Distrib. Comput. (2008) 21:55–84
DOI 10.1007/s00446-008-0056-2

# Consensus and collision detectors in radio networks

**Gregory Chockler · Murat Demirbas · Seth Gilbert · Nancy Lynch · Calvin Newport · Tina Nolte**

Received: 29 September 2006 / Accepted: 21 February 2008 / Published online: 11 March 2008


**Abstract** We consider the fault-tolerant consensus problem in radio networks with crash-prone nodes. Specifically, we develop lower bounds and matching upper bounds for this problem in single-hop radios networks, where all nodes are located within broadcast range of each other. In a novel break from existing work, we introduce a collision-prone communication model in which each node may lose an arbitrary subset of the messages sent by its neighbors during each round. This model is motivated by behavior observed in empirical studies of these networks. To cope with this communication unreliability we augment nodes with receiver-side *collision detectors* and present a new classification of these detectors in terms of accuracy and completeness. This classification is motivated by practical realities and allows us to determine, roughly speaking, how much collision detection capability is enough to solve the consensus problem efficiently in this setting. We consider nine different combinations of completeness and accuracy properties in total, determining for each whether consensus is solvable, and, if it is, a lower bound on the number of rounds required. Furthermore, we distinguish anonymous and non-anonymous protocols—where "anonymous" implies that devices do not have unique identifiers—determining what effect (if any) this extra information has on the complexity of the problem. In all relevant cases, we provide matching upper bounds.

**Keywords** Wireless ad hoc networks · Consensus · Collision detectors · Fault-tolerance

This work is supported by MURI–AFOSR SA2796PO 1-0000243658, USAF–AFRL #FA9550-04-1-0121, NSF Grant CCR-0121277, NSF-Texas Engineering Experiment Station Grant XS64961-CS, and DARPA F33615-01-C-1896.

G. Chockler
IBM Haifa Research Laboratory, Mount Carmel,
31905 Haifa, Israel
e-mail: chockler@il.ibm.com

M. Demirbas
CSE, University at Buffalo, SUNY, Buffalo, NY 14260, USA
e-mail: demirbas@cse.buffalo.edu

S. Gilbert
EPFL IC, Lausanne, Switzerland
e-mail: seth.gilbert@epfl.ch

N. Lynch · C. Newport (✉) · T. Nolte
CSAIL, MIT, Office 32-G670, The Stata Center, 32 Vassar Street,
Cambridge, MA 02139, USA
e-mail: cnewport@mit.edu

N. Lynch
e-mail: nlynch@mit.edu

T. Nolte
e-mail: tnolte@mit.edu

## 1 Introduction

As wireless technology continues to improve and miniaturize, we observe an increasing interest in large-scale, widely-deployed radio networks. Many services and applications in these environments (e.g., TDMA scheduling, remote management and re-programming of sensors, temperature and climate control, and assembly line monitoring) require wireless devices to coordinate their actions in the face of failures resulting from hardware malfunction, physical damage, battery depletion, or enforced hibernation. Fault-tolerant agreement, that is, *consensus*, is an essential building block for these applications as it facilitates consistent distributed behavior.

In this paper, we study the fault-tolerant consensus problem in radio networks with crash-prone devices. We focus on solving consensus in *single-hop* networks where the devices

are located within communication range of each other and are sufficiently synchronized to operate in a synchronous (i.e., round by round) manner. Radio networks introduce several challenges not found in typical distributed systems.

First, communication in wireless networks is unreliable: collisions, wireless interference, and other electromagnetic anomalies may cause significant message disruption. Second, the deployment of devices cannot be carefully controlled, so the number of deployed devices (and, perhaps, the density of the deployment) is a priori unknown. Moreover, the devices may be "anonymous," meaning that they have no unique identifiers.

To cope with undetectable message loss, we augment the devices with *collision detectors*. Collision detectors monitor the broadcast medium and attempt to deliver notifications when message loss is detected. They do not provide any information with respect to the number of lost messages or the identities of their senders. Moreover, there is no guarantee that a device performing a transmission can detect collisions (unlike, for example, Ethernet networks [29]).

In a novel break from prior work, we consider collision detectors that may be *unreliable*. Inspired by [6], we classify collision detectors according to their *completeness*, the ability to detect collisions, and *accuracy*, the ability to report only real collisions (no false positives). For each collision-detector class that we introduce, we show how to solve consensus and provide matching lower bounds.

We consider two accuracy properties: permanent accuracy and eventual accuracy. While permanently accurate collision detectors are more powerful, eventually accurate collision detectors are more realistic, as they result in algorithms that are robust in the face of false positives (caused, perhaps, by electromagnetic noise and broadcasts by nearby devices). The latter is particularly important for multi-hop algorithms in which neighboring regions might interfere with local collision detection.

Since most current collision detector implementations can occasionally miss a collision, we also consider several ways of weakening the assumption of completeness. In particular, we consider: (1) a *majority complete* collision detector that guarantees to detect a collision only if half or more of the messages sent in a round are not received, (2) a *half complete* collision detector that guarantees to detect a collision only if more than half of the messages sent in a round are not received, and (3) a *0-complete* collision detector that guarantees to detect a collision only if all the messages sent in a round are lost. Not all of these classes are meant to correspond to specific hardware devices. They are used, instead, to probe the exact completeness threshold at which the complexity of agreement changes. For example, the difference between the *majority* and *half* complete properties is small. We show, however, that moving from one to the other introduces a significant increase in complexity.

We consider the eight collision detector classes obtained by combining these completeness and accuracy properties. We also consider the possibility of no collision detector. We analyze the computational power of each of these classes in the context of two other relevant network parameters: eventual stabilization of message loss behavior, and the presence/absence of unique identifiers. We consider (1) the ability to solve consensus, (2) the solution complexity, and (3) the robustness to message loss. Our results provide separation among many of these classes in terms of the parameters above (Fig. 1).

An important contribution of our analysis is in providing feedback to hardware/firmware designers with respect to the requirements for collision detectors. In fact, one of the main questions motivating this research is the question of how reliable a collision detector is really needed in a radio network. While there recently has been significant progress in implementing collision detection [13,32,38], there has been little formal analysis of the minimal requirements. We show that reasonable and readily implementable collision detectors are sufficient.

### 1.1 Network model overview

In Sects. 4, 5, and 6 we formally define our model. This model captures a single-hop radio network in which all nodes are within broadcast range of each other. (Note, the focus on a single hop is common practice when studying coordination problems in a wireless setting; cf., [12,25]).

Because we are interested in ad hoc deployments, we assume that the number of devices participating in the network is a priori unknown. We consider both devices with and without unique identifiers. We call the former "non-anonymous" and the latter "anonymous." Indeed, one of the questions we address in this study is the power gained by having reliable identification.

Algorithms proceed in synchronized rounds. We note that in real radio systems, the length of a communication round might be long relative to the time required to transmit a single packet. (Tuning a round to the exact time required to transmit, though often assumed in theory, is impractical in real world deployments. Clock drift and dynamic link layers make such precision difficult to achieve.) This allows for the possibility of *multiple messages* being received in a single round. Our model is the first, that we know of, to capture this empirically-inspired potential complexity.

Any subset of the messages sent during a given round can be lost at any receiver (with the trivial exception that senders always receive their own messages). To mitigate this loss we introduce (potentially unreliable) collision detectors. As mentioned, we describe these detectors in terms of *completeness* and *accuracy*. A goal of this study is to determine, roughly speaking, how much collision information is enough

to solve consensus. To allow progress, we consider networks that satisfy an *eventual collision freedom* property that states that, eventually, if only a single device transmits, all devices receive the message. This is a strict generalization of the broadcast behavior assumed in most studies of this setting. (These prior studies maintain that if a single device transmits its message is *always* received. We note that unexpected interference might make this property hard to maintain in every round.)

Finally, we also introduce a service which we call a contention manager. This service encapsulates the task of reducing contention on the broadcast channel. In each round, the manager suggests that each device either be *active* or *passive*. Informally, the former is meant to indicate that a device can try to broadcast in the upcoming round, and the latter indicates that a device should be silent. Most reasonable contention managers should eventually stabilize on only a small number of devices (e.g., 1) being labeled as *active*, thus allowing, in executions satisfying eventual collision freedom, for messages to be delivered without collision.

The contention manager captures the challenge of distributed medium access control. This problem is well-studied in practice [4,7,18,19,21,29,33]. Our goal is to separate the complexity of reducing contention from the complexity of solving specific coordination problems such as consensus. Accordingly, our termination bounds are given relative to the stabilization point of the contention manager.

### 1.2 Upper bounds overview

In Sects. 7 and 8, we describe our consensus algorithms.

Algorithm 1 assumes a majority complete collision detector (which reports a collision if the process did not receive at least a majority of the messages transmitted). It solves consensus in a constant number of rounds after the eventual collision freedom property holds and the contention manager has stabilized. The algorithm proceeds in round pairs. In the first round of the pair, active processes propose a decision value. In the second round, processes that failed to receive a proposal transmit a veto. The collision detector ensures that if a veto is sent, then all processes will either receive this veto or a collision notification and, therefore, know not to decide. The detector also prevents partitions in the proposal phase. If two processes receive two different proposals, the detector notifies them of their loss.

Algorithm 2 weakens the collision detector to satisfy only 0-completeness (which guarantees to report a collision only if *all* messages are lost). It terminates in $O(\lg |V|)$ rounds, where $V$ is the set of potential decision values. This algorithm also uses a propose/veto structure. The difference, however, is that the propose phase has expanded to include one round for each bit in the proposal. The single-round proposal of Algorithm 1 cannot work in this setting, as the weak guarantees of a 0-complete detector might allow processes to partition without knowing it.

Algorithm 3 weakens the network model so that it no longer guarantees any message delivery, even if only a single process is transmitting. Processes are left to communicate, in the worst case, through collision notifications. Using these binary semaphores, processes walk through a search tree of decision values trying to identify a valid initial value to decide. The algorithm terminates in $O(f \lg |V|)$ rounds, where $f$ is the total number of failures. (A failed process can cause the processes to start over at the top of the tree.)

The first two algorithms assume only eventual accuracy. That is, eventually the collision detectors stop reporting false positives. The third algorithm requires permanent accuracy. All three algorithms are anonymous. For the sake of completeness, we address, in Sect. 8.3, the non-anonymous setting in which the space of possible IDs is smaller than the space of decision values (a rare case). We sketch an algorithm that runs leader election on the IDs, and terminates in $O(\lg |I|)$ rounds, where $I$ is the ID space.

### 1.3 Lower bounds overview

In Sect. 9 we prove lower bounds.

We start with Theorem 4 which shows consensus to be impossible without collision detection. The proof employs a partitioning argument. It follows directly, in Theorem 5, that consensus is also impossible with a detector that never guarantees accuracy. (Permanent false positives render the detector useless.)

In Theorem 6, we show that an anonymous consensus solution using a detector that is no better than half-complete requires $\Omega(\lg |V|)$ rounds. This proves Algorithm 2 tight. The proof uses a counting argument to show that there exist two initial values that generate the same sequence of transmissions for the first $\lg |V|$ rounds. During these rounds, the simultaneous communication can prevent two partitions, each starting with one of these two initial values, from learning about the other—delaying decision. In Theorem 7, we extend the argument to non-anonymous algorithms and refine the bound to $\Omega\left(\min\left\{\lg |V|, \lg \frac{|I|}{n}\right\}\right)$, showing the algorithm described in Sect. 8.3 to be (close to) tight. [1]

In Theorem 8, we show consensus to be impossible in an environment with eventual accuracy and no eventual collision freedom guarantee. This result proves Algorithm 3's requirement of an accurate detector to be optimal. The proof argues that in a setting where no messages are delivered, false collision notifications can be arranged to communicate any arbitrary value (violating validity).

[1] The described algorithm runs in $\lg |I|$ not $\lg \frac{|I|}{n}$ rounds.

**Algorithm 1: Solving consensus with ECF and a collision detector from maj-$\diamond\mathcal{AC}$.**

```
1  Process P_i:
2    estimate_i ∈ V, initially set to the initial value of process P_i
3    phase_i ∈ {proposal, veto}, initially proposal
4    For each round r, r ≥ 1 do:
5      if (phase_i = proposal) then
6        if CM()_i = active then
7          bcast(estimate_i)_i
8        messages_i ← SET(recv()_i)
9        CD-advice_i ← CD()_i
10       if (CD-advice_i ≠ ±) and (|messages_i| > 0) then
11         estimate_i ← min{messages_i}
12       phase_i ← veto
13     else if (phase_i = veto) then
14       if (CD-advice_i = ±) or (|messages_i| > 1) then
15         bcast(veto)_i
16       veto-messages_i ← recv()_i
17       CD-advice_i ← CD()_i
18       if (veto-messages_i = ∅) and (CD-advice_i = null) and (|messages_i| = 1) then
19         decide(estimate_i)_i
20       phase_i ← proposal
21
```

**Algorithm 2: Solving consensus with ECF and a 0-$\diamond\mathcal{AC}$ collision detector.**

```
1  Process P_i:
2    estimate_i ∈ V^{0,1}, initially set to a binary rep. of P_i's initial value
3    phase_i ∈ {prepare, propose, accept}, initially prepare
4    size ← ⌈lg |V|⌉
5    For each round r, r ≥ 1 do:
6      if (phase_i = prepare) then
7        if CM()_i = active then
8          bcast(estimate_i)_i
9        messages_i ← SET(recv()_i)
10       CD-advice_i ← CD()_i
11       if (CD-advice_i ≠ ±) and (|messages_i| > 0) then
12         estimate_i ← min{messages_i}
13       decide_i ← true
14       bit_i ← 1
15       phase_i ← propose
16     else if (phase_i = propose) then
17       if (estimate_i[bit_i] = 1) then
18         bcast(veto)_i
19       votes_i ← recv()_i
20       CD-advice_i ← CD()_i
21       if ((|votes_i| > 0) or (CD-advice_i = ±)) and (estimate_i[bit_i] = 0) then
22         decide_i ← false
23       bit_i ← bit_i + 1
24       if (bit_i > size) then
25         phase_i ← accept
26     else if (phase_i = accept) then
27       if (not decide_i) then
28         bcast(veto)_i
29       veto-messages_i ← recv()_i
30       CD-advice_i ← CD()_i
31       if (|veto-messages_i| = 0) and (CD-advice_i ≠ ±) then
32         decide(estimate_i)_i
33       phase_i ← prepare
```

**Algorithm 3: Solving consensus with a 0-$\mathcal{AC}$ collision detector but without ECF.**

```
1  Process P_i:
2    estimate_i ∈ V, initially set to the initial value of process P_i
3    phase_i ∈ {vote-val, vote-left, vote-right, recurse}, initially vote-val
4    curr_i, A node pointer, initially set to the root of a balanced binary search tree representation of V
5    For each round r, r ≥ 1 do:
6      if (phase_i = vote-val) then
7        if (estimate_i = val[curr_i]) then
8          bcast('vote')_i
9        msgs(1)_i ← recv()_i
10       CD(1)_i ← CD()_i
11       phase_i ← vote-left
12     else if (phase_i = vote-left) then
13       if (estimate_i ∈ left[curr_i]) then
14         bcast('vote')_i
15       msgs(2)_i ← recv()_i
16       CD(2)_i ← CD()_i
17       phase_i ← vote-right
18     else if (phase_i = vote-right) then
19       if (estimate_i ∈ right[curr_i]) then
20         bcast('vote')_i
21       msgs(3)_i ← recv()_i
22       CD(3)_i ← CD()_i
23       phase_i ← recurse
24     else if (phase_i = recurse) then
25       if (|msgs(1)_i| > 0) or (CD(1)_i = ±) then
26         decide(val[curr_i])_i
27       else if (|msgs(2)_i| > 0) or (CD(2)_i = ±) then
28         curr_i ← left[curr_i]
29       else if ((|msgs(3)| > 0) or (CD(3)_i = ±)) then
30         curr_i ← right[curr_i]
31       else
32         curr_i ← parent[curr_i]
33       phase_i ← vote-val
34
```

Finally, in Theorem 9, we show that consensus with an accurate detector and no eventual collision freedom requires $\Omega(\lg |V|)$ rounds. This proves Algorithm 3 tight. The proof (sketch) notes that collision notifications are the only means of communication if messages are never guaranteed to be delivered. A notification leaks only one bit of information. To spell out a decision value, therefore, requires one round for each bit (Fig. 1).

## 2 Related work

Local radio broadcast is inherently unreliable due to the possibility of message collisions. Many solutions have been proposed to mitigate some of this uncertainty. For example, the most widely-used MAC layers in wireless ad hoc networks make use of physical carrier sensing and exponential backoff to help reduce contention on the channel; cf. [1,32,36,39]. For *unicast* communication with a known recipient virtual carrier sensing (the use of *clear to send* and *ready to send* control messages) can be used to help eliminate the well-known *hidden terminal problem* and *exposed terminal problem* (see [5] for a more extensive discussion of these common problems and how virtual carrier sensing attempts to solve them). Similarly, in these situations where the recipients are known, link-layer acknowledgments can be used to help the sender verify the success or failure of its transmission and potentially trigger re-transmissions as needed.

