $n = 1$.*

5.1.2 Random Separation Protocol (RSP).

The Random Separation Protocol solves probabilistic LD in WCD systems consisting of 2 or more processes. If $n > 1$, then RSP verifies that $n > 1$ in $2\frac{log(1/\epsilon)}{n-1}$ rounds with probability at least $1 - \epsilon$ where $\epsilon \in (0, 1]$. However, if $n = 1$, then RSP does not terminate. We use RSP to solve probabilistic LD for all values of n in Section 5.1.3.

The protocol is identical to BSP except that IDs in RSP are infinite-bit strings in which the bits are chosen independently and uniformly at random. For each process i, let the infinite-bit ID of process i be denoted id_i, and let $id_i[k]$ denote the k-th bit of id_i. In each phase k, if $id_i[k] = 1$, then i transmits in the Transmit round; otherwise i listens in the Transmit round. It can be verified easily that RSP terminates in the first phase k in which for some pair of processes i and j, $id_i[k] \neq id_j[k]$. The pseudocode is shown in Algorithm 4.

Next, we show that RSP verifies if $n > 1$ in $2\frac{log(1/\epsilon)}{n-1}$ rounds with probability at least $1 - \epsilon$ (where $\epsilon \in (0, 1]$).

Theorem 18. *In RSP, if $n > 1$ and $\epsilon \in (0, 1]$, then for every process i, the event $alone(false, r)_i$ occurs within the first $\frac{log(1/\epsilon)}{n-1}$ phases; that is, $r \leq 2\frac{log(1/\epsilon)}{n-1}$, with probability at least $1 - \epsilon$.*

Algorithm 4 Random Separation Protocol.

```
Let $m \in \mathcal{M}$ denote some special message that $i$ sends to signal its presence in the system.
Execution proceeds in phases. Each phase consists of two rounds: a Transmit round and an Ack round.
Process $i$ executes the following:
    for $k$ := 1 to $\infty$
        toss an unbiased coin $p \in \{0, 1\}$ to determine the $k$-th bit of the ID
        Transmit round:
            if $p = 1$ then perform $send(m, 2k-1)_i$
            else perform $send(\perp, 2k-1)_i$
            wait until $receive(m', 2k-1)_i$
        Ack round:
            if ($p = 0$ and $m' \neq \perp$) then perform $send(``ack", 2k)_i$
            else perform $send(\perp, 2k)_i$
            wait until $receive(m'', 2k)_i$
            if ($m'' \neq \perp$) then perform $alone(false, 2k)_i$; halt.
    endfor
```

Proof. Assume $n > 1$. Recall the definition of a failed phase from Section 5.1.1: a phase k is said to be a failed phase if in the Transmit round of phase k, all the processes in the system that have not halted perform the same action (either transmit or remain silent). A phase that is not failed is called a *non-failed phase*.

Applying Lemma 9 (originally for BSP) to RSP we see that RSP does not terminate until the first non-failed phase, and applying Lemma 10 (originally for BSP) to RSP we see that RSP terminates (outputting events $alone(false, *)_*$) right after the first non-failed phase.

Since RSP uses an unbiased coin, for each phase k and each process i, the probability of i transmitting in the Transmit round of phase k is exactly $\frac{1}{2}$. Applying this observation to all the processes, and considering the two cases where either all the processes transmit or all the processes do not transmit in a given Transmit round, we see that the probability that a given phase is a failed phase is $2 \cdot (\frac{1}{2})^n = (\frac{1}{2})^{n-1}$.

Therefore, for any given $\epsilon \in (0, 1]$, the probability that RSP does not terminate in $\frac{log(1/\epsilon)}{n-1}$ phases is the probability that all of the first $\frac{log(1/\epsilon)}{n-1}$ phases are failed which is $((\frac{1}{2})^{n-1})^{(\frac{log(1/\epsilon)}{n-1})} = \epsilon$. □

By substituting $r = 2\frac{\log(1/\epsilon)}{n-1}$ in Theorem 18 we get the following corollary.

Corollary 19. *In RSP, for every process i, an $alone()_i$ event occurs within r rounds with probability at least $1 - (\frac{1}{2})^{\frac{r(n-1)}{2}}$.*

Often, the error probability ϵ is expressed as functions of n or u; common values of ϵ are 2^{-n} and 2^{-u}. Let $\epsilon = 2^{-n}$; we substitute the value of ϵ in Theorem 18 and obtain the following result.

Corollary 20. *In RSP, if $n > 1$, then for every process i the event $alone(false, r)_i$ occurs within the first $2\frac{n}{n-1}$ rounds with probability at least $1 - 2^{-n}$.*

Thus, for $n > 1$, the number of rounds within which RSP terminates with high probability is always at most 4, and as n increases, it converges to 2.

Similarly, if we substitute $\epsilon = 2^{-u}$ in Theorem 18, we get:

Corollary 21. *In RSP, if $n > 1$, then for every process i the event $alone(false, r)_i$ occurs within the first $2\frac{u}{n-1}$ rounds with probability at least $1 - 2^{-u}$.*

5.1.3 Combined Separation Protocol (CSP).

Even though RSP terminates in fewer rounds than BSP with high probability, it terminates only if $n > 1$. In other words, RSP can verify that a process is not alone, but it cannot confirm that a process is alone. We can overcome this problem with the Combined Separation Protocol (CSP) which interleaves RSP and BSP such that we execute BSP in the odd rounds and RSP in the even rounds; CSP terminates when either BSP or RSP terminates.

Theorem 22. *CSP solves probabilistic LD on WCD systems and is ρ_{LD}-round-bounded where*

$$\rho_{LD}(n,\epsilon) = \begin{cases} 4\lceil \log u \rceil & \textit{if } n = 1, \\ 4\min\left((\lceil \log u \rceil - \lceil \log n \rceil + 1), \frac{\log(1/\epsilon)}{n-1}\right) & \textit{if } n > 1. \end{cases}$$

Proof. The correctness follows from Theorems 16 and 18. The remainder of this proof demonstrates that the LD service is ρ_{LD}-round-bounded.

We consider two cases: (1) $n = 1$, and (2) $n > 1$.

Case (1) If $n = 1$, then we know from Lemma 15 that BSP terminates in $2\lceil \log u \rceil$ rounds. Since BSP runs in odd numbered rounds, CSP terminates in at most $4\lceil \log u \rceil$ rounds.

Case (2) If $n > 1$, the following arguments hold. Since BSP runs in the odd rounds of the CSP algorithm, and from Lemma 15 we know that BSP terminates in at most $2(\lceil \log u \rceil - \lceil \log n \rceil + 1)$ rounds, therefore, CSP terminates with probability 1 in $4(\lceil \log u \rceil - \lceil \log n \rceil + 1)$ rounds.

Also, from Theorem 18 we know that RSP terminates in $2\frac{log(1/\epsilon)}{n-1}$ rounds with probability at least $1 - \epsilon$. Since RSP runs in even numbered rounds of CSP, for any given $\epsilon \in (0, 1]$, CSP terminates in $4\frac{log(1/\epsilon)}{n-1}$ rounds with probability at least $1 - \epsilon$. □

By substituting $r = 4\frac{\log(1/\epsilon)}{n-1}$ in Theorem 22 we get the following corollary.

Corollary 23. *If the underlying physical system in a WCD systems in which the duration of each round is* 1 *real-time unit, then CSP solves probabilistic LD in at most* $4(\lceil \log u \rceil - \lceil \log n \rceil + 1)$ *real-time units with probability at least* $(1 - 2^{-\frac{r(n-1)}{4}})$ *if* $n > 1$ *and in at most* $4(\lceil \log u \rceil - \lceil \log n \rceil + 1)$ *real-time units with probability* 1 *if* $n = 1$.

5.2 Lower Bounds for Loneliness Detection in Weak Collision Detection Systems

In this section, we present lower bounds for both probabilistic and deterministic LD problems in WCD systems. For the probabilistic case, in Theorem 26, we show that for any $n > 1$, and any $\epsilon \in (0, 1]$, no algorithm that solves probabilistic LD can guarantee termination in fewer than $\min\left(\frac{\log(\frac{1}{\epsilon})}{n}, \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 1\right)$ rounds with probability greater than $1 - \epsilon$. That is, for any such n and ϵ, and for any algorithm A that solves probabilistic LD, there exists a set of processes, J_{LB}[4], where $|J_{LB}| = n$, such that the probability of A terminating within $\min\left(\frac{\log(\frac{1}{\epsilon})}{n}, \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2\right)$ rounds, when A is run on the system consisting of all processes in J_{LB}, is at most $1 - \epsilon$. We also show in Theorem 26 that for $n = 1$, no algorithm that solves probabilistic LD can guarantee termination in fewer than $\lceil \log(u) \rceil - 1$ rounds. For the deterministic case and $n > 1$, we show in Corollary 27 that no algorithm that solves deterministic LD can guarantee termination in fewer than $\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 1$ rounds. These results show that time complexities of algorithms BSP and CSP, presented in Theorems 16 and 22, respectively, match the lower bounds.

[4] The subscript LB in J_{LB} stands for *lower bound.*

Lemma 24. *For any algorithm A that solves probabilistic LD in WCD systems, any $n > 1$, and any positive integer $r \leq \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2$, there exists a set J_{LB} of n processes, such that the probability of A not terminating within r rounds, when A is run on the system consisting of all the processes in J_{LB}, is at least $(\frac{1}{2})^{rn}$.*

Proof Sketch. For each process i, we consider executing A on system S_i consisting of only process i. Consider the probability space of all admissible executions of S_i, and denote the probability of events in this space Pr_i. For each such execution and each round $z \geq 1$, define $trans_i(z)$ to be $true$ if i transmits in round z, and $false$ otherwise. We define the boolean function dtd_i (*dominating transmission decision*) on $\{1, ..., r\}$ recursively. $dtd_i(1)$ is assigned the value that is more likely to be taken by $trans_i(1)$, *i.e.*, $dtd_i(1) = true$ iff $Pr_i\{trans_i(1) = true\} \geq \frac{1}{2}$. Let $DTD_{i,1}$ denote the event in the probability space Pr_i that $trans_i(1) = dtd_i(1)$. For each $z \geq 2$, we define $dtd_i(z)$ to be the value that is more likely to be taken by $trans_i(z)$, conditioned on $DTD_{i,z-1}$, *i.e.*, $dtd_i(z) = true$ iff $Pr_i\{trans_i(z) = true | DTD_{i,z-1}\} \geq \frac{1}{2}$.

