Theoretical Computer Science 410 (2009) 546–569

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Of malicious motes and suspicious sensors
On the efficiency of malicious interference in wireless networks

Seth Gilbert [a,*], Rachid Guerraoui [a], Calvin Newport [b]

[a] *École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland*
[b] *Massachusetts Institute of Technology, Cambridge, MA 02139, USA*

ARTICLE INFO

Keywords:
Wireless radio networks
Ad hoc networks
Sensor networks
Fault tolerance
Algorithms

ABSTRACT

How efficiently can a malicious device disrupt a single-hop wireless network? Imagine two honest players attempting to exchange information in the presence of a malicious adversary that can disrupt communication by jamming or overwriting messages. Assume the adversary has a broadcast budget of β—unknown to the honest players. We show that communication can be delayed for $2\beta + \Theta(\lg |V|)$ rounds, where V is the set of values that the honest players may transmit. We then derive, via reduction to this 3-player game, round complexity lower bounds for several classical n-player problems: $2\beta + \Omega(\lg |V|)$ for reliable broadcast, $2\beta + \Omega(\log n)$ for leader election, and $2\beta + \Omega(k \lg |V|/k)$ for static k-selection. We also consider an extension of our adversary model that includes up to t crash failures. Here we show a bound of $2\beta + \Theta(t)$ rounds for binary consensus. We provide tight, or nearly tight, upper bounds for all four problems. From these results we can derive bounds on the efficiency of malicious disruption, stated in terms of two new metrics: *jamming gain* (the ratio of rounds delayed to adversarial broadcasts) and *disruption-free complexity* (the rounds required to terminate in the special case of no disruption). Two key conclusions of this study: (1) all the problems considered feature semantic vulnerabilities that allow an adversary to disrupt termination more efficiently than simple jamming (i.e., all have a jamming gain greater than 1); and (2) for all the problems considered, the round complexity grows linearly with the number of bits to be communicated (i.e., all have a $\Omega(\lg |V|)$ or $\Omega(\lg n)$ disruption-free complexity.)

1. Introduction

Ad hoc networks of wireless devices hold significant promise for the future of ubiquitous computing. Unfortunately, such networks are particularly vulnerable to adversarial interference due to their use of a shared, public communication medium and their deployment in unprotected environments. For example, a committed adversary can disrupt an ad hoc network by jamming the communication channel with noise. Continuous jamming, however, might be unwise: it depletes the adversary's energy, allows the honest devices to detect its presence, and simplifies its localization—and subsequent destruction. The adversary, therefore, would rather be more efficient, disrupting the protocol using a minimal number of transmissions.

* Corresponding author.
E-mail addresses: seth.gilbert@epfl.ch (S. Gilbert), rachid.guerraoui@epfl.ch (R. Guerraoui), cnewport@mit.edu (C. Newport).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.10.008

1.1. Metrics for adversarial efficiency

We introduce two new metrics for measuring the efficiency with which an adversary can disrupt a protocol: *Jamming Gain* and *Disruption-Free Complexity*. Jamming gain captures how efficiently the adversary can disrupt the long-term completion of the protocol; disruption-free complexity captures how efficiently the adversary can delay the protocol in the short term.

Jamming gain

The efficiency of the adversary can be quantified, roughly speaking, by comparing the *duration* of the disruption to the adversary's *cost* for causing the disruption. In the systems literature, this metric has been informally referred to as *jamming gain* (e.g., [1]). In the context of round-based protocols (time-slotted wireless radio channels), the jamming gain can be defined as follows. Let $D_P(t)$ be the minimal number of broadcasts needed by the adversary to delay protocol P from terminating for t rounds, for some initial value. Then the *jamming gain* of protocol P is:

$$\mathsf{JG}(P) = \lim_{T\to\infty} \frac{T}{\max(D_P(T), 1)}.$$

For example, if the adversary must broadcast in every round, the jamming gain is 1. By contrast, if the adversary need *never* broadcast to prevent termination, then the jamming gain is infinite.

Disruption-free complexity

A second metric, *disruption-free complexity*, measures how long the adversary can disrupt a protocol without performing *any* broadcasts. The uncertainty introduced by the *possibility* of adversarial broadcasts is sufficient to slow down many protocols. We formalize this metric as:

$$\mathsf{DF}(P) = \max\{t : D_P(t) = 0\}.$$

If a protocol has large disruption-free complexity, then the adversary can significantly reduce the throughput of multiple consecutive executions, while avoiding the disadvantages of actually jamming. For example, if the fear of potential adversarial interference required the addition of a complex initialization procedure to your protocol, this would be captured by a large disruption-free complexity value.

1.2. The 3-player game

We begin by analyzing a 3-player game that captures many of the fundamental difficulties of wireless coordination in this setting. We will then extend these results to several classical n-player problems: reliable broadcast, leader election, static k-selection and consensus.

The 3-player game consists of two honest players—Alice and Bob, and a third malicious player, Collin (the *Collider*). All three players share a time-slotted single-hop wireless radio channel. Alice and Bob each begin with a value to communicate. Collin is determined to prevent them from communicating, in either direction, for as long as possible. Collin can broadcast in any time slot (i.e., round), potentially destroying honest messages or overwhelming them with malicious data. In order to precisely measure the efficiency of a malicious adversary, we endow Collin with a budget of β messages, and analyze how long Alice and Bob can be disrupted. The size of β is not known *a priori* to Alice and Bob. (If it were, then Alice and Bob could communicate reliably by repeating each message $2\beta + 1$ times.)

1.3. The 3-player game lower bounds

We show that Collin can delay Alice and Bob's communication for

$$2\beta + \log|V|/2$$

rounds, where V is the set of possible values that Alice and Bob may communicate. An immediate corollary is that no protocol for Alice and Bob can achieve a jamming gain better than 2. This result is surprising as it implies that every such communication protocol has some semantic vulnerability that the adversary can exploit to gain extra efficiency. A second corollary is that the disruption-free complexity is $\Omega(\log|V|)$. Therefore for large V, the passive presence of Collin can significantly reduce Alice and Bob's communication throughput. We prove these lower bounds (in Section 4) by exhibiting a strategy for Collin to delay Alice and Bob, exploiting the fact that they can never trust any message, since Collin could have *overwhelmed* it with a fake message.

In Section 5 we focus on the natural sub-problem of Alice communicating to Bob, who does not broadcast. We explicitly model Alice's broadcast budget as $\beta + \Delta$ (note, in the bounds of Section 4, no restrictions are placed on Alice or Bob's broadcast budget). We first show that communication is impossible for $\Delta \leq 0$, as, in this circumstance, Collin can effectively simulate Alice starting from a different value—confusing Bob. We then show for $\Delta \geq 1$ that no communication protocol can

terminate in less than

$$2\beta + \max\left(\frac{2\Delta|V|^{\frac{1}{2\Delta}}}{e} - 2\Delta,\ \frac{\log|V|}{2}\right)$$

rounds. This implies that for $\Delta = \Omega(\log|V|)$ the lower bound is the same as the infinite-energy bound of Section 4. For $\Delta = o(\log|V|)$, however, the disruption-free complexity increases toward $O(|V|)$ as Δ approaches 1. This highlights an inherent tradeoff between Alice's message complexity and her throughput.

1.4. The 3-player game upper bounds

For our upper bound (Section 6), we consider the setting where Alice needs to transmit a value to Bob, who does not broadcast any messages. We exhibit a protocol that allows Alice—using $\beta + \Delta$ broadcasts—to transmit her value to Bob in

$$2\beta + \max\{2\Delta|V|^{\frac{1}{2\Delta}},\ 4\log|V|\}$$

rounds. (Notice that if $\Delta < 1$, Alice's task is impossible.) For $\Delta = \Omega(\log|V|)$, the protocol matches our un-restricted lower bound of Section 4. For smaller Δ, it matches the limited-energy bound of Section 5.

Finally, we consider a variant of the 3-player game in which Alice and Bob do not start in the same round; Bob is activated asynchronously by the adversary. We present a protocol that solves this problem and still terminates within $2\beta + \Theta(\log|V|)$ rounds (assuming Alice has an unrestricted message budget).

1.5. The n-player implications

The trials and tribulations of Alice and Bob capture something fundamental about how efficiently malicious devices can disrupt wireless coordination in more general problems.

Lower bounds for n-player problems

In Section 8, we derive new lower bounds – via reduction to our 3-player game – for several classical n-player problems:

Reliable broadcast:	$2\beta + \Omega(\log|V|)$;
Leader election:	$2\beta + \Omega(\log n)$;
Static k-selection:	$2\beta + \Omega(k\log\frac{|V|}{k})$.

For the last problem, k represents the number of participants contending to transmit their initial value. These represent, to the best of our knowledge, the first complexity bounds for these problems in a wireless network with an adversary that can arbitrarily disrupt communication. As before, we draw immediate corollaries regarding the jamming gain and disruption-free complexity, resulting in a jamming gain of 2, and disruption-free complexity of $\Omega(\log|V|)$, $\Omega(\log n)$, and $\Omega(k\log\frac{|V|}{k})$, respectively.

Consensus lower bound for model with crash failures

We next consider a more general framework that also includes crash failures: the malicious adversary can both broadcast β messages *and* crash up to t honest devices. We study binary consensus as an archetypal problem in this framework, and derive a lower bound of

$$2\beta + \Theta(t)$$

rounds. The $\Theta(t)$ term is established by a tree-based technique that maintains the indistinguishability of two univalent configurations for t rounds. The 2β term then follows from a (partial) reduction of the three-player game to consensus. This shows a jamming gain of 2, as before. By contrast, the disruption-free complexity, $\Theta(t)$, is significantly larger than for the crash-free models: notice that if no nodes are allowed to crash, then consensus can be solved by a simple reliable broadcast of only one bit. (By contrast, if the adversary cannot disrupt communication, then crash failures have no effect on the round complexity as we are in a synchronous *broadcast* model in which each message is delivered to every other node).

Upper bounds

Finally, in Section 9, we present tight upper bounds for reliable broadcast and consensus and nearly tight bounds for leader election and static k-selection.

1.6. Motivations

Underlying our results on jamming gain and disruption-free complexity is an analysis of how long the adversary can disrupt communication given a limited broadcast budget. This interpretation is interesting in its own right: a limited broadcast budget models the (limited) energy available to a set of malicious devices. (Notice, when we assume one malicious adversary with a budget of β messages, we might model a network with several malicious devices with a combined broadcast budget of β.)

Authentication—for example, using cryptographic keys—impacts our lower bounds. With authentication, the 3-player communication game completes in $\beta+1$ rounds, resulting in a jamming gain and disruption-free complexity of 1. Intuitively, jamming gain arises from semantic vulnerabilities in the protocol; cryptographic techniques can eliminate this vulnerability by preventing Collin from spoofing honest communication. In general, however, deploying cryptographic solutions in wireless networks can be difficult. Public key authentication schemes are often expensive both in computation and, to some extent, communication. Symmetric key schemes (such as MACs) have been deployed in wireless networks (see, e.g., [2, 3]), yet the focus has generally been link-level security, rather than authenticated broadcast, and there remain issues with key distribution. For example, if only a single key is used, the system is easily compromised by a single corrupted node; if multiple keys are used, then keys must be exchanged and communication is complicated.

One interpretation of our bound is that authentication should be deployed only if its cost is less than the cost of waiting the additional $\beta + \Theta(\log |V|)$ rounds imposed in settings without this capability. In fact, our protocols can be viewed as low-cost (in terms of computation and setup) alternatives to cryptographic authentication. The output of a value by our reliable broadcast protocol, for example, confirms its validity. Further efficiency can be achieved by first broadcasting a message directly, and then using our "secure" reliable broadcast protocol to transmit a short hash of the message.

2. Related work

This paper explores the damage that can be caused by a genuinely malicious (Byzantine) device that can reliably disrupt communication in a wireless ad hoc network. Koo [4], Bhandari and Vaidya [5], as well as Pelc and Peleg [6], study "t-locally bounded" Byzantine failures in wireless networks, in which the number of Byzantine nodes in a region is bounded. In these papers, the Byzantine devices are required to follow a strict TDMA schedule, thus preventing them from interfering with honest communication. Others have considered models with probabilistic message corruption [7,8]. Wireless networks with crash failures (but not Byzantine failures) have also been studied extensively in both single hop (e.g., [9,10]) and multihop (e.g., [11,12]) contexts. By contrast, we consider a malicious adversary that can choose to send a message in any round, potentially destroying honest messages or overwhelming them with malicious data.

Koo, Bhandari, Katz, and Vaidya [13] have recently considered a model where the adversary has a limited broadcast budget and can send a message in any round, overwhelming honest messages. A key difference, however, is that they assume that the adversary's budget is fixed *a priori* and known to all participants. By contrast, we do not assume that β is known in advance. Due to this assumption, it is no longer sufficient to simply repeat each message $2\beta + 1$ times as is done in [13]. Furthermore, an unknown β allows us to directly derive efficiency bounds (e.g., in terms of *jamming gain* and *disruption free complexity*). In addition, this lack of knowledge better captures real world systems; we cannot always assume that devices know such details about adversaries. We also note that [13] focuses primarily on feasibility, that is, determining the threshold density of dishonest players for which multihop broadcast is possible. By contrast, our paper focuses on the time complexity of the protocols and the efficiency of the adversary. Furthermore, we also consider the impact of combining crash failures with a malicious adversary, and move beyond broadcast to consider other problems such as leader election and consensus.