In many cases, however, the recipients are unknown, rendering virtual carrier sensing and link-layer acknowledgments unusable. In practical multicast communication, though physical carrier sensing goes a long way toward reducing message loss on the wireless medium, it does not eliminate it. To verify this reality, consider empirical studies of ad hoc networks, such as [17,24,37,40], which show that even with sophisticated collision avoidance mechanisms (e.g., 802.11 [1], B-MAC [32], S-MAC [39], and T-MAC [36]), and even assuming low traffic loads, the fraction of messages being lost can be as high as 20–50%. Accordingly, algorithm design for these networks *must* take into account the expectation of lost messages.

### 2.1 Practical implementations of collision detectors and contention managers

*Collision detectors* Initial experiments with collision detector implementations have shown that simple detection

| Collision Detector Properties | Collision Freedom Assumption | |
|---|---|---|
| | **Eventual Collision Freedom** | **No Eventual Collision Freedom** |
| **Accuracy & Completeness** | **Time:** $\Theta(1)$<br>Upper: Alg. 1 in Sec. 8.1<br>Lower: Immediate | **Time:** $O(f \lg \lvert V \rvert)$<br>Upper: Alg. 3 in Sec. 8.4<br>Anon. Lower: Thm. 9 in Sec. 9.5<br>Non-Anon. Lower: Open |
| **Accuracy & Maj. Completeness** | **Time:** $\Theta(1)$<br>Upper: Alg. 1 in Sec. 8.1<br>Lower: Immediate | **Time:** $O(f \lg \lvert V \rvert)$<br>Upper: Alg. 3 in Sec. 8.4<br>Anon. Lower: Thm. 9 in Sec. 9.5<br>Non-Anon. Lower: Open |
| **Accuracy & Half Completeness** | **Anon. Time:** $\Theta(\lg \lvert V \rvert)$<br>**Non-Anon. Time:** $\Theta(\min \lg \lvert V \rvert, \lg \lvert I \rvert)$<br>Anon. Upper: Alg. 2 in Sec. 8.2<br>Non-Anon. Upper: Sec. 8.3<br>Anon. Lower: Thm. 6 in Sec. 9.3.3<br>Non-Anon. Lower: Thm. 7 & Cor. 3 in Sec. 9.3.4 | **Time:** $O(f \lg \lvert V \rvert)$<br>Upper: Alg. 3 in Sec. 8.4<br>Anon. Lower: Thm. 9 in Sec. 9.5<br>Non-Anon. Lower: Open |
| **Accuracy & Zero Completeness** | **Anon. Time:** $\Theta(\lg \lvert V \rvert)$<br>**Non-Anon. Time:** $\Theta(\min \lg \lvert V \rvert, \lg \lvert I \rvert)$<br>Anon. Upper: Alg. 2 in Sec. 8.2<br>Non-Anon. Upper: Sec. 8.3<br>Anon. Lower: Thm. 6 in Sec. 9.3.3<br>Non-Anon. Lower: Thm. 7 & Cor. 3 in Sec. 9.3.4 | **Time:** $O(f \lg \lvert V \rvert)$<br>Upper: Alg. 3 in Sec. 8.4<br>Anon. Lower: Thm. 9 in Sec. 9.5<br>Non-Anon. Lower: Open |
| **Eventual Acc. & Completeness** | Same as Acc. and Compl. | Impossible<br>Thm. 8 in Sec. 9.4 |
| **Eventual Acc. & Maj. Completeness** | Same as Acc. and Maj. Compl. | Impossible<br>Thm. 8 in Sec. 9.4 |
| **Eventual Acc. & Half Completeness** | Same as Acc. and Half Compl. | Impossible<br>Thm. 8 in Sec. 9.4 |
| **Eventual Acc. & Zero Completeness** | Same as Acc. and Zero Compl. | Impossible<br>Thm. 8 in Sec. 9.4 |
| **No CD** | Impossible<br>Thm. 4 in Sec. 9.1 | Impossible<br>Thm. 4 in Sec. 9.1 |
| **No Accuracy** | Impossible<br>Thm. 5 in Sec. 9.2 | Impossible<br>Thm. 5 in Sec. 9.2 |

**Fig. 1** Summary of results for each combination of collision detector property and collision freedom assumption. Each cell includes a tight running time and pointers to the sections containing the corresponding upper and lower bounds. When not otherwise specified, a bound is assumed to hold for both the anonymous and non-anonymous case. The value $V$ represents the size of the consensus value space, $I$ the size of the ID space, and $f$ the bound on the number of failures

schemes can achieve zero completeness in 100% of rounds, and majority completeness in over 90% of rounds. We are confident that with further refinement the majority completeness property can be satisfied in much closer to 100% of rounds. Our approach, which is admittedly rudimentary, relies on a combination of carrier sensing and failed CRC checks to indicate interference. (See [9] for a more detailed discussion) More experienced electrical engineers could, undoubtedly, produce significantly better heuristics for detecting this behavior. Indeed, in a study by Deng et al. [13], it is suggested that there currently exists no technical obstacle to adding carrier-sensing based collision detection support to the current 802.11 protocol. Even better results can be obtained if we consider hardware modifications. The use of a separate busy-tone channel [20], for example, would make zero complete detection trivial.

*Contention managers* In our simulation and testbed implementations of the consensus algorithms that follow (Sect. 8), we relied on a simple contention manager driven by random coin-flips. Specifically, all contention managers initially return *active*. If contention is detected on the channel, the contention manager switches to *passive* with probability 0.5. If silence is detected on the channel, a manager switches from *passive* to *active* with the same probability. In our experiments, this simple approach consistently yielded stabilization (collision free broadcast) in approximately $\lg n$ rounds [10].

Beyond our simple implementation, the general problem of contention on a multiple access channel has been well-studied. Starting with early work on protocols such as Aloha [2] and the binary exponential backoff used in Ethernet [29], the development and analysis of contention management on a shared channel has enjoyed considerable attention; cf., [4,7,15,18,19,21,22,29,30,33]. More recently, for example, Jurdzinski and Stachowiak [22] prove a $\Omega((\log n \log(1/\epsilon))/(\log\log n + \log\log(1/\epsilon))$ round lower bound on isolating a single process in a single-hop radio network with uniform algorithms, known $n$, and a global round counter. (A bound of $\Omega(n \log n)$ exists for unknown $n$ and local round counters.) Farach-Colton et al. [15] tighten the lower bound to $\Omega(\log n \log(1/\epsilon))$ rounds, which matches an upper bound provided in [22]. It remains open how these bounds would change in our model of variable-strength receiver collision detection.

## 2.2 Consensus in wireless networks

There has been extensive prior work focused on fault-tolerant consensus in synchronous [28], partially synchronous [14], asynchronous with failure detectors [27,6] and fully asynchronous [16] message passing systems with reliable or

eventually reliable point-to-point channels. In particular, to tolerate message loss the work of [14,27] assumes an eventually connected majority component and an a priori known number of participants. Both of these assumptions are unavailable in the wireless ad hoc environments we consider.

Santoro and Widmayer [34,35] study consensus in the presence of unreliable communication, and show that consensus is impossible if as few as $(n-1)$ of the $n^2$ possible messages sent in a round can be lost. In this study, we circumvent this impossibility result with both our collision detectors and contention managers; which can be used in executions that satisfy eventual collision freedom to provide eventual message reliability. Also, algorithms in [35] are not applicable in our setting because they rely on a priori knowledge of the number of participants, and they do not tolerate node failures.

In [26], Kumar presents a quorum-based solution that solves fault-tolerant consensus among subsets of nodes in a multi-hop wireless sensor network. The model, however, differs from ours in that it requires nodes to have significant advance knowledge of the network topology, and failure behavior is constrained to maintain specific redundancy guarantees.

Aspnes et al. [3] present a solution for consensus in wireless networks with anonymous but reliable nodes, and reliable communication. Although anonymity is not a primary focus of our paper, most of our algorithms are, in fact, anonymous as they do not use node identifiers. In addition, our algorithms work under more realistic environment assumptions as they tolerate unreliable communication and node crashes.

Koo [23] presents an (almost) tight lower bound for the maximum fraction of Byzantine neighbors that still allows atomic broadcast to be solved in radio networks where each node adheres to a pre-defined transmission schedule. We do not consider Byzantine failures and, unlike Koo, we do assume unreliable broadcast.

We presented the justification and main properties of our model in [8]. Many of the algorithms and lower bounds examined in this study were first described in [11]. And, in [9], we discussed how to implement the elements of our model in practice.

## 3 Preliminaries

The following supplementary definitions aid the description of our model, upper bounds, and lower bounds.

- Given two multisets $M_1$ and $M_2$, $M_1 \subseteq M_2$ indicates that for all $m \in M_1$: $m \in M_2$ and $m$ does not appear in $M_1$ more times than it appears in $M_2$.
- Given two multisets $M_1$ and $M_2$, $M_1 \bigcup M_2$ indicates the multiset union of $M_1$ and $M_2$ in which any element $m \in M_1$ (resp. $m \in M_2$) appears the total number of times that $m$ appears in $M_1$ and $M_2$.
- We say a multiset $M$ is *finite* if it is described by only a finite number of (value, number) pairs.
- For a finite multiset $M$, described by a sequence of (value, number) pairs, we use $|M|$ to indicate the sum of the number components of these pairs, that is, the total number of instances of all values in $M$.
- For a finite set of values $V$, we use $Multi(V)$ to indicate the set of all possible finite multisets defined over $V$.
- For a finite set $S$, we use $MS(S)$ to indicate the multiset containing one of each element in $S$.
- For a finite multiset $M$, we use the notation $SET(M)$ to indicate the set containing every value that appears in $M$.

## 4 The system model

We model a synchronous single-hop broadcast network with non-uniform message loss, contention management, and collision detection. Formally, we define $I$ to be the finite set of all possible process indices, and $M$ to be a fixed message alphabet. We then provide the following definitions:

**Definition 1 (Process)** A *process* is some automaton $A$ consisting of the following five components:

1. $states_A$, a potentially infinite set of *states*.
2. $start_A$, a non-empty subset of $states_A$ known as the *start states*.
3. $fail_A$, a single state from $states_A$ known as the *fail state*.
4. $msg_A$, a message generation function that maps

   $$states_A \times \{active, passive\} \rightarrow M \bigcup \{null\}$$

   where $M$ is our fixed message alphabet and *null* is a placeholder indicating no message. We assume:

   $$msg_A(fail_A, *) = null$$

5. $trans_A$, a state transition function mapping

   $$states_A \times Multi(M) \times \{\pm, null\} \times \{active, passive\} \rightarrow states_A$$

   where $Multi(M)$ is the set of all possible finite multisets defined over $M$. We assume:

   $$trans_A(fail_A, *, *, *) = fail_A$$

In this definition, $\{\pm, null\}$ represent the possible inputs from the collision detector, and $\{active, passive\}$ the possible advice from the contention manager. The $fail_A$ state models crash failures.

**Definition 2 (Algorithm)** An algorithm is a mapping from $I$ to processes.

An algorithm specifies which process runs on each node. Notice, an algorithm $\mathcal{A}$ can encode $i$ in the state of each automaton $\mathcal{A}(i)$. This models unique IDs. For settings in which we desire no unique IDs, we use the following definition:

**Definition 3 (Anonymous)** An algorithm $\mathcal{A}$ is *anonymous* if and only if: $\forall i, j \in I, \mathcal{A}(i) = \mathcal{A}(j)$.

Next, we define a *P-transmission trace*, defined over a non-empty subset $P$ of $I$. It describes, at each round of an execution involving processes in $P$, how many processes broadcast a message (notated $c_i$) and the number of messages received at each process (notated $T_i$).

**Definition 4 ($P$-transmission trace)** A $P$-transmission trace, where $P$ is a non-empty subset of $I$, is an infinite sequence of ordered pairs $(c_1, T_1), (c_2, T_2), ...$ where each $c_i$ is a natural number less than or equal to $|P|$, and each $T_i$ is a mapping from $P$ to $\{0, ..., c_i\}$.

*P-CD trace* describes what collision detector advice each process receives in each round.

**Definition 5 ($P$-CD trace)** A $P$-CD trace, where $P$ is a non-empty subset of $I$, is an infinite sequence of mappings, $CD_1$, $CD_2$, ... where each $CD_i$ maps from $P$ to $\{\pm, null\}$.

We can now formally define a collision detector, for a given set $P$ of indices, as a function from $P$-transmission traces to a set of $P$-CD traces. That is, given a description of how many messages are sent in each round, and how many messages each process receives in each round, the collision detector describes which sequences of collision detector advice are valid. Notice, this definition prevents the collision detector from making use of the identity of the senders or the contents of the messages.

**Definition 6 ($P$-Collision detector)** A $P$-collision detector, where $P$ is a non-empty subset of $I$, is a function from $P$-transmission traces to non-empty sets of $P$-CD traces.

To define a contention manager, we first define, as we did for the collision detector, the relevant type of trace. Here, this is a $P$-CM trace which simply describes which contention manager advice (either $active$ or $passive$) is returned to each process during each round.

**Definition 7 ($P$-CM trace)** A $P$-CM trace, where $P$ is a non-empty subset of $I$, is an infinite sequence of mappings, $CM_1$, $CM_2$, ... where each $CM_i$ maps from $P$ to $\{active, passive\}$.

We can now formally define a contention manager, for a set $P \subseteq I$, as a function from some set $A \subseteq P$ to a set of $P$-CM traces. $A$ can be, for example, the correct processes in the system (those which never enter the $fail$ state), allowing for manager definitions that stabilize to only correct process(es) being active.

**Definition 8 ($P$-Contention manager)** A $P$-contention manager, where $P$ is a non-empty subset of $I$, is a function from each subset of $P$ to a non-empty set of $P$-CM traces.

Next we define an environment, which describes a group of process indices, a collision detector, and a contention manager. Roughly speaking, an environment describes the platform on which we can run an algorithm.

**Definition 9 (Environment)** An environment in our model consists of:

- $P$, a non-empty subset of $I$,
- a $P$-collision detector, and
- a $P$-contention manager.

For a given environment $E$, we use the notation $E.P$ to indicate the set of process indices described by $E$, $E.CD$ to indicate the collision detector described by $E$, and $E.CM$ to indicate the contention manager described by $E$.

Finally, we define a system, which is the combination of an environment with a specific algorithm. Because an environment describes a set of process indices, and an algorithm is a mapping from process indices to processes, a system describes a set of specific processes and the collision detector and contention manager that they have access to. Notice, because we can combine any algorithm with any environment, the processes described by a system will have no *a priori* knowledge of the number of other processes also in the system (beyond, of course, the loose upper bound $|I|$ on the ID space).

**Definition 10 (System)** A system in our model is a pair $(E, \mathcal{A})$, consisting of an environment $E$, and an algorithm $\mathcal{A}$.

### 4.1 Executions and indistinguishability

An **execution** $\alpha$ of a system $(E, \mathcal{A})$, is an infinite sequence

$$C_0, M_1, N_1, D_1, W_1, C_1, M_2, N_2, D_2, W_2, C_2, ...$$

where $C_r$ describes the process states in round $r$, $M_r$ the sent messages in $r$, $N_r$ the received messages in $r$, and $D_r$

and $W_r$ the collision detector and contention manager advice, respectively, in $r$, along with the appropriate well-formedness criteria. (See Definition 19 in Appendix A for the formal definition.)

That is, we assume the states evolve according to the transition function for $\mathcal{A}$ and the message receive behavior upholds integrity and no-duplication. A process that transmits in $r$ receives its own message in $r$. Loss at other processes is unconstrained (i.e., in a given round, any process can fail to receive any arbitrary subset of messages transmitted by other processes). We assume that the collision detector and contention manager advice is consistent with the definitions of the detector and manager described by $E$. That is, the collision advice is consistent with one of the $E.P$-CD traces returned by passing $E.CD$ the $E.P$-transmission trace described by $\alpha$, and the contention advice is consistent with one the $E.P$-CM traces, returned by passing $E.CM$ the set of processes that never enter their $fail$ state in $\alpha$.

Following the normal convention, we say two executions defined over systems $(E, \mathcal{A})$ and $(E', \mathcal{A})$, are **indistinguishable** with respect to a process $\mathcal{A}(i)$, $i \in E.P \cap E'.P$, through round $k$, if the process has the same states, received messages, collision detector and contention manager advice through the first $k$ rounds. (See Definition 20 in Appendix A for the formal definition).