Recall that r is a positive integer and $r \leq \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2$. Since for each process i and each round $z \geq 1$, there are two possible values for $dtd_i(z)$, there are 2^r possible values for dtd_i sequences spanning r rounds. Since $2^r < \frac{u}{n-1}$, by the pigeonhole principle, there exists a set J_{LB} of n processes that have identical sequences of dominating transmission decisions, *i.e.*, $\forall i, j \in J_{LB}, dtd_i = dtd_j$. For each z, $1 \leq z \leq r$, let $cdtd(z)$ denote the common dominating transmission decision of the processes of J_{LB} in round z.

Let S be the system consisting of exactly the processes in J_{LB}. Consider an admissible execution α of system S. If, for each process $i \in J_{LB}$ and each round $z \leq r$ of α, $trans_i(z) = cdtd(z)$, then for each $i \in J_{LB}$ there exists an execution β_i in S_i such that i cannot distinguish α from β_i in the first r rounds.[5] However, in α, the only valid output is $alone(false, *)_*$ whereas in β_i, the only valid output is $alone(true, *)_*$. Hence, i cannot terminate within r rounds. By induction we show that the probability that for each $i \in J_{LB}$ and z, $1 \leq z \leq r$, $trans_i(z) = cdtd(z)$ is at least $(\frac{1}{2})^{rn}$. Hence, with probability at least $(\frac{1}{2})^{rn}$, A does not terminate within r rounds.

Proof. Assume for the sake of contradiction that there exists an algorithm A that solves probabilistic LD in WCD systems, there exists $n > 1$ and there exists $r \leq \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2$, such that for every set J_{LB} of n processes, the probability that A, when run on in a WCD system consisting of exactly the process in J_{LB}, terminates within r rounds is strictly greater than $1 - (\frac{1}{2})^{rn}$. We force a contradiction by showing that for the given values of n and r (assumed above) there exists an execution where A violates safety properties of the LD problem.

For each process i, we consider executing A in a system S_i consisting of only process i. Consider the space of all admissible executions of S_i. Denote the probability of events in this space Pr_i. For each such execution and each round z, $1 \leq z \leq r$, define the transmission decision of i in round z, denoted by the random variable $trans_i(z)$, to be equal to $true$ if i transmits in round z, and $false$ otherwise. We define the boolean function dtd_i, standing for *dominating transmission decision*, on $\{1, ..., r\}$ recursively. $dtd_i(1)$ is assigned the value that is more likely to be taken by $trans_i(1)$, i.e.,

$$dtd_i(1) = \begin{cases} true & \text{if } Pr_i\{trans_i(1) = true\} \geq \frac{1}{2}, \\ false & \text{otherwise.} \end{cases}$$

Let $DTD_{i,1}$ denote the event in the probability space Pr_i consisting of those executions in which $trans_i(1)$ $= dtd_i(1)$. For each z, $z \geq 2$, we define $dtd_i(z)$ to be the value that is more likely to be taken by $trans_i(z)$,

[5] Note the overloading of the random variable $trans_i(z)$. Here, $trans_i(z)$ denotes the transmission decision of process i at round z in system S. Earlier, $trans_i(z)$ denoted the same event in the system S_i.

conditioned on $DTD_{i,z-1}$, i.e.,

$$dtd_i(z) = \begin{cases} true & \text{if } Pr_i\{trans_i(z) = true | DTD_{i,z-1}\} \geq \frac{1}{2}, \\ false & \text{otherwise.} \end{cases}$$

$DTD_{i,z}$ denotes the event in the probability space Pr_i consisting of those executions in which, for every round r',$1 \leq r' \leq z$, $trans_i(r') = dtd_i(r')$.

Since for each process i and each round z, $1 \leq z \leq r$, there are two possible values for $dtd_i(z)$, there are 2^r possible values for a dtd_i sequence spanning r rounds. Since $2^r \leq 2^{\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2} < 2^{(\log(u)+1)-(\log(n-1)-1)-2} = \frac{u}{n-1}$, by the pigeonhole principle, there exists a set J_{LB} of n processes such that all processes in J_{LB} have identical sequences of dominating transmission decisions, i.e., for every pair of processes $i, j \in J_{LB}$ and each round z where $1 \leq z \leq r$, we have $dtd_i(z) = dtd_j(z)$. For each z, $1 \leq z \leq r$, let $cdtd(z)$ denote the common dominating transmission decision of the processes of J_{LB} in round z.

Let S denote the system consisting of exactly the processes in J_{LB}. Consider the space of all admissible executions of A in system S. We denote the probability of events in this space by Pr_S. For each z, $1 \leq z \leq r$, and each $i \in J_{LB}$, we define $C_{i,z}$ to be the event in this space where for each round r', $1 \leq r' \leq z$, $trans_i(r') = cdtd(r')$; naturally, $C_{i,z-1} \supseteq C_{i,z}$. Now, for every z, $1 \leq z \leq r$, define $G_z = \bigcap_{i \in J_{LB}} C_{i,z}$, and therefore, $G_{z-1} \supseteq G_z$.

Note that for each z, $1 \leq z \leq r$, for any execution in G_z, for each process $i \in J_{LB}$, the response that i receives from the channel for each round r', $1 \leq r' \leq z$ is equal to the message that i transmitted if $trans_i(r') = true$, and $\perp$ otherwise. Let Π_z denote the set of z-round prefixes of all executions in G_z.

Indistinguishability Claim: For every z, $1 \leq z \leq r$, for every process i in system S, for each z-round prefix $\alpha \in \Pi_z$, there exists a z-round prefix β_i of some execution of system S_i such that the execution is in $DTD_{i,z}$. In other words, for every z-round execution prefix $\alpha \in \Pi_z$, there exists a z-round execution prefix β_i such that process i cannot distinguish execution prefix α of system S from execution prefix β_i of system S_i. In effect, for the first r rounds, the actions at each process i in an execution of system S in G_r are identical to the actions at i in an execution of system S_i in $DTD_{i,r}$.

To get to the contradiction, we prove the following claim.

Claim. For each z, $1 \leq z \leq r$, $Pr_S\{G_z\} \geq (\frac{1}{2})^{zn}$.

Proof. The proof is by induction on z. For the base case, let $z = 1$. Note that each process $i \in J_{LB}$ is in the same initial state in systems S and S_i. By the definition of dominating transmission decision, we know that in system S_i, $Pr_i\{trans_i(1) = cdtd(1)\} \geq \frac{1}{2}$, i.e., $Pr_i\{DTD_{i,1}\} \geq \frac{1}{2}$. Since i makes the same random choices in both S and S_i in the first round, $Pr_S\{C_{i,1}\} = Pr_i\{DTD_{i,1}\}$, and hence, $Pr_S\{C_{i,1}\} \geq \frac{1}{2}$.

Note that the random choices of different processes in J_{LB} are independent of each other in the first round, and hence, for all $i \in J_{LB}$, the events $C_{i,1}$ are independent (and recall that for each such i, $Pr_S\{C_{i,1}\} \geq \frac{1}{2}$). Therefore, $Pr_S\{G_1\} = Pr_S\{\cap_{i \in J_{LB}} C_{i,1}\} \geq (\dfrac{1}{2})^n$.

For the inductive step, assume that for some z, $2 \leq z < r$, $Pr_S\{G_{z-1}\} \geq (\frac{1}{2})^{(z-1)n}$. By the definition of dominating transmission decision, we have the following: for each $i \in J_{LB}, Pr_i\{trans_i(z) = cdtd(z) | DTD_{i,z-1}\} \geq \frac{1}{2}$, and therefore, $Pr_i\{DTD_{i,z} | DTD_{i,z-1}\} \geq \frac{1}{2}$. Note that the probability distribution of random choices of process i in round z of executions in system S, conditioned on G_{z-1}, is identical to the probability distribution of random choices of process i in round z of executions in system S_i, conditioned on $DTD_{i,z-1}$. Therefore, $Pr_S\{C_{i,z} | G_{z-1}\} = Pr_i\{DTD_{i,z} | DTD_{i,z-1}\} \geq \frac{1}{2}$.

Since, for each pair of processes i and j, the events $C_{i,z}$ and $C_{j,z}$, conditioned on G_{z-1}, are independent, and $G_z = \cap_{i \in J_{LB}} C_{i,z}$, we obtain that $Pr_S\{G_z | G_{z-1}\} \geq (\frac{1}{2})^n$. Therefore, from the inductive hypothesis, and noting that $G_z \subseteq G_{z-1}$, we have $Pr_S\{G_z\} = Pr_S\{G_z | G_{z-1}\} \cdot Pr_S\{G_{z-1}\} \geq (\frac{1}{2})^{zn}$. Thus, we have proved that for each z, $1 \leq z \leq r$, $Pr_S\{G_z\} \geq (\frac{1}{2})^{zn}$. □

In particular, letting $z = r$, we see that $Pr_S\{G_r\} \geq (\frac{1}{2})^{rn}$. Now, recall that we assumed that for every set of processes with size n, the probability that A, when run on those n processes, terminates by r rounds is strictly greater than $1 - (\frac{1}{2})^{rn}$. Therefore, the probability that A, when run on system S, terminates by r rounds is strictly greater than $1 - (\frac{1}{2})^{rn}$. Since we know that $Pr_S\{G_r\} \geq (\frac{1}{2})^{rn}$, there must exist an execution γ of algorithm A in system S where γ terminates by r rounds and γ is in G_r. Let γ_r denote the prefix of γ that is r rounds in length. By definition, γ_r is in the set Π_r (which denotes the set of r-round prefixes of executions in G_r). Hence, we can apply the indistinguishability claim (from above) to γ_r and state that there exists an execution β_i in system S_i such that process i cannot distinguish γ from β_i up to the end of round r.