Adversarial jamming of physical layer radio communication is a well studied problem in the electrical engineering community (see, e.g., [14]). In the context of wireless ad hoc networks, there has been recent interest in studying the jamming problem at the MAC layer and above. See, for example, [1,15–17], which analyze specific MAC and network layer protocols, highlighting semantic vulnerabilities that can be leveraged to gain increased jamming efficiency.

3. Preliminaries

We now specify the details of our communication model. and highlight the assumptions underlying our tight bound and its corollaries. We then describe the 3-player game that forms the foundation for the results to follow.

3.1. Network model

We assume a synchronous round-based Multiple Access Channel (MAC) model with receiver collision detection. We consider n honest devices, *the players*, named from the set $[1, n]$, and one additional malicious device incarnating *the adversary*. In each round, each device can decide to broadcast a message or listen. If there are no broadcasts in a round, then none of the players receives a message. If exactly one message is broadcast, then all players receive the message.[1] If two or more messages are broadcast, then each player can either: (1) receive exactly one of the broadcast messages; or (2) detect noise on the channel, i.e., a collision. (This channel behavior represents the unpredictability of real networks, for example, shadowing effects [18].) Without loss of generality, we assume that the adversary determines for each honest player whether option 1 or 2 occurs; in case of option 1, we assume (without loss of generality) that the adversary's message is systematically received.

[1] For simplicity, we assume that a player receives its own messages; this is, if some player sends message m in round r, then the player itself receives message m in round r.

Throughout this paper, we endow the adversary with a budget of β broadcast messages, where β is *a priori* unknown to the players. We assume no message authentication capabilities. That is, a player cannot necessarily distinguish a message sent from the adversary from a message sent by a fellow honest player.

3.2. The 3-player game

The basic game we consider involves two honest players, Alice and Bob, and an adversary named Collin. For some value domain V, Alice is initialized with value $v_a \in V$ and Bob with $v_b \in V$, where $|V| > 1$ and V is known to all. Each player attempts to communicate its initial value to the other. Specifically, the players can perform output(v) for any $v \in V$ such that the following two properties are satisfied:

(1) **Safety:** Bob only outputs v_a and Alice only outputs v_b; and
(2) **Liveness:** Eventually, either Alice or Bob outputs a value.

4. Lower bound for the 3-player game

In this section, we prove a lower bound on the round complexity of the 3-player communication game. The bound holds even if Alice and Bob have an unlimited budget of messages. To obtain our result, we describe a strategy for Collin to frugally use his β messages to prevent communication. Two assumptions are key to this strategy: (1) Collin's budget of messages β is unknown to Alice and Bob; (2) Alice and Bob cannot distinguish a message sent by Collin from an honest message. Thus, when Bob (for example) receives a message m, he cannot be certain that Alice sent message m. A *silent* round, on the other hand, cannot be faked: if Bob (for example) receives no message and no collision notification, then he can be certain that Alice did not broadcast a message. Therefore, in order to prevent Alice and Bob from communicating, it is sufficient, roughly speaking, for Collin to disturb silent rounds. The main theorem shown in this section is as follows:

Theorem 1. *Every three-player communication protocol for Alice, Bob, and Collin requires at least* $2\beta + \log|V|/2$ *rounds to terminate.*

The proof proceeds as follows. First, we identify two values, v and w, for which Alice and Bob both behave in a similar manner for the first $\log|V|/2-1$ rounds. We then describe a set of behavioral rules for Collin to delay Alice and Bob by filling in silent rounds. We next show that neither Alice nor Bob can output a value while Collin continues to follow this strategy. Finally, we argue that Collin can afford to pursue this strategy for $\log|V|/2 + 2\beta - 1$ rounds using only β broadcasts.

Assume, for the sake of contradiction, a protocol A that defies this worst-case performance. For any value $v \in V$, denote by $\gamma(v)$ the $\log|V|/2-1$ round execution prefix of A where Alice and Bob both begin with initial value v, and Collin performs no broadcasts. We begin with the following lemma:

Lemma 2. *There exist two values* $v, w \in V$ *where* $v \neq w$ *such that:*

- *Alice broadcasts in round* r *of* $\gamma(v)$ *if and only if Alice broadcasts in round* r *of* $\gamma(w)$.
- *Bob broadcasts in round* r *of* $\gamma(v)$ *if and only if Bob broadcasts in round* r *of* $\gamma(w)$.

Proof. In each round, there are four possibilities: (1) Alice broadcasts alone, (2) Bob broadcasts alone, (3) Alice and Bob both broadcast, and (4) neither Alice nor Bob broadcasts. Accordingly, for a sequence of c rounds, there are 4^c possible patterns of broadcast behavior. Thus, there are at most $4^{\log|V|/2-1} = \frac{|V|}{4}$ possible broadcast patterns that result from the $|V|$ possible γ executions. It follows by the pigeonhole principle that at least two such executions have the same behavioral pattern. □

Fix v and w to be the two values identified by Lemma 2. Define $\alpha(v)$ (resp. $\alpha(w)$) to be the execution of A in which Alice and Bob both begin with initial value v (resp. w) and Collin applies the α-rules described in Fig. 1 for as long as his broadcast budget persists (once Collin depletes his budget he remains silent for every subsequent round). Specifically, we construct $\alpha(v)$ and $\alpha(w)$ one round at a time. At the beginning of a given round, we first determine what behavior Alice and Bob will take in both executions. Find the corresponding row in Fig. 1 such that the two columns under "Alice" describe Alice's behavior during this round in the two executions, and the two columns under "Bob" describe the same with respect to Bob. Reference the first two columns under "Collin" to determine how Collin will behave in both executions during this round.

The symbol "—" indicates silence, and m, m', m'', m''' represent different messages that Alice and Bob may broadcast. The first two columns under "Result" describe the resulting messages transmitted during this round in $\alpha(v)$ and $\alpha(w)$, respectively.

For example, assume that in the current round Alice will broadcast in $\alpha(v)$ but will be silent in $\alpha(w)$, and Bob will be silent in both executions. This behavior is described by the first row of Rule #1. Looking to the Collin columns, we see that the adversary will be silent in $\alpha(v)$ but will transmit m—the message Alice broadcasts in $\alpha(v)$—in $\alpha(w)$. The Result columns correctly indicate that m is the single message broadcast in both executions during this round.

We define $\rho(w, v)$ (resp. $\rho(v, w)$) to be the execution of A in which Alice begins with initial value w (resp. v), Bob begins with initial value v (resp. w), and Collin applies the ρ-rules described in Fig. 2, for as long as his broadcast budget persists. Specifically, Collin's behavior in round r of a ρ execution is determined entirely by Alice and Bob's behavior in $\alpha(v)$ and $\alpha(w)$ during this round. This construction relies on the fact (to be shown) that (1) $\rho(w, v)$ is indistinguishable from $\alpha(v)$ with respect to Bob and indistinguishable from $\alpha(w)$ with respect to Alice, and (2) $\rho(v, w)$ is indistinguishable from $\alpha(w)$ with respect to Bob and indistinguishable from $\alpha(v)$ with respect to Alice.

Rule	Alice		Bob		Collin		Result	
	$\alpha(v)$	$\alpha(w)$	$\alpha(v)$	$\alpha(w)$	$\alpha(v)$	$\alpha(w)$	$\alpha(v)$	$\alpha(w)$
# 1	m	$-$	$-$	$-$	$-$	m	m	m
	$-$	m	$-$	$-$	m	$-$	m	m
# 2	m	m'	$-$	$-$	$-$	$-$	m	m'
	$-$	$-$	m	m'	$-$	$-$	m	m'
# 3	$-$	$-$	m	$-$	$-$	m	m	m
	$-$	$-$	$-$	m	m	$-$	m	m
# 4	m	$-$	m'	$-$	$-$	m'	m, m'	m'
	$-$	m	$-$	m'	m'	$-$	m'	m, m'
# 5	m	$-$	$-$	m'	$-$	m	m	m, m'
	$-$	m	m'	$-$	m	$-$	m, m'	m
# 6	m	m'	m''	$-$	$-$	$-$	m, m''	m'
	m	m'	$-$	m''	$-$	$-$	m	m', m''
# 7	m	$-$	m'	m''	$-$	$-$	m, m'	m''
	$-$	m	m'	m''	$-$	$-$	m'	m, m''
# 8	m	m'	m''	m'''	$-$	$-$	m, m''	m', m'''
	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$

Fig. 1. Collin's behavioral rules for executions $\alpha(v)$ and $\alpha(w)$. For each of the possible behaviors of Alice and Bob in $\alpha(v)$ and $\alpha(w)$, Collin behaves in the α executions as specified by the indicated row in the table. The "Result" column indicates the total set of messages broadcast in the two executions under certain conditions. In rounds where Collin broadcasts concurrently with Alice or Bob, assume that Collin's message is the only one received by the non-broadcasting player.

Rule	Alice		Bob		Collin		Result	
	$\alpha(v)$	$\alpha(w)$	$\alpha(v)$	$\alpha(w)$	$\rho(w,v)$	$\rho(v,w)$	$\rho(w,v)$	$\rho(v,w)$
# 1	m	$-$	$-$	$-$	m	$-$	m	m
	$-$	m	$-$	$-$	$-$	m	m	m
# 2	m	m'	$-$	$-$	m	m'	m, m'	m, m'
	$-$	$-$	m	m'	m'	m	m, m'	m, m'
# 3	$-$	$-$	m	$-$	$-$	m	m	m
	$-$	$-$	$-$	m	m	$-$	m	m
# 4	m	$-$	m'	$-$	$-$	m'	m'	m, m'
	$-$	m	$-$	m'	m'	$-$	m, m'	m'
# 5	m	$-$	$-$	m'	m	$-$	m	m, m'
	$-$	m	m'	$-$	$-$	m	m, m'	m
# 6	m	m'	m''	$-$	$-$	m'	m', m''	m, m'
	m	m'	$-$	m''	m	$-$	m, m'	m, m''
# 7	m	$-$	m'	m''	m''	$-$	m', m''	m, m''
	$-$	m	m'	m''	$-$	m'	m, m'	m', m''
# 8	m	m'	m''	m'''	$-$	$-$	m', m''	m, m'''
	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$

Fig. 2. Collin's behavioral rules for $\rho(w, v)$ and $\rho(v, w)$ executions. For each of the possible behaviors of Alice and Bob in $\alpha(v)$ and $\alpha(w)$, Collin behaves in the ρ executions as specified by the indicated row in the table. The "Result" column indicates the total set of messages broadcast in the two executions under certain conditions. In rounds where Collin broadcasts concurrently with Alice or Bob, assume that Collin's message is the only one received by the non-broadcasting player.

For example, in the case considered above—Alice broadcasts only in $\alpha(v)$, and Bob is silent in both—Collin broadcasts m in $\rho(w, v)$ and broadcasts nothing in $\rho(v, w)$. In rounds when Collin transmits concurrently with either Alice or Bob, we assume that Collin's message is the only one received by the non-broadcasting player.

We use the notation $\beta_{\alpha(v)}$, $\beta_{\alpha(w)}$, $\beta_{\rho(v,w)}$, and $\beta_{\rho(w,v)}$ to describe Collin's broadcast budget in $\alpha(v)$, $\alpha(w)$, $\rho(v, w)$, and $\rho(w, v)$, respectively. We now make the following claim:

Lemma 3. *Assuming that Collin has sufficiently large values $\beta_{\alpha(v)}$, $\beta_{\alpha(w)}$, and $\beta_{\rho(w,v)}$ that allow him to follow the rules in Figs. 1 and 2 for t rounds in all three executions $\alpha(v)$, $\alpha(w)$ and $\rho(w, v)$. It follows that, through round t: $\rho(w, v)$ is indistinguishable from $\alpha(v)$ with respect to Bob and indistinguishable from $\alpha(w)$ with respect to Alice.*

Proof. We prove this result by induction on the round number, r, $0 \leq r \leq t$. Because Bob begins with value v in both $\rho(w, v)$ and $\alpha(v)$, and Alice begins with value w in both $\rho(w, v)$ and $\alpha(w)$, the base case ($r = 0$) is immediate. Consider the possible behaviors for Alice, Bob, and Collin during round $r + 1$. By the definition of our executions, Alice and Bob's behavior is determined by deterministic algorithm A, and Collin's behavior is determined by the relevant rule in Figs. 1 and 2.

We start with Alice. If she broadcasts in $\alpha(w)$ in round $r + 1$ then, by our inductive hypothesis, she will also broadcast in $\rho(w, v)$ in round $r+1$ since she cannot distinguish the two executions through round r. Thus, at the end of round $r+1$, Alice cannot distinguish the two executions: in each case, she simply receives her message and nothing else. Assume, therefore, that Alice does not broadcast in $\alpha(w)$ in round $r+1$. This restricts our attention to rules $1(a), 2(b), 3, 4(a), 5(a), 7(a)$ and $8(b)$ in Figs. 1 and 2. For each row, consider the "Result" column for $\alpha(w)$ and $\rho(w, v)$, which indicate the messages broadcast in round $r + 1$ (and thus received by Alice) in *alpha*(w) and $\rho(w, v)$, respectively. (In the case where two messages are indicated, consider only the message broadcast by Collin—according to our execution definitions, these will be the only messages received by Alice).