The following helper functions will also prove useful in our discussion. For a given execution $\alpha$ defined over system $(E, \mathcal{A})$, we define:

- $t_T(\alpha)$ to be the $E.P$-transmission trace $(c_1, T_1), (c_2, T_2), \ldots$ where for all $i > 0$: $c_i$ describes the number of messages transmitted in round $i$, and $T_i[j]$, for all $j \in E.P$, describes the number of messages received by $\mathcal{A}(j)$ in $i$.
- $t_{CD}(\alpha)$ to be the $E.P$-CD trace $CD_1, CD_2, \ldots$ where for all $i > 0$ and $j \in E.P$, $CD_i[j]$ describes the collision advice given to $\mathcal{A}(j)$ in $i$.
- $t_{CM}(\alpha)$ to be the $E.P$-CD trace $CM_1, CM_2, \ldots$ where for all $i > 0$ and $j \in E.P$, $CM_i[j]$ describes the contention manager advice given to $\mathcal{A}(j)$ in $i$.
- $t_C(\alpha)$ to be the largest possible subset $A$ of $P$ where for all $j \in A$, process $j$ never enters its $fail$ state in $\alpha$.

(See Appendix A for more formal definitions).

### 4.2 Process failures and message loss

*Process failures* Any number of processes can fail by crashing (that is, permanently stop executing). This is captured in our model by the $fail$ state of each process. As described in our execution definition, any process, during any round, can be non-deterministically transitioned into its fail state. Once there, by the definition of a process, it can never leave the fail state and never broadcast any message. We say a process is **correct** in a given execution, if and only if it never enters its fail state. Notice, $t_C(\alpha)$ describes the complete set of correct processes in execution $\alpha$. (See Definition 21 in Appendix A for the formal definition).

*Message loss* As described above, any process in any round can fail to receive any subset of messages sent by other processes. Recall, however, that in real systems, if only a single process broadcasts during a given round, we might reasonably expect that message to be successfully received. This might not *always* be true, as, for example, interference from outside of our single-hop area could occasionally cause non-uniform message disruption, but we could expect this property to hold *eventually*.[2] Accordingly, we define a communication property, which we refer to as the **eventual collision freedom (ECF)** property, that captures this behavior. If an execution satisfies ECF, then there exists a round $r_{\text{cf}}$, such that in all rounds $r \geq r_{\text{cf}}$, if only one process transmits, all processes receive this single message. (See Definition 1 in Appendix A for the formal definition).

## 5 Contention managers

In our model, the contention manager encapsulates the task of reducing contention on the broadcast channel. In each round, the manager suggests that each process either be *active* or *passive*. Informally, the former is meant to indicate that a process can try to broadcast in the upcoming round, and the latter indicates that a process should be silent. Most reasonable contention managers should eventually stabilize on a single process being labeled as *active* in each round, thus allowing, in executions satisfying eventual collision freedom, for messages to be delivered without collisions.

Notice there are some similarities between contention managers and failure detectors. A contention manager that stabilizes to a single *active* process, for example, resembles $\Omega$. Failure detectors, however, provide information on failures and help processes cope with asynchrony. Contention managers, on the other hand, provide contention resolution. Failure detectors encapsulate partial synchrony. Contention managers encapsulate randomized back-off strategies.

### 5.1 The wake-up and leader election services

A natural contention manager property can be defined as follows: a given $P$-contention manager, $S_{CM}$ is a **wake-up**

[2] As is often the case in distributed system definitions, the notion that a property holds for the rest of an execution starting at a certain, unknown point, is a generalization of the more realistic assumption that the property holds for a sufficiently long duration.

**service**, if there exists a round $r_{\text{wake}}$ such that for all rounds $r \geq r_{\text{wake}}$, only a single correct process is advised to be $active$.

A reasonable extension of this property might guarantee stabilization to a *single* leader. We provide the following definition: a $P$-contention manager $S_{CM}$ is a **leader election service**, if there exists a round $r_{\text{lead}}$ such that for all rounds $r \geq r_{\text{lead}}$, the same single correct process is the only process advised to be $active$. (See Properties 2 and 3 in Appendix A for the formal definitions).

Notice, by definition, a leader election service is also a wake-up service. To obtain the strongest possible results, we will use the stronger leader election service when constructing lower bounds and the weaker wake-up service when constructing the matching upper bounds.

### 5.2 Contention manager classes

A contention manager class is simply the set of *all* contention managers that satisfy a specific property. In this paper, we consider three such classes:

1. The **WS** class includes all wake-up services.
2. The **LS** class includes all leader-election services.
3. To aid the definition of our third class, we first define the $P$-contention manager $NOCM_P$, where $P$ is a non-empty subset of $I$, to be the trivial contention manager that assigns $active$ to all process indices in all rounds. Using this definition, we define the **NoCM** class to be the set consisting of $NOCM_P$ for all non-empty subsets $P \subseteq I$.

### 5.3 The maximal leader election service

To aid the construction of lower bounds, it will prove useful to define a contention manager that captures, for a given set $P$ of process indices, all possible contention manager behaviors that satisfy the leader election service property for this set. We call this the *maximal leader election service for* $P$ as it represents the maximal element in the set of all $P$-contention managers that satisfy the leader election service property. Formally, we use the notation $MAXLS_P$ to refer to this contention manager for a given $P$. The formal definition is given in Definition 22 in Appendix A.

## 6 Collision detectors

We classify collision detectors in terms of their *completeness* and *accuracy* properties. The former describes the conditions under which a detector guarantees to report a collision. The latter describes the conditions under which a detector guarantees *not* to report a collision when none actually occurred.

### 6.1 Completeness properties

Here we provide informal definitions for the completeness properties considered in this study. See Properties 4, 5, 6, and 7 in Appendix A for the formal definitions.

We say that a collision detector satisfies **completeness** if it guarantees to report a collision at any process that lost even one message. (Property 4 in Appendix A).

As we discuss in the introduction, in many practical scenarios, the MAC layer can reliably detect collisions only if a certain fraction of the messages being broadcast in a round is lost. It is reasonable, therefore, to consider weaker completeness properties, such as the following:

A collision detector satisfies **majority completeness** if it guarantees to report a collision at any process that loses half or more of the messages sent during the round. Equivalently: it returns a notification if the process did not receive a majority of the messages sent. (Property 5 in Appendix A).

A closely related property: a collision detector satisfies **half completeness** if it guarantees to report a collision at any process loses more than half of the messages sent during the round. Equivalently: it returns a notification if the process did not receive at least half of the messages sent. (Property 6 in Appendix A).

Notice the close similarity between half and majority completeness. The two properties differ only by, at most, a single message. That is, the half completeness property allows a process to lose at most one more message than the majority completeness property before guaranteeing to report a collision. As we show in Sect. 9, this small difference generates a significant complexity gap in the rounds required to solve consensus.

Finally, a collision detector satisfies **zero completeness** if it guarantees to report a collision at any process that loses *all* of the messages broadcast during that round. (Property 7 in Appendix A). This final definition is appealing because of its practicality. It requires only the ability to distinguish silence from noise (a problem solved by the carrier sensing capabilities integrated into many existing wireless MAC layers).

### 6.2 Accuracy properties

Here we provide informal definitions for the accuracy properties considered in this study. See Properties 8 and 9 in Appendix A for the formal definitions.

A collision detector satisfies **accuracy** if it guarantees to report a collision to a process *only* if that process failed to receive a message. (Property 8 in Appendix A). In order to account for the situation in which arbitrary noise can be

| | Complete | maj-Complete | half-Complete | 0-Complete |
|---|---|---|---|---|
| **Accurate** | $\mathcal{AC}$ | maj-$\mathcal{AC}$ | half-$\mathcal{AC}$ | 0-$\mathcal{AC}$ |
| **Eventually Accurate** | $\Diamond\mathcal{AC}$ | maj-$\Diamond\mathcal{AC}$ | half-$\Diamond\mathcal{AC}$ | 0-$\Diamond\mathcal{AC}$ |

**Fig. 2** A summary of collision detector classes

mistaken for collisions (for example, colliding packets from a neighboring region of a multi-hop network) we will also consider collision detectors satisfying a weaker accuracy property. Specifically, we say that a collision detector satisfies **eventual accuracy** if in every execution there exists a round $r_{\text{acc}}$ such that for all $r \geq r_{\text{acc}}$ the detector becomes accurate. (Property 9 in Appendix A). This might capture, for example, the time after which higher-level coordination protocols ensure that neighboring regions of the network become silent. Because this round differs in different executions, algorithms cannot be sure of when this period of accuracy begins, so they must be resilient to false detections.

Notice that we do not consider eventual completeness properties. It is easy to show that consensus is impossible if a collision detector might satisfy no completeness properties for an *a priori* unknown number of rounds. In this setting, the processes can be split into two partitions. In one partition all processes start with initial value 0. In the other partition, processes start with initial value 1. If all the messages are prevented from travelling between the partitions, and the completeness properties of the detectors are suspended long enough for the processes to decide, these decisions occur without the partitions having learned of each other—violating agreement.

It remains an interesting open question, however, to consider what might be possible with detectors that guarantee a weak completeness property at all times and satisfy a stronger completeness property eventually. For example, using such a detector, can one design an algorithm that terminates quickly in the case where the strong property holds from the first round?

### 6.3 Collision detector classes

In this paper, we focus, for the most part, on collision detectors that satisfy various combinations of the completeness and accuracy guarantees described above. To aid this discussion we define several *collision detector classes*, where a collision detector class is simply the set of *all* collision detectors that satisfy a specific collection of properties. The main classes we consider are described in Fig. 2. You will notice that we provide notation for eight different classes, each representing a different combination of the two accuracy and four completeness properties presented in this section. For example, the half-$\Diamond\mathcal{AC}$ class is the set of all collision detectors, defined over all index sets $P$, that satisfy both half completeness and eventual accuracy.

When we construct upper bounds, we assume they will work with any detector from a given class. When we derive lower bounds for a given class, we, as the lower bound designer, are free to choose which detector from the class to consider.

Before continuing, we introduce two special collision detection classes for which notation is not included in Fig. 2. The first is the **NoACC** class, which we define to include all collision detectors that satisfy completeness, but not necessarily accuracy.

To aid the definition of our second special class, we first define the $P$-collision detector $NOCD_P$, where $P$ is a non-empty subset of $I$, to be the trivial detector that assigns $\pm$ to all process indices in all rounds for all $P$-transmission traces. Using this definition, we define the **NoCD** class to be the set consisting of $NOCD_P$ for all non-empty subsets $P \subseteq I$. We establish the following straight-forward relation which will aid our lower bound construction:

**Lemma 1** *The collision detector class NoCD is a subset of the class NoACC (NoCD ⊆ NoACC).*

*Proof* Follows directly from the definitions. □

### 6.4 Maximal collision detectors

It will prove useful, in the construction of lower bounds, to define collision detectors that capture all possible behaviors for a given class. Specifically, we use the notation $MAXCD_P(C)$ to describe the $P$-collision detector that returns, for a given $P$-transmission trace, every $P$-CD trace that results from a $P$-collision detector in $C$. See Definition 23 in Appendix A for the formal definition.

### 6.5 The noise lemma

Before continuing, we note the following lemma (and associated corollary), that captures an important guarantee about the behavior shared by all collision detector classes considered in this study:

**Lemma 2** *For any execution $\alpha$ of system $(E, \mathcal{A})$, where $E.CD$ satisfies zero completeness, the following guarantee is satisfied: For all $r > 0$ and $i \in E.P$, if $t_T(\alpha)(r) = (c, T)$ and $c > 0$, then either $T(i) > 0$ or $t_{CD}(\alpha)(r)(i) = \pm$. That is, if one or more processes broadcast in round $r$, then all processes either receive something or detect a collision.*

*Proof* The zero completeness properties guarantees a collision notification in the case where one or messages are broadcast but none are received. □

Notice that, by definition, completeness, majority completeness, and half completeness all imply zero completeness.

Accordingly, Lemma 2 holds for systems containing a collision detector that satisfies *any* of our completeness properties.

**Corollary 1** *For any execution $\alpha$ of system $(E, \mathcal{A})$, where $E.CD$ satisfies zero completeness, the following guarantee is satisfied: For all $r > 0$ and $i \in E.P$, if $t_T(\alpha)(r) = (c, T)$ and $T(i) = 0$ and $t_{CD}(\alpha)(r)(i) = null$, then $c = 0$. That is, if any process receives nothing and detects no collision, then no message is broadcast.*

*Proof* Follows directly from Lemma 2. □

## 7 The consensus problem and related definitions

In the consensus problem, each process receives as input, at the beginning of the execution, a value from a fixed set $V$, and eventually decides a value from $V$ or fails.[3] We say the consensus problem is *solved* in this execution if and only if the following three properties are satisfied:

1. **Agreement:** no two processes decide different values.
2. **Strong validity:** if a process decides value $v$, then $v$ is the initial value of some process. A variant to this property is **Uniform validity**, which requires that if all processes share the same initial value $v$, then $v$ is the only possible decision value. To obtain the strongest possible results, we consider uniform validity (the weaker of the two) when proving our lower bounds, and strong validity when proving our matching upper bounds.
3. **Termination:** all correct processes eventually decide.

These properties should hold regardless of the number of process failures. To reason about the guarantees of a given consensus algorithm we need a notation for describing exactly the conditions under which the algorithm solves consensus. To accomplish this, we first offer the following two definitions that describe large classes of environments that share similar properties:

**Definition 11** ($\mathcal{E}(D, M)$) For any set of collision detectors $D$, and set of contention managers $M$, $\mathcal{E}(D, M) = \{E | E$ is an environment such that $E.CD \in D$ and $E.CM \in M\}$.

**Definition 12** ($\mathcal{E}^n(D, M)$) For any set of collision detectors $D$, set of contention managers, $M$, and positive integer $n$, $\mathcal{E}^n(D, M) = \{E | E \in \mathcal{E}(D, M)$ and $|E.P| = n\}$.

[3] To capture the notion of an "input value" in our model, assume a process has one initial state for each possible initial value. Therefore, the collection of initial states at the beginning of an execution (that is, the vector $C_0$) describes the initial value assignments for that execution. To capture the notion of "deciding" in our model, assume each process has one (or potentially many) special decide states for each initial value. By entering a decide state for $v$, the process decides $v$.

To obtain the strongest possible results, we use the first definition when proving upper bounds and the second (chosen for any arbitrary $n > 1$), when proving lower bounds.

We now offer two different notations that capture useful classes of algorithms. The first class describes algorithms that solve consensus in executions that satisfy eventual collision freedom, while the second, more general class, describes algorithms that solve consensus regardless of the collision behavior.

**Definition 13** (**($\mathcal{E}$,$V$,ECF)-consensus algorithm**) For any set of environments $\mathcal{E}$, and value set $V$, we say algorithm $\mathcal{A}$ is an ($\mathcal{E}$,$V$,ECF)-consensus algorithm if and only if for all executions $\alpha$ of system $(E, \mathcal{A})$, where $E \in \mathcal{E}$, initial values are assigned from $V$, and $\alpha$ satisfies eventual collision freedom, consensus is solved in $\alpha$.

**Definition 14** (**($\mathcal{E}$,$V$,NOCF)-consensus algorithm**) For any set of environments $\mathcal{E}$, and value set $V$, we say algorithm $\mathcal{A}$ is an ($\mathcal{E}$,$V$,NOCF)-consensus algorithm if and only if for all executions $\alpha$ of system $(E, \mathcal{A})$, where $E \in \mathcal{E}$, and initial values are assigned from $V$, consensus is solved in $\alpha$.

Finally, before addressing specific algorithms, we present the following general definition, and associated lemma, which will facilitate the discussion to follow:

**Definition 15** (**Communication stabilization time (CST)**) Let $\alpha$ be an execution of system $(E, \mathcal{A})$, where $\alpha$ satisfies eventual collision freedom, $E.CM$ is a wake-up service, and $E.CD$ satisfies eventual accuracy. The *Communication stabilization time* of $\alpha$ (also denoted $CST(\alpha)$) is equal to:

$$max\{r_{cf}, r_{acc}, r_{wake}\}$$

where $r_{cf}$, $r_{acc}$, and $r_{wake}$ are the rounds posited by the eventual collision freedom, eventual accuracy, and wake-up service properties, respectively.

**Lemma 3** *Let $\alpha$ be an execution of system $(E, \mathcal{A})$, where $\alpha$ satisfies eventual collision freedom, $E.CM$ is a wake-up service, and $E.CD$ satisfies eventual accuracy. For any round $r \geq CST(\alpha)$ where no process is advised to be $passive$ by the contention manager broadcasts, the following conditions are true*:

1. *Each process receives every message broadcast in round $r$.*
2. *No process detects a collision in round $r$.*

*Proof* Because the $CST(\alpha)$ occurs at or after $r_{wake}$, only a single process will be advised to be $active$ by the contention manager in round $r$. By assumption, therefore, if any process broadcasts during $r$, it will be this single process. Because the

execution satisfies eventual collision freedom and $CST(\alpha) \geq r_{cf}$, if this process broadcasts, then every process receives its message. And, finally, because $CST(\alpha) \geq r_{acc}$, we are guaranteed no spurious collision notifications in round $r$. The two conclusions follow directly. □

## 8 Consensus algorithms

In this section, we describe the algorithms that will match the bounds established in Sect. 9. We sketch the main ideas of the correctness proofs—providing the necessary information to understand the key intuition behind the algorithms. A more rigorous treatment of these arguments can be found in [31].