Since $n > 1$, we know from the properties of the LD problem that the output of each process $i \in J_{LB}$ in γ is equal to $alone(false, *)_i$. By assumption, the output event $alone(false, *)_i$ occurs within r rounds. Applying the indistinguishability claim, we see that the output event $alone(false, *)_i$ occurs in the execution β_i as well. However, since there is only one process in the system S_i, this output in execution β_i is incorrect and violates the safety property of the LD problem.

□

Lemma 25. *For $n = 1$, no algorithm that solves probabilistic LD in WCD systems guarantees termination in fewer than $\lceil \log(u) \rceil - 1$ rounds.*

Proof. The proof is by contradiction. Assume for the sake of contradiction that there exists an algorithm A that solves probabilistic LD in WCD systems and always terminates within $\lceil \log(u) \rceil - 2$ rounds when $n = 1$. We show that this leads to a contradiction by showing that there exists an execution where A violates the safety properties of the LD problem.

Let $r = \lceil \log(u) \rceil - 2$. For each process i, we consider executing A in a system S_i consisting of only process i, and let E_i be the set of all possible admissible executions of A in system S_i. Choose an arbitrary execution $\beta_i \in E_i$. Note that from the properties of the LD problem, the output event in β_i must be $alone(true, *)_i$. We define boolean function $trans_i$ over $\{1, ..., r\}$ as follows. For each z, $1 \leq z \leq r$, $trans_i(z)$ is equal to $true$ if i transmits in round z of β_i, and $false$ otherwise.

Since there are 2^r different possible sequences of values for each $trans_i$, and $2^r = 2^{\lceil \log(u) \rceil - 2} < 2^{\log(u) - 1} < u$, by the pigeonhole principle, there exist two processes j and j' such that for every z, $1 \leq z \leq r$, $trans_j(z) = trans_{j'}(z)$. Let S be the system consisting of processes j and j'.

Consider the execution α of system S where processes j and j', take exactly the same steps as they take in β_j and $\beta_{j'}$, respectively. We show that execution α is a valid execution for algorithm A in system S. First, we show for each z, $1 \leq z \leq r$, the execution prefix of α consisting of the first z rounds is a valid execution prefix for algorithm A in system S. Next, we show that α violates the safety properties of the LD problem.

We show that for each z, $1 \leq z \leq r$, the execution prefix of α consisting of the first z rounds is a valid execution prefix for algorithm A in system S, by induction on z. For the base case, let $z = 1$. It is clear that the execution prefix of α consisting of the first round is a valid execution in system S because the response that each process $i \in \{j, j'\}$ receives from the channel is its own transmitted message if $trans_i(1) = true$ and is $\perp$ otherwise. For the inductive step, assume that the execution prefix of α consisting of the first $z - 1$ rounds is a valid execution for algorithm A is system S. Since the response that each process $i \in j, j'$ receives from the channel is equal to its transmitted message if $trans_l(z) = true$ and is equal $\perp$ otherwise, the execution prefix of α consisting of the first z rounds is a valid execution in system S.

Note that j and j' cannot distinguish α from β_j and $\beta_{j'}$, respectively. Hence, in α, each process i outputs $alone(true, r)_i$ at the end of round r. However, this output is incorrect for the LD problem in S. $\square$

Theorem 26. *For any algorithm A that solves probabilistic LD in a WCD system, and any $n > 1$, there exists a set J_{LB} of n processes, such that, for any $\epsilon \in (0, 1]$, the probability that A terminates in $\min\big(\frac{\log(\frac{1}{\epsilon})}{n}, \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2\big)$ rounds, when run on the system consisting of all processes of J_{LB}, is at most $1 - \epsilon$. For $n = 1$, no algorithm solves probabilistic LD in WCD systems can guarantee termination in fewer than $\lceil \log(u) \rceil - 1$ rounds.*

Proof. For $n > 1$, let $r(\epsilon) = \min\big(\frac{\log(\frac{1}{\epsilon})}{n}, \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2\big)$. From Lemma 24 and since $r(\epsilon) \leq \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2$, there exists a set of processes, J_{LB}, where $|J_{LB}| = n$, such that the probability of A not terminating within $r(\epsilon)$ rounds, when A is run on the system consisting of all processes in J_{LB}, is at least $(\frac{1}{2})^{r(\epsilon)\, n}$, and is therefore at most $1 - (\frac{1}{2})^{\log(\frac{1}{\epsilon})} = 1 - \epsilon$.

Similarly, for $n = 1$, the proof follows directly from Lemma 25. $\square$

Corollary 27. *For $n > 1$, no algorithm that solves deterministic LD in WCD systems guarantees termination in fewer than $\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 1$ rounds. For $n = 1$, no algorithm that solves deterministic LD guarantees termination in fewer than $\lceil \log(u) \rceil - 1$ rounds.*

Proof. For the case where $n > 1$, assume, for the purpose of contradiction, that there exists an algorithm A that solves deterministic LD in WCD systems and guarantees termination in fewer than $\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 1$ rounds. Since every solution to deterministic LD is also a solution to probabilistic LD, we conclude that A solves probabilistic LD in WCD systems and guarantees termination in fewer than $\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 1$ rounds. However, this conclusion contradicts Lemma 24.

Similarly, for $n = 1$, the proof follows analogously from Lemma 25. $\square$

Corollary 28. *For $n > 1$, solving deterministic LD in WCD systems requires $\Omega(\log(u) - \log(n-1))$ rounds. For $n = 1$, solving deterministic LD requires $\Omega(\log(u))$ rounds.*

Proof. The proof follows from Corollary 27, directly. $\square$

Recall from Section 4.2.1 that solving deterministic LD is straightforward in SCD systems with just one round of communication. In Corollary 28, we showed that solving deterministic LD requires $\Omega(\log(u) - \log(n-1))$ rounds of communication in WCD systems if $n > 1$, and $\Omega(\log(u))$ rounds of communication in WCD systems if $n = 1$. Furthermore, we showed that SCD channels can be implemented on top of WCD systems using an LD service. Thus, in a precise sense, deterministic LD captures the difference between SCD and WCD systems.

5.3 Revisiting SCD on WCD Systems

Recall that in Section 4.2.2, we presented an implementation of an SCD channel on top of a WCD system using an LD service; in Section 5.1, we presented the BSP and CSP algorithms that solve deterministic and probabilistic LD, respectively, on top of a WCD system. Therefore, by replacing the LD service in Section 4.2.2 with the LD algorithms from Section 5.1, we should arguably be able to implement an SCD system on top of a WCD system without using any special auxiliary services.

In the following two theorems, we demonstrate this claim by showing that BSP and CSP can be used as the LD service for the construction presented in Section 4.2.2; we also study the time-bound functions of the resulting SCD channels. Let S_{BSP} denote the system which uses BSP (from Algorithm 3) as the LD service and executes Algorithm 2. Similarly, let S_{CSP} denote the system which uses CSP (from Section 5.1.3) as the LD service and executes Algorithm 2.

Theorem 29. *System S_{BSP} implements a B_{SCD}-time-bounded deterministic SCD channel on top of a B_{WCD}-time-bounded WCD system where $B_{SCD}(n, r) = B_{WCD}(n, \lceil \log u \rceil - \lceil \log n \rceil + 1 + 2r)$.*

Proof. In Theorem 16, we showed that BSP solves the deterministic LD problem in a WCD system. Consider an execution of BSP and suppose it terminates in round k, i.e., for each process i, an $alone(a_{LD}, k)_i$ event occurs. Note that BSP does not send any messages on the WCD channel after round k. Hence, Algorithm 2 uses the WCD channel after round k in isolation; that is, no other component within the system sends or receives messages on the WCD channel. Therefore, BSP can be used as an LD service in Algorithm 2.

In Theorem 16, we also showed that the BSP algorithm terminates in at most $\lceil \log u \rceil - \lceil \log n \rceil + 1$ rounds, i.e., $k \leq \lceil \log u \rceil - \lceil \log n \rceil + 1$. If we use BSP as the LD service in Algorithm 2, we know from Theorem 5 that Algorithm 2 implements an SCD channel on top of a B_{WCD}-time-bounded WCD system. Using Theorem 5, we conclude that a time-bound function B_{SCD} for the resulting SCD channel is given by $B_{SCD}(n, r) = B_{WCD}(n, \lceil \log u \rceil - \lceil \log n \rceil + 1 + 2r)$. □

Theorem 30. *System S_{CSP} implements a β_{SCD}-time-bounded probabilistic SCD channel on top of a B_{WCD}-time-bounded WCD system where*

$$\beta_{SCD}(n, r, \epsilon) = \begin{cases} B_{WCD}\big(n, 4\lceil \log u \rceil + 2r\big) & \textit{if } n = 1, \\ B_{WCD}\big(n, 4\min((\lceil \log u \rceil - \lceil \log n \rceil + 1), \frac{\log(1/\epsilon)}{n-1}) + 2r\big) & \textit{if } n > 1. \end{cases}$$

Proof. In Theorem 22, we showed that CSP solves the probabilistic LD problem in WCD systems. Consider an execution of CSP and suppose that it terminates in round k, i.e., for each process i, $alone(a_{LD}, k)_i$ event occurs. Similar to the proof in the deterministic case, note that CSP does not send any messages on the WCD channel after round k. Hence, Algorithm 2 uses the WCD channel after round k in isolation; that is, no other component within the system sends or receives messages on the WCD channel. Therefore, CSP can be used as an LD service in Algorithm 2. Also, from Theorem 22 we know that CSP implements a LD service and is ρ_{LD}-round-bounded where

$$\rho_{LD}(n, \epsilon) = \begin{cases} 4\lceil \log u \rceil & \text{if } n = 1, \\ 4\min((\lceil \log u \rceil - \lceil \log n \rceil + 1), \frac{\log(1/\epsilon)}{n-1}) & \text{if } n > 1. \end{cases}$$