Specifically, the result column for $\alpha(w)$ in Fig. 1 is the combination of Alice, Bob, and Collin's behavior in $\alpha(w)$, as described in this row. The result column for $\rho(w, v)$ in Fig. 2 is the combination of Alice's behavior in $\alpha(w)$, Bob's behavior in $\alpha(v)$, and Collin's behavior as described by $\rho(w, v)$. Since Alice cannot distinguish $\rho(w, v)$ from $\alpha(w)$ through the end of round r, and since Bob cannot distinguish $\rho(w, v)$ from $\alpha(v)$ through the end of round r, this behavior captures Alice's and Bob's broadcasts in round $r + 1$.

Notice that these two result columns are equal for all the rows we are considering (or, in rows with more than one message, there is at least one message that is sent in both cases). Therefore, Alice receives the same message in both $\alpha(w)$ and $\rho(w, v)$ in every case during which she does not broadcast. Indistinguishability is maintained. A symmetric arguments shows the same to hold true for Bob with respect to $\alpha(v)$. □

The symmetric claim is true for executions $\alpha(v)$, $\alpha(w)$, and $\rho(v, w)$:

Lemma 4. *Assuming that Collin has sufficiently large values of $\beta_{\alpha(v)}$, $\beta_{\alpha(w)}$, and $\beta_{\rho(v,w)}$ that allow him to follow the rules in Fig. 1 for t rounds in all three executions $\alpha(v)$, $\alpha(w)$, and $\rho(v, w)$. It follows that, through round t: $\rho(v, w)$ is indistinguishable from $\alpha(w)$ with respect to Bob and indistinguishable from $\alpha(v)$ with respect to Alice.*

Proof. The argument is symmetric to Lemma 3. □

Let β be the broadcast budget given by Theorem 1. To prove this theorem we show that one of the two α executions requires only β broadcasts by Collin during the first $2\beta + \log|V|/2 - 1$ rounds. Remember, due to our previously established indistinguishability, neither Alice nor Bob can output during the rounds in which Collin's budget remains non-empty.

Proof of Theorem 1. Let $\beta_{\alpha(v)}$, $\beta_{\alpha(w)}$, $\beta_{\rho(v,w)}$, and $\beta_{\rho(w,v)}$ be sufficiently large to allow Collin to broadcast, when required by the rules in Figs. 1 and 2, for the first $t = 2\beta + \log|V|/2 - 1$ rounds of $\alpha(v)$, $\alpha(w)$, $\rho(v, w)$, and $\rho(w, v)$, respectively. By Lemmas 3 and 4, Alice cannot distinguish $\alpha(v)$ from $\rho(v, w)$ or $\alpha(w)$ from $\rho(w, v)$, and Bob cannot distinguish $\alpha(v)$ from $\rho(w, v)$ or $\alpha(w)$ from $\rho(v, w)$, through round t. It follows that Alice and Bob cannot output during the first t rounds of either α execution. If one of the players did output a value, by the demonstrated indistinguishability, the player would have to output the same value in the corresponding ρ execution—violating the safety of A. For example, if Alice output v prior to round t of $\alpha(v)$, then she would also output v prior to round t of $\rho(w, v)$, thus reporting the wrong value for Bob.

We start by considering rounds 1 through $\log|V|/2 - 1$ of $\alpha(v)$ and $\alpha(w)$. We show by induction on the round number, r, $0 \leq r < \log|V|/2 - 1$, that Collin does not broadcast in either α execution during these rounds.

The base case ($r = 0$) is immediate. For $r + 1$ we note that, by our inductive hypothesis, Collin has not yet broadcast. Therefore, $\alpha(v)$ and $\alpha(w)$ are indistinguishable from $\gamma(v)$ and $\gamma(w)$, respectively, through round r. We can apply Lemma 2 to show that Alice (resp. Bob) broadcasts in round $r + 1$ of $\alpha(v)$ if and only if Alice (resp. Bob) broadcasts in round $r + 1$ of $\alpha(w)$. Notice, however, that by the rules in Fig. 1, Collin only broadcasts in an α execution in situations of asymmetric silence; i.e., when Alice (resp. Bob) broadcasts in one α execution but not the other. Therefore, Collin will not broadcast in either α execution during $r + 1$.

We next turn our attention to the 2β rounds that follow. By the rules in Fig. 1, Collin never broadcasts in both $\alpha(v)$ and $\alpha(w)$ during the same round. Therefore, by a simple counting argument, it is impossible for Collin to broadcast in more than half of these 2β rounds in both α executions. Without loss of generality, let $\alpha(v)$ be the execution requiring no more than β broadcasts during the first $2\beta + \log|V|/2 - 1$ rounds. It is valid, therefore, to define $\beta_{\alpha(v)} = \beta$. This makes $\alpha(v)$ a valid execution under the constraints of the theorem statement. By our argument above neither Alice nor Bob can output during the first $2\beta + \log|V|/2 - 1$ rounds of this execution—proving our claim. □

We conclude with an immediate corollary of Theorem 1:

Corollary 5. *Any 3-player communication protocol has a jamming gain of at least 2, and a disruption-free complexity of $\Omega(\log|V|)$.*

5. Energy-aware lower bounds for the 3-player game

In Section 4 we proved a lower bound of $2\beta + \Omega(\log |V|)$ rounds for the communication game, regardless of the size of the honest players' message budgets. Here, we recast the bound to take into account this budget size. For simplicity, we focus on the natural special case of the communication problem where Alice is broadcasting a value to Bob, and Bob does not broadcast. (It remains an interesting open problem to extend these bounds to the general case of two-way communication.)

In the following, we assume Alice's broadcast budget is expressed as $\beta+\Delta$. We assume that Bob knows that Alice's budget has at least Δ more broadcasts than Collin's, but, as before, Bob does not know β. We must make a similar assumption for Alice: she knows Δ, but does not know her total budget of $\beta + \Delta$ (as this would indicate the value of β, which is unknown).

In Section 5.1 we show that communication is impossible for $\Delta \leq 0$. In Section 5.2, we prove a lower bound of:

$$2\beta + \max\left(\frac{2\Delta|V|^{\frac{1}{2\Delta}}}{e} - 2\Delta, \frac{\log|V|}{2}\right)$$

rounds for $\Delta > 0$. This implies that for $\Delta = \Omega(\log |V|)$ the lower bound is the same as the infinite-energy bound of Section 4. For $\Delta = o(\log |V|)$, however, the first term in the max statement increases exponentially to $\Theta(|V|)$ as Δ approaches 1.

5.1. Impossibility result for $\Delta \leq 0$

Theorem 6. *There exists no protocol that allows Alice to transmit a value to Bob using less than or equal to β broadcasts.*

Proof. Assume, for the sake of contradiction, that such a protocol, A, exists. Let α_0 be an execution of A in which Alice is initialized with value 0 and Collin simulates, using A, Alice starting with value 1. Let α_1 be an execution of A in which Alice is initialized with value 1 and Collin simulates, using A, Alice starting with value 0. Because both Alice and Collin have the same broadcast budgets, it is easy to see that these two executions will appear indistinguishable with respect to Bob. Assume, without loss of generality, that Bob outputs 0 in execution α_0. Bob, therefore, also outputs 0 in α_1, resulting in a contradiction. □

5.2. Lower bound for $\Delta > 0$

We extend the lower bound of Theorem 1 to explicitly consider Alice's broadcast budget. At the core of our argument is the observation that there should not exist two sequences of rounds in which Alice behaves the same for two different initial values, and uses $\geq 2\Delta$ broadcasts. As we saw in Theorem 1, for sequences in which Alice behaves the same for two different values, Collin can maintain indistinguishability for Bob without having to broadcast. If Alice uses up her budget advantage over Collin during rounds in which he does not broadcast, then she has left him with enough power to effectively spoof her until her budget expires during the rounds that follow. Accordingly, for small Δ values, Alice must use many more silent rounds when communicating with Bob to prevent exhausting her extra broadcasts with undue haste.

Theorem 7. *Let:*

$$k = \max\left(\frac{2\Delta|V|^{\frac{1}{2\Delta}}}{e} - 2\Delta, \frac{\log|V|}{2}\right).$$

If Alice has a budget of size $\beta + \Delta$ where $\Delta > 0$, then there exists no protocol that allows Alice to transmit her initial value to Bob in less than $2\beta + k$ rounds.

We are considering a restricted case of the general lower bound presented in Section 4. If $k = \frac{\log|V|}{2}$, the theorem follows directly from our general bound. We assume for the remainder of the proof that $k = \frac{2\Delta|V|^{\frac{1}{2\Delta}}}{e} - 2\Delta$. Assume, for the sake of contradiction, the existence of a protocol A that defies our bound.

We begin by defining $\gamma(v)$, for all $v \in V$, as before, to be the execution that results from starting Alice with value v and running the protocol A until termination, with no interference from Collin (i.e., Collin never broadcasts).

To continue, we define $t(v)$, for all $v \in V$, to be the *minimum* of: (1) the round in $\gamma(v)$ in which Alice uses her 2Δth broadcast; (2) round $k-1$. Finally, let $\gamma_t(v)$, for all $v \in V$, to be the execution prefix of $\gamma(v)$ through round $t(v)$. We claim the following:

Lemma 8. *There exist two values $v, w \in V$ where $v \neq w$, such that Alice broadcasts in round r of $\gamma_t(v)$ if and only if Alice broadcasts in round r of $\gamma_t(w)$.*

Proof. We describe Alice's broadcast behavior in the first $k-1$ rounds of some execution ψ as a binary string of at most $k-1$ bits $B(\psi, k-1)$: bit i in $B(\psi, k-1)$ equals 1 if and only if Alice broadcasts during round i of ψ. (If ψ contains fewer than $k-1$ rounds, then pad the string with 0 bits until it reaches length $k-1$). Consider the set of all possible executions γ_t:

$$S = \{B(\gamma_t(v), k-1) \mid v \in V\}.$$

We want to bound the size of this set. By the definition of γ_t, we know that there are no more that 2Δ broadcasts by Alice in any γ_t execution prefix. Moreover, notice that there are at most $\binom{(k-1)+2\Delta}{2\Delta}$ binary broadcast sequences of length $k-1$ that include no more than 2Δ ones. Thus, we bound the size of the set S:

$$\begin{aligned}
|S| &\leq \binom{(k-1)+2\Delta}{2\Delta} \\
&< \binom{k+2\Delta}{2\Delta} \\
&\leq \left(\frac{e(k+2\Delta)}{2\Delta}\right)^{2\Delta} \\
&= \left(\frac{e\left(\left[\left(\frac{|V|^{\frac{1}{2\Delta}}}{e}\right)2\Delta - 2\Delta\right] + 2\Delta\right)}{2\Delta}\right)^{2\Delta} \\
&= \left(|V|^{\frac{1}{2\Delta}}\right)^{2\Delta} \\
&= |V|.
\end{aligned}$$

Having established that $|S| < |V|$, it follows, by the pigeonhole principle, that there must exist two distinct values $v, w \in V$ such that $B(\gamma_t(v), k-1) = B(\gamma_t(w), k-1)$. It follows directly that for these two values v and w, if Alice broadcasts in some round i in $\gamma_t(v)$ (resp. $\gamma_t(w)$) then Alice broadcasts in round i in $\gamma_t(w)$ (resp. $\gamma_t(v)$). □

Fix v and w to be the two values identified by Lemma 8. We next argue that executions $\gamma_t(v)$ and $\gamma_t(w)$ are both of length $k-1$.

Lemma 9. *Both execution prefixes $\gamma_t(v)$ and $\gamma_t(w)$ are of length $k-1$ rounds.*

Proof. Assume for the sake of contradiction that $\gamma_t(v)$ is of length $< k-1$ rounds; that is, $t(v) < k-1$. This implies that Alice broadcasts 2Δ messages in execution $\gamma_t(v)$. Since by Lemma 8, Alice broadcasts in some round r of $\gamma_t(v)$ if and only if she also broadcasts in round r of $\gamma_t(w)$, we can also conclude that Alice expends 2Δ messages in the first $t(v) < k-1$ rounds of $\gamma_t(w)$, and hence $t(w) = t(v) < k-1$ as well.

Define execution $\delta(v)$ of protocol A, as follows. Alice starts with initial value v. For the first $t(v)$ rounds Collin does not broadcast. After this prefix, Collin simulates Alice running A with initial value w, until he exhausts his budget (i.e., after β broadcasts). In the case where Collin and Alice broadcast simultaneously, assume Bob receives neither message—and therefore detects a collision. We argue that Bob cannot output a value in execution $\delta(v)$, contradicting the termination property of A.

To prove this claim, we first construct a second execution: $\delta(w)$. Alice starts this execution with initial value w. During the first $t(v)$ rounds, Collin simulates Alice running A with initial value v, that is, Collin broadcasts the message that Alice sends in $\delta(v)$. During these initial rounds we assume Collin's messages overwhelm Alice's, and are, therefore, the only messages received by Bob. Alice does not notice this behavior as they are broadcast on the same schedule, since by assumption, Alice broadcasts a message in round r when starting with initial value v if and only if Alice broadcasts a message when starting with initial value w. For the rounds that follow, Collin continues to simulate Alice running A with initial value v. As in $\delta(v)$, if simultaneous broadcasts occur during these later rounds, both messages are lost.