*Pseudocode conventions* To simplify the presentation of the algorithms we introduce the following pseudocode conventions: for a given round and process $p_i$, $\mathsf{bcast}(m)_i$ specifies the message $m$, broadcast by $p_i$ during the current round and $\mathsf{recv}()_i$ describes the multiset of messages (potentially empty) that $p_i$ receives during the current round. As defined in 3, we use the notation $SET(\mathsf{recv}()_i)$ to indicate the set containing every unique value in the multiset $recv()_i$. We use $\mathsf{CD}()_i$ and $\mathsf{CM}()_i$ to refer to the advice returned to $p_i$, during the current round, by its collision detector and contention manager, respectively. In Algorithm 2, we use the convention $V^{0,1}$ to indicate a binary representation of value set $V$. That is, $V^{0,1}$ replaces each value in $V$ with a unique binary string. We assume that these sequences are each of length $\lceil \lg |V| \rceil$ (which is, of course, enough to encode $|V|$ unique values). Similarly, we use bracket-notation to access a specific bit in one of these strings. For example, if $estimate_i \in V^{0,1}$, then $estimate_i[b]$, for $1 \leq b \leq \lceil \lg |V| \rceil$, indicates the $b$th bit in the binary sequence $estimate_i$. And, finally, we use $decide(v)_i$ to indicate that process $p_i$ decides value $v$.

*Halting* Notice, our model does not offer an explicit treatment of *halting*, which is relevant in certain practical applications. One could imagine, for example, in such settings, the use of an extra round in which processes broadcast to indicate they have not yet decided. A decided process could then safely halt after hearing this round to be silent. As it turns out, for all but the first algorithm in this section, it is safe to simply halt immediately upon deciding. (That is, there is never a need for a decided process to aid undecided processes).

*Roadmap* We start in Sect. 8.1 by describing an anonymous algorithm that solves consensus in executions satisfying eventual collision freedom using a wake-up service and any collision detector from maj-$\diamond\mathcal{AC}$. As, by definition, $\mathcal{AC}$, $\diamond\mathcal{AC}$, and maj-$\mathcal{AC}$ are all subsets of the class maj-$\diamond\mathcal{AC}$, this algorithm solves consensus for these detectors as well. The algorithm guarantees termination in a constant number of rounds after the communication stabilization time.

We then proceed in Sect. 8.2 to describe an anonymous algorithm that solves consensus in executions satisfying eventual collision freedom using a wake-up service and any collision detector from 0-$\diamond\mathcal{AC}$. All other collision detector classes we consider (with the exception of NoCD and NoACC) are subsets of 0-$\diamond\mathcal{AC}$, making this a general solution to the problem in all practical contexts. The algorithm guarantees termination in $\Theta(\lg(|V|)$ rounds after the communication stabilization time. In Sect. 8.3, we describe a non-anonymous variant of this algorithm that guarantees termination in $\min\{\lg|V|, \lg|I|\}$ rounds after the communication stabilization time.

Finally, in Sect. 8.4 we describe an anonymous algorithm that solves consensus, even in executions that do not satisfy eventual collision freedom, using any collision detector from 0-$\mathcal{AC}$. The algorithm terminates in $O(\lg(|V|)$ rounds after failures cease.

### 8.1 Anonymous consensus with ECF and collision detectors in maj-$\diamond\mathcal{AC}$

The pseudo-code in Algorithm 1 describes an anonymous ($\mathcal{E}$(maj-$\diamond\mathcal{AC}$,WS),$V$,ECF)-consensus algorithm. That is, it is guaranteed to solve consensus in any execution satisfying eventual collision freedom in an environment with a wake-up service and collision detector from maj-$\diamond\mathcal{AC}$. This implementation tolerates any number of process failures and terminates by round $CST + 2$.

The algorithm consists of two alternating phases: a *proposal* phase and a *veto* phase. In the proposal phase, every process that is advised to be *active* by its contention manager broadcasts its current estimate. If a process hears no collisions and receives at least one value, then it updates its estimate to the minimum value received. If a process detects a collision, or receives no messages, then it does not update its estimate.

During the next round, which is a *veto*-phase round, a process broadcasts a "veto" message if it heard a collision notification in the *proposal* phase or received two or more different values in the *proposal* phase. We are, therefore, using a negative acknowledgment scheme in which processes use the veto phase to notify other processes about bad behavior observed in the preceding phase. A process can decide its estimate if it makes it through a veto-phase round without receiving a veto message[4] or detecting a collision.

The basic idea is that a "silent" veto round indicates that no process has any reason to complain about the

[4] Remember, by the definition of our model, processes always receive their own broadcasts, so if a process broadcasts a veto it will definitely not decide this round.

preceding proposal round. If no process has any reason to complain about a proposal round, this means that each process received a single value and no collision notifications. If a process received no collision notification, then it received a majority of the messages (by the definition of majority completeness). Therefore, because majority sets intersect, we conclude that all processes must have received the *same* value. Therefore, any process making it through a "silent" veto round can safely decide—even if false collision notifications delay other processes from deciding that round—because it can be assured that no value, other than its decision value, is currently alive in the network. We formalize this argument as follows.

The proofs of validity, agreement, and termination rely on the following two lemmas:

**Lemma 4** *For $r \geq 0$, let $E_r = \{v \mid v$ equals the estimate value of some non-crashed process after $r$ rounds.$\}$. For any $s$ and $r$, where $0 \leq r \leq s$, $E_s \subseteq E_r$.*

*Proof* (*Sketch*) Let $r + 1$ be a *proposal* round (*estimate* values cannot be modified in a *veto* round). Processes that transmit in $r+1$ will transmit their *estimate* value. Processes that receive messages will only update their *estimate* to the smallest received value. Therefore, each process in $r + 1$ either keeps its *estimate* or updates its *estimate* to that of another process. In both cases: $E_{r+1} \subseteq E_r$ □

**Lemma 5** *If, for every process $p_i$ that is not crashed after proposal-round $r$, $|messages_i| = 1$ and $CD\text{-}advice_i = null$, then $|E_r| = 1$.*

*Proof* (*Sketch*) By the lemma assumptions, each process receives exactly one value and no collision notification during round $r$. Assume two processes $i$ and $j$, each received a different value: $i$ received $v$ and $j$ received $v'$. By majority completeness, we know $i$ and $j$ each received a majority of the messages sent in $r$. But majority sets intersect, making it impossible for $i$ to receive only $v$ and $j$ to receive only $v'$. This implies that every process updates its estimate to the same value. A contradiction. □

**Lemma 6 (Validity)** *If some process decides value $v$, then $v$ is the initial value of some process.*

*Proof* (*Sketch*) From Lemma 4, we know $E_{r-1} \subseteq E_0$, where $E_0$ is the set of initial values. Processes decide their *estimate* value. □

**Lemma 7 (Agreement)** *No two processes decide different values.*

*Proof* (*Sketch*) Let $r$ be the first round in which a process decides. Let $p_i$ be a process that decides $v$ in round $r$. By the definition of the algorithm (line 18) we know $p_i$ received (only) one value in round $r - 1$, so, by the noise lemma (Lemma 2), all processes received either a value or collision notification in round $r-1$. In round $r$, however, $p_i$ received no messages or collision notification. By Corollary 1, it follows that no process broadcast a veto in round $r$. This implies that all processes received no collision notification and only one value in round $r - 1$.

We apply Lemma 5 which provides that $|E_{r-1}| = 1$, and Lemma 4, which provides that for $r' \geq r$, $E_{r'} \subseteq E_{r-1} = \{v\}$. This makes $v$ the only possible decision value in any future round. □

**Lemma 8 (Termination)** *All correct processes decide by round $CST + 2$.*

*Proof* Let $r$ equal the first *proposal*-phase round such that $r \geq CST$. Because Algorithm 1 has only *active* processes (that is, processes that were returned *active* from the contention manager) broadcast during the *proposal* phase we can apply Lemma 3 to $r$, which provides that: (1) every process receives every message broadcast in $r$; (2) no process receives a collision notification in $r$. Because $r \geq r_{\text{wake}}$ a single correct process broadcasts. Every process receives this value (call it $v_r$) and no collision notification. Therefore, every process adopts $v_r$ as its estimate. No process will subsequently veto in round $r+1$. And because $r+1 > r_{\text{acc}}$ there will be no false positive collision notifications. This allows all processes to decide $v_r$ in $r + 1$. □

**Theorem 1** *For any non-empty value set $V$, Algorithm 1 is an anonymous ($\mathcal{E}$(maj-$\Diamond\mathcal{AC}$,WS)) that terminates by round $CST + 2$.*

*Proof* Correctness follows from Lemmas 6, 7 and 8. □

### 8.2 Anonymous consensus with ECF and collision detectors in 0-$\Diamond\mathcal{AC}$

The pseudo-code in Algorithm 2 describes an anonymous ($\mathcal{E}$(0-$\Diamond\mathcal{AC}$,WS),$V$,ECF)-consensus algorithm. That is, it is guaranteed to solve consensus in any execution satisfying eventual collision freedom in an environment with a wake-up service and collision detector from 0-$\Diamond\mathcal{AC}$. This implementation tolerates any number of process failures and terminates by round $CST + 2(\lceil \lg |V| \rceil + 1)$.

Algorithm 2 consists of three alternating phases. In the first phase, called *prepare*, every process advised to be *active* by its contention manager broadcasts its current estimate. Every process that receives at least one estimate and no collision notifications will adopt the minimum estimate it receives. In the second phase, called *propose*, the processes attempt to check that they all have the same estimate. There is one round dedicated to each bit in the estimate. If a process has an estimate with a one in the bit associated with that

round, then it broadcasts a message. If a process has an estimate with a zero in the bit associated with that round, it listens for broadcasts, and decides to reject (by setting $decide \leftarrow false$) if it hears any broadcasts or collisions. In the third phase, called *accept*, any processes that decided to reject in the previous phase will broadcast a veto. Any process that receives a veto message or collision notification realizes that there is a lack of consistency, and will cycle back to the first phase.

The basic idea is that if two processes have different estimates, there will be at least one round during the *propose* phase where one process is broadcasting and one is listening. The listening process will receive either a message or a collision notification, so it will successfully discover the lack of agreement so far. It can now veto in the *accept* phase to prevent any process from deciding a value.

The proofs of validity, agreement, and termination rely on the following two lemmas:

**Lemma 9** *For $r > 0$, let $E_r = \{v \mid v$ equals the estimate value of some non-crashed process after $r$ rounds. $\}$ For any $s$ and $r$, where $0 \leq r \leq s$, $E_s \subseteq E_r$.*

*Proof* (*Sketch*) During *prepare* phases, processes only broadcast their *estimate* values. Processes only modify their *estimate* to equal a *prepare* message. □

**Lemma 10** *If all non-crashed processes begin accept-phase round $r$ with $decide = true$, then all non-crashed processes begin $r$ with the same estimate value.*

*Proof* (*Sketch*) Preceding round $r$, each process executed one *propose*-phase round for each bit of their *estimate* value. Each process broadcasts only during rounds corresponding to bits that equaled 1. If a process receives a message or collision notification during a round where it does not broadcast, then that process sets $decide \leftarrow false$.

Because all processes begin $r$ with $decide = true$, we know that no process receives a message or collision notification during a *propose*-phase round in which it did not transmit. It follows from Corollary 1 that all non-crashed processes transmitted on the same schedule. Therefore, all non-crashed processes must have begun $r$ with the same *estimate*. □

**Lemma 11 (Validity)** *If some process decides value $v$, then $v$ is the initial value of some process.*

*Proof* (*Sketch*) From Lemma 9, we know $E_{r-1} \subseteq E_0$, where $E_0$ is the set of initial values. Processes decide their *estimate* value. □

**Lemma 12 (Agreement)** *No two processes decide different values.*

*Proof (Sketch)* Let $r$ be the first round in which a process decides. Let $p_i$ be a process that decides in $r$. Assume it decides $v_r$. This process heard no vetos in round *accept*-phase round $r$. By Corollary 1, we conclude that no process broadcasts a veto during $r$ It follows that all non-crashed started $r$ with $decide = true$. By Lemma 10 these processes share the same *estimate* of $v_r$. Finally, Lemma 9 provides that $v_r$ is the only possible decision value. □

**Lemma 13 (Termination)** *All correct processes decide by round $CST + 2(\lceil \log |V| \rceil + 1)$.*

*Proof (Sketch)* Let $r$ be the first *prepare*-phase round such that $r \geq CST$. Because Algorithm 2 has only *active* processes broadcast during the *prepare* phase (line 7), we can apply Lemma 3 to round $r$, which provides that for this round: (1) every process receives every message broadcast; (2) no process receives a collision notification. Because $CST \geq r_{\text{wake}}$, we know that a single correct process will broadcast in $r$. All non-crashed processes receive this value and no collision notification. During the *propose* phase, therefore, all non-crashed processes will transmit on the same schedule. because $CST \geq r_{\text{acc}}$ there will be no false collision notifications during the silent rounds, and all non-crashed processes will therefore begin the *accept* phase with $decide = true$. Accordingly, no process vetoes. Again, because $CST \geq r_{\text{acc}}$, no collision notification is received, and all non-crashed processes decide. □

**Theorem 2** *For any non-empty value set $V$, Algorithm 2 is an anonymous $(\mathcal{E}(0\text{-}\Diamond\mathcal{AC}, WS), V, ECF)$-consensus algorithm that terminates by round $CST + 2(\lceil \lg |V| \rceil + 1)$.*

*Proof* Correctness follows from Lemmas 11, 12 and 13. □

### 8.3 Non-anonymous consensus with ECF and collision detectors in 0-$\Diamond\mathcal{AC}$

In this section, we briefly describe a non-anonymous $(\mathcal{E}(0\text{-}\Diamond\mathcal{AC}, \text{WS}), V, \text{ECF})$-consensus algorithm, based on Algorithm 2, that can solve consensus faster than Algorithm 2 in the special case where the space of possible IDs ($I$) is small relative to the space of decision values ($V$). This algorithm (almost) matches[5] our non-anonymous lower bound for this setting (Corollary 3 in 9).

We do not provide formal pseudo-code or a rigorous correctness proof as we maintain that Algorithm 2 is the best

[5] The lower bound presented in Corollary 3 requires $\Omega(\min\{\lg |V|, \lg \frac{|I|}{n}\})$ rounds, whereas our upper bound presented here works in $\Theta(\min\{\lg |V|, \lg |I|\})$ rounds. Therefore, in one case, there is a gap of $1/n$, within the lg term, between the two. The $n$ factor appears to be an artifact of some looseness in the counting argument used in the lower bound calculation. Indeed, in Conjecture 1, we claim that $\Omega(\min\{\lg |V|, \lg |I|\})$ is, in fact, the real lower bound.

option for an ($\mathcal{E}$(0-$\Diamond\mathcal{AC}$,WS),$V$,ECF)-consensus algorithm. The version described here outperforms Algorithm 2 only in the unlikely case of an ID space being smaller then the consensus value space, and we present it only for completeness. It works as follows:

- If $|V| \leq |I|$, then every process runs Algorithm 2 without modification.
- If $|V| > |I|$, then every process divides up the rounds into repeated groups of three consecutive phases, which we will call phase 1, phase 2, and phase 3. During the phase 1 rounds, each process runs an instance of Algorithm 2 on the set of possible IDs, using its own ID as its initial value. The decision value of this instance of Algorithm 2 describes a leader. Once a process has been identified as a leader, it begins to broadcast its real initial value (from $V$) during phase 2 rounds. Every process that has not yet heard the leader's value by phase 2 round $r$, will broadcast "veto" in phase 3 round $r+1$. The leader keeps broadcasting its value in phase 2 until it hears a silent phase 3 round. Non-leaders decide the value in the first phase 2 message that they receive. They then halt. The leader decides its own value and halts after it hears a silent phase 3 round following a phase 2 broadcast.

In the first case ($|V| \leq |I|$), this algorithm finishes by $CST + \Theta(\lg|V|)$. In the second case ($|V| > |I|$), the leader election finishes by $CST + \Theta(\log|I|)$. The first successful broadcast and subsequent silent veto round will happen within two rounds after whichever comes later: leader election or $CST$. This provides a worst case termination of $CST + \Theta(\log|I|)$. Combined, we get a termination guarantee of $CST + \Theta(\min\{\lg|V|, \lg|I|\})$ rounds.