If we use CSP as the LD service in Algorithm 2, we know from Theorem 6 that Algorithm 2 implements a β_{SCD}-time-bounded probabilistic SCD channel on top of a B_{WCD}-time-bounded WCD system where $\beta_{SCD}(n, r, \epsilon) = B_{WCD}(n, \rho_{LD}(n, \epsilon) + 2r)$. Hence, for the resulted SCD channel, we have

$$\beta_{SCD}(n, r, \epsilon) = \begin{cases} B_{WCD}\big(n, 4\lceil \log u \rceil + 2r\big) & \text{if } n = 1, \\ B_{WCD}\big(n, \min(4(\lceil \log u \rceil - \lceil \log n \rceil + 1), \frac{4\log(1/\epsilon)}{n-1}) + 2r\big) & \text{if } n > 1. \end{cases}$$

□

6 Algorithms and Lower Bounds for Leader Election

In this section, we study the LE problem in both SCD and WCD systems. For LE in SCD systems, we present a deterministic algorithm BLEP and a randomized algorithm CLEP in Section 6.1. The randomized algorithm CLEP we present in Section 6.1.2 is obtained by simply interleaving the algorithm by Nakano and Olariu [10] with our deterministic algorithm BLEP. We also present lower bounds for deterministic and probabilistic LE in SCD systems in Section 6.2 that match our upper bounds. For LE on WCD systems, we again present both deterministic and randomized algorithms in Section 6.3. For this purpose, we combine the LE algorithms for SCD systems with the algorithms of Section 5.3 that implement an SCD channel over a WCD channel.

We also argue that the lower bounds for LE in SCD systems hold for WCD systems as well, and then, we show that the time complexity of the aforementioned algorithms match the respective lower bounds.

6.1 Upper Bounds for LE in SCD Systems

In this section, we present two algorithms: Bitwise Leader Election Protocol (BLEP) and Combined Leader Election Protocol (CLEP). The former is a deterministic algorithm that solves deterministic LE in SCD systems. The latter is a randomized algorithm which interleaves BLEP and Nakano and Olariu's algorithm in [10] to solve probabilistic LE in SCD systems. Later, in Section 6.2, we present lower bounds for deterministic and probabilistic LE that match the upper bounds from BLEP and CLEP, respectively.

6.1.1 Deterministic Upper Bound: Bitwise Leader Election Protocol.

BLEP solves deterministic LE in SCD systems in $\mathcal{O}(\log u)$ rounds. Let the ID of each process i be represented as a sequence of bits denoted id_i such that some bit in the sequence id_i is 1.[6] Since the ID space is of size u, each such sequence is $\lfloor \log(u) \rfloor + 1$ bits long. Starting from the least significant bit, number the bits from 1 to $\lfloor \log u \rfloor + 1$, and let $id_i[k]$ denote the k-th bit of process i's ID.

In BLEP, every process is *active* when contending to be the leader, *inactive* when it is not; a Boolean variable $active$ denotes whether or not a process is active. Initially, all the processes are $active$. The execution proceeds from round 1 to round $\lfloor \log u \rfloor + 1$. In each round r, every process i that is $active$, and for which $id_i[r] = 1$, transmits its ID id_i; all other processes are silent in round r. At the end of round r, every process j receives some response from the SCD channel. If j receives a collision notification $\top$ at the end of round r, and j was $active$ but did not transmit its ID in round r (because $id_j[r] = 0$), then j ceases to be $active$ (becomes inactive) at the end of round r and therefore stops contending to be the leader. On the other hand, if the response at the end of round r is the ID of some process l, then j elects l as the leader, outputs $leader(l)_j$, and halts. If j receives $\perp$ at the end of round r, then j does nothing. The execution proceeds to round $r + 1$, and so on, until round $\lfloor \log u \rfloor + 1$. The pseudocode is shown in Algorithm 5.

Lemma 31. *In every execution of BLEP, for every process i, there exists at most one $leader()_i$ event.*

Proof. Follows from the pseudocode in Algorithm 5. □

Lemma 32. *In every execution of BLEP, if an event $leader(l)_i$ occurs for some process i, then l is the ID of some process in the system.*

[6] Note that id_i may be represented as a natural number from the set $[1, u]$. That is, the ID consisting of 0s for all the bits is not a valid ID in the ID space. The reason for such restriction is the following. In BLEP, a necessary condition for a process to be elected as the leader in round (say) k is that the k-th bit of it's ID be 1; therefore, if a process i were to have an all-0-bit ID, then it will never be elected leader. This becomes problematic in systems consisting of just process i.

Algorithm 5 Bitwise Leader Election Protocol

```
Let $id[1 \ldots \lfloor \log u \rfloor + 1]$ denote the sequence of bits of a process $i$ where $id[1]$ denotes the least significant bit.
We assume that at least one of bits in $id[1 \ldots \lfloor \log u \rfloor + 1]$ is 1.
Execution proceeds in rounds.
Process $i$ executes the following:
    $active := true$
    for $r$ := 1 to $\lfloor \log(u) \rfloor + 1$
        if ($active = true$ and $id[r] = 1$) then perform $send(id, r)$
        else perform $send(\perp, r)$
        wait until $receive(m', r)$
        if ($m' \in I$) then perform $leader(m')$; halt.
        if ($m' = \top$) then $active := (active)\&(id[r] = 1)$
    endfor
```

Proof. From the pseudocode in Algorithm 5, if an event $leader(l)_i$ occurs for some process i, then $l = m'$ at process i when the event occurs. Note that m' is some message received by i. Since every message sent in Algorithm 5 contains the ID of some process in the system, the lemma holds. □

Lemma 33. *In every execution of BLEP, for every pair of processes i and j and round r, if event $leader(l)_i$ occurs in round r, then event $leader(l)_j$ occurs in round r as well.*

Proof. Let round r' be the earliest round at the end of which for some process (say) i, a $leader(l)_i$ event occurs. From the pseudocode in Algorithm 5, we know that if event $leader(l)_i$ occurs at the end of round r', then $l = m'$ at process i where m' is some message received by i at the end of round r' and m' is the ID of some process in the system. From the properties of the SCD system, every other process j also receives m' at the end of round r'. Therefore, for every other process j, the event $leader(l)_j$ occurs at the end of round r'. In other words, for every pair of processes i and j and a round r, if event $leader(l)_i$ occurs in round r, then event $leader(l)_j$ occurs in round r as well. □

Lemma 34. *In every execution of BLEP, for every process i, some $leader()_i$ event occurs within $\lfloor \log u \rfloor + 1$ rounds.*

Proof. If there is exactly one process i in the system (that is, $n = 1$), then by construction we know that there exists some $r \leq \lfloor \log u \rfloor + 1$ such that $id_i[r] = 1$. Let r_{min} be the smallest such r. From the pseudocode, we see that a $leader()_i$ event occurs at the end of round r_{min}. Thus, the lemma is proved for $n = 1$. For the remainder of this proof, we assume $n > 1$.

For contradiction, assume that no $leader()$ event occurs by the end of round $\lfloor \log u \rfloor + 1$. Since processes have unique IDs, for each pair of distinct processes i and j, there exists some $r' \leq \lfloor \log u \rfloor + 1$ such that $id_i[r'] \neq id_j[r']$. Therefore, at the end of round r', it is not the case that both i and j remain active. Hence, at the end of $\lfloor \log u \rfloor + 1$ at most one process remains active. Thus, we establish the following claim.

Claim 1. At most one process is active at the end of round $\lfloor \log u \rfloor + 1$.

Note that if a process (say) i becomes inactive in a round (say) r, then there exists some other process (say) j which is active and transmits in round r. However, from the pseudocode, we see that such a process j remains active at the end of round r. Hence, in any execution, it is not possible for all the processes to become inactive. Thus, we establish the following claim.

Claim 2. At least one process is active at the end of round $\lfloor \log u \rfloor + 1$.

From Claims 1 and 2, we see that exactly one process is active at the end of round $\lfloor \log u \rfloor + 1$; let i denote that process. Consider the earliest round $r \leq \lfloor \log u \rfloor + 1$ at the end of which i is the only active process. By construction, multiple processes are active at the beginning of round r. For i to be the only process active at the end of round r, the following must be true: $id_i[r] = 1$, and for each process $j \neq i$ that is active in round r, $id_j[r] = 0$. Therefore, from the pseudocode, in round r, only i transmits its ID. From the properties of SCD channels, at the end of round r, all the processes receive i's ID. Therefore, for each process j in the system, $leader()_j$ event occurs in round $r \leq \lfloor \log u \rfloor + 1$. However, this contradicts our assumption that no $leader()$ event occurs by the end of round $\lfloor \log u \rfloor + 1$.

Therefore, we know that for some process i, a $leader()_i$ event occurs in round $r \leq \lfloor \log u \rfloor + 1$. Now, invoking Lemma 33 for process i, every other process j, and round r, we see that for every other process j, $leader()_j$ event occurs in round r. □

Theorem 35. *BLEP solves deterministic LE and is R_{LE}-round-bounded where $R_{LE}(n) = \lfloor \log(u) \rfloor + 1$.*

Proof. From Lemmas 31, 32, and 33, we know that BLEP satisfies the safety properties of LE, and from Lemma 34 we know that BLEP satisfies the deterministic liveness properties of LE, and BLEP terminates in at most $\lfloor \log u \rfloor + 1$ rounds. □

6.1.2 Probabilistic Upper Bound: Combined Leader Election Protocol

Nakano and Olariu [10] presented a randomized LE algorithm for SCD systems that terminates in $\mathcal{O}(1)$ rounds if $n = 1$, and in $\mathcal{O}(\log \log n + \log(\frac{1}{\epsilon}))$ rounds with probability at least $1 - \epsilon$, where $\epsilon \in (0, 1]$, if $n > 1$. To get a randomized LE algorithm for SCD systems with time complexity that matches the lower bound in Section 6.2, we interleave Nakano-Olariu's algorithm with BLEP and we call this algorithm CLEP. In CLEP, we execute BLEP in the odd rounds and Nakano-Olariu in the even rounds.