Let β be Collin's broadcast budget at the beginning of execution $\delta(v)$, and $\beta' = \beta + \Delta$ be Collin's broadcast budget at the beginning of $\delta(w)$. (Since Collin's broadcast budget is unknown to Alice and Bob, it can be different in the two executions.) Thus, at the beginning of $\delta(v)$, Alice has a broadcast budget of $\beta + \Delta$, and at the beginning of $\delta(w)$, Alice has a broadcast budget of $\beta' + \Delta = \beta + 2\Delta$ messages.

We now argue that $\delta(v)$ and $\delta(w)$ are indistinguishable with respect to Bob at all points during each execution. In both cases, Bob sees the same messages during the first $t(v)$ rounds: in $\delta(v)$, these messages are sent by Alice; in $\delta(w)$, these messages are sent by Collin.

In $\delta(v)$, Bob receives up to $\beta - \Delta$ additional messages broadcast by Alice using the $\beta + \Delta - 2\Delta = \beta - \Delta$ messages that remain in her budget after the $t(v)$ round prefix; these messages attempt to convince Bob to output v. Bob will also receive up to β messages broadcast by Collin who is using his full budget of size β; these messages attempt to convince Bob to output w.[2]

In $\delta(w)$, after the initial $t(v)$ rounds, Bob will then receive up to $\beta - \Delta$ additional messages broadcast by Collin with the $\beta' - 2\Delta = (\beta + \Delta) - 2\Delta = \beta - \Delta$ broadcasts that remain in his budget after the 2Δ expended during the $t(v)$ prefix;

[2] In both cases, as in the two that follow, "receive" means either receiving the message or detecting a collision, which could only occur if both message types where broadcast.

these are the same messages that Alice sent in $\delta(v)$ to try to convince Bob to output v. Bob will also receive up to β messages broadcast by Alice with the $\beta' + \Delta - 2\Delta = (\beta + \Delta) + \Delta - 2\Delta = \beta$ broadcasts that remain in her budget after the 2Δ expended during the $t(v)$ round prefix); there are the same messages that Collin sent in $\delta(v)$.

Notice, in both executions, Bob receives the same set of up to $\beta - \Delta$ messages that recommend value v and the same set of up to β messages that recommend value w, following the initial indistinguishable $t(v)$ round prefix. Thus, the two executions are indistinguishable. If Bob outputs v in $\delta(v)$ then this violates safety in the context of $\delta(w)$; if Bob outputs w in $\delta(v)$ then Bob violates safety in the context of $\delta(v)$. □

We now define $\alpha(v)$, $\alpha(w)$, $\rho(v, w)$ and $\rho(w, v)$ in the same manner as in Section 4. Notice, we can ignore the rows in Fig. 1 that have Bob broadcast as we are considering the special case where only Alice communicates.

Proof of Theorem 7. Let $\beta_{\alpha(v)}$, $\beta_{\alpha(w)}$, $\beta_{\rho(v,w)}$ and $\beta_{\rho(w,v)}$ be sufficiently large to allow Collin to broadcast, when required by the rules in Fig. 1, for the first $t = 2\beta + k - 1$ rounds of $\alpha(v)$, $\alpha(w)$, $\rho(v, w)$ and $\rho(w, v)$. Notice, because Bob does not start with an initial value in the case we consider here, we can disregard the second parameter to the ρ executions. For simplicity, we will refer to $\rho(v, w)$ as $\rho(v)$ and $\rho(w, v)$ as $\rho(w)$. Because we defined these executions the same as in Section 4, we can apply Lemma 3 and Lemma 4, which provide that Bob cannot distinguish $\alpha(v)$ from $\rho(w)$, or distinguish $\alpha(w)$ from $\rho(v)$, through round t. As in the proof for Theorem 1, it follows that Bob cannot output during the first t rounds of either α execution.

Let $\ell = min\{\mathsf{length}(\gamma_t(v)), \mathsf{length}(\gamma_t(w))\}$. Consider rounds 1 through ℓ of $\alpha(v)$ and $\alpha(w)$. We show by induction on the round number, r, $0 \leq r < \ell$, that Collin does not broadcast in either α execution during these rounds.

The base case ($r = 0$) is immediate. For $r + 1$ we note that, by our inductive hypothesis, Collin has not yet broadcast. Therefore, $\alpha(v)$ and $\alpha(w)$ are indistinguishable from $\gamma_t(v)$ and $\gamma_t(w)$, respectively, through round r. We can apply Lemma 8 to show that Alice broadcasts in round $r+1$ of $\alpha(v)$ if and only if Alice broadcasts in round $r+1$ of $\alpha(w)$. Notice, however, that by the rules in Fig. 1, Collin only broadcasts in an α execution in situations of asymmetric silence; i.e., when Alice broadcasts in one α execution but not the other. Therefore, Collin will not broadcast in either α execution during $r + 1$.

We now turn our attention to the 2β rounds that follow. By the rules in Fig. 1, Collin never broadcasts in both $\alpha(v)$ and $\alpha(w)$ during the same round. Therefore, as with the proof of 1, we note that by a simple counting argument, it is impossible for Collin to broadcast in more than half of these 2β rounds in both α executions. Without loss of generality, let $\alpha(v)$ be the execution requiring no more than β broadcasts during the first $2\beta + \ell$ rounds. It is valid, therefore, to define $\beta_{\alpha(v)} = \beta$. This makes $\alpha(v)$ a valid execution under the constraints of the Theorem statement. Remember, by our argument above Bob can output during the first $2\beta + \ell$ rounds of this execution.

Recall that we have already shown in Lemma 9 that $\ell \geq k - 1$, since both $\gamma_t(v)$ and $\gamma_t(w)$ are at least $k - 1$ rounds, concluding the proof. □

6. Upper bounds for the 3-player game

We prove in this section that our round complexity lower bounds are tight (within a constant factor) by demonstrating a protocol that achieves a matching running time. To strengthen our result, we consider the (seemingly) harder problem of Alice transmitting her value to Bob in a setting where Bob does not broadcast. Specifically, we give a protocol that, assuming Alice has a budget of $\beta + \Delta$ messages, $\Delta > 0$, transmits Alice's input value to Bob in $2\beta + \max(\log|V|, \Delta|V|^{\frac{1}{\Delta}})$ rounds. As in Section 5, the value of Δ is known to Alice and Bob, while β is unknown.

We begin in Section 6.1 by presenting a protocol that transmits a single bit from Alice to Bob. Then, in Section 6.2, we show how Alice encodes her message to Bob as a sequence of bits so that it can be sent using at most $\beta + \Delta$ messages. (If $\Delta = \Omega(\log|V|)$, then the natural binary encoding is used.)

Throughout the rest of this paper, we use the following pseudocode convention: for a given value v, each invocation of bcast-rcv(v) executes a single round of wireless communication. If value $v = \perp$, then no message is broadcast. The bcast-rcv invocation returns any messages received in that round, or $\perp$ if no message is received. Moreover if a collision is detected during the communication round, the bcast-rcv invocation returns the special symbol $\pm$.

6.1. Transmitting one bit

The basic protocol, described in Algorithm 1, transmits a single bit b from Alice to Bob. The key idea is to alternate data rounds and veto rounds, using the former to transmit data, and the later to verify that the data has arrived correctly. (Each iteration of lines 5–21 by Alice and lines 5–12 by Bob implements two rounds of communication.)

In the data phase, Alice transmits a message if $b = 1$ and remains silent otherwise. (Lines 7–9 describe the case where $b = 0$ and Alice broadcasts nothing; lines 10–11 describe the case where $b = 1$ and Alice broadcasts a vote message.) The veto phase is used to confirm the accuracy of the preceding data phase. Silence in the veto phase indicates that the preceding data phase was accurate and Alice can terminate (lines 20–21). A broadcast in the veto phase, on the other hand, indicates potential trouble, therefore requiring Alice to try again with a new pair of data and veto rounds.

This provides Collin two means by which to delay termination. First, he can broadcast in the data phase when Alice would otherwise be silent (i.e., when $b = 0$). Alice will detect this phony data phase broadcast and subsequently indicate this deception by broadcasting in the veto phase (line 9). Second, Collin can broadcast in the veto phase when Alice would

Algorithm 1: Single-Bit Transmission Protocol

```
1  ▷ Alice transmits bit b ∈ {0,1} to Bob:
2  Alice-bcast(b)
3     ▷ Repeat the data and veto phases until done.
4     repeat:
5        ▷ Data phase:
6        do-veto ← false
7        if (b = 0) then
8           m ← bcast-rcv(⊥)
9           if (m ≠ ⊥) then do-veto ← true
10       else if (b = 1) then
11          m ← bcast-rcv(vote)
12
13       ▷ Veto phase:
14       if (do-veto = false) then
15          m ← bcast-rcv(⊥)
16       else
17          m ← bcast-rcv(veto)
18
19       ▷ Check if success:
20       if (m = ⊥) then
21          return success
22
23 ▷ Bob receives a bit b ∈ {0,1} from Alice:
24 Bob-recv()
25    ▷ Repeat the data and veto phases until done.
26    repeat:
27       ▷ Data phase:
28       m1 ← bcast-rcv(⊥)
29
30       ▷ Veto phase:
31       m2 ← bcast-rcv(⊥)
32       if (m2 = ⊥) then
33          if (m1 = ⊥) then return 0
34          else if (m1 ≠ ⊥) then return 1
```

have remained silent. Alice will detect this phony veto (lines 20–21) and then try again with a new pair of rounds. (Note: Alice knows the veto is phony, but Bob does not, necessitating that Alice continues with two further rounds, just as if she had sent the veto. A minor optimization would be to have Bob ignore a veto message if the preceding data round was empty. In this instance it is clear that Alice did not send the veto message.)

Bob simply listens in both the data and veto phases, broadcasting nothing. If the veto phase is silent, he returns a value based on the message (or lack thereof) from the data phase. Notice that Alice and Bob terminate the protocol at the same round: the first silent veto phase round.

We now discuss some properties of Algorithm 1:

Lemma 10. *Assume that Alice has at least one more transmission in her broadcast budget than Collin, and that Alice begins with initial value $b \in \{0, 1\}$. If Alice and Bob both invoke Algorithm 1 in the same round, then:*

(1) Termination: *Alice returns at the end of round r if and only if Bob returns at the end of round r.*

(2) Safety: *Assume Bob terminates at the end of round r. Then Bob returns value b.*

(3) Alice's energy consumption: *If Alice has not terminated by the end of round r, where r is a* veto*-phase, then Alice has expended at most $r/2$ broadcasts. If Alice has terminated by the end of round r then:*
   - Case 1: $b = 0$: *Alice expends at most $r/2 - 1$ broadcasts;*
   - Case 2: $b = 1$: *Alice expends at most $r/2$ broadcasts.*

(4) Collin's Energy Consumption: *If Alice and Bob have not terminated by the end of round r, where r is a* veto*-phase, then Collin has expended at least $r/2$ broadcasts.*

Proof. We address each of the properties in turn:

(1) *Termination:* Notice that Alice and Bob both terminate after the first silent veto phase (lines 20–21 for Alice, lines 10–12 for Bob). By the definition of our model, a silent round cannot be faked by Collin: either both processes receive *something* (be it a message or collision notification), or both processes receive nothing.

(2) *Alice's energy consumption:* Notice that Alice never broadcasts in both the data phase and the veto phase: if $b = 0$, Alice will broadcast only in the veto phase (see lines 7–8); if $b = 1$, then Alice will broadcast only in the data phase. We conclude that if Alice has not terminated by the end of round r, a veto-phase, then Alice has expended no more than $r/2$ broadcasts.

If Alice *does* terminate at the end of round r, a veto-phase, we first note that by the above argument, through round $r - 2$ Alice has expended at most $(r - 2)/2$ broadcasts. By our assumption that Alice terminates after r and the fact that Alice and Bob terminate only after a silent round, we know that Alice does not broadcast in round r. (A broadcast in r would indicate a veto, requiring the protocol to continue.) If $b = 0$, then Alice does not broadcast in round $r - 1$. If $b = 1$, then Alice does broadcast in $r - 1$. In the former case, Alice expends at most $(r - 2)/2 = r/2 - 1$ broadcasts (note that r is even). In the latter, Alice expends at most $r/2$ broadcasts.

(3) *Collin's energy consumption:* If Collin does not broadcast in either the data phase or the veto phase, then the protocol terminates: if there is no broadcast in the veto phase, then Alice and Bob both terminate; moreover, Alice only broadcasts in the veto phase if Collin broadcasts in the data phase when $b = 0$. It follows: if the protocol has not terminated at the end of round r, a veto-phase, then Collin has broadcast in either the data or the veto round for each pair of proceeding rounds, leading to the conclusion that Collin has expended at least $r/2$ broadcasts.

(4) *Safety:* Assume that Alice and Bob terminate at the end of round r. By definition, round r is a veto-phase. If Alice and Bob terminate after round r, then they had not terminated by the end of round $r - 2$, also a veto phase. By our two claims regarding energy consumption, shown above, Alice has expended at *most* $(r - 2)/2$ broadcasts through round $r - 2$ and Collin has expended at *least* the same amount. By assumption, Alice begins with at least one more broadcast in her budget than Collin, and thus has at least one broadcast left in her budget at the beginning of round $r - 1$. Consider the two cases for b:

If $b = 1$, Alice broadcasts in round $r - 1$. Bob either receives this message or detects a collision. Because Alice and Bob terminate in round r we know that round r, the veto phase, is silent. Thus Bob outputs '1'.