This algorithm, as described so far, is not fault-tolerant. Specifically, a leader can fail after being elected but before it broadcasts its value. Fortunately, there is an easy criterion for detecting the failure of a leader: a silent phase 2 round after a phase 1 decision has been reached. Any process that notices these conditions knows definitively that the leader has failed. This can trigger a new leader election among the remaining processes.

There are, however, difficulties in coordinating the start of this new leader election, as false collision notifications can prevent all processes from learning of the leader's death during the same round. To circumvent this problem, processes could run consecutive instances of consensus. During the first instance they try to elect a leader as specified. They then move directly into the second instance, setting their *estimate* value back to their unique ID. The trick is that during this new instance, processes do not broadcast in the *prepare* phase unless they detect the current leader to be failed. By receiving a *prepare* message in this phase, therefore, a process learns of the leader's failure in the previous phase. This ensures that the second run of consensus cannot terminate until all non-crashed processes have detected the current leader's failure. (For any process to terminate in the second phase, all processes would have had to have received a *prepare*-phase message).

If the second leader crashes, the same rules will ensure all processes participate in the third instance of consensus, etc. After each leader failure, all non-crashed processes will eventually learn of the failure and participate fully in the current instance of consensus, electing a new leader. Eventually, a correct process will be elected and successfully broadcast its value. Notice, that the time required for the processes to converge on a correct leader is captured in $CST$, so the running time in unaffected.

### 8.4 Anonymous consensus with NOCF and collision detectors in 0-$\mathcal{AC}$

It is a natural question to ask whether some collision detector classes can be powerful enough to solve consensus even if message loss is unrestricted. Surprisingly, the answer to this question is yes. Algorithm 3 can be used to solve the problem in $O(\log|V|)$ rounds with a collision detector in 0-$\mathcal{AC}$. This algorithm circumvents the problem of never-ending collisions by performing a search through a balanced binary search tree representation of the possible initial value space. Specifically, each iteration of the search is represented by four consecutive phases. In the first phase, called *vote-val*, processes can vote for the value represented by the current node in the tree by broadcasting a message. A process will vote in this phase if and only if this value is its initial value. In the second phase, called *vote-left*, processes can vote to descend to the left child of the current node by broadcasting. A process will vote in this phase if and only if its initial value is in the sub-tree rooted at this child. In the third phase, called *vote-right*, processes behave symmetrically to *vote-left*. In the fourth phase, called *recurse*, processes decide what action to take depending on the results of the voting from the previous three phases. If they registered a vote in the *vote-val* phase, they will decide the current value. If, instead, they registered a vote in only one of the *left* and *right* phases, they will descend to the appropriate child. If they register a vote for both, they will, by default, descend to the left child. And, finally, if no votes are registered (due to a process failure), they ascend to the parent of the current node.

The alert reader will notice that the *recurse* phase does not need its own round, as no message is broadcast and the receive set is ignored. For the sake of efficiency, this final phase could be appended to the end of the *vote-right* phase as an additional local computation. We leave it as its own

round only to simplify the presentation and description of the algorithm. By eliminating this round we could, however, reduce the factor of 8 to a factor of 6 in the termination bound.

Notice, also, that this algorithm does not use a contention manager. This is because it is designed for executions that do not necessarily satisfy eventual collision freedom. Without this property, identifying a single broadcaster is no longer important, as its messages are not guaranteed to ever be delivered (as they would be in an ECF execution).

Finally, note that the termination of Algorithm 3 is affected by failures. Imagine, for example, that a certain process, with a small initial value, leads all other processes deep into the left side of the search tree. Assume this process then crashes before it can vote for its value. Under certain initializations, all other processes might have initial values that are found in the right subtree of the root. This would then require all processes to traverse all the way back up the root, and then descend again into the right sub-tree before they can decide. In other words, this one failure added a $O(\log |V|)$ cost to our time complexity. To capture this reality, we introduce the term $f$, which, following convention, represents the number of failures in a given execution. Clearly, our termination bound will be, among other things, a function of $f$. Because the 0-$\mathcal{AC}$ collision detector class maintains accuracy at every round, we can extend Lemma 2 and Corollary 1 to the following, more powerful claim:

**Lemma 14** *For any round $r$ of an execution of Algorithm 3, one of the following two behaviors occurs*:

1. *Every process receives at least one message or a collision notification in $r$.*
2. *Every process receives no messages and no collision notification in $r$.*

*Proof* Lemma 2 provides that if any process broadcasts in $r$, then every process receives at least one message or a collision notification. By the definition of accuracy, if no process broadcasts in $r$, then no process will receive a collision notification (and, by the definition of an execution, no process will receive a message either). □

This allows us to make the following strong claim:

**Lemma 15** *For any round $r$, all non-crashed processes start $r$ with $curr$ pointing to the same node in the binary search tree.*

*Proof* (*Sketch*) Processes navigate the search tree based upon the sequence of noise and silence received in the preceding rounds. By Lemma 14, this pattern is the same for all processes. □

**Lemma 16 (Validity)** *If some process decides value $v$, then $v$ is the initial value of some process.*

*Proof* (*Sketch*) A process decides if and only if it hears noise during the *vote* phase. Because we assume accurate collision detectors, noise indicates that a process transmitted during this phase. It follows that the decided value is the voter's initial value. □

**Lemma 17 (Agreement)** *No two processes decide different values.*

*Proof* (*Sketch*) A process decides if it hears noise during a *vote* phase. By Lemma 14, if one process hears noise, then all processes hear noise. Therefore all non-crashed processes decide the value corresponding their current location in the search tree. Finally, by Lemma 15, we know all non-crashed processes are located at the same node in the tree. □

**Lemma 18 (Termination)** *All correct processes decide within $(8f + 4)\lg |V|$ rounds.*

*Proof* A failure, in the worst case, can require the processes to traverse $\lg |V|$ nodes down toward a leaf then another $\lg |V|$ back nodes up to the root: adding no more than $8 \lg |V|$ rounds. Therefore, $f$ failures add no more than $8f \lg |V|$ extra rounds. It may then take, in the worst case, $4 \lg |V|$more rounds to descend to be lead to the decision value by a correct process. The total running time is therefore bounded by $(8f + 4)\lg |V|$ □

**Theorem 3** *For any non-empty value set $V$, Algorithm 3 is an anonymous ($\mathcal{E}$(0-$\mathcal{AC}$,NoCM),$V$,ECF)-consensus algorithm that terminates in at most $(8f + 4)\lg |V|$ rounds.*

*Proof* Correctness follows from Lemmas 16, 17 and 18. □

## 9 Lower bounds

In this section, we show lower bounds that match (or, in the case of Theorem 7, come close to matching) the upper bounds of the previous section. We start, in Sect. 9.1, by examining systems with collision detectors from the NoCD class. That is, a system with effectively no collision detection. We show with Theorem 4 that consensus is impossible in this context; even if the system includes a leader election service and we consider only executions that satisfy eventual collision freedom. This highlights the necessity of collision detection, and underscores the following observation: *Eventual reliable communication* (*i.e., as provided by eventual collision freedom and a leader election service*) *is not useful without a means to determine when this period of reliability has begun*

(*i.e., a non-trivial collision detector*). It then follows directly from Lemma 1 (in Sect. 6)—which states that the collision detector class NoCD is a subset of the class NoACC—that consensus is also impossible in systems with collision detectors from the NoACC class. That is, in systems in which collision detector satisfies no accuracy properties. This is formalized with Theorem 5 in Sect. 9.2.

Next, in Sects. 9.3.1, 9.3.2, and 9.3.3, we examine systems with anonymous algorithms and collision detectors from the half-$\mathcal{AC}$ class. We show with Theorem 6 that, in this context, consensus cannot be solved in a constant number of rounds after the communication stabilization time; even if the system includes a leader election service and we consider only executions that satisfy eventual collision freedom. Specifically, we prove the existence of an execution that does not terminate before round $CST + \Theta(\log |V|)$.

We continue, in Sect. 9.3.4, to consider this same question in the context of non-anonymous algorithms. We prove with Theorem 7 the existence of an execution that does not terminate before round $CST + \lg\left(\frac{|V||I|}{n|V|+|I|}\right)\frac{1}{2}$. With Corollary 3 we simplify this expression to obtain the cleaner asymptotic result: $CST + \Omega\left(\min\left\{\log |V|, \log \frac{|I|}{n}\right\}\right)$. We conclude this particular line of questioning by conjecturing, in Conjecture 1, that the real bound is $CST + \Omega(\min\{\log |V|, \log |I|\})$.

The anonymous bound is matched by Algorithm 2 from Sect. 8, and the non-anonymous bound is (almost) matched by the variant of Algorithm 2 described in Sect. 8.3. Note: because we demonstrated in Sect. 8 a constant-round solution that uses a detector from the maj-$\diamond\mathcal{AC}$ class, these results demonstrate a substantial complexity gap between the half-complete and majority-complete properties.

We next consider executions that do not necessarily satisfy eventual collision freedom. One might expect that under such conditions consensus cannot be solved. Indeed, with Theorem 8, in Sect. 9.4, we show that consensus cannot be solved with a collision detector that does not satisfy accuracy in all rounds. With an accurate detector, however, consensus *is* solvable. This was demonstrated by Algorithm 3 which solves consensus in $O(\lg |V|)$ rounds using a detector from 0-$\mathcal{AC}$ and no contention manager. We show, with Theorem 9, in Sect. 9.5, that this algorithm is optimal by proving that its logarithmic complexity is necessary for any solution to consensus in this context.

To obtain the strongest possible results, all bounds that follow assume the weaker *uniform validity* property for consensus, as defined in Sect. 7. We also assume the stronger leader election service property for the contention managers used in this section, whereas the matching upper bounds use the weaker wake-up service property. Finally, we assume, for every execution considered in this section, that there are **no process failures**. This highlights the perhaps surprising reality that message loss, coupled with the lack of knowledge of number of participants, not failures, are the source of time complexity in this setting.

### 9.1 Impossibility of consensus with no collision detection

We show that no algorithm can solve consensus in a system with a collision detector from the NoCD class. This holds even if we only consider executions that satisfy eventual collision freedom, and we assume the system contains a leader election service.

**Theorem 4** *For every value set $V$, where $|V| > 1$, there exists no ($\mathcal{E}$(NoCD,LS),$V$,ECF)-consensus algorithm.*

*Proof* Assume by contradiction that an ($\mathcal{E}$(NoCD, LS),$V$,ECF)-consensus algorithm, $\mathcal{A}$, exists. First, we fix two disjoint and non-empty subsets of $I$, $P_a$ and $P_b$. Next, we define three environments $A, B, C$ as follows: Let $A.P = P_a$, $B.P = P_b$, and $C.P = P_a \cup P_b$. Let $A.CD = NOCD_{P_a}$, $B.CD = NOCD_{P_b}$, and $C.CD = NOCD_{P_a \cup P_b}$. And let $A.CM = MAXLS_{P_a}$, $B.CM = MAXLS_{P_b}$, and $C.CM = MAXLS_{P_a \cup P_b}$. By definition, $A, B, C \in \mathcal{E}$(NoCD,LS).

Next, we construct an execution $\alpha$, of the system $(A, \mathcal{A})$, and an execution $\beta$, of the system $(B, \mathcal{A})$, as follows:

1. Fix the executions so there is no message loss in either $\alpha$ or $\beta$.
2. In $\alpha$, fix the contention manager, starting with round 1, to return $active$ only to the process described by $\min(P_a)$. In $\beta$, fix the contention manager to behave the same, with respect to $\min(P_b)$.
3. Fix the collision detector in both executions to return $\pm$ to all processes in all rounds (the only allowable behavior for the NoCD class).
4. In $\alpha$, have all process start with initial value $v$, and in $\beta$ have all processes start with initial value $v'$, where $v, v' \in V$ and $v \neq v'$.

It is clear that these executions satisfy the constraints of their environments, as, in both, the contention managers satisfy the leader election service property, and the collision detector returns $\pm$ to all processes in all rounds (the only allowable behavior from a $NOCD$ detector). Furthermore, we notice that both executions trivially satisfy eventual collision freedom (as there is no message loss). Therefore, by the definition of an ($\mathcal{E}$(NoCD,LS),$V$,ECF)-consensus algorithm, consensus is solved in both. Let $k$ be the smallest round after which all processes have decided in both $\alpha$ and $\beta$.

We next construct an execution $\gamma$, of the system $(C, \mathcal{A})$, as follows:

1. Fix the execution such that for the first $k$ rounds all processes described by indices in $P_a$ lose all (and only)

messages from processes described by indices in $P_b$, and vice versa. Starting with round $k+1$, there is no further message loss.
2. Fix the collision detector to return $\pm$ to all processes in all rounds, as it must.
3. Fix the contention manager, for the first $k$ rounds, to return $active$ only to the processes described by $\min(P_a)$ and $\min(P_b)$. Starting with round $k+1$, the contention manager returns $active$ only to the process described by $\min(P_a)$.
4. All process described by indices in $P_a$ start with initial value $v$, and all processes described by indices in $P_b$ start with initial value $v'$.

Again, it is clear that this execution satisfies the constraints of its environment. The contention manager satisfies the leader election service property by stabilizing to a single $active$ process (in round $k+1$) and the collision detector returns $\pm$ to all processes in all rounds, as required by its definition. Furthermore, we note that this execution satisfies eventual collision freedom as message loss ceases at round $k+1$. Once again, by the definition of an ($\mathcal{E}$(NoCD,LS),$V$,ECF)-consensus algorithm, consensus is solved in $\gamma$.

To reach a contradiction, we first note that, by construction, for all $i$ in $P_a$, the execution $\gamma$ is indistinguishable from $\alpha$, with respect to $i$, through round $k$. And for all $j$ in $P_b$, the execution $\gamma$ is indistinguishable from $\beta$, with respect to $j$, through round $k$. Therefore, by round $k$, all processes described by indices in $P_a$ will decide the same value in both $\alpha$ and $\gamma$, and all processes described by indices in $P_b$ will decide the same value in both $\beta$ and $\gamma$. By uniform validity, however, processes decide $v$ in $\alpha$ and $v'$ in $\beta$; thus both values will be decided in $\gamma$—violating agreement. A contradiction. □

## 9.2 Impossibility of consensus with no accuracy guarantees

**Theorem 5** *For every value set $V$, where $|V| > 1$, there exists no ($\mathcal{E}$(NoACC,LS),$V$,ECF)-consensus algorithm.*

*Proof* Lemma 1, from Sect. 6, establishes that NoCD $\subseteq$ NoACC. Therefore, if an algorithm $\mathcal{A}$ is an ($\mathcal{E}$(NoACC, LS),$V$,ECF)-consensus algorithm, then $\mathcal{A}$ is an ($\mathcal{E}$(NoCD,LS) ,$V$,ECF)-consensus algorithm. By Theorem 4, there exists no ($\mathcal{E}$(NoCD,LS),$V$,ECF)-consensus algorithm. Therefore, there exists no ($\mathcal{E}$(NoACC,LS),$V$,ECF)-consensus algorithm. □

## 9.3 Impossibility of constant round consensus with ECF and half-$\mathcal{AC}$

We next show that no algorithm can guarantee to always solve consensus in a constant number of rounds after the communication stabilization time if half of the messages sent in a round can be lost without detection. Specifically, we provide two main results. In Theorem 6, presented in Sect. 9.3.3, we show that for any anonymous ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm, there exists an execution that does not terminate before $CST + \Theta(\log|V|)$. In Corollary 3, presented in Sect. 9.3.4, we show that for any non-anonymous ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm, there exists an execution that does not terminate before $CST + \Omega(\min\{\log|V|, \log\frac{|I|}{n}\})$.

We start, however, with some general defintions and lemmas, presented in Sects. 9.3.1 and 9.3.2, which aid the discussion to follow.

### *9.3.1 Definitions*

**Definition 16 (Basic Broadcast Count Sequence)** The Basic Broadcast Count Sequence of an execution $\alpha$ is the infinite sequence of values drawn from $\{0, 1, 2+\}$ where, for all $r > 0$, the $r$th position in the sequence is:

- 0 if and only if no process broadcasts during round $r$ of $\alpha$,
- 1 if and only if exactly one process broadcasts during round $r$ of $\alpha$,
- 2+ if and only if two or more processes broadcast during round $r$ of $\alpha$.

We say two executions $\alpha$ and $\beta$, have the same broadcast count sequence through round $k$, for some $k > 0$, if and only if the basic broadcast count sequence of both executions are the same through the first $k$ values.

Next, we introduce two definitions that will help us identify a specific type of "well-behaved" execution:

**Definition 17 ($V$-start algorithm)** Let $V$ be a non-empty set of values. We say algorithm $\mathcal{A}$ is a *$V$-start algorithm* if and only if for all $i \in I$, $\mathcal{A}(i)$ has exactly $|V|$ initial states described by the set $\{\text{init}_i(v) | v \in V\}$.