Theorem 36. *CLEP solves the probabilistic LE problem in SCD systems and is ρ_{LE}-round bounded where*

$$\rho_{LE}(n, \epsilon) = \begin{cases} \mathcal{O}(1) & \textit{if } n = 1, \\ \mathcal{O}(\min(\log u, \log \log n + \log(\frac{1}{\epsilon})) & \textit{if } n > 1. \end{cases}$$

Proof. The correctness of CLEP follows from the correctness of BLEP and Nakano-Olariu. The proof of ρ_{LE}-round-boundedness follows from Theorem 35 and Lemma 3.3 in [10]. □

6.2 Lower bounds for LE in both SCD and WCD Systems

In this section, we present lower bounds for deterministic and probabilistic LE in SCD systems in Theorems 38 and 41, respectively. These lower bounds match (respectively) the upper bounds presented in Theorems 35 and 36. Also, since SCD systems have stronger assumptions about the collision detection abilities of processes, these lower bounds hold for WCD systems as well. In order to demonstrate the lower bounds for LE, we consider a variant of LE, denoted *n-LE*, in which the number of nodes n in the system is known to all processes. The lower bounds that apply to n-LE apply to LE as well.

Lemma 37. *For any $n > 1$ and any algorithm A that solves probabilistic n-LE in SCD systems, there exist some set $J \subseteq I$ of processes where $|J| = n$ such that A, when run on J, is not guaranteed to terminate in fewer than $\lceil \log(u) \rceil - \lfloor \log(n - 1) \rfloor - 1$ rounds.*

Proof Sketch. Assume for contradiction that there exists an $n > 1$ and an algorithm A that solves probabilistic n-LE and always terminates within $\lceil \log u \rceil - \lfloor \log(n - 1) \rfloor - 2$ rounds. We construct executions of A for each process i in which i receives $\top$ from the channel in each round that it transmits and receives $\perp$ otherwise. Using techniques from the proof of Lemma 24, we show that in such executions on some set J_{LB} of n processes, each process $i \in J_{LB}$ elects itself as the leader. This violates the properties of n-LE and forces the contradiction.

Proof. Assume for the sake of contradiction that there exists an $n > 1$ and an algorithm A that solves probabilistic n-LE such that for any set $J \subseteq I$ of n processes, algorithm A, when run on J, always terminates within $\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2$ rounds.

Let $k = \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2$. For each process i, for each z, $1 \leq z \leq k$, a *transmission decision* $trans_i(z)$ is equal to $true$ if i transmits in round z, and $false$ otherwise. For each process i, the *leader decision* $leader_i$ is equal to $true$ if i elects itself the leader, and $false$ otherwise. Note that for each process i, $trans_i$ and $leader_i$ are functions of executions of A on any set of processes that includes i. For each execution of A, and for each process i in that execution, we define an *action sequence* to be a Boolean sequence of the form $a^i_1, a^i_2, \ldots, a^i_{k+1}$, where the first k elements are *transmission decisions* and the last element is the *leader decision*.

Any execution of A is said to be *totally failed* if, for each process i in the system, and each z, $1 \leq z \leq k$, if i transmits a message in round z, then i receives $\top$ from the channel, and i receives $\perp$ otherwise.

For each process i, we consider executions of A in a (fake) system S_i consisting of only process i such that each such execution is totally failed.[7] Let Pr_i denote the probability distribution on the space of all totally-failed admissible executions in system S_i. Let AS_i be the set of all action sequences for i associated with executions of S_i in Pr_i. Since for each z, $1 \leq z \leq k+1$, there are two possible values for a_z, $|AS_i| \leq 2^{k+1}$. Therefore, since $2^{k+1} = 2^{\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 2} < 2^{(\log(u)+1)-(\log(n-1)-1)-2} = 2^{\log(u)-\log(n-1)} \leq \frac{u}{n-1}$, from the pigeonhole principle, there are at least n processes that share some action sequence. That is, there exists a subset J_n of I, $|J_n| = n$ and $\cap_{i \in J_n} AS_i \neq \emptyset$. Let cas, standing for *c*ommon *a*ction *s*equence, be an arbitrary action sequence in $\cap_{i \in J_n} AS_i$. Let $cas[z]$ denote the value of the z-th element of cas. For each process i and the system S_i, let α_i denote a fixed admissible execution in S_i such that the action sequence associated with α_i for the first k steps and the final decision is equal to cas.

Now fix an SCD system S consisting of the processes in J_n. Let Pr_S denote the probability distribution on the space of all admissible executions of algorithm A in system S. Recall that in all such executions, the algorithm terminates within k rounds. Next we show that in this space, there is some execution in which outputs of A are an incorrect solution for the n-LE problem. Specifically, we show that a sequence of steps α in which each process takes steps corresponding to the common action sequence cas is an execution.

Let each process $i \in J_n$ takes the same step in the first round of α as it does in the first round of α_i. Since process i is in the same initial state in systems S_i and S, the action associated with process i in α_i is allowed for process i in system S. Thus, in the first round, either all the processes transmit concurrently or all the processes are silent. Therefore, in α, the response process i receives from the SCD channel in the first round is identical to the response i receives in execution α_i.

Similarly, we see that each round z, where $1 \leq z < k$, of α may be constructed as follows: each process i takes the same step in round z of α as i does in round z of α_i. By the construction of α_i for each process i, we see that the step by i in round z is allowed from the preceding state of i. Since, the step by process i in round z follows $cas[z]$, the response from the SCD channel to process i in round z is $\top$ is i transmits in round z, and $\perp$ otherwise. Therefore, the state of process i at the end of round z is the same in α_i and α. Thus, the sequence α in which each process takes steps corresponding to the common action sequence cas is an execution.

However, note that in α, either all the processes elect themselves as the leader or no process elects itself as the leader and $n > 1$. Therefore, the final result of each process i is denoted by $cas[k+1]$: the outputs

[7] Note that in any SCD system consisting of just process i, the sole process never receives $\top$; consequently, we consider a fake system S_i where we assume that executions are always totally failed. We use such fake executions and demonstrate an indistinguishability (with respect to i) between these fake executions and real executions in a system containing more than one process (including process i). Such indistinguishability is what is used to establish the lower bound.

are an incorrect solution to the n-LE problem.

□

Theorem 38. *For any $n > 1$, no deterministic LE algorithm in SCD systems can guarantee terminating in fewer than $\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 1$ rounds.*

Proof. Note that deterministic n-LE algorithms are just a special case of randomized n-LE algorithms. Therefore, by Lemma 37, for any $n > 1$, no deterministic n-LE algorithm in SCD systems can guarantee terminating in fewer than $\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 1$ rounds. Thus, we know that no deterministic LE algorithm in SCD systems with $n > 1$, can guarantee terminating in fewer than $\lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor - 1$ rounds. □

Thus, we have demonstrated a lower bound for deterministic LE, and note that the lower bound matches the upper bound in Section 6.1.1 when $u \gg n$. Next, we determine a lower bound for probabilistic LE.

Theorem 39. *Let A be any 2-LE algorithm in SCD systems and suppose that $k < \lceil \log(u) \rceil - 1$ and $u \geq 2$. There exist two processes such that the probability of termination of A by the end of the k-th round, when A is run on the system of those two processes, is at most $1 - (\frac{1}{4})^{k+1}$.*

Proof Sketch. The proof structure is similar to that of Lemma 24. The key difference is the following. In the proof of Lemma 24 we considered some special executions of an LD algorithm A_{LD} in WCD systems with just one process and showed that such executions are locally indistinguishable from some (other) special executions of A_{LD} in a WCD system with a specific set of n processes. In SCD systems, such a construction is not feasible for the following reason. In WCD systems when a transmitting process always receives the same feedback from the channel; on the other hand, in SCD systems, a transmitting process could receive different feedback depending on whether or not a collision occurred. To circumvent this issue, we consider executions of A — a solution to 2-LE problem in SCD systems — in a fake scenario where a process receives $\top$ in every round that it transmits and receives $\bot$ in every round that it remains silent. We use such executions to demonstrate that with probability at least $(\frac{1}{4})^{r+1}$, A does not terminate in r rounds where $r < \lceil \log u \rceil - 1$.

Proof. The proof is by contradiction. Suppose that there exists an algorithm A that solves the 2-LE problem such that the probability of termination of A by the end of the round (say) k, where $0 < k < \lceil \log(u) \rceil - 1$, is strictly greater than $1 - (\frac{1}{4})^{k+1}$. We transform algorithm A to an algorithm B for the 2-LE problem that always terminates in k rounds but might have outputs that are incorrect solutions to the 2-LE problem. The transformation is as follows: algorithm B executes algorithm A for k rounds, if A terminates by the end of round k, B outputs the same leader IDs that A outputs, and if A does not terminate by the end of round k, each process outputs its own ID as the leader at the end of round k. Note that if A terminates by round k, the outputs of B are correct solutions for the 2-LE problem. Therefore, by the assumption that for any two processes, A terminates in at most k rounds with probability greater than $1 - (\frac{1}{4})^{k+1}$, we know that for any two processes, the outputs of B, when B is run on the system of those two processes, are correct solutions for the 2-LE problem with probability strictly greater than $1 - (\frac{1}{4})^{k+1}$. Next we show that this leads to a contradiction by showing that there exist two processes such that when B is run on the system of those two processes for k rounds, the probability that outputs of B are incorrect solutions to the 2-LE problem is at least $(\frac{1}{4})^{k+1}$.