If $b = 0$, Alice is silent in round $r - 1$. If Collin had broadcast in round $r - 1$ then Alice would have received this broadcast or detected a collision, and in this case, she would have used her (at least) one remaining broadcast to veto in round r. Because Alice and Bob terminate, however, we know that round r is silent. This implies that Collin did not broadcast in round $r - 1$. Both rounds, therefore, are silent, and Bob correctly outputs '0'. □

6.2. Encoding the message one bit at a time

In order to transmit a non-binary value, Alice transmits her message to Bob one bit at a time using Algorithm 1. If the value to transmit is encoded as a binary string in the natural manner, this leads to a string of length $\log |V|$.

Notice, however, that in order to successfully transmit a '1', Alice expends one more broadcast than Collin. Consider, for example, the case where Alice is transmitting a '1' to Bob, and both Alice and Bob terminate in round r. By Lemma 10: through (veto-phase) round $r - 2$, Alice and Collin have expended the same number of broadcasts (assuming Collin is being as frugal as possible); in order to terminate in round r, Alice must broadcast in $r - 1$, while Collin does not. By contrast, to transmit a '0', Alice does not need to broadcast in $r - 1$. Thus it is cheaper for Alice to transmit '0's than '1's.

If $\Delta \geq \log |V|$, Alice can use the standard binary encoding for a value v. In the worst-case, value v consists of a sequence of $\log |V|$ 1's. Since Alice's broadcast budget is at least $\log |V|$ larger than Collin's, she can afford to transmit this value v.

If $\Delta < \log |V|$, then it is necessary to encode the value v more carefully. Specifically, Alice and Bob both know that Alice has a broadcast budget at least Δ larger then Collin. Thus Alice encodes her messages as a bit string of length k containing at most Δ bits with value '1'. If the bit string is of length k, this implies it can encode up to $\binom{k}{\Delta}$ messages. Some simple approximation allows us to calculate that if $k = \Delta |V|^{\frac{1}{\Delta}}$ then:

$$\begin{aligned}\binom{k}{\Delta} &= \binom{\Delta |V|^{\frac{1}{\Delta}}}{\Delta} \\ &\geq \left(\frac{\Delta |V|^{\frac{1}{\Delta}}}{\Delta}\right)^{\Delta} \\ &\geq |V|.\end{aligned}$$

Thus, using bit strings of length $\Delta |V|^{\frac{1}{\Delta}}$, Alice can encode all the messages in the set V using no more than Δ bits with value '1' in each encoding.

Once Alice and Bob have chosen an encoding based on the relationship between Δ and the size of V, Alice then sends the encoded message one bit at a time using the single bit transmission protocol in Algorithm 1, at which point Bob decodes it

in the natural way. From this, we conclude the following theorem:

Theorem 11. *If Alice begins with a budget of $\beta + \Delta$ broadcasts, $\Delta > 0$, and encodes her message $v \in V$ as a bit string of length* $\max(\log|V|, \Delta|V|^{\frac{1}{\Delta}})$ *with at most Δ bit of value* '1'*, and transmits it to Bob one bit at a time using Algorithm* 1*, then:*

(1) Safety: *Bob outputs value v.*
(2) Liveness: *Alice terminates and Bob produces an output in at most:*

$$2\beta + 2\max(\log|V|, \Delta|V|^{\frac{1}{\Delta}})$$

rounds.

Proof. Fix sequence S for Alice to transmit to Bob. By Lemma 10, we know that each bit of S will be transmitted correctly as long as Alice has more transmissions remaining in her broadcast budget than Collin when the single-bit protocol begins for that bit. For the first bit, this is true by definition as we assume $\Delta > 1$. If this bit is 0, then by Lemma 10 Alice will expend no more broadcasts than Collin to transmit it to Bob. If this bit is 1, then by the same lemma Alice must expend at most one more broadcast then Collin to transmit it to Bob. By definition, no more than Δ bits in S are 1. Because Alice has Δ more broadcasts than Collin, it follows by simple induction that Alice will have sufficient budget to transmit each bit. Safety follows from this observation.

We now proceed to calculate the number of rounds required to transmit all the bits. Let R be the total number of rounds required to send the full sequence. We divide the rounds into $R/2$ pairs of consecutive data and veto rounds. Each such pair of rounds either: (1) terminates Algorithm 1 for one of the bits in the sequence; (2) does not terminate Algorithm 1 due to interference from Collin. Given Collin's budget of β, at most β of these pairs fall into the second category. Because there are $\max(\log|V|, \Delta|V|^{\frac{1}{\Delta}})$ bits to transmit, exactly this many pairs fall under the first option. It follows that:

$$R/2 \leq \beta + \max(\log|V|, \Delta|V|^{\frac{1}{\Delta}}).$$

Therefore:

$$R \leq 2\beta + 2\max(\log|V|, \Delta|V|^{\frac{1}{\Delta}})$$

which concludes the proof. □

7. Synchronizing Alice and Bob: The wake-up case

In this section, we consider a variation of the basic broadcast problem where Alice and Bob need to synchronize, that is, where Alice and Bob do not necessarily begin the broadcast protocol in the same round.

We assume that Alice is always awake, that is, Alice begins her protocol at the very beginning of the execution. At this point, Alice may not yet have received a value to transmit to Bob; however she still may need to counter Collin's malicious behavior, preventing Collin from tricking Bob into receiving an incorrect value. In some round $r_b \geq 0$, Bob arrives and begins executing his protocol. Consider, for example, the case where Alice is a base station, and Bob is a sensor or mobile device that occasionally wakes up to receive data from Alice.

At some point (either before or after Bob arrives) in round $r_a \geq 0$, Alice is provided with a value v to transmit. (We model this in the pseudocode as Alice requesting the value in each round via a call to get-value(); once the value is available, this function returns the value; until then, it returns $\perp$.)

The goal is that Bob receives this value as soon as possible. The *duration* of the protocol is $r_{out} - \max(r_a, r_b)$, where r_{out} is the round during which Bob first outputs Alice's value. Because r_a and r_b are unbounded, we do not restrict Alice's energy budget. The presented protocol has a duration of $2\beta + O(\log|V|)$ rounds.

As in Section 6, the basis of our protocol is a simple subroutine that transmits a single bit of information from Alice to Bob. It is necessary, however, that Alice and Bob *synchronize*, and they must continually monitor the channel for Collin's attempts to disrupt synchronization. The synchronization compensates for Alice and Bob's different starting rounds, and is used to help them start the single-bit subroutine at the same time.

Notice that in the pseudocode (Algorithms 2–5), the state for Alice is presented in Algorithm 2 and is shared by all the subroutines used by Alice; similarly, the state for Bob is presented in Algorithm 3 and is shared by all the subroutines used by Bob.

7.1. Synchronization

The synchronization protocol is presented in Algorithm 4. Alice and Bob synchronize via a long sequence of broadcasts. The key idea is that the number of broadcasts is sufficiently long that it is prohibitively expensive for Collin to fake the synchronization sequence.

Specifically, when Alice wants to begin synchronization, she first broadcasts a synch message for $5\log|V|$ rounds, and then remains silent in round $5\log|V| + 1$ of the sequence (see lines 3–12, Algorithm 4). Notice that the Alice's synchronization only completes when there is a silent round (lines 8–12, Algorithm 4); Collin can delay the completion of Alice's synchronization by disrupting the last round.

When Bob wants to synchronize, he listens for a long sequence of broadcasts, followed by a silent round. Whenever he observes $5\log|V|$ consecutive rounds in which a message is broadcast, followed by a single silent round, he starts (or

Algorithm 2: Alice Transmission Protocol

```
1  State:
2      size, initially, the size of the bit string to send
3      value ∈ {0,1}^size ∪ {⊥}, initially, ⊥
4      phase ∈ {synch, not-in-synch}, initially, not-in-synch
5      count ∈ ℤ, initially, 0
6      bit ∈ [−1, size], initially, −1
7      m, a message
8
9  Alice()
10     repeat:
11         ▷ Check whether Alice's value is ready to send
12         if (value = ⊥) then value ← get-value()
13
14         ▷ Choose which case to perform:
15         if (value = ⊥) then
16             ▷ Alice's value is not ready:
17             if (phase = in-synch) then
18                 ▷ Prevent false synch.
19                 if (Alice-bcast(0) = success) then
20                     phase ← not-in-synch
21             else if (phase = not-in-synch) then
22                 ▷ Listen for a round:
23                 m ← bcast-rcv(⊥)
24                 synch-check(m)
25         else if (value ≠ ⊥) then
26             ▷ Alice's value is ready:
27             if (phase = not-in-synch) then
28                 ▷ Perform synchronization.
29                 synch-Alice()
30             else if (phase = in-synch) and (bit = −1) then
31                 ▷ Finish synchronization:
32                 if (Alice-bcast(1) = success) then
33                     bit ← bit + 1
34             else if (phase = in-synch) and (bit ≥ 0) and (bit ≤ size) then
35                 ▷ Transmit a bit:
36                 if (Alice-bcast(value[bit]) = success) then
37                     bit ← bit + 1
38             else if (phase = in-synch) and (bit > size) then
39                 ▷ Start again:
40                 phase ← not-in-synch
```

re-starts) the protocol. The synch-check routine (lines 20–28, Algorithm 4) counts the number of consecutive rounds in which a non-$\perp$ message has been received; when the *count* has reached $5 \log |V|$, it waits for a silent round and then initiates the restart, resetting the *phase*, *bit* and *count*.

If Alice begins a synch-Alice() any time after Bob begins a synch-Bob(), then both Alice and Bob will return from the synchronization protocol in the same round, and both will begin their respective transmission/receive protocols at the same time.

7.2. The single-bit transmission sub-protocol

The pseudo-code for the single-bit transmission protocol is depicted in Algorithm 5, and is nearly identical to the single-bit protocol presented in Algorithm 1 for the case where Alice and Bob are already synchronized. As before, Alice alternates data and veto rounds until Bob successfully receives the specified bit of information.

Algorithm 3: Bob Receive Protocol

```
1  State:
2      size, initially, the size of the bit string to send
3      value ∈ {0,1}^size ∪ {⊥}, initially, ⊥
4      phase ∈ {synch, not-in-synch}, initially, not-in-synch
5      count ∈ ℤ, initially, 0
6      bit ∈ [−1, size], initially, −1
7      m, a message
8
9  Bob()
10     repeat:
11         ▷ Check whether synchronized or not:
12         if (phase = not-in-synch) then
13             synch-Bob()
14         else if (phase = in-synch) then
15             m ← Bob-recv()
16             if (bit = −1) then
17                 if (m = 1) then
18                     ▷ Get ready to received real data:
19                     bit ← bit + 1
20                 else if (m = 0) then
21                     ▷ Abort the false synchronization:
22                     phase ← not-in-synch
23             else if (bit ≥ 0) and (bit ≤ size) and (m ≠ ⊥) then
24                 value[bit] ← m
25                 bit ← bit + 1
26         if (bit > size) then
27             return value
```

In this case, however, the basic protocol is complicated by the need to detect "false synchronization". In particular, Collin may choose to broadcast in 5 log $|V|$ consecutive rounds, thus faking the synchronization sequence. Moreover, Bob may wake up just in time to detect this synchronization sequence, and hence may begin receiving data bits (via Bob-recv) immediately after hearing the 5 log $|V|+1$ synchronization sequence. Thus, Alice too must restart the protocol in this case, retransmitting the entire bit sequence.

The main difference, then, between algorithm Algorithm 1 and Algorithm 5 is that Alice and Bob each call the synch-check routine after each round, checking whether a restart is necessary. If so, they abort the transmission/reception of this particular bit (see lines 13–14 and 22–23 for Alice, Algorithm 5; line 7 and 11 for Bob, Algorithm 5).

7.3. The overall protocol

The overall protocol is presented in Algorithm 2 (for Alice) and Algorithm 3 (for Bob). Alice and Bob begin by synchronizing, and then Alice transmits the value to Bob one bit at a time.

There is one further step needed to ensure that Collin cannot trick Bob. Consider the case where Alice has not yet received the value to transmit. In this case, if Collin broadcasts the synchronization sequence, then Alice must continually be on guard to prevent Bob from receiving false data. In particular, Alice must broadcast in *every* veto round to indicate that she is not ready to transmit data; this would be very costly, in terms of Alice's energy budget. Instead, we use the following technique: in order to ensure that the string is coming from Alice, rather than Collin, Alice appends a '1' to the beginning of her string; if Bob receives a string beginning with a '0' then he knows that it is not coming from Alice. Moreover, if Alice detects a false synchronization, she transmits a '0' to Bob, ensuring that Bob can determine that she is not ready to send the value. In this way, Alice can abort a false synchronization in an energy-efficient manner.

We now proceed to describe Alice's protocol (Algorithm 2) in more detail. The protocol has two stages:

- *Pre-value:* Initially, Alice has not yet received a value to transmit; at this point (lines 15–24, Algorithm 2), her only goal is to prevent Bob from receiving an incorrect value. Thus, if Collin has successfully faked a synchronization (line 17, Algorithm 2), then Alice broadcasts a '0' to indicate that she is not ready to transmit her value. Otherwise (lines 21–24, Algorithm 2), Alice listens passively to any messages broadcast by Collin in order to detect fake synchronization.