Notice that any algorithm that solves consensus over a value set $V$ is a $V$-start algorithm. This holds because a consensus algorithm must have a unique initial state for each possible initial value. Throughout this section, whenever we discuss a $V$-start algorithm, $\mathcal{A}$, that solves consensus for value set $V$, we assume for all $i \in I$ and $v \in V$, that initial state $\text{init}_i(v)$ for $\mathcal{A}(i)$ is the initial state of this process that corresponds to initial value $v$.

We now define a specific execution type for $V$-start algorithms:

**Definition 18 ($\alpha_P(v)$ (Alpha execution))** Let $\mathcal{A}$ be a $V$-start algorithm, where $V$ is some non-empty set of values,

$v \in V$, and $P$ is a non-empty subset of $I$. Let $E_P$ be an environment with $E_P.P = P$, $E_P.CD = MAXCD_P(\mathcal{AC})$, and $E_P.CM = MAXLS_P$. Then $\alpha_P(v)$ describes the unique execution of system $(E_P, \mathcal{A})$ that results when we:

1. Fix $\mathcal{A}(i)$, for all $i \in P$, to start with initial state $\text{init}_i(v)$,
2. Fix $E_P.CM$ to designate only the process corresponding to $\min(P)$ as *active*,
3. Fix the execution such that in any given round, if a single process broadcasts, then all processes receive the message, if more than one process broadcasts, then (as required by the model) the receivers each receive their own message, but all other messages are lost,
4. Fix $E_P.CD$ to satisfy completeness and accuracy (as it must by the definition of $E_P$), and
5. Fix the execution such that there are no failures.

This execution satisfies the constraints of $E_P$ as the collision detector, by definition, satisfies completeness and accuracy, and the contention manager satisfies the leader election service property by stabilizing to a single *active* process starting in the first round.

A few points to notice: first, by definition, $E_P \in E$(half-$\mathcal{AC}$,LS) (a complete detector is a half-complete detector). We also note that this execution satisfies eventual collision freedom (assumption 3 makes this explicit). Thus, if $\mathcal{A}$ happens to be an ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm (as it will be when we use this definition later in the section), then any alpha execution defined over $\mathcal{A}$, solves consensus.

### 9.3.2 Key lemmas

We first introduce a lemma, and an associated corollary, that prove some important properties regarding the behavior of anonymous algorithms:

**Lemma 19** *Let $\mathcal{A}$ be an anonymous $V$-start algorithm, where $V$ is a non-empty set of values, let $P$ and $P'$ be two disjoint subsets of $I$ such that $|P| = |P'| > 0$, let $f$ be a bijection $f : P \to P'$ such that $f(\min(P)) = \min(P')$, and let $v$ be an element of $V$. For every $i \in P$, the sequence of states, message receive sets, contention manager advice, and collision detector advice, describing the execution of $\mathcal{A}(i)$ in $\alpha_P(v)$, is the same as the sequence describing the execution of $\mathcal{A}(f(i))$ in $\alpha_{P'}(v)$, where both alpha executions are defined over $\mathcal{A}$.*

*Proof* (*Sketch*) The only difference between the two systems is the set of participating process indices. Because, however, we assume $\mathcal{A}$ is anonymous, this difference cannot affect the executions, leading to the same behavior in both $\alpha_P(v)$ and $\alpha_{P'}(v)$. □

**Corollary 2 (Lemma 19)** *Let $\mathcal{A}$ be an anonymous $V$-start algorithm, where $V$ is a non-empty set of values. Let $P$ and $P'$ be two disjoint subsets of $I$ such that $|P| = |P'| > 0$. For all $v \in V$, and integer $r > 0$, $\alpha_P(v)$ and $\alpha_{P'}(v)$ have the same basic broadcast count sequence through the first $r$ rounds, where both $\alpha$ executions are defined over $\mathcal{A}$.*

*Proof* The decision to broadcast in a given round is a function of a process's state at the beginning of the round and the contention manager advice during the round. Let $f$ be the bijection defined by Lemma 19. Therefore, by Lemma 19, we know that for every $i \in P$, process $\mathcal{A}(i)$ broadcasts in round $r$ of $\alpha_P(v)$ if and only if process $\mathcal{A}(f(i))$ broadcasts in round $r$ of $\alpha_{P'}(v)$. Because $f$ is a bijection from $P$ to $P'$, the corollary follows directly. □

We next present two counting arguments that bound the number of basic broadcast sequences that can exist among pairs of executions over short execution prefixes. Lemma 20 considers anonymous algorithms, and Lemma 21 considers non-anonymous algorithms.

**Lemma 20** *Let $\mathcal{A}$ be an anonymous $V$-start algorithm, where $V$ is a set of values such that $|V| > 1$, and let $P$ be a non-empty subset of $I$. There exist two alpha executions, $\alpha_P(v)$ and $\alpha_P(v')$, defined over $\mathcal{A}$, where $v, v' \in V$, $v \neq v'$, and $\alpha_P(v)$ and $\alpha_P(v')$ have the same basic broadcast count sequence through the first $\frac{\lg|V|}{2} - 1$ rounds.*

*Proof* We have $|V|$ different alpha executions to consider; one for each value in $V$. For any sequence of $k$ rounds, there are at most $3^k$ possible broadcast count sequences (in each round, three behaviors can occur that are relevant to the basic broadcast count: no process broadcasts; one process broadcasts; or more than one process broadcasts). We claim that for $k = \frac{\lg|V|}{2} - 1$, the total number of sequences of length $k$ is less than $|V|$. Thus, by the pigeon-hole principle, at least two values in $V$ must produce the same sequence. We verify this claim by plugging in for $k$ and solving:

$$\begin{aligned} 3^{\left(\frac{\lg|V|}{2}-1\right)} &< 3^{\left(\frac{\lg|V|}{\lg 3}-\log_3 2\right)} \\ &= 3^{(\log_3|V|)}3^{(-\log_3 2)} \\ &= \frac{|V|}{2} \\ &< |V| \end{aligned}$$

**Lemma 21** *Let $\mathcal{A}$ be a $V$-start algorithm, where $V$ is a set of values such that $|V| > 1$, and let $n$ be an integer such that $1 < n \leq \lfloor \frac{|I|}{2} \rfloor$. There exist two alpha executions, $\alpha_P(v)$ and $\alpha_{P'}(v')$, defined over $\mathcal{A}$, where $P, P' \subseteq I$, $|P| = |P'| = n$, $P \cap P' = \emptyset$, $v, v' \in V$, $v \neq v'$, and $\alpha_P(v)$ and $\alpha_{P'}(v')$ have the same basic broadcast count sequence through the first $\lg\left(\frac{|V||I|}{n|V|+|I|}\right)\frac{1}{2}$ rounds.*

*Proof* Let $\Pi$ be a partition of $I$ into disjoint sets of size $n$. Let $S$ be the set of alpha executions defined over $\mathcal{A}$, all index sets in $\Pi$, and all values in $V$. It follows that we have $|V||\Pi|$ different alpha executions in $S$ to consider. Note that for any $P \in \Pi$ and $v \in V$, there are exactly $|V| + |\Pi| - 1$ alpha executions in $S$ of the form $\alpha_P(*)$ or $\alpha_*(v)$ (that is, defined over the same process index set $P$ or value $v$). Also note, as argued above, for any sequence of $k$ rounds, there are at most $3^k$ basic broadcast count sequences. We claim that for $k = \lg\left(\frac{|V||I|}{n|V|+|I|}\right)\frac{1}{2}$: $\frac{|V||\Pi|}{3^k} \geq |V|+|\Pi|$. If true, this implies, by the pigeon-hole principle, that there exist at least $|V|+|\Pi|$ alpha executions in $S$ that share the same basic broadcast count sequence. Because no more than $|V| + |\Pi| - 1$ executions can share the same process set or value, then at least two of these $|V| + |\Pi|$ sequence-sharing executions must be defined over different process index sets and values. These are the two executions posited by our Lemma statement.
We verify this claim by plugging in for $k$ and showing that the following equation holds:

$$\frac{|V||\Pi|}{3^k} \geq |V| + |\Pi|$$

First, however, we note that $|\Pi| = \lfloor\frac{|I|}{n}\rfloor$, (the floor captures the fact that $I$ might not divide evenly by $n$), allowing us to state:

$$\frac{|V||I|}{n3^k} \geq |V| + \lfloor\frac{|I|}{n}\rfloor$$

Next, we replace $k$ with the following larger expression: $k' = \lg\left(\frac{|V||I|}{n|V|+|I|}\right)\lg^{-1} 3$. This is valid because, clearly, if our above equation is true for $k' > k$ then it is also true for $k$. We now subsitute for $k'$ and simplify:

$$\begin{aligned}
\frac{|V||\Pi|}{n3^{k'}} &= \frac{|V||\Pi|}{n3^{\lg\left(\frac{|V||I|}{n|V|+|I|}\right)\lg^{-1} 3}} \\
&= \frac{|V||\Pi|}{n3^{\log_3\left(\frac{|V||I|}{n|V|+|I|}\right)}} \\
&= \frac{|V||\Pi|}{n\frac{|V||I|}{n|V|+|I|}} \\
&= \frac{|V||I|\,(n|V| + |I|)}{n|V||I|} \\
&= \frac{n|V| + |I|}{n} \\
&= |V| + \frac{|I|}{n} \\
&\geq |V| + \lfloor\frac{|I|}{n}\rfloor
\end{aligned}$$

Thus, our claim is verified. □

We conclude this subsection with a general indistinguishability lemma, involving alpha executions with similar basic broadcast count sequences. We call this the Pasting Lemma, as it involves the "pasting together" of two executions.

**Lemma 22 (Pasting Lemma)** *Let $\mathcal{A}$ be a $V$-start algorithm, where $V$ is a set of values such that $|V| > 1$. Suppose $v, v' \in V$, $k > 0$, and $R, R' \subseteq I$, such that $v \neq v'$, $|R| = |R'| > 1$, and $R \cap R' = \emptyset$. Suppose alpha executions $\alpha_R(v)$ and $\alpha_{R'}(v')$, defined over $\mathcal{A}$, have the same basic broadcast count sequence for the first $k$ rounds. Let $E_{R\cup R'}$ be an environment where $E_{R\cup R'}.P = R \cup R'$, $E_{R\cup R'}.CD = MAXCD_{R\cup R'}$(half-$\mathcal{AC}$), and $E_{R\cup R'}.CM = MAXLS_{R\cup R'}$. Then there exists an execution, $\gamma$ of system $(E_{R\cup R'}, \mathcal{A})$, that satisfies eventual collision freedom, such that $\gamma$ is indistinguishable from $\alpha_R(v)$ (resp. $\alpha_{R'}(v')$), through round $k$, with respect to all $i \in R$ (resp. $j \in R'$).*

*Proof* We start by constructing an execution $\gamma$ that satisfies our desired indistinguishabilities and eventual collision freedom. We then show that this execution satisfies the constraints of its environment. Specifically, let $\gamma$ be the unique execution of system $(E_{R\cup R'}, \mathcal{A})$ where:

1. For every $i \in R$, $\mathcal{A}(i)$ starts with state $init_i(v)$, and for all $j \in R'$, $\mathcal{A}(j)$ starts with state $init_j(v')$.
2. For the first $k$ rounds, we fix the execution to generate the following receive behavior: if, among processes in $R$, a single process broadcasts, then all processes in $R$ receive this message. Similarly, if, among processes in $R'$, a single process broadcasts, then all processes in $R'$ receive this message. Broadcasters always receive their own message (as required by the model). All other messages are lost. Notice, therefore, that for these $k$ rounds, a process in $R$ (resp. $R'$) never receives a message from a process in $R'$ (resp. $R$). For example, if two processes in $R$ broadcast, the two broadcasters receive their own message, but no other process in $R$ receives anything. On the other hand, if exactly one process in $R$ and one process in $R'$ broadcast, all processes in $R$ get the $R$ message and all processes in $R'$ get the $R'$ message. Starting with round $k + 1$, there is no further message loss.
3. For the first $k$ rounds, $E_{R\cup R'}.CD$ returns $\pm$ to $\mathcal{A}(i)$ for some $i \in R$ (resp. $\mathcal{A}(j)$ for some $j \in R'$) if and only if it returned $\pm$ to $\mathcal{A}(i)$ (resp. $\mathcal{A}(j)$) during this round of $\alpha_R(v)$ (resp. $\alpha_{R'}(v')$). Starting with round $k + 1$, the detector returns *null* to all processes.
4. For the first $k$ rounds, $E_{R\cup R'}.CM$ returns *active* to $\mathcal{A}(\min(R))$ and $\mathcal{A}(\min(R'))$. Starting with round $k + 1$, it returns *active* only to $\mathcal{A}(\min(R))$.

We constructed $\gamma$ such that for every $i \in R$, $\alpha_R(v)$ is indistinguishable from $\gamma$, with respect to $i$, through round $k$, and for every $j \in R'$, $\alpha_{R'}(v')$ is indistinguishable from $\gamma$, with respect to $j$, through round $k$. The collision detector and contention manager advice for these rounds, by definition, are the same with respect to the alpha executions. To see why the message receive behavior is the same, we turn to assumption

2 of our $\gamma$ definition. First, notice that no process $\mathcal{A}(i)$, such that $i \in R$, ever receives a message from process $\mathcal{A}(j)$, such that $j \in R'$, and vice versa. Second, process $\mathcal{A}(i)$, $i \in R$ (resp. $\mathcal{A}(j)$, $j \in R'$) only receives a message $m$ if a single process $\mathcal{A}(i), i \in R$ (resp. $\mathcal{A}(j)$, $j \in R'$) broadcasts (and it broadcast $m$), and/or the receiving process broadcast itself. This matches the definition of receive behavior in our alpha executions. Also notice that $\gamma$ satisfies eventual collision freedom as message loss stops at round $k+1$.

We must next show that $\gamma$ is valid. In other words, we must show that the contention manager and collision detector behavior we describe satisifes the constraints of the environment. It is easy to see that this is the case for the contention manager, as, by construction, it stabilizes to a single *active* process in round $k+1$, thus satisfying the leader election service property. The collision detector behavior is more complicated. Because we specified that $E_{R\cup R'}.CD = MAXCD_{R\cup R'}$(half-$\mathcal{AC}$) we must ensure that neither half-completeness nor accuracy is ever violated in $\gamma$. This is obvious starting with round $k+1$, so we focus only on the first $k$ rounds.

Two factors are key in this argument: first, the indistinguishability between $\gamma$ and the alpha executions for these first $k$ rounds, and second, the fact that the basic broadcast count sequence is the same for both of these alpha executions for these first $k$ rounds. Let us examine the possible cases from the point of view of an arbitrary process $\mathcal{A}(i)$, for a single round $r \leq k$, where we assume, without loss of generality, that $i \in R$.

- *Case 1 $\mathcal{A}(i)$ receives null from the collision detector.* If $\mathcal{A}(i)$ receives *null* in this round of $\gamma$, then, by assumption 3 of our $\gamma$ definition, $\mathcal{A}(i)$ receives *null* in this round of $\alpha_R(v)$ as well. By the definition of an alpha execution, this means either a single process or no process broadcast during this round of $\alpha_R(v)$. By our indistinguishability and basic broadcast count equality, this implies that either: (a) no process broadcast in this round of $\gamma$; or (b) exactly one process described by an index in $R$ and one process described by an index in $R'$ broadcast in this round of $\gamma$. Accuracy is trivially satisfied in both (a) and (b) (as the detector returned *null* in both). And half-completeness is satisfied in both, as in (a) no messages are lost, and in (b) $\mathcal{A}(i)$ lost exactly half of the messages—making it acceptable for it to return *null* by the definition of half-completeness. (This is where we first notice the separation between half-completeness and its close neighbor majority completeness. If we were dealing with a majority complete collision detector, returning *null* in case b would be unacceptable.)
- *Case 2 $\mathcal{A}(i)$ receives $\pm$ from the collision detector.* If $\mathcal{A}(i)$ receives $\pm$ in this round of $\gamma$, then, by assumption 3 of our $\gamma$ definition, $\mathcal{A}(i)$ receives $\pm$ in this round of $\alpha_R(v)$ as well. By the definition of an alpha execution this means two or more processes broadcast during this round of $\alpha_R(v)$. By our indistinguishability and basic broadcast count equality, two or more processes described by indices in $R$ and two or more processes described by indices in $R'$ broadcast during this round of $\gamma$. Therefore, by assumption 2 of our $\gamma$ definition, all processes lose at least one message in this round (as the only messages received in this case are broadcasters receiving their own message). Because there was message loss, and the detector returned $\pm$, half-completeness and accuracy are clearly satsified.