For each process i, we define an *action sequence* for i to be a sequence of k ordered pairs of states with an output and an input after each state pair plus a final state and a leader ID at the end of the sequence: $(state_1, state'_1), out_1, in_1, (state_2, state'_2), out_2, in_2, \ldots, (state_k, state'_k), out_k,\ in_k,\ state_{k+1},\ leader.$

An action sequence starts with a state pair. For each z, $1 \leq z \leq k$, $state_z$ represents the state of process i at the start of round z round and $state'_z$ represents the state of process i after it has made the random choices for round z. For each $z \geq 1$, out_z represents the message that process i sends in round z. That is, $out_z \in M \cup \{\perp\}$ (where $out_z = \perp$ if i does not send any message in round z) and is a determined by $state'_z$. For each $z \geq 1$, in_z represents the response that process i receives from the channel in round z. That is, $in_z \in M \cup \{\perp, \top\}$ where $in_z = \perp$ if i receives silence in round z and $in_z = \top$ if i receives a collision notification in round z. Also, $state_{k+1}$ denotes the state of process i after k rounds and $leader$ denotes the ID that i outputs as the ID of the leader. In this model, $state_1$ denotes the (unique) initial state of process i. Also, for each $z \geq 2$, $state_z$ is determined by $state'_{z-1}$, out_{z-1} and in_{z-1}

A *totally-failed* sequence is an action sequence in which, for each z, $1 \leq z \leq r$, $in_z = \perp$ if $out_z = \perp$, and $in_z = \top$ otherwise. In totally-failed sequences, since in_{z-1} is determined by out_{z-1} which is in turn determined by $state'_{z-1}$, we see that $state_z$ is determined by just $state'_{z-1}$. Similarly, $leader$ is determined by $state_{k+1}$.

Now we define a probability distribution on the space of admissible executions corresponding to totally-failed sequences of i, and we denote this probability distribution by Pr_i. In each round r, if i is in state s_1 at the start of round r, then we denote the probability that process i is in state s_2 after making the random choices of this round by $p_{s_1}(s_2)$, $0 \leq p_{s_1}(s_2) \leq 1$. For each execution α in Pr_i, the probability that α occurs is the product of probabilities of transitions between the two entries of each of its state pairs, i.e.,

$$\prod_{z=1}^{k} \Big(p_{state_z}(state'_z) \Big).$$

For each totally-failed sequence at process i, we define a random variable *transmission decision*, $trans_i$ on $\{1, ..., k\}$ as follows. For each z, $1 \leq z \leq k$, if $out_z \neq \perp$, then $trans_i(z) = true$, and if $out_z = \perp$, then $trans_i(z) = false$. We define the Boolean function dtd_i, standing for *dominating transmission decision*, on $\{1, ..., k\}$ recursively. $dtd_i(1)$ is assigned the value that is more likely to be taken by $trans_i(1)$, i.e.,

$$dtd_i(1) = \begin{cases} true & \text{if } Pr_i\{trans_i(1) = true\} \geq \frac{1}{2}, \\ false & \text{otherwise.} \end{cases}$$

Let $DTD_{i,1}$ denote the event in the probability space Pr_i that $trans_i(1) = dtd_i(1)$. For each z, $z \geq 2$, we define $dtd_i(z)$ to be the value that is more likely to be taken by $trans_i(z)$, conditioned on $DTD_{i,z-1}$, i.e.,

$$dtd_i(z) = \begin{cases} true & \text{if } Pr_i\{trans_i(z) = true | DTD_{i,z-1}\} \geq \frac{1}{2}, \\ false & \text{otherwise.} \end{cases}$$

$DTD_{i,z}$ denotes the event in the probability space Pr_i consisting of those executions in which, for every round r', $1 \leq r' \leq z$, $trans_i(r') = dtd_i(r')$.

Recall that $leader$ denotes the ID that i outputs as the ID of the leader. We define the random variable $result_i$ to be $true$ if $leader = i$, and $false$ otherwise. We also define the dominating result of i, dr_i, to be the value that is more likely to be taken by $result_i$, conditioned on $DTD_{i,k}$, i.e.,

$$dr_i = \begin{cases} true & \text{if } Pr_i\{result_i = true | DTD_{i,k}\} \geq \frac{1}{2}, \\ false & \text{otherwise.} \end{cases}$$

Note that for any k, there are 2^{k+1} possibilities for a sequence of dominating transmission decisions through k rounds and the dominating result. Hence, if $k < \lceil \log(u) \rceil - 1$, there exist two processes that have identical sequences of dominating transmission decisions for the first k rounds and identical dominating results.

Choose one such pair (i, j) of distinct processes. Let $cdtd$ be the common dominating transmission decision sequence of i and j. Also, let cdr be the common dominating result of i and j. Fix a system S to be the SCD system consisting of i and j and executing algorithm B. We define an *execution-sequence* of system S to be a sequence of k ordered 4-tuples of states with a 2-tuple output and a 2-tuple input after each state 4-tuple plus a 2-tuple final state and a 2-tuple leader, e.g., $(state^i_1, state'^i_1, state^j_1, state'^j_1)$, (out^i_1, out^j_1), (in^i_1, in^j_1), $(state^i_2, state'^i_2, state^j_2, state'^j_2)$, (out^i_2, out^j_2), (in^i_2, in^j_2), $\ldots$, $(state^i_k, state'^i_k, state^j_k, state'^j_k)$, (out^i_k, out^j_k), (in^i_k, in^j_k), $(state^i_{k+1}, state^j_{k+1})$, $(leader^i, leader^j)$. An execution-sequence satisfies the following properties.

The sequence starts with a state 4-tuple. For each $z \geq 1$, and each $l \in \{i, j\}$, $state^l_z$ represents the state of the process l at the start of round z and $state'^l_z$ represents the state of process l after it has made its random choices for round z. For each such z and l, out^l_z represents the message that process l takes in round z and is determined by $state'^l_z$. That is, $out^l_z \in M \cup \{\bot\}$. For each such z and l, in^l_z represents the response that process l receives from the channel in round z and is determined by the properties of SCD channels and the actions out^i_z and out^j_z. That is, $in^l_z = M \cup \{\bot, \top\}$. Also, $state^l_{k+1}$ denotes the final state of process l after k rounds and $leader^l$ denotes the ID that l outputs as the ID of the leader. Here, $state^l_1$ denotes the (unique) initial state of process l. Also, for each $z \geq 2$ and $l \in \{i, j\}$, $state^l_z$ is determined by $state'^l_{z-1}$, out^l_{z-1} and in^l_{z-1}; however, since in^l_{z-1} is determined by $(out^i_{z-1}, out^j_{z-1})$ which is in turn determined by $state'^i_{z-1}$ and $state'^j_{z-1}$, we have that $state^l_z$ is determined by just $state'^i_{z-1}$ and $state'^j_{z-1}$. Also, $leader^l$ is determined by $state^l_{k+1}$.

Now we define a probability distribution on the space of executions of system S and denote this probability distribution by Pr_S. In each round and for each $l \in \{i, j\}$, if process l is in state s_1 at the start of this round, then we denote the probability that process l is in state s_2 after making the random choices of this round by $p^l_{s_1}(s_2)$, $0 \leq p^l_{s_1}(s_2) \leq 1$. For each execution-sequence β in system S, the probability that an execution corresponding to β occurs is

$$\prod_{z=1}^{k} \left(p^i_{state^i_z}(state'^i_z) \ \cdot \ p^j_{state^j_z}(state'^j_z) \right)$$

where ($state^i_z$, $state'^i_z$, $state^j_z$, $state'^j_z$) is the z-th state 4-tuple in β.

For each z, $1 \leq z \leq k$, and each $l \in \{i, j\}$, we define $C_{l,z}$ to be the the set of executions for which in each round r', $1 \leq r' \leq z$, $trans_l(r') = cdtd(r')$; naturally, $C_{l,z-1} \supseteq C_{l,z}$. We also state that an execution is in $C_{l,k+1}$ if the execution is in $C_{l,k}$ and $dr_l = cdr$. Now, for every z, $1 \leq z \leq k+1$, define $G_z = C_{i,z} \cap C_{j,z}$, and therefore, $G_{z-1} \supseteq G_z$.

Note that for each z, $1 \leq z \leq k$, if $trans_i(z) = cdtd(z)$ and $trans_j(z) = cdtd(z)$, then input that i and j receive from the channel in round z is in the form stated for the totally-failed sequences, i.e., for each $l \in \{i, j\}$, in^l_z is equal to $\bot$ if $trans_l(z) = \bot$, and $\top$ otherwise. Thus, we establish the following claim.

Failure-Round claim. If an execution is in G_z, then the corresponding z-round prefix of the action sequences for i and j are z-round prefixes of a totally-failed sequence.

Now, in order to get to a contradiction, we prove that the probability of the event in the probability space Pr_S that the leader IDs output in system S are incorrect solutions to the 2-LE problem is at least $(\frac{1}{4})^{k+1}$. Note that in every execution in G_{k+1}, processes output their own IDs as the leader, and therefore, the outputs are an incorrect solution to the 2-LE problem. Hence, for getting to contradiction, it is sufficient to show that $Pr_S\{G_{k+1}\} \geq (\frac{1}{4})^{k+1}$. Next, we prove that for each z, $1 \leq z \leq k+1$, $Pr_S\{G_z\} \geq (\frac{1}{4})^z$ by induction on z.

For the base case, let $z = 1$. Note that by the definition of dominating transmission decision, we know that for each $l \in \{i, j\}$, the probability that $trans_l(1) = cdtd(1)$ is at least $\frac{1}{2}$, i.e., $Pr_l\{DTD_{l,1}\} \geq \frac{1}{2}$. Since for each such l, the probability distribution of random choices of process l in first round is exactly the same as probability distribution of random choices of that process in first round of totally-failed sequences, we have $Pr_S\{C_{l,1}\} = Pr_l\{DTD_{l,1}\}$. Since the random choices of i and j in the first round are independent of each other, we establish that $Pr_S\{G_1\} \geq \frac{1}{4}$.