Algorithm 4: Synchronization Protocol

```
1  synch-Alice()
2     repeat:
3        if (count ≤ 5·size) then
4           m ← bcast-rcv(synch)
5           count ← count + 1
6        else
7           m ← bcast-rcv(⊥)
8           if (m = ⊥) then
9              phase ← in-synch
10             bit ← −1
11             count ← 0
12             return synched
13
14 synch-Bob()
15    repeat:
16       m ← bcast-rcv(⊥)
17       if (synch-check(m) = restart) then
18          return synched
19
20 synch-check(m)
21    if (m = ⊥) and (count > 5·size) then    ▷ If synch complete:
22       phase ← in-synch    ▷ (Re)start phase.
23       bit ← −1
24       count ← 0
25       return restart
26    if (m ≠ ⊥) then count ← count + 1   ▷ Increment synch count.
27    else count ← 0  ▷ Reset synch count.
28    return ok
```

- *Post-value:* Eventually, Alice receives the value to transmit (line 12, Algorithm 2), and attempts to transmit it to Bob (lines 25–40, Algorithm 2). In this case, there are three sub-cases. (1) If Alice and Bob are not yet synchronized (lines 27–29, Algorithm 2), then Alice calls synch-Alice in order to synchronize. (2) If Alice and Bob have completed the synchronization protocol, but Alice has not yet sent an initial bit (lines 30–33, Algorithm 2), then she broadcasts a '1' indicating a valid synchronization. (3) If Alice and Bob are synchronized, and if they have completed the initial validation bit, then Alice transmits her value bit-by-bit (lines 34–37, Algorithm 2). When this is complete, Alice resets the synchronization and begins again (38–40).

Bob's protocol (Algorithm 3) is somewhat simpler. Initially, Bob waits to detect a synchronization sequence by Alice (lines 12–13, Algorithm 3). Once synchronization is complete, Bob tries to receive a single-bit; (lines 16–22, Algorithm 3). If the bit is a '1', then Bob prepares to receive data; otherwise, if the bit is a '0', then Bob resets the phase (line 22, Algorithm 3), and again waits for synchronization. Once fully synchronized, Bob continues to receive bits (lines 23–25, Algorithm 3) until the entire value is received.

7.4. Analysis

We now demonstrate that Alice successfully transmits her value to Bob, and that the protocol terminates in at most $2\beta + \Theta(\log |V|)$ rounds. As a notational convention, we use the subscript A to indicate Alice's state and B to indicate Bob's state. For example, $value_A$ represents Alice's view of the value, while $value_B$ represents Bob's view.

We begin by examining the modified single-bit transmission protocol (Algorithm 5). The following lemma is almost identical to Lemma 10 with three caveats: (1) No assumption is made regarding Alice's broadcast budget, as we have already assumed that Alice has a sufficient number of broadcasts remaining in her budget. (2) Unlike Algorithm 1, however, Alice or Bob may terminate without succeeding (when one of the players detects a synchronization sequence); in this case, we show that either both succeed or both fail in the transmission protocol. (3) We assume that initially, Alice and Bob begin with the same view of the current synchronization protocol, i.e., $count_A = count_B$.

Lemma 12. *Assume that Alice begins with initial value $b \in \{0, 1\}$. If Alice and Bob both invoke Algorithm 5 in the same round and $count_A = count_B$, then:*

(1) Termination 1: *Alice returns at the end of round r if and only if Bob returns at the end of round r.*
(2) Termination 2: *Alice returns* success *if and only if Bob returns a value $\neq \perp$.*

Algorithm 5: Modified Single-Bit Transmission Protocol

```
1  ▷ Alice transmits bit b ∈ {0,1} to Bob:
2  Alice-bcast(b)
3     ▷ Repeat the data and veto phases until done.
4     repeat:
5        ▷ Data phase:
6        do-veto ← false
7        if (b = 0) then
8           m ← bcast-rcv(⊥)
9           if (m ≠ ⊥) then do-veto ← true
10       else if (b = 1) then
11          m ← bcast-rcv(vote)
12
13       ▷ Check if synchronized:
14       if (synch-check(m) = restart) then return failed
15
16       ▷ Veto phase:
17       if (do-veto = false) then
18          m ← bcast-rcv(⊥)
19       else
20          m ← bcast-rcv(veto)
21
22       ▷ Check if synchronized:
23       if (synch-check(m) = restart) then return failed
24
25       ▷ Check if success:
26       if (m = ⊥) then return success
27
28 ▷ Bob receives a bit b ∈ {0,1} from Alice:
29 Bob-recv()
30    ▷ Repeat the data and veto phases until done.
31    repeat:
32       ▷ Data phase:
33       m1 ← bcast-rcv(⊥)
34       if (synch-check(m1) = restart) then return ⊥
35
36       ▷ Veto phase:
37       m2 ← bcast-rcv(⊥)
38       if (synch-check(m2) = restart) then return ⊥
39       if (m2 = ⊥) then
40          if (m1 = ⊥) then return 0
41          else if (m1 ≠ ⊥) then return 1
```

(3) Safety: *Assume Bob terminates at the end of round r. Then Bob either returns value b or* $\perp$.
(4) Alice's energy consumption: *If Alice has not terminated by the end of round r, where r is a* veto *phase, then Alice has expended at most* $r/2$ *broadcasts. If Alice has terminated by the end of round r then:*
   - Case 1: $b = 0$*: Alice expends at most* $r/2 - 1$ *broadcasts;*
   - Case 2: $b = 1$*: Alice expends at most* $r/2$ *broadcasts.*

(5) Collin's energy consumption: *If Alice and Bob have not terminated by the end of round r, where r is a* veto *phase, then Collin has expended at least* $r/2$ *broadcasts.*

Proof. The only modifications from the proof in Lemma 10 involve checking for synchronization; the safety and energy consumption properties follow exactly as in Lemma 10. The added synchronization checks, however, can effect termination, and hence we must argue that they hold.

First, we observe that throughout the invocation of Algorithm 5, we know that $count_A = count_B$. This fact holds initially. In each round, the *count* variable is modified only by synch-check, which depends only on whether Alice or Bob received

a message or silence. Since Collin cannot fake a silent round, we know that Alice and Bob either both receive a message or both receive silence. Thus, Alice and Bob both update *count* in the same manner when they call synch-check in each round.

Next, we observe that if either Alice or Bob terminates without succeeding, then both Alice and Bob terminate in the same round without succeeding. This follows immediately from the observation that $count_A = count_B$ at the end of each round, and thus if synch-check returns restart for one player, then it returns restart for both players in the same round.

Finally, if neither player returns without succeeding, then it follows exactly as in Lemma 10 that Alice and Bob terminate in the same round. □

We next show that the synchronization protocol works correctly, that is, if Bob has phase set to in-synch, then Alice also believes that the system is synchronized, and both Alice and Bob have identical values of *count*.

Lemma 13. *Assume that at the end of round r,* $phase_B =$ in-synch *and also that neither Alice nor Bob have terminated at the end of round r. Then* (1) $phase_A =$ in-synch, (2) $count_A = count_B$, *and* (3) $bit_A = bit_B$.

Proof. First, we argue that $count_A \geq count_B$ throughout the entire execution (regardless of the phase). Notice that when Bob is first awakened, he has $count_B = 0$ and Alice has $count_A \geq 0$; we need to show that the desired inequality is maintained. There are two cases to consider:

- Assume that Bob increments $count_B$: In this case, Bob has received a message $\neq \perp$ that causes him to increment the count. We can conclude that either Alice or Collin must have broadcast a message, and hence Alice also must have received a message or detected a collision in that round. Thus, Alice also increments $count_A$.
- Assume that Alice decreases $count_A$ in some round: In this case, Alice must have received a message $m = \perp$ (either line 11 or 24 of Algorithm 4). Thus we can conclude that Bob also received $m = \perp$ (due to the collision detector functionality), and hence also resets $count_B = 0$ (line 24, Algorithm 4).

Thus in either case, $count_A$ remains no smaller than $count_B$.

We now prove inductively that the three claims hold. First, we consider the round r in which Bob sets $phase_B =$ in-synch. In this case, we can conclude that $count_B = 0$ and $bit_b = -1$ at the end of round r, since the count is reset whenever the phase is set to in-synch (see lines 21–25, Algorithm 4). We now argue that Alice also executes either lines 21–25 of *synch-check* or lines 9–12 of *synch-Alice* (Algorithm 4), which result in appropriate values for $phase_A$, $count_A$, and bit_A. Specifically, we know that in round r, Bob detected only silence, and hence Alice too detected silence, since Collin cannot fake a silent round. Moreover, we know that $count_A \geq count_B$, and also that $count_B \geq 5 \cdot size$ (since Bob detected a complete synchronization sequence). Thus, $count_A \geq 5 \cdot size$, and hence Alice also detects a complete synchronization sequence. Thus, the three claims holds at the end of round r.

Next, we argue that the three claims continue to hold as long as $phase_B =$ in-synch. First, it is easy to see that bit_A continues to equal bit_B: by the Termination 2 property of the single-bit transmission protocol (Lemma 12), we know that Alice-bcast returns success if and only if Bob-recv returns a value $\neq \perp$; thus Alice increments bit_A (lines 33 and 37, Algorithm 2) if and only if Bob increments bit_B (lines 19 and 25, Algorithm 3).

Next, observe that in each round, Alice receives a non-$\perp$ message if and only if Bob receives a non-$\perp$ message, and hence the synch-check function modifies the *count* and *phase* of Alice and Bob in the same way. Moreover, the *count* is modified only by the synch-check function; thus $count_A = count_B$ at the end of each round.

There are, however, two other places in Algorithm 2 where $phase_A$ is set to not-in-synch: line 20 (Algorithm 2), where a false synchronization is detected, and line 40 (Algorithm 2), where the transmission completes. In each case, it remains to argue that Bob either sets $phase_B$ to not-in-synch or terminates.

- *False synchronization is detected (line 20, Algorithm 2):* This can occur only immediately after $phase_A$ has been set to in-synch, which is also immediately after Bob has set $phase_B$ to in-synch. Moreover, it implies that Alice-bcast(0) returned success. Thus, we conclude by Lemma 12 (Termination 1 and 2, and Safety properties) that the function Bob-recv returns '0', the value sent by Alice. Hence Bob too sets $phase_B =$ not-in-synch, as required.
- *Transmission completes (line 40, Algorithm 2):* If Alice sets $phase_A$ to not-in-synch because $bit_A > size$, then we can conclude the $bit_B > size$, and hence Bob returns $value_B$ (line 27, Algorithm 3), terminating the protocol. □

We conclude the section with the main theorem:

Theorem 14. *If Alice receives initial value v, then Algorithm 2 and Algorithm 3 guarantee:*

(1) Safety: *Bob outputs value v.*
(2) Liveness: *Bob terminates at most* $2\beta + \Theta(\log |V|)$ *rounds after round* $\max(r_a, r_b)$, *i.e., after both Alice receives her initial value and Bob wakes up.*

Proof. By combining Lemma 12 with Lemma 13, it is easy to see that Bob outputs value v only if Alice receives value v as her input. Specifically, Lemma 13 shows that once synchronized, Alice and Bob continue to have the same *count*. This implies (by induction) via Lemma 12 that each invocation of the single-bit transmission protocol terminates in the same round, and thus the following invocation of the single-bit transmission protocol *begins* in the same round. Finally, from Lemma 12, we conclude that Bob outputs the bit broadcast by Alice. Since $bit_A = bit_B$ throughout, Bob correctly updates $value[bit]$ with the correct bit of Alice's value.

We now argue that Bob eventually terminates. Specifically, the first time Alice calls synch-Alice after Bob awakes, Bob will receive an appropriate synchronization sequence and return synched. Moreover, each iteration of Alice-bcast and Bob-recv terminate as per Lemmas 12 and 13. With no disruption from Collin, the synchronization and data transmission require $\Theta(\log |V|)$ rounds. More specifically, with no delays by Collin, Alice and Bob will finish in $\max(r_a, r_b) + 14 \log |V| + 5$ round: in the worst case, Bob awakes one round after Alice begins a synchronization sequence, resulting in two iterations of synchronization (each of which takes $5 \log |V| + 3$ rounds), and two iterations of bit transmission ($2 \log |V|$), less the round Bob arrived too late.

Collin's β broadcasts may: (1) delay synchronization for up to β rounds; (2) force Alice and Bob to resynchronize, or (3) disrupt an individual bit transmission. In the first case, each broadcast by Collin causes one round of delay. In the second case, we can amortize the delay against Collin's broadcasts and observe that each broadcast causes delay < 1. At worst, resynchronization delays Alice and Bob by $2 \log |V| + 2$, more specifically, the $2 \log |V| + 1$ cost of retransmitting all but the last bit in Alice's initial value, and the cost of the first "validation" bit when $bit = -1$. Bob's cost for forcing the resynchronization is at least $5 \log |V|/2$, more specifically, his optimal strategy is to alternate broadcasts with Alice as she attempts (and fails) to broadcast a '1' using $5 \log |V|/2$ broadcast. Thus each broadcast by Collin causes delay < 1. Finally, in the third case, it follows from Lemma 12 that Collin can delay each bit transmission by only two rounds with each broadcast.

Thus we conclude that the total running time is bounded by $2\beta + \max(r_a, r_b) + 14 \log |V| + 5$. □

8. Lower bounds for *n*-player problems

We now generalize our results to n-player coordination problems. In Section 8.1, we show how to derive lower bounds for several problems by relating them to the three player game that we have studied in previous sections. In Section 8.2, we consider the impact of combining malicious behavior with crash failures.