Thus, our claim is verified. □

### 9.3.3 *Impossibility of constant round consensus with an anonymous ($\mathcal{E}$(half-$\mathcal{AC}$,LS),V,ECF)-consensus algorithm*

**Theorem 6** *Let $V$ be a value set such that $|V| > 1$, and let $n$ be an integer such that $1 < n \leq \lfloor \frac{|I|}{2} \rfloor$. For any anonymous ($\mathcal{E}$(half-$\mathcal{AC}$,LS),V,ECF)-consensus algorithm, $\mathcal{A}$, there exists an environment $E \in \mathcal{E}^n$(half-$\mathcal{AC}$,LS), and an execution $\alpha$ of the system $(E, \mathcal{A})$, where $\alpha$ satisfies eventual collision freedom, $CST(\alpha) = 1$, and some process in $\alpha$ does not decide until after round $\frac{\lg |V|}{2} - 1$.*

*Proof* Let $\mathcal{A}$ be any anonymous ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm. Fix $P$ and $P'$ to be two disjoint subsets of $I$ such that $|P| = |P'| = n$. In this proof we will consider alpha executions defined over $\mathcal{A}$, $P$ or $P'$, and values from $V$. (Notice, by virtue of being a consensus algorithm, $\mathcal{A}$ is clearly also a $V$-start algorithm). These executions satisfy eventual collision freedom, have a communication stabilization time of 1, and are defined by an environment in $\mathcal{E}^n$(half-$\mathcal{AC}$,LS). Therefore, if we can find such an alpha execution that does not decide for a logarithmic number of rounds, our theorem will be proved.

First, we apply Lemma 20 to $\mathcal{A}$, $V$, and $P$, which provides two alpha executions, $\alpha_P(v)$ and $\alpha_P(v')$, that have the same basic broadcast count sequence through the first $\frac{\lg |V|}{2} - 1$ rounds. By Corollary 2, we know this, therefore, is also true of $\alpha_P(v)$ and $\alpha_{P'}(v')$ (by this corollary, $\alpha_{P'}(v')$ has the same basic broadcast count sequence as $\alpha_P(v')$). We can now apply the Pasting Lemma (Lemma 22) to $\alpha_P(v)$, $\alpha_{P'}(v')$, and $k = \frac{\lg |V|}{2} - 1$. This produces an execution $\gamma$ of system $(E_{P\cup P'}, \mathcal{A})$—where $E_{P\cup P'}.P = P \cup P'$, $E_{P\cup P'}.CD = MAXCD_{P\cup P'}$(half-$\mathcal{AC}$), and $E_{P\cup P'}.CM = MAXLS_{P\cup P'}$—that satisfies eventual collision freedom, such that $\gamma$ is indistinguishable from $\alpha_P(v)$ (resp. $\alpha_{P'}(v')$), through round $k$, with respect to processes described by indices in $P$ (resp. $P'$).

Let us assume, for the sake of contradiction, that both $\alpha_P(v)$ and $\alpha_{P'}(v')$ terminate by round $k = \frac{\lg|V|}{2} - 1$. By the definition of an ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm, $\gamma$ must solve consensus. By assumption, in both $\alpha_P(v)$ and $\alpha_{P'}(v')$, all processes decide by round $k$ in these executions. By our indistinguishability, these processes decide the same values in $\gamma$. By uniform validity, processes described by indices in $P$ decide $v$, and processes described by indices in $P'$ decide $v'$. Thus, both values are decided in $\gamma$—violating agreement. A contradiction. □

*Making the bound tight* We match this lower bound with Algorithm 2, described in Sect. 8, which is an anonymous ($\mathcal{E}$(0-$\Diamond\mathcal{AC}$,WS),$V$,ECF)-consensus algorithm that guarantees termination by $CST + \Theta(\lg|V|)$.

### 9.3.4 Impossibility of constant round consensus with a non-anonymous ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm

We now turn our attention to the case of non-anonymous algorithms. Here, we derive a more complicated bound, but then show, in Corollary 3, that for reasonable parameters, it is no worse, roughly speaking, than its anonymous counterpart.

**Theorem 7** *Let $V$ be a value set such that $|V| > 1$, and let $n$ be an integer such that $1 < n \leq \lfloor \frac{|I|}{2} \rfloor$. For any ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm, $\mathcal{A}$, there exists an environment $E \in \mathcal{E}^n$(half-$\mathcal{AC}$,LS), and an execution $\alpha$ of the system $(E, \mathcal{A})$, where $\alpha$ satisfies eventual collision freedom, $CST(\alpha) = 1$, and some process in $\alpha$ does not decide until after round* $\lg(\frac{|V||I|}{n|V|+|I|})\frac{1}{2}$.

*Proof* Let $\mathcal{A}$ be any ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm. For this proof we consider alpha executions defined over algorithm $\mathcal{A}$, value set $V$, and all subsets of size $n$ of $I$. These executions satisfy eventual collision freedom, have a communication stabilization time of 1, and are defined by an environment in $\mathcal{E}^n$(half-$\mathcal{AC}$,LS). Therefore, if we can find such an alpha execution that does not decide for the desired number of rounds, our theorem will be proved.

First, we apply Lemma 21, which provides two such executions, $\alpha_P(v)$ and $\alpha_{P'}(v')$, where $|P| = |P'| = n$, $P \cap P' = \emptyset$, and both have the same basic broadcast count sequence through the first $\lg(\frac{|V||I|}{n|V|+|I|})\frac{1}{2}$ rounds. We can now apply the Pasting Lemma (Lemma 22) to $\alpha_P(v)$, $\alpha_{P'}(v')$, and $k = \lg(\frac{|V||I|}{n|V|+|I|})\frac{1}{2}$, which, as before, provides an execution $\gamma$ of system $(E_{P\cup P'}, \mathcal{A})$—where $E_{P\cup P'}.P = P \cup P'$, $E_{P\cup P'}.CD = MAXCD_{P\cup P'}$(half-$\mathcal{AC}$), and $E_{P\cup P'}.CM = MAXLS_{P\cup P'}$—that satisfies eventual collision freedom, such that $\gamma$ is indistinguishable from $\alpha_P(v)$ (resp. $\alpha_{P'}(v')$), through round $k$, with respect to processes described by indices in $P$ (resp. $P'$).

Let us assume, for the sake of contradiction, that both $\alpha_P(v)$ and $\alpha_{P'}(v')$ terminate by round $k = \lg(\frac{|V||I|}{n|V|+|I|})\frac{1}{2}$ By the definition of an ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm, $\gamma$ solves consensus. By assumption, in both $\alpha_P(v)$ and $\alpha_{P'}(v')$, all processes decide by round $k$. By our indistinguishability, these processes decide the same values in $\gamma$. By uniform validity, processes described by indices in $P$ decide $v$, and processes described by indices in $P'$ decide $v'$. Thus, both values are decided in $\gamma$—violating agreement. A contradiction. □

The obvious next question to ask is how the result of Theorem 7 compares to the result of Theorem 6. At first glance, the two results seem potentially incomparable, as the former contains both $|I|$ and $n$ in a somewhat complex fraction, while the latter does not contain either of these two terms. In the following corollary, however, we show that these two results are, in reality, quite similar:

**Corollary 3** *Let $V$ be a value set such that $|V| > 1$, and let $n$ be an integer such that $1 < n \leq \lfloor \frac{|I|}{2} \rfloor$ and $|I| = nk$ for some integer $k > 1$. For any ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm, $\mathcal{A}$, there exists an environment $E \in \mathcal{E}^n$(half-$\mathcal{AC}$,LS), and an execution $\alpha$ of the system $(E, \mathcal{A})$, where $\alpha$ satisfies eventual collision freedom, $CST(\alpha) = 1$, and some process in $\alpha$ does not decide for $\Omega(\min\{\log|V|, \log\frac{|I|}{n}\})$ rounds.*

*Proof* We consider the two possible cases:
*Case 1* $\min\{\log|V|, \log\frac{|I|}{n}\} = \log|V|$.
This implies that $|V| \leq \frac{|I|}{n}$. Therefore, we can express the two terms as follows, where $c$ is a constant greater than or equal to 1:

$$\frac{|I|}{n} = c|V|$$

Solving for $|I|$ we get $|I| = nc|V|$. We can now make this substitution for $|I|$ in the bound from Theorem 7 and simplify:

$$\begin{aligned}
k &= \lg\left(\frac{|V||I|}{n|V| + |I|}\right)\frac{1}{2} \\
&= \lg\left(\frac{|V|nc|V|}{n|V| + nc|V|}\right)\frac{1}{2} \\
&= \lg\left(\frac{nc|V|^2}{(c+1)n|V|}\right)\frac{1}{2} \\
&= \lg\left(\frac{c}{c+1}|V|\right)\frac{1}{2} \\
&= (\lg\left(\frac{c}{c+1}\right) + \lg(|V|))\frac{1}{2} \\
&= \Omega(\lg|V|)
\end{aligned}$$

*Case 2* $\min\{\log|V|, \log\frac{|I|}{n}\} = \log\frac{|I|}{n}$.

This implies that $\frac{|I|}{n} \leq |V|$. As before, we can express the two terms as follows, where $c$ is a constant greater than or equal to 1:

$$|V| = \frac{c|I|}{n}$$

We can now make this substitution for $|V|$ in the bound from Theorem 7 and simplify:

$$\begin{aligned} k &= \lg\left(\frac{|V||I|}{n|V| + |I|}\right)\frac{1}{2} \\ &= \lg\left(\frac{\frac{c|I|}{n}|I|}{n\frac{c|I|}{n} + |I|}\right)\frac{1}{2} \\ &= \lg\left(\frac{c|I|^2}{n(c+1)|I|}\right)\frac{1}{2} \\ &= \lg\left(\frac{c|I|}{(c+1)n}\right)\frac{1}{2} \\ &= \lg\left(\frac{c}{c+1}\right) + \lg\left(\frac{|I|}{n}\right)\frac{1}{2} \\ &= \Omega\left(\lg\frac{|I|}{n}\right) \end{aligned}$$

And, of course, for the case where $|V| = \frac{|I|}{n}$, we can set $c = 1$ in either equation to reduce the result of Theorem 7 to either $\Omega(\lg|V|)$ or $\Omega\left(\lg\frac{|I|}{n}\right)$; meaning any tie-breaking criteria for the min function is fine. □

*Making the bound tight* To match this bound, we can use the algorithm informally described in Sect. 8.3. This algorithm uses Algorithm 2 when $|I| \geq |V|$, and runs Algorithm 2 on the IDs—to elect a leader which can then broadcast its value—in the case where $|I| < |V|$. It runs in time $CST + \Theta(\min\{\lg|V|, \lg|I|\})$ which comes within a factor of $1/n$ of our lower bound. In the following conjecture we posit that this algorithm is, in fact, optimal, and that this gap can be closed through a more complicated counting argument in the lower bound.

*Conjecture 1* Let $V$ be a value set such that $|V| > 1$, and let $n$ be an integer such that $1 < n \leq \lfloor\frac{|I|}{2}\rfloor$ and $|I| = nk$ for some integer $k > 1$. For any ($\mathcal{E}$(half-$\mathcal{AC}$,LS),$V$,ECF)-consensus algorithm, $\mathcal{A}$, there exists an environment $E \in \mathcal{E}^n$(half-$\mathcal{AC}$,LS), and an execution $\alpha$ of the system $(E, \mathcal{A})$, where $\alpha$ satisfies eventual collision freedom, $CST(\alpha) = 1$, and some process in $\alpha$ does not decide for $\Omega(\min\{\lg|V|, \lg|I|\})$ rounds.

The $\frac{|I|}{n}$ term in our previous result stems from the counting argument in Lemma 21, where we consider only $\frac{|I|}{n}$ non-overlapping subsets of $I$. This restriction simplifies the counting argument, but potentially provides some extra information to the algorithm by restricting the sets of processes that can be participating in an execution. We conjecture that a more complicated counting argument, that considers more possible sets of $n$ nodes (some overlapping), could replace this term $\lg|I|$.

### 9.4 Impossibility of consensus with eventual accuracy but without ECF

In this section and the next, we consider executions that do not necessarily satisfy eventual collision freedom. This might represent, for example, a noisy network where processes are never guaranteed to gain solo access to the channel long enough to successfully transmit a full message. We start by showing that consensus is impossible in this model if the collision detector is only eventually accurate.

**Theorem 8** *For every value set $V$, where $|V| > 1$, there exists no ($\mathcal{E}(\Diamond\mathcal{AC}$,LS),$V$,NOCF)-consensus algorithm.*

*Proof* Assume by contradiction that an (E($\Diamond\mathcal{AC}$,LS), V,NOCF)-consensus algorithm, $\mathcal{A}$, exists. First, we fix two disjoint and non-empty subsets of $I$, $P_a$ and $P_b$. Next, we define three environments $A, B, C$ as follows: Let $A.P = P_a$, $B.P = P_b$, and $C.P = P_a \cup P_b$. Let $A.CD = MAXCD_{P_a}(\Diamond\mathcal{AC})$, $B.CD = MAXCD_{P_b}(\Diamond\mathcal{AC})$, and $C.CD = MAXCD_{P_a\cup P_b}(\Diamond\mathcal{AC})$. Let $A.CM = MAXLS_{P_a}$, $B.CM = MAXLS_{P_b}$, and $C.CM = MAXLS_{P_a\cup P_b}$. By definition, $A, B, C \in E(\Diamond\mathcal{AC}$,LS). We next define an execution $\gamma$, of the system $(C, \mathcal{A})$, as follows:

1. Fix the execution such that all processes described by indices in $P_a$ lose all (and only) messages from processes described by indices in $P_b$, and vice versa.
2. Fix the collision detector to satisfy completeness and accuracy in all rounds.
3. Fix the contention manager to return *active* only to the process described by $\min(P_a)$.
4. Fix the execution so that all processes described by indices in $P_a$ start with initial value $v$, and all processes described by indices in $P_b$ start with initial value $v'$, where $v, v' \in V$, $v \neq v'$.

It is clear that $\gamma$ satisfies the constraints of its environment, as, by definition, the collision detector satisfies completeness and eventual accuracy (in fact, it satisfies accuracy), and the contention manager stabilizes to a single *active* process starting in the first round. Therefore, by the definition of an ($\mathcal{E}(\Diamond\mathcal{AC}$,LS),$V$,NOCF)-consensus algorithm, consensus is solved in $\gamma$. Assume all processes decide by round $k$. Let $x \in \{v, v'\}$ be the single value decided.

We will now construct an execution $\alpha$, of the system $(A, \mathcal{A})$, and an execution $\beta$, of the system $(B, \mathcal{A})$, as follows:

1. All processes in $\alpha$ are initialized with $v$, and all processes in $\beta$ are initialized with $v'$.

2. Fix the environments so there is no message loss in either execution.
3. In $\alpha$, fix the contention manager to return $active$ only to the process described by $\min(P_a)$, in $\beta$, for the first $k$ rounds, fix the contention manager to return $passive$ to all processes, and, starting at round $k+1$, have it return $active$ only to the process described by $\min(P_b)$.
4. For all $i \in P_a$ and $r$, $1 \leq r \leq k$, we fix $A.CD$ to return $\pm$ to $\mathcal{A}(i)$ during round $r$, if and only if $\mathcal{A}(i)$ received a collision notification during round $r$ of $\gamma$. We define $B.CD$ in the same way with respect to $P_b$. Starting with round $k+1$, we fix the collision detectors, in both executions, to satisfy completeness and accuracy.

We now validate that $\alpha$ and $\beta$ satisfy the constraints of their respective environments. The contention manager in both executions stabilizes to a single $active$ process (starting in round 1 in $\alpha$, and round $k+1$ in $\beta$). As there is no message loss, then clearly the collision detector satisfies completeness. Finally, we note note that the detector satisfies eventual accuracy as, starting with round $k+1$, by construction, the detectors in both executions become accurate.

Next, we note, by construction, for all $i$ in $P_a$, the execution $\gamma$ is indistinguishable from $\alpha$, with respect to $i$, through round $k$. And for all $j$ in $P_b$, the execution $\gamma$ is indistinguishable from $\beta$, with respect to $j$, through round $k$. As noted above, all processes decide $x \in \{v, v'\}$, by round $k$ in $\gamma$. Therefore, all processes also decide $x$ in their respective $\alpha$ or $\beta$ execution. Assume, without loss of generality, that $x = v$. This implies processes decide $v$ in $\beta$—violating uniform validity. A contradiction. □

### 9.5 Impossibility of anonymous constant round consensus with accuracy but without ECF

In this section, we consider the consensus problem with accurate collision detectors but no ECF guarantees. In Sect. 8, we presented Algorithm 3, an anonymous algorithm which solves consensus in $O(\lg |V|)$ rounds using a collision detector in 0-$\mathcal{AC}$ and no contention manager (i.e., the trivial $NOCM$ contention manager that returns $active$ to all processes in all rounds). Here, we show this bound to be optimal by sketching a proof for the necessity of $\lg |V|$ rounds for any anonymous ($\mathcal{E}(\mathcal{AC}$,NoCM),$V$,NOCF)-consensus algorithm to terminate. Intuitively, this result should not be surprising. Without the ability to ever successfully deliver a message, processes are reduced to binary communication in each round (i.e., silence = 0, collision notification = 1). At a rate of one bit per round, it will, of course, require $\lg |V|$ rounds to communicate an arbitrary decision value from $V$.