Now, for the inductive step, assume that for some z, $1 \leq z \leq k+1$, $Pr_S\{G_{z-1}\} \geq (\frac{1}{4})^{z-1}$. First, consider the case that $z \leq k$. By the definition of dominating transmission decision, we know that for each $l \in \{i, j\}$ and for each z', $2 \leq z' \leq z$, we have $Pr_l\{trans_l(z') = cdtd(z')|DTD_{l,z'-1}\} \geq \frac{1}{2}$. Note that by the Failure-Round Claim, if an execution is in $G_{z'-1}$, then the action sequences for processes i and j corresponding to the first $z'-1$ rounds of the execution are $z'-1$ round prefixes of totally-failed sequences. Therefore, we can say that for each such z', conditioned on $G_{z'-1}$, for each $l \in \{i, j\}$, the probability distribution of random choices of process l in round z' of system S is exactly the same as probability distribution of random choices of process l in round z' of totally-failed sequences conditioned on $DTD_{l,z'-1}$. Hence, $Pr_S\{trans_l(z) = cdtd(z)|G_{z-1}\} = Pr_l\{trans_l(z) = cdtd(z)|DTD_{l,z-1}\} \geq \frac{1}{2}$. Since the random choices of i and j in round z, when conditioned on G_{z-1}, are independent of each other, we have $Pr_S\{G_z|G_{z-1}\} \geq \frac{1}{4}$. Therefore, from the inductive hypothesis, and noting that $G_z \subseteq G_{z-1}$, we have $Pr_S\{G_z\} = Pr_S\{G_z|G_{z-1}\} \cdot Pr_S\{G_{z-1}\} \geq (\frac{1}{4})^z$.

Now, consider the case that $z = k+1$. By the definition of dominating result, we know that for each $l \in i, j$, $Pr_l\{result_l = true|DTD_{l,k}\} \geq \frac{1}{2}$. By an argument similar to above, we have $Pr_S\{result_l = cdr|G_k\} = Pr_l\{result_l = dr_l|DTD_{l,k}\} \geq \frac{1}{2}$. Therefore, noting the independence of random choices of i and j, when conditioned on G_k, we have $Pr_S\{G_{k+1}|G_k\} \geq \frac{1}{4}$. Thus, applying the inductive hypothesis, and noting that $G_{k+1} \subseteq G_k$, we have $Pr_S\{G_{k+1}\} = Pr_S\{G_{k+1}|G_k\} \cdot Pr_S\{G_k\} \geq (\frac{1}{4})^{k+1}$.

Thus, there exist two processes such that the output leader IDs, when B is run with the system consisting of those two processes, are incorrect solutions to the 2-LE problem with probability at least $(\frac{1}{4})^k$. □

Theorem 40. *For any ID space I of size u, $u \geq 2$, any ϵ, $0 < \epsilon \leq 1$, and any algorithm A that solves probabilistic 2-LE in SCD systems, there exist two processes with IDs from I such that, when A is run with just those two processes, the probability that A terminates in* $\min(\log(\frac{1}{4\epsilon})/2, \lceil \log(u) \rceil - 2)$ *rounds, when run on the system of those two processes, is at most* $1 - \epsilon$.

Proof. Fix a 2-LE algorithm A in SCD systems. For any $\epsilon \in (0, 1]$, let $k(\epsilon) = \min(\log(\frac{1}{4\epsilon})/2, \lceil \log(u) \rceil - 2)$. Since $k(\epsilon) < \lceil \log(u) \rceil - 1$, by Theorem 39, we know that there exist two processes for which the probability that A terminates in $k(\epsilon)$ rounds when run on the system of those two processes, is at most $1 - (\frac{1}{4})^{k(\epsilon)+1}$ and is therefore at most $1 - (\frac{1}{4})^{\log(\frac{1}{4\epsilon})/2+1} = 1 - \epsilon$. □

Theorem 41. *For any ID space I of size u, $u \geq 1$, any ϵ, $0 < \epsilon \leq 1$, any n', $1 \leq n' \leq \frac{u}{2}$, and any algorithm A that solves probabilistic $2n'$-LE in SCD systems, there exist $n = 2n'$ processes with IDs from I such that, when A is run with just those n processes, the probability that A terminates in at most* $\min(\log(\frac{1}{4\epsilon})/2, \lceil \log(\frac{u}{n}) \rceil - 2)$ *rounds is at most* $1 - \epsilon$.

Proof Sketch. Assume for the sake of contradiction that there exists some algorithm A that solves probabilistic $2n'$-LE and terminates within $\min(\log(\frac{1}{4\epsilon})/2, \lceil \log(\frac{u}{n}) \rceil - 2)$ rounds with probability greater than $1 - \epsilon$. Consider an ID space I^* of size $u^* = \lfloor \frac{u}{n'} \rfloor$. Using A we construct an algorithm A^* that solves probabilistic 2-LE by emulating A on groups of processes and terminates when A does. Since A terminates within $\min(\log(\frac{1}{4\epsilon})/2, \lceil \log u^* \rceil - 2)$ rounds with probability greater than $1 - \epsilon$ where $\epsilon \in (0, 1]$, the same bounds apply to A^* as well, and this contradicts Theorem 40.

Proof. Assume for the sake of contradiction that there exists an ID space I^* of size u^*, $u^* \geq 1$, an ϵ, $0 < \epsilon \leq 1$, an n', $1 \leq n' \leq \frac{u^*}{2}$, and an algorithm A^* that solves probabilistic $2n'$-LE such that for any set of $n = 2n'$ processes with IDs from I^*, the probability that A^* terminates in at most $\min(\log(\frac{1}{4\epsilon})/2, \lceil \log(\frac{u^*}{n}) \rceil - 2)$ rounds is strictly greater than $1 - \epsilon$. Let $u = \lfloor \frac{u^*}{n'} \rfloor$. Consider an ID space of size u and call it I. We construct another algorithm A that solves the 2-LE problem for the system of any pair of processes with IDs from I in at most $\min(\log(\frac{1}{4\epsilon})/2, \lceil \log u \rceil - 2)$ rounds with probability strictly greater than $1 - \epsilon$.

Let the IDs in I^* and the IDs in I be totally ordered and for any $j \geq 1$, let $id(j)$ and $id^*(j)$ represent the j-th ID in the ordering of I and I^*, respectively. We refer to all the processes in the ID space I as *real processes* and the processes in the ID space I^* as *virtual processes*.

With each real process i, we associate the virtual processes with IDs in the set $v_i = \{id^*(j) | (i-1) \cdot n' + 1 \leq j \leq i \cdot n'\}$. For each real process i, algorithm A on this process simulates the algorithm A^* when executed in the set v_i as follows. In each round r, if exactly one of the processes in $i^* \in v_i$ transmits in A^*, then i transmits the same message in round r. If none of the processes in v_i transmit a message in round r, then i remains silent in round r. If multiple processes in v_i transmit messages in round r, then i transmits a special collision message (SCM) which denotes that round r experienced a collision. Also, if a real process i receives a message that is not SCM, or it receives either $\bot$ or $\top$, it passes that message, or $\bot$, or $\top$ (respectively) to each process in the set v_i that i is simulating. If a real process i receives SCM, then i passes $\top$ to each process in the set v_i.

Now, for any two processes from I, consider the scenario where system consists of those two processes and both of processes simulate A^* in the above manner. Then, in each round of A, if at most one virtual process transmits, all virtual processes receive the same message as they would receive in A^*. If more than one of the virtual processes transmit, either the transmitting virtual processes are from the same real process or there are two of them that are from different real processes. In former case, both real processes receive SCM and in latter case, both real processes receive $\top$. Therefore, similar to what happens in A^*, all virtual processes receive $\top$ in both cases.

In effect, virtual processes in A take exactly the same steps as they would take in A^* and receive exactly the same messages as they would receive in A^*. When algorithm A^*, simulated in A, terminates, one virtual process is chosen as the leader and all virtual processes output the ID of that virtual leader. In A, real processes output the ID of the real process responsible for the chosen leader, i.e., if the ID of the chosen virtual leader is from the set v_i, then real processes output the ID of process i.

From the above simulation we know that if A^*, when run on system of any $n = 2n'$ processes from I^*, solves the $2n'$-LE problem in at most $\min(\log(\frac{1}{4\epsilon})/2, \lceil \log(\frac{u^*}{n}) \rceil - 2)$ rounds with probability strictly greater $1 - \epsilon$, algorithm A, when run on any pair of processes from I, solves the 2-LE problem in at most $\min(\frac{\log(\frac{1}{4\epsilon})}{2}, \lceil \log(\frac{u^*}{n}) \rceil - 2) < \min(\frac{\log(\frac{1}{4\epsilon})}{2}, \lceil \log(\frac{n'(u+1)}{2n'}) \rceil - 2) \leq \min(\frac{\log(\frac{1}{4\epsilon})}{2}, \lceil \log(\frac{u+1}{2}) \rceil - 2) < \min(\frac{\log(\frac{1}{4\epsilon})}{2}, \lceil \log(u) \rceil - 2)$ rounds with probability strictly greater than $1 - \epsilon$. This is in contradiction with the Theorem 40. □

6.3 Algorithms for Leader Election in Weak Collision Detection Systems

In this section, we show that the LE problem in WCD systems can be solved in time complexities that match the lower bounds presented in Section 6.2 for both deterministic and probabilistic cases. Recall that in Section 5.3, we presented implementations of deterministic and probabilistic SCD channels on top of a WCD system. Also, recall that in Section 6.1, we presented BLEP and CLEP as, respectively, deterministic and randomized LE algorithms for SCD systems. To solve LE in WCD systems, we use these LE algorithms on top of the aforementioned implementations of SCD system. We study the time complexities of the resulting deterministic and randomized LE algorithms on WCD systems as a function of the time complexities of the deterministic and probabilistic SCD implementations (respectively), and the time complexities of BLEP and CLEP (respectively).