8.1. *n*-player reductions

We demonstrate how Alice and Bob can together simulate an arbitrary n-player protocol. We then use this simulation to derive lower bounds, via reduction from the 3-player communication game, for several n-player problems: reliable broadcast, leader election, and static k-selection. None of our round-complexity lower bounds restrict the message budget of honest players.

Simulation

A simulation by Alice and Bob is defined by a 5-tuple: $\{A, n, S_A, S_B, I\}$, where: (1) A is the n-player protocol being simulated; (2) S_A and S_B partition the n players into two non-empty and non-overlapping sets; (3) I is a mapping of players to their respective initial values.

Alice simulates the players in S_A, initializing them according to I. (Alice is provided only the initial values for nodes in S_A, i.e., $I|S_A$.) In each round, if any of the players in S_A choose to broadcast, Alice arbitrarily chooses one of their messages to broadcast (the remaining messages are ignored). She simulates the receipt of this message at each of the players in S_A. If no player in S_A broadcasts, then Alice listens during the round, receiving m, and simulates the receipt of m at each player in S_A (i.e., m is either: a message, a collision notification, or silence). Alice outputs any values output by her simulated players. Bob behaves symmetrically, with respect to S_B. We prove the following about the fidelity of these simulations.

Theorem 15. *Consider simulation $\{A, n, S_A, S_B, I\}$. For all r-round executions of the simulation, there exists an r-round execution α of A, initialized according to I, where the outputs of Alice and Bob are equivalent to the outputs in α, and Collin broadcasts the same number of messages in the simulation and α.*

Proof. We prove this claim by a straightforward induction on the round number, showing that after r' rounds: (1) the state of the simulated players corresponds to some r'-round legal execution of A; and (2) Collin has performed the same number of broadcasts in both the simulation and the execution of A.

There are two cases of interest. First, consider the case where two or more of Alice's simulated players broadcast (resp. Bob's players), and the simulation algorithm has Alice (resp. Bob) choose only one message to broadcast. In our α execution, this matches the case in which a single message overwhelms others broadcast in the same round. Another interesting case occurs when Collin broadcasts in the simulation. In our α execution, this matches the case where Collin broadcasts the same message, overwhelming other messages and/or causing collisions in the same pattern seen in the simulator (which will depend on whether Alice (resp. Bob) receives Collin's message or a collision notification). This is the only case in which we require Collin to broadcast in the α execution, preserving the second property of our hypothesis. □

We now leverage our simulation to reduce the 3-player game to a several classic n-player problems—obtaining new lower bounds.

Reliable broadcast

In reliable broadcast, one player – the *source* – is provided with an input value $v_0 \in V$. The source must communicate this value to all other players. *Safety* requires that each player output only v_0, i.e., perform output(v) only if $v = v_0$. *Liveness* requires that all players eventually perform an output.

Theorem 16. *Any reliable broadcast protocol requires at least* $2\beta + \log|V|/2$ *rounds to terminate.*

Proof. Assume by contradiction that A is a reliable broadcast protocol that terminates in $R < 2\beta + \log|V|/2$ rounds for all initial values. We reduce 3-player communication, for value domain V, to A. Alice and Bob simulate A for n players, where: (1) S_A contains the source, S_B contains all other players, and (2) I maps the source to v_a, Alice's initial value. Bob outputs the first value output by a simulated player. By Theorem 15, Bob always outputs $v_0 = v_a$ by round R, contradicting Theorem 1. □

Leader election

In leader election, a group of *participants* from among the n players contend to become the leader. All n players should learn the leader, i.e., perform output(ℓ), for the same participant ℓ. To prevent trivial solutions, we assume the full set of participants is not known *a priori*. Instead, each players begins with a binary initial value that specifies whether or not it should participate in the election. The remaining (non-participating) nodes may help with the protocol, but may not be elected leader. *Safety* requires that no two processes output a different leader, and that the common leader output is a participant (i.e., had an initial value of 1). *Liveness* requires every player to perform an output.

Theorem 17. *Any leader election protocol requires at least* $2\beta + \log(n-1)/2$ *rounds to terminate.*

Proof. Assume by contradiction that A is a leader election protocol that terminates in $R < 2\beta + \log(n-1)/2$ rounds for all choices of participants. Let V be the value set containing every integer between 1 and $n-1$. We reduce the 3-player game, for value domain V, to A.

Alice and Bob simulate A for n players where: (1) S_A contains players 1 through $n-1$, S_B contains player n, and (2) I designates player $v_a \in S_A$ (and no one else) to be a participant. The single player in S_B is designated as not participating in the election. Let i be the leader output by Bob's simulated player. Bob outputs $i = v_a$, as required.

By Theorem 15 Bob always outputs v_a within R rounds, contradicting Theorem 1, as $2\beta + \log V/2 = 2\beta + \log(n-1)/2$. □

Static k-selection

In static k-Selection, k *participants* are each provided with a (potentially different) value in V. Each player must receive and output all k values. *Safety* requires that the first k outputs of a player equal the k initial values. *Liveness* requires that all players eventually perform at least k output actions. The protocol *terminates* when all players have performed at least k output actions. (The selection problem is well-studied in radio networks, e.g., [19,20].)[3]

Theorem 18. *Any static k-selection protocol requires at least* $2\beta + \Omega(k \log \frac{|V|}{k})$ *rounds to terminate.*

Proof. Assume by contradiction that A is a protocol that terminates in $R < 2\beta + o(k \log|V|/k)$ rounds, for all initial values and choices of participants. Let value domain V' contain one unique value for every unique multiset of k values drawn from value domain V. We assume a well-known mapping between the values in V' and the multisets. We reduce the 3-player game, for value domain V', to A.

Alice and Bob simulate A for n players where: (1) S_A contains players 1 through k, S_B contains the remaining players, and (2) I activates players 1 through k, and provides each a different value from the multiset mapped to $v_a \in V'$. Given k simulated outputs, Bob can reconstruct and output the unique multiset described by these values. By Theorem 15 Bob will always output v_a in R rounds, contradicting Theorem 1, since $2\beta + \log|V'|/2 = 2\beta + \log\frac{|V|^k}{k!}/2 = 2\beta + \Theta(k \log\frac{|V|}{k})$ rounds. □

We conclude with an immediate corollary of the previous theorems:

Corollary 19. *Any protocol for reliable broadcast, leader election or static k-selection has a jamming gain of at least* 2 *and a disruption-free running time of* $\Omega(\log|V|)$, $\Omega(\log(n-1))$, *and* $\Omega(k \log\frac{|V|}{k})$, *respectively.*

8.2. Combining malicious and crash behavior

We now study the impact of combining malicious behavior with crash failures. We assume that the adversary, in addition to having a budget of β messages, can also crash up to t players. We consider *binary consensus* as an archetypal problem in this context. In consensus, the the n honest players each propose a value. The following properties must be maintained:

(1) *Liveness:* all non-crashed players eventually decide a value.
(2) *Agreement:* all players that decide choose the same value.
(3) *Validity:* if all non-crashed players propose the same value, then all deciding players choose that value.

[3] Often k-selection is oblivious to initial values. We allow a dependence on the initial values, strengthening the lower bound.

By a simple indistinguishability argument, it is easy to see that consensus is impossible if $n \leq 2t$: one cannot distinguish a correct player from a crashed player that is simulated by the adversary; thus no player can decide in an execution in which t players propose '0' and t propose '1'.

We therefore assume that $n = 2t + 1$, and establish a lower bound of $2\beta + \Theta(t)$ on the round complexity of consensus. Our bound reveals the interesting fact that the possibility of crashed honest devices increases the power of the malicious adversary. This is perhaps surprising as, if there is no malicious adversary, crash-failures have no effect on termination (in a synchronous *broadcast* network).

As before, we use a simulation by Alice and Bob of the (t-resilient) n-player consensus protocol. The simulation, however, is more challenging than those used for the n-player problems studied previously, as we must now compensate for the crash failures. Indeed, we do not start the simulation from the initial configuration of our consensus protocol, but instead from one of two univalent configurations arising after t rounds. These configurations are constructed in Lemma 21, which is interesting in its own right as it exhibits executions in which information (about initial values) is transmitted at the rate of at most one bit per round. By combining this with valency arguments, we show how the 3-player game can then be employed to finalize our lower bound.

Theorem 20. *Any t-resilient binary consensus protocol requires at least $2\beta + t$ rounds to terminate.*

Assume A is a protocol that defies our theorem. We fix the environment such that if multiple messages are sent in a round, and the adversary does not broadcast, then the message sent by the player with the smallest id is received by everyone. An execution (or prefix) of A is *failure-free* if it includes no crashes or broadcasts by the adversary.

Given these assumptions, it is clear that each initial configuration gives rise to a single deterministic failure-free, disruption-free execution. We represent all of these possible failure-free, disruption-free executions as a single tree $T(A)$. Every execution begins at the root, and a node at depth r represents the execution at the beginning of round r. Each node at depth r contains one outgoing edge for every possible message m that may be received in round r. There is also one outgoing edge for a silent round (labeled $\perp$) if silence is possible in round r in some execution that passes through this node. By definition, every failure-free execution of A is represented by a single path in $T(A)$, where the edge between nodes at depth r and $r + 1$ describe what message was received in this execution during round r For each initial configuration c, we say that a node $x \in T(A)$ is *reachable* from c—with respect to A—if the path associated with c's failure-free execution includes node x.

Notice that if a depth r node x is reachable for two initial configurations c and c', and some player i has the same initial value in c and c', then at the beginning of round r, player i cannot distinguish a failure-free execution starting from c with one starting from c'. Through this round, i has received the same messages in both executions. If c is 0-valent (meaning that '0' is the only possible decision starting from configuration c), and c' is 1-valent (meaning that '1' is the only possible decision starting from configuration c'), then i cannot decide prior to round r.

Lemma 21. *There exists a path of length t in $T(A)$, starting at the root, and ending at node R_t, where R_t is reachable from two initial configurations, c_0 and c_1, such that some player p_t has the same initial value in c_0 and c_1, and every crash-free extension of c_0 is 0-valent and every crash-free extension of c_1 is 1-valent, with respect to A.*

Proof. Starting at the root of $T(A)$, given an initial configuration c_0, construct a path of length t by applying the following: (1) If there exists ≥ 1 outgoing message edges, choose the message from the player with the smallest id. (2) Otherwise, follow the $\perp$ edge. Let R_t be the node reached after t iterations.

Configuration c_0 contains either a majority of '0's or a majority of '1's. Notice that a majority contains at least $t + 1$ players, since $n = 2t + 1$. Assume without loss of generality that a majority of players (i.e., at least $t + 1$) propose '0' in c_0. This implies that any crash-free extension of c_0 must decide '0', since any such execution is indistinguishable from one in which all players propose '0', and those $\leq t$ players that seem to be proposing '1' are actually nodes that crashed at the beginning of the execution and are now being emulated by the adversary—in which case a decision of '1' violates validity.

We now construct an initial configuration c_1. Denote by P the set of players that broadcast messages which were received along the path to R_t. Note that P contains $\leq t$ players (one player for each non-$\perp$ edge followed in the path). Choose c_1 such that the players in P propose the same initial value as in c_0, and the remaining players (of which there are at least $t + 1$) all propose '1'. Choose some $p_t \in P$. (If $|P| = 0$, then arbitrarily choose one player p_t to have the same initial value in c_0 and c_1.) By the same reasoning applied to c_0, all crash-free extensions of c_1 must decide '1'. (A majority of processes propose '1', therefore the $\leq t$ that appear not to might be emulated by the adversary). We can show, by a straightforward induction argument, that R_t is reachable from c_1. The base case (the root) is trivial. Our hypothesis posits a node $R_{t'}$, on the path to R_t, such that $R_{t'}$ is reachable from c_1. Let e be the outgoing edge from $R_{t'}$ on the path to R_t. There are two cases:

(1) Edge e is associated with a message m broadcast by some player $i \in P$. Since player i begins with the same initial value in both c_0 and c_1, and has seen the same sequence of messages and silence up to this point, it broadcasts the same message m in the failure-free execution generated by c_1. No player with a smaller id can also broadcast in this round of the c_1 execution as this contradicts the path construction (which would have chosen that edge, not e, when constructing R_t).
(2) Edge e is labeled $\perp$. By the path construction algorithm, $\perp$ must be the only outgoing edge from $R_{t'}$ (It is only chosen if no edges are labeled with messages). Therefore, it must describe the behavior in this round for the failure-free c_1 execution as well.

In both cases we arrive at a new node, $R_{t'+1}$, that is one step closer to R_t on our path. □

With this lemma established, we can now prove our main theorem statement. Our strategy will be to note that the failure-free execution prefixes described by the path R_t are indistinguishable with respect to p_t. Therefore, he cannot have yet decided. This provides a t rounds delay.

To obtain the additional 2β rounds, we defer to our Alice and Bob simulation. In this case, Alice is attempting to send a binary value $v_a \in \{0, 1\}$ to Bob. We have Bob simulate p_t and Alice simulate the rest of the nodes, initializing them with the initial values specified by c_0, if $v_a = 0$, and the initial values specified by c_1, if $v_a = 1$. Instead, however, of starting the simulation from the initial state of A, Alice and Bob start the simulation from the state after the initial t rounds. (This state is the same for p_t regardless of which case Alice chooses, so Bob can perform this initialization without knowing Alice's initial value.) Bob outputs what p_t decides. If p_t can decide in less than 2β rounds than Alice and Bob can solve binary communication in less than 2β rounds—defying our bound on the 3-player communication game.

Proof of Theorem 20. Let α_0 (resp. α_1) denote the failure-free execution prefix starting from c_0 (resp. c_1) and proceeding as described by the path to the R_t in $T(A)$. Executions α_0 and α_1 are indistinguishable with respect to p_t; hence p_t has not decided prior to round t. To this point, the adversary has expended no broadcasts. To achieve a further 2β delay, we defer to Alice and Bob, who can solve the binary communication game by performing a *crash-free* simulation of the n-player protocol A. Specifically, Alice simulates all players except p_t, starting them in their states at the end of α_0, if $v_a = 0$, and their states at the end of α_1, if $v_a = 1$. Bob simulates p_t, starting it in its state after α_0 (which is identical to its state after α_1). Bob outputs the value decided by node p_t.

By Theorem 15, and our assumption that A defies our bound, p_t will decide, and Bob will therefore output, within no more than 2β rounds of the simulation. The execution of A generated by this simulation contains no crash-failures. By our assumptions on the univalency of c_0 and c_1 in crash-free executions (from Lemma 21), we know p_t will decide v_a. Therefore, our simulation solves the binary consensus problem in no more than 2β rounds. This violates Theorem 1. A contradiction. □

We conclude with an immediate corollary of Theorem 20:

Corollary 22. *Any t-resilient binary consensus protocol has a jamming gain of at least 2 and a disruption-free complexity of $\Omega(t)$.*

9. Upper bounds for the n-player problems

We now briefly present protocols for reliable broadcast, leader election, static k-selection, and binary consensus. The round complexities for reliable broadcast and consensus match the lower bounds described in Section 8, within constant factors. Those for leader election and k-selection leave a gap.

Reliable broadcast

An algorithm for reliable broadcast follows immediately from the algorithm in Section 6. The source runs Alice's protocol, and all other players run Bob's protocol, resulting in a running time of $2\beta + O(\log |V|)$, matching the lower bound. This protocol requires the source to have a budget of $\beta + \log |V|$.

Binary consensus

Assuming t crashes, consensus can be achieved using reliable broadcast: each of $2t + 1$ players transmits their initial value, one at a time. Notice that if one or more of these $2t + 1$ are crashed, we can make no guarantee as to which value an honest player will receive from the crashed player: if there is no malicious interference, the protocol results in each honest player receiving a '0' (as silence is interpreted as '0'); on the other hand, Collin can maliciously trick honest players into receiving a '1' with only a single broadcast.

Each player, after receiving $2t + 1$ values, decides the value received at least $t + 1$ times, i.e., the majority value. The running time is $2\beta + \Theta(t)$: each broadcast by Collin can delay the honest nodes by at most 2 rounds. Each player needs a budget of $\beta + 1$ broadcasts.

Leader election

Recall that in the leader election problem, up to k players are initially contending to become the leader; we refer to these contending players as *participants*. The goal of the algorithm is to elect one of the participants. The algorithm is parameterized by an integer $c \geq 1$.

In order to elect a leader, we use a *tournament tree*, a binary tree with n leaves, each labeled with a player's id. Each player maintains such a tournament tree as a local data structure, and maintains a pointer into that tree. At the beginning of the protocol, each player begins at the root of the tournament tree, and at each step descends to a child or ascends to the parent. At each step, the protocol determines whether any of the nodes represented by the leaves of the left subtree are participants, or whether any of the nodes represented by the leaves of the right subtree are participants. If there are any participants in the left subtree, the protocol follows the edge to the left child; if there are any participants in the right subtree, the protocol

follows the edge to the right subtree; otherwise, the protocol ascends to the parent. (The occurrence of this case implies that there was some earlier malicious interference which led the protocol to an incorrect subtree.)

In more detail, each step consists of two c-round phases—the left-child phase and the right-child phase. In the left-child phase, every participating player identified with a leaf of the left subtree broadcasts for at most c consecutive rounds. Conversely, players in the right subtree broadcast up to c times during the right-child phase. A phase ends after the first silent round, or after c non-silent rounds. In the latter case, we say that the phase was *successful*. If the left-child phase was successful, the protocol descends to the left; otherwise, if the right-child phase was successful, the protocol descends to the right; if neither round was successful, the protocol ascends to the parent. If there is no parent – because the current tree node is the root – then there are no participating players. If there is no malicious interference, the protocol reaches a leaf in $2c \log n$ rounds.

On reaching a leaf, the protocol ensures that the identified player is in fact a participant—not simply a product of malicious interference. To achieve this, the identified player uses reliable broadcast to transmit a '1' if she is participating, and a '0' otherwise. In the latter case, the protocol ascends to the parent and continues. The protocol requires each participant to have a budget of $2c \log n + \beta + 1$ broadcasts.

Theorem 23. *The leader election protocol terminates after* $2\beta\frac{c+1}{c} + 2c \log n + 2$ *rounds, for all* $c \geq 1$.

Proof (*Sketch*). We say an edge in the tree connecting a parent to a child is *good* if there is a participant in the subtree of the child. Once the protocol traverses a good edge descending the tree, it never re-ascends that edge, since the edge is good. Thus, at each of the $\log n$ levels of the tournament tree, the protocol traverses only one good edge, resulting in a cost of $2c \log n$. The traversal of each bad edge results in at most $2c + 2$ rounds (amortized): $2c$ rounds to descend and 2 rounds to later ascend. With β' broadcasts, the adversary can cause the protocol to traverse at most β'/c bad edges, resulting in a cost of at most $2\beta'(c + 1)/c$. Finally, the one-bit broadcast to identify the leader takes $2\beta'' + 2$ rounds, assuming Collin expends β'' broadcasts in delaying it. Together, this implies the final running time. □

Static k-selection

In the problem of static k-selection, k players are provided with values to transmit. Each player must receive and output each of the k values. In many ways, this problem combines leader election and broadcast: each of the k players must be identified, and then each should broadcast its value.

A protocol for static k-selection can be obtained by repeating the leader election protocol k times: each time a leader is elected, it uses reliable broadcast to transmit its value, and then ceases to contend in future iterations of leader election. The protocol completes when leader election finds no further contenders. Each participant needs a budget of $2kc \log n + \beta + \log |V|$ broadcasts.

Theorem 24. *For all* $c \geq 1$, *the k-selection protocol terminates in at most:*

$$2\beta\frac{c+1}{c} + 2kc \log n + k \log V + 2k + 2$$

rounds. Under the (common) assumption that $\log n = O(\log |V|)$, *this implies termination within:*

$$2\beta\frac{c+1}{c} + O(ck \log |V|)$$

rounds. □

10. Concluding remarks

We have shown how our 3-player game bounds can be interpreted in a larger n-player context to help derive bounds for several classical problems in distributed computing. We discuss below other interpretations of our 3-player game results.

One possible interpretation can be given in terms of the relative cost of sending a message—as compared to listening—on a radio channel. Assume that it takes s units of energy to send a message, and ℓ units of energy to listen for a message. In real systems, the ratio of s to ℓ varies depending on the network configuration and the underlying hardware. For example, by filtering signals below a certain energy threshold, a network designer can increase the cost of sending a message. In some networks, s is equal to ℓ; in others, s is larger than ℓ. Assume that Alice, Bob, and Collin all begin with the same amount of energy. Our result shows that Collin can prevent Alice and Bob from communicating if and only if $s \leq 2\ell$. In other words, communication can be made more robust against malicious devices if the inherent cost of broadcasting is high relative to listening.

Another possible interpretation can be given in terms of the cost of provoking a collision versus the actual cost of spoofing messages. In our 3-player game, we represented the energy available to the adversary, Collin, in terms of the *total* number of messages, β, he can broadcast. In practice, one might distinguish the cost of sending a message, K, from the cost of causing a collision, k, where k would typically be smaller than K. In this sense, our tight bound indicates that, if $k < K/2$, then the best strategy for Collin is to jam Alice and Bob with collisions in every round; if $k \geq K/2$, then the best strategy for Collin is to follow the silence-filling approach described in Section 4. In other words, for $k < K/2$, the cost of causing a collision

in every round is less than the cost of the deploying the strategy from our main lower bound (which requires Collin to send spoofed messages).

Finally, it is important to recall that we assumed that Alice and Bob can distinguish a "silent" round from a round in which a collision occurred. That is, we assume Alice and Bob can detect some electromagnetic noise in the case where all messages are lost due to collision, which is realistic in practice (carrier sensing is well-studied). Without such an assumption, a silent round can no longer be used to encode information. Collin can create any arbitrary sequence, of length β, consisting of messages and silences. Because β is unknown, it is easy to see that no communication protocol can ever safely terminate. In this sense, the absence of collision detection provides Collin with infinite power.

While the lower bounds proved in this paper were for deterministic players, we conjecture that they hold even for randomized protocols. For example, consider the case where Alice and Bob are non-adaptive, i.e., the broadcast schedules and the contents of the messages are determined only as a function of the initial values and the random choices. In this case, Collin can delay Alice and Bob for $2\beta + \Theta(\log V)$ rounds by ignoring rounds in which the behavior is strictly probabilistic, and filling-in rounds (as in the deterministic lower bound) where the behavior is guaranteed. Following a similar argument as presented in Section 4, it is possible to show that, in expectation, Alice and Bob do not terminate for 2β rounds. Deriving an adaptive lower bound is an interesting open problem.

Interestingly, it is straightforward to demonstrate that our lower bounds hold for weaker games. That is, Lemmas 2 and 3 imply that calculating equality, *bitwise-and* or *bitwise-or* have the same round complexity as the 3-player game.

An obvious future research direction is to extend our results to multihop environments. For example, when considering reliable broadcast over multiple hops, will the single-hop bound scale naturally with the hop count, or are the players (or adversary) able to gain additional advantage in this new setting?

Acknowledgments

We are grateful to H. Attiya for her helpful comments, and to G. Chockler for many long discussions and key insights.

References

[1] T.X. Brown, J.E. James, A. Sethi, Jamming and sensing of encrypted wireless ad hoc networks, Technical Report CU-CS-1005-06, UC Boulder, 2006.
[2] A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, D.E. Culler, Spins: Security protocols for sensor networks, Wireless Networks 8 (5) (2002) 521–534.
[3] C. Karlof, N. Sastry, D. Wagner, Tinysec: A link layer security architecture for wireless sensor networks, in: Embedded Networked Sensor Systems, 2004.
[4] C.Y. Koo, Broadcast in radio networks tolerating byzantine adversarial behavior, in: Principles of Distributed Computing, 2004, pp. 275–282.
[5] V. Bhandari, N.H. Vaidya, On reliable broadcast in a radio network, in: Principles of Distributed Computing, 2005, pp. 138–147.
[6] A. Pelc, D. Peleg, Broadcasting with locally bounded byzantine faults, Information Processing Letters 93 (3) (2005) 109–115.
[7] V. Drabkin, R. Friedman, M. Segal, Efficient byzantine broadcast in wireless ad hoc networks, in: Dependable Systems and Networks, 2005, pp. 160–169.
[8] A. Pelc, D. Peleg, Feasibility and complexity of broadcasting with random transmission failures, in: Principles of Distributed Computing, 2005, pp. 334–341.
[9] A. Clementi, A. Monti, R. Silvestri, Optimal f-reliable protocols for the do-all problem on single-hop wireless networks, in: Algorithms and Computation, 2002, pp. 320–331.
[10] B.S. Chlebus, D.R. Kowalski, A. Lingas, The do-all problem in broadcast networks, in: Principles of Distributed Computing, 2001, pp. 117–127.
[11] E. Kranakis, D. Krizanc, A. Pelc, Fault-tolerant broadcasting in radio networks, in: European Symposium on Algorithms, 1998, pp. 283–294.
[12] A. Clementi, A. Monti, R. Silvestri, Round robin is optimal for fault-tolerant broadcasting on wireless networks, Journal of Parallel Distributed Computing 64 (1) (2004) 89–96.
[13] C.Y. Koo, V. Bhandari, J. Katz, N.H. Vaidya, Relibable broadcast in radio networks: The bounded collision case, in: Principles of Distributed Computing, 2006.
[14] M. Stahlberg, Radio jamming attacks against two popular mobile networks, in: Helsinki University of Technology Seminar on Network Security, 2000.
[15] R. Negi, A. Perrig, Jamming analysis of mac protocols, Technical Report, Carnegie Mellon University, 2003.
[16] Y. Hu, A. Perrig, A survey of secure wireless ad hoc routing, IEEE Security and Privacy Magazine 02 (3) (2004) 28–39.
[17] V. Gupta, S. Krishnamurthy, S. Faloutsos, Denial of service attacks at the mac layer in wireless ad hoc networks, in: Military Communications Conference, 2002.
[18] A. Woo, K. Whitehouse, F. Jiang, J. Polastre, D. Culler, Exploiting the capture effect for collision detection and recovery, in: Workshop on Embedded Networked Sensors, 2005, pp. 45–52.
[19] A. Clementi, A. Monti, R. Silvestri, Selective families, superimposed codes, and broadcasting on unknown radio networks, in: Symposium on Discrete algorithms, Philadelphia, PA, USA, 2001, pp. 709–718.
[20] D.R. Kowalski, On selection problem in radio networks, in: Principles of Distributed Computing, ACM Press, New York, NY, USA, 2005, pp. 158–166.