6.3.1 Deterministic LE in WCD systems

Theorem 42. *For a B_{WCD}-time-bounded WCD system with IDs from an ID-space of size u and consisting of n processes, $1 \leq n \leq u$, there exists an algorithm that solves deterministic LE and is R_{LE}-round-bounded where $R_{LE}(n) = B_{WCD}(n, \mathcal{O}(\log u))$.*

Proof. In Theorem 29, we showed an implementation of a deterministic SCD channel on top of a WCD system with a time-bound function $B_{SCD}(n, r) = B_{WCD}(n, \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor + 2r)$. We also showed in Theorem 35 that the BLEP algorithm solves the LE problem in SCD channels in at most $\lceil \log(u) \rceil$ rounds of the SCD channel. Therefore, when the BLEP algorithm uses the aforementioned SCD channel implementation, BLEP terminates in at most $\lceil \log(u) \rceil$ rounds of the aforementioned implementation of SCD channel or equivalently by time $B_{SCD}(n, \lceil \log(u) \rceil) = B_{WCD}(n, \lceil \log(u) \rceil - \lfloor \log(n-1) \rfloor + 2 \lceil \log(u) \rceil)$ $= B_{WCD}(n, \mathcal{O}(\log u))$. □

Note that the time complexity presented above, $\mathcal{O}(\log u)$, matches the $\Omega(\log u - \log n)$ lower bound presented in Theorem 37 asymptotically when the size of the ID space is much larger than the actual number of processes in the system, i.e., $u \gg n$.

6.3.2 Probabilistic LE in WCD systems

Theorem 43. *For a B_{WCD}-time-bounded WCD system with IDs from an ID-space of size u and consisting of n processes, $1 \leq n \leq u$, there exists an algorithm that solves probabilistic LE and is ρ_{LE}-round-bounded where*

$$\rho_{LE}(n, \epsilon) = \begin{cases} B_{WCD}(n, \mathcal{O}(\log(u))) & \text{if } n = 1 \\ B_{WCD}(n, \mathcal{O}(\min(\log u, \log\log n + \log(\frac{1}{\epsilon}))) & \text{if } n > 1. \end{cases}$$

Proof. In Theorem 30, we showed an implementation of a β_{SCD}-time-bounded probabilistic SCD channel on top of a B_{WCD}-time-bounded WCD system where

$$\beta_{SCD}(n, r, \epsilon) = \begin{cases} B_{WCD}\big(n, 4\lceil \log u \rceil + 2r\big) & \text{if } n = 1, \\ B_{WCD}\big(n, \min(4(\lceil \log u \rceil - \lfloor \log(n-1) \rfloor), \frac{4\log(1/\epsilon)}{n-1}) + 2r\big) & \text{if } n > 1. \end{cases}$$

Also, in Theorem 36, we showed that CLEP solves the LE problem in SCD systems and terminates in $\mathcal{O}(1)$ rounds if $n = 1$, and in $\mathcal{O}(\min(\log u, \log\log n + \log(\frac{1}{\epsilon}))$ rounds with probability at least $1 - \epsilon$, where $\epsilon \in (0, 1]$, if $n > 1$. Therefore, using the Union Bound, we know that the randomized LE algorithm in WCD systems that we get by using CLEP on top of the aforementioned SCD channel implementation outputs all the $leader()$ events within $\rho_{LE}(n, 2\epsilon)$ time units with probability at least $1 - 2\epsilon$ where $\epsilon \in (0, 0.5]$, and

$$\rho_{LE}(n, 2\epsilon) = \begin{cases} B_{WCD}(n, 4\lceil \log u \rceil + 2\,\mathcal{O}(1)) & \text{if } n = 1 \\ B_{WCD}(n, \min(4(\lceil \log u \rceil - \lfloor \log(n-1) \rfloor), \frac{4\log(1/\epsilon)}{n-1}) & \\ \quad + 2\,\mathcal{O}(\min(\log u, \log\log n + \log(\frac{1}{\epsilon})))) & \text{if } n > 1. \end{cases}$$

Hence, we have

$$\rho_{LE}(n,\epsilon) = \begin{cases} B_{WCD}(n, \mathcal{O}(\log(u))) & \text{if } n = 1 \\ B_{WCD}(n, \mathcal{O}(\min(\log u, \log\log n + \log(\frac{1}{\epsilon})))) & \text{if } n > 1. \end{cases}$$

where $\epsilon \in (0, 1]$. □

Here, we argue that the upper bounds presented above match the respective lower bounds. For the case of $n = 1$, Theorem 26, along with the reduction of LD to LE in Section 4.3, shows an $\Omega(\log u)$ lower bound for solving LE in WCD systems. Hence, the $\mathcal{O}(\log(u))$ upper bound presented above matches the the respective lower bound. For the case of $n > 1$, the $\mathcal{O}(\min(\log u, \log\log n + \log(\frac{1}{\epsilon}))$ upper bound presented above matches the lower bound presented in Theorem 41, when $u \gg n$ and $\epsilon = \mathcal{O}(\frac{1}{n})$.

7 Conclusion

We studied the problem of Leader Election (LE) in single-hop wireless systems subject to collisions. We presented matching upper and lower bounds for solving deterministic and randomized leader election in strong detection (SCD) systems: deterministic LE can be solved in SCD systems in $\Theta(\log u)$ rounds whereas probabilistic LE can be solved in SCD systems in $\Theta(\log \frac{1}{\epsilon})$ rounds with probability at least $1 - \epsilon$.

In order to demonstrate that the same bounds apply to the LE problem in weak collision detection (WCD) systems as well, we introduced the problem of Loneliness Detection (LD). LD is a key subproblem of leader election and as a discriminator between strong and weak collision detection. We showed that LD can be used to implement an SCD channel on top of a WCD channel with a continuing overhead of two WCD-channel rounds per the implemented SCD-channel round. We showed that LD can be solved in SCD systems trivially, and we presented matching upper and lower bounds for solving deterministic and probabilistic loneliness detection in WCD systems: deterministic LD can be solved in WCD systems in $\Theta(\log u - \log n)$ rounds whereas probabilistic LD can be solved in WCD systems in $\Theta(1)$ rounds w.h.p., if $n > 1$, and in $\Theta(\log u)$ rounds, otherwise.

Discussion. Loneliness Detection is an elegant abstraction that captures the difference between SCD and WCD systems. Since SCD systems provide more information about collisions to processes than WCD systems do, designing protocols for SCD systems is potentially simpler and easier than designing protocols for WCD systems. However, as shown in this article, efficient implementations of LD in WCD systems and efficient implementation of SCD channels on top of WCD channels (using LD) enables us to run SCD-based algorithms in WCD systems as well. Thus, LD helps in simplifying the design of protocols for WCD systems.

Although we have discussed LD only in the context of systems with static membership, it may be extended to systems with dynamic membership (including multi-hop systems). However, the utility of LD as the discriminator between SCD and WCD systems in dynamic (and multi-hop) systems is not immediately apparent and could potentially warrant a separate investigation.

Future Work. Several avenues of future research remain open. The LE problem can be extended across multiple dimensions. Consider the following extensions. (1) We can consider single-shot leader election in multi-hop systems with static membership. (2) We may also consider multi-shot leader election in systems with dynamic membership; this problem can ensure the availability of a leader within the system infinitely-often despite continual changes in the membership of the system. (3) We can consider multi-shot leader election in single-hop network with static membership subject to fairness constraints which ensure that each process becomes a leader infinitely often; solutions to the aforementioned problem act as a contention-resolution protocol in single-hop systems and ensure a reliable local broadcast similar to the abstract MAC layer [8].

The abstract MAC layer provides a reliable local broadcast service with timing guarantees expressed in terms of abstract delay functions. Although the abstract MAC layer and related problems of contention resolution, collision resolution, and local broadcast have been studied in the past, existing work focuses either on systems with no collision detection or strong collision detection. The time costs for solving these problems in weak collision detection systems remains unknown.

References

[1] Jacir Luiz Bordim, Yasuaki Ito, and Koji Nakano. Randomized leader election protocols in noisy radio networks with a single transceiver. In *Proceedings of the 4th International Symposium on Parallel and Distributed Processing and Applications*, pages 246–256, 2006.

[2] John I. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE transactions on information theory*, 25(5):505–515, 1979.

[3] A. E. F. Clementi, A. Monti, and R. Silvestri. Distributed broadcast in radio networks with unknown topology. *Theoretical Computer Science*, 302:337–364, 2003.

[4] J.F. Hayes. An adaptive technique for local distribution. *IEEE transactions on communication*, 26:1178–1186, 1978.

[5] Dilsun K. Kaynar, Nancy A. Lynch, Roberto Segala, and Frits Vaandrager. The theory of Timed I/O Automata, second edition. *Synthesis Lectures on Distributed Computing Theory*, 1(1):1–137, 2010.

[6] Dariusz Kowalski and Andrzej Pelc. Broadcasting in undirected ad hoc radio networks. *Distributed Computing*, 18(1), 2005.

[7] Dariusz Kowalski and Andrzej Pelc. Leader election in ad hoc radio networks: A keen ear helps. In *International Conference on Automata, Languages and Programming*, pages 521–533, 2009.

[8] Fabian Kuhn, Nancy Lynch, and Calvin Newport. The abstract mac layer. In *Proceedings of the 23rd International Conference on Distributed Computing*, pages 48–62, 2009.

[9] Koji Nakano and Stephan Olariu. Randomized leader election protocols in radio networks with no collision detection. In *Proceedings of the 11th International Conference of Algorithms and Computation*, pages 362–373, 2000.

[10] Koji Nakano and Stephan Olariu. Uniform leader election protocols for radio networks. *IEEE transactions on parallel and distributed systems*, 13(5), 2002.

[11] Johannes Schneider and Roger Wattenhofer. What is the use of collision detection (in wireless networks)? In *Proceedings of the International Symposium on Distributed Computing*, pages 133–147, 2010.

[12] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1995.

[13] Mariëlle Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, the Netherlands, April 2002. Available via `http://www.soe.ucsc.edu/~marielle`.

[14] D. E. Willard. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM Journal of Computing*, 15:468–477, 1986.