**Theorem 9** *Let $V$ be a value set such that $|V| > 1$, and let $n$ be an integer such that $1 < n \leq \lfloor \frac{|I|}{2} \rfloor$. For any anonymous ($\mathcal{E}(\mathcal{AC}$,NoCM),$V$,NOCF)-consensus algorithm, $\mathcal{A}$, there exists an environment $E \in \mathcal{E}^n(\mathcal{AC}$,NoCM), and an execution $\alpha$ of the system $(E, \mathcal{A})$, where some process in $\alpha$ does not decide until after round* $\lg |V| - 1$

*Proof (Sketch)* With no unique identifiers or meaningful contention manager advice to break the symmetry, if we start all processes with the same initial value, and fix the execution such that all messages are lost (except, of course, for senders receiving their own message), then the processes will behave identically. That is, in each round, either all processes broadcast the same message, or all processes are silent.

For a given $n$ value, $1 < n \leq \lfloor \frac{|I|}{2} \rfloor$, and $v \in V$, let $\beta(v)$ be such an execution containing $n$ processes. Let the *binary broadcast sequence* of execution $\beta(v)$ be the infinite binary sequence defined such that position $r$ is 1 if and only if processes broadcast in round $r$ of $\beta(v)$.

For simplicity, assume $|V|$ is a power of 2. By a simple counting argument (i.e., as we saw in Lemma 20), we can show that there must exist two values, $v, v' \in V$ $(v \neq v')$ such that $\beta(v)$ and $\beta(v')$ have the same binary broadcast sequence through round $\lg |V| - 1$. Specifically, there are $2^k$ different binary broadcast count sequences of length $k$. Therefore, for $k = \lg |V| - 1$ there are $2^{\lg |V|-1} = |V|/2$ different sequences. Because we have $|V|$ different $\beta$ executions, one for each value in $V$, by the pigeon-hole principle at least two such executions must have the same binary broadcast count sequence through round $k$. We obtain our needed result through the expected indistinguishability argument (i.e., in the style of the Pasting Lemma). If we compose these two $\beta$ executions into a larger execution $\gamma$, processes cannot distinguish this composition until after round $\lg |V| - 1$. Before this point, there is never a round in which processes from one partition are broadcasting while processes from the other are silent. Therefore, it cannot be the case that processes decide in both $\beta$ executions by round $k$, as they would then decide the same values in $\gamma$—violating agreement. □

*Making the bound tight* This bound is matched by Algorithm 3, which is an anonymous ($\mathcal{E}$(0-$\mathcal{AC}$,NoCM),$V$,NOCF)-consensus algorithm that terminates by round $\Theta(\lg |V|)$.[6]

*The non-anonymous case* It remains an interesting open question to prove a bound for the case where processes have access to IDs and/or a leader election service. Both cases break the symmetry that forms the core of the simple argu-

[6] This upper bound holds after failures cease. Because, however, there are no failures in the executions considered in our above proof, it matches the lower bound. It remains an interesting open question to see if either: (1) one can construct an ($\mathcal{E}$(0-$\mathcal{AC}$,NoCM),$V$,NOCF)-consensus algorithm that terminates in $\Theta(\lg |V|)$ rounds regardless of failure behavior; or (2) one can refine the previous bound to account for delays caused by failures.

ment presented above. Intuitively, however, this extra information should not help the processes decide faster. Without guaranteed message delivery, they are still reduced to, essentially, binary communication. Even if we explicitly provided each process with $P$ for the system, this still would not circumvent the need for some process to spell out its initial value, bit by bit—therefore requiring $\lg |V|$ rounds.

## 10 Conclusion

In this study, we investigated the fault-tolerant consensus problem in a single-hop wireless network. In a novel break from previous work, we considered a realistic communication model in which any arbitrary subset of broadcast messages can be lost at any receiver. To help cope with this unreliability, we introduced (potentially weak) receiver-side collision detectors and defined a new classification scheme to precisely capture their power. We considered, separately, devices that have unique identifiers, and those that do not, as well as executions that allow messages to be delivered if there is a single broadcaster, and executions that do not.

For each combination of these properties—collision detector, identifiers, and message delivery behavior—we explored whether or not the consensus problem is solvable, and, if it was, we proved a lower bound on the round complexity. In all relevant cases, matching upper bounds were also provided. Our results produced the following observations regarding the consensus problem in a realistic wireless network model:

- Consensus *cannot* be solved in a realistic wireless network model without *some* collision detection capability.
- Consensus *can* be solved efficiently (i.e., in a constant number of rounds) if devices are equipped with receiver-side collision detectors that can detect the loss of half or more of the messages broadcast during the round.
- For small value spaces (e.g., deciding to *commit* or *abort*), consensus *can still* be solved efficiently even with a very weak receiver-side collision detector that can only detect the loss of all messages broadcast during the round.
- Collision detectors that produce false positives *are tolerable* so long as they stabilize to behaving properly and the network eventually allows a message to be transmitted if there is only a single broadcaster.
- In the adversarial case of a network that never guarantees to transmit a message, consensus *can still* be solved so long as devices have collision detectors that never produce false positives.
- Perfect collision detection—a detector that detects all message loss—*does not* provide significant advantages over "pretty good" detection—a detector that detects if half or or more of the messages are lost—for solving consensus.
- Unique identifiers *do not* facilitate consensus unless the space of possible identifiers is smaller than the set of values being decided.

There are, of course, many interesting open questions motivated by this research direction. For example, what properties, besides the six completeness and accuracy properties described here, might also be useful for defining a collision detector? Similarly, the zero complete detector seems, intuitively, to be the "weakest" useful detector for solving consensus. Is this true? Are there weaker properties that are still powerful enough to solve this problem? It might also be interesting to consider occasionally well-behaved detectors. For example, a collision detector that is always zero complete and occasionally fully complete. Given such a service, could we design a consensus algorithm that terminates efficiently during the periods where the detector happens to behave well? Such a result would be appealing as this definition of a detector matches what we might expect in the real world (i.e., a device that can usually detect any lost message, but, occasionally—for example, under periods of heavy message traffic—it cannot do better than the detection of all messages being lost).

Another interesting question concerns our assumption that processes start during the same round. If we remove this constraint, can we devise algorithms that still solve consensus? Will this introduce a fundamental complexity gap?

We plan to extend our formal model to describe a multihop network. We are interested in exploring the consensus problem in this new environment, as well as reconsidering already well-studied problems, such as reliable broadcast, and seeing if we can replicate, extend, or improve existing results within this framework.

In conclusion, we note that much of the early work on wireless ad hoc networks did not use detailed communication models. This was sufficient for obtaining the best-effort guarantees needed for many first-generation applications, such as data aggregation. In the future, however, as more and more demanding applications are deployed in this context, there will be an increased need for stronger safety properties. These stronger properties require models that better capture the reality of communication on a wireless medium. As we show in this study, in such models, collision detection is needed to solve even basic coordination problems. Accordingly, we contend that as this field matures, the concept of collision detection should be more widely studied and employed by both theoreticians and practitioners.

**Acknowledgments** We thank the three anonymous reviewers for their insightful and comprehensive editorial suggestions. The resulting paper is much stronger due to their efforts. We also thank Vassos Hadzilacos

for his helpful commentary and assistance in guiding this work through the refereeing process. We additionally acknowledge Rachid Guerraoui for his crucial feedback on the original conference version of this work and his discussions on the potential connection between our formalisms and failure detectors. Finally, we salute Michael Bender for sharing his expertise on back-off protocols.

## Appendix A: Detailed definitions

Here, we described the detailed definitions and properties that were omitted, for the sake of concision, from the main body of the paper.

**Helper definitions related to executions**

- A *state assignment* for $E.P$ is a mapping $S$ from $E.P$ to $\bigcup_{i \in E.P} states_{\mathcal{A}(i)}$, such that for every $i \in E.P$, $S(i) \in states_{\mathcal{A}(i)}$.
  It will be used, in the context of an execution, to describe, for a single round, the current state of each process in the system.
- A *message assignment* for $E.P$ is a mapping from $E.P$ to $M \cup \{null\}$. It will be used, in the context of an execution, to describe, for a single round, the message broadcast (if any) by each process in the system.
- A *message set assignment* for $E.P$ is a mapping from $E.P$ to $Multi(M)$. It will be used, in the context of an execution, to describe, for a single round, the messages received (if any) by each process in the system.
- A *collision advice assignment* for $E.P$ is a mapping from $E.P$ to $\{null, \pm\}$. It will be used, in the context of an execution, to describe, for a single round, the collision detector advice returned to each process in the system.
- A *contention advice assignment* for $E.P$ is a mapping from $E.P$ to $\{active, passive\}$. It will be used, in the context of an execution, to describe, for a single round, the contention manager advice returned to each process in the system.
- Given an infinite sequence $\beta$, of the form $C_0, M_1, N_1, D_1, W_1, C_1, M_2, N_2, D_2, W_2, C_2, ...$, where each $C_r$ is a state assignment for $E.P$, each $M_r$ is a message assignment for $E.P$, each $N_r$ is a message set assignment for $E.P$, each $D_r$ is a collision advice assignment for $E.P$, and each $W_r$ is a contention advice assignment for $E.P$, we define:
  - $t_T(\beta)$ to be the $E.P$-transmission trace $(c_1, T_1), (c_2, T_2), ...$ where for all $i > 0$: $c_i = |\{j | j \in E.P \text{ and } M_i[j] \neq null\}|$; and, for all $i > 0$ and $j \in E.P$: $T_i[j] = |N_i[j]|$. That is, $t_T(\beta)$ is the unique $E.P$-transmission trace described by the message assignments in $\beta$.
  - $t_{CD}(\beta)$ to be the $E.P$-CD trace $CD_1, CD_2, ...$ where for all $i > 0$ and for all $j \in EP$: $CD_i[j] = D_i[j]$. That is, $t_{CD}(\beta)$ is the unique $E.P$-CD trace described by the collision advice assignments in $\beta$.
  - $t_{CM}(\beta)$ to be the $E.P$-CM trace $CM_1, CM_2, ...$ where for all $i > 0$ and for all $j \in E.P$: $CM_i[j] = W_i[j]$. That is, $t_{CM}(\beta)$ is the unique $E.P$-CM trace described by the contention advice assignments in $\beta$.
  - $t_C(\beta)$ to be the set $\{j | j \in E.P \text{ and } \forall i \geq 0, C_i[j] \neq fail_{\mathcal{A}}\}$. That is, $T_C(\beta)$ is the set of processes that never enter the $fail$ state.

We can now provide the following formal definition of an execution:

**Definition 19 (Execution)** An *execution* $\alpha$ of a system $(E, \mathcal{A})$ is an infinite sequence

$C_0, M_1, N_1, D_1, W_1, C_1, M_2, N_2, D_2, W_2, C_2, ...$

where each $C_r$ is a state assignment for $E.P$, each $M_r$ is a message assignment for $E.P$, each $N_r$ is a message set assignment for $E.P$, each $D_r$ is a collision advice assignment for $E.P$, and each $W_r$ is a contention advice assignment for $E.P$. We assume the following constraints:

1. For all $i \in E.P$: $C_0[i] \in start_{\mathcal{A}(i)}$.
2. For all $i \in E.P$ and $r > 0$: either $C_r[i] = trans_{\mathcal{A}(i)}(C_{r-1}[i], N_r[i], D_r[i], W_r[i])$ or $C_r[i] = fail_{\mathcal{A}(i)}$.
3. For all $i \in E.P$ and $r > 0$: $M_r[i] = msg_{\mathcal{A}(i)}(C_{r-1}[i], W_r[i])$.
4. $N_r[i] \subseteq \bigcup_{j \in E.P} MS(\{M_r[j]\} - \{null\})$.
5. If $M_r[i] \neq null$, then $M_r[i] \in N_r[i]$.
6. $t_{CD}(\alpha) \in E.CD(t_T(\alpha))$.
7. $t_{CM}(\alpha) \in E.CM(t_C(\alpha))$.

Informally speaking, $C_r$ represents the system state after $r$ rounds, while $M_r$ and $N_r$ represent the messages that are sent and received at round $r$, respectively. $D_r$ describes the advice returned from the collision detector to each process in round $r$, and $W_r$ describes the advice returned from the contention manager to each process in round $r$.

Constraints 19.1 and 19.2 require that each process start from an initial state and subsequently evolve its state according to its transition function. Notice, in constraint 19.2 it is possible for a process to instead enter its fail state. Once here, by the constraints of our process definition, it can never leave this state or broadcast messages for the remainder of an execution. We use this to model crash failures.

Constraint 19.3 requires that processes broadcast according to their message transition function. Constraint 19.4 requires the receive behavior to uphold integrity and no-duplication, as it specifies that the receive set of a process for a given round must be a sub-multiset of the multiset defined by the union of all messages broadcast that round.

Constraint 19.5 requires broadcasters to always receive their own message. Notice, however, that message loss is otherwise un-constrained. *Any process can lose any arbitrary subset of messages sent by other processes during any round.* Similarly, we never force message loss. Even if every process in the system broadcasts, it is still possible that all processes will receive all messages. Finally, constraints 19.6 and 19.7 require the collision advice and contention advice to conform to the definitions of the environments collision detector and contention manager, respectively.

We use the terminology *k-round execution prefix* to describe a prefix of an execution sequence that describes only the first $k$ rounds (i.e., the sequence through $C_k$).

The following definitions also prove useful:

**Definition 20 (Indistinguishability)** Let $\alpha$ and $\alpha'$ be two executions, defined over systems $(E, \mathcal{A})$ and $(E', \mathcal{A})$, respectively—that is, the same algorithm in possibly different environments. For a given $i \in E.P \cap E'.P$, we say $\alpha$ is indistinguishable from $\alpha'$, with respect to $i$, through round $r$, if $C_0[i]$ is the same in both executions, and, for all $k$, $1 \leq k \leq r$, the state $(C_k[i])$, message $(M_k[i])$, message set $(N_k[i])$, collision advice $(D_k[i])$, and contention advice $(W_k[i])$ assignment values for round $k$ and index $i$ are also the same in both. That is, in $\alpha$ and $\alpha'$, $\mathcal{A}(i)$ has the same sequence of states, the same sequence of outgoing messages, the same sequence of incoming messages, and the same sequence of collision detector and contention manager advice up to the end of round $r$.

**Definition 21 (Correct)** Let $\alpha$ be an execution of system $(E, \mathcal{A})$. For a given $i \in E.P$, we say process $\mathcal{A}(i)$ is correct in $\alpha$ if and only if for all $C_r \in \alpha$, $C_r[i] \neq fail_{\mathcal{A}(i)}$. That is, $\mathcal{A}(i)$ never enters its fail state during $\alpha$.

*Property 1* (*Eventual Collision Freedom*) Let $\alpha$ be an execution of system $(E, \mathcal{A})$. We say $\alpha$ satisfies the eventual collision freedom property if there exists a round $r_{\text{cf}}$ such that for all $r \geq r_{\text{cf}}$ and all $i \in E.P$: if $t_T(\alpha)(r) = (c, T)$ and $c = 1$, then $T(i) = 1$. That is, there exists a round $r_{\text{cf}}$ such that for any round greater than or equal to $r_{\text{cf}}$, if only a single process broadcasts then all processes receive its message.

*Property 2* (*Wake-up Service*) A given $P$-contention manager, $S_{CM}$, is a wake-up service if for every $A$, where $A$ is a non-empty subset of $P$, and for every $P$-CM trace $t_{CM} \in S_{CM}(A)$ there exists a round $r_{\text{wake}}$ such that for all $r \geq r_{\text{wake}}$: $|\{i | i \in A \text{ and } t_{CM}(r)(i) = active\}| = 1$. That is, for all rounds greater than or equal to $r_{\text{wake}}$, only a single correct process is told to be $active$.

*Property 3* (*Leader Election Service*) A given $P$-contention manager, $S_{CM}$, is a leader election service if for every $A$, where $A$ is a non-empty subset of $P$, for every $P$-CM trace $t_{CM} \in S_{CM}(A)$, there exists a round $r_{\text{lead}}$ such that for all $r \geq r_{\text{lead}}$, $|\{i | i \in A \text{ and } t_{CM}(r)(i) = active\}| = 1$, and for all $r > r_{\text{lead}}$ , if $t_{CM}(r)(i) = active$, then $t_{CM}(r-1)(i) = active$. That is, for all rounds greater than or equal to $r_{\text{lead}}$, the same single correct process is told to be $active$.

**Definition 22** ($MAXLS_P$) Let $P$ be any non-empty subset of $I$, and let $CM_P$ be the set of all $P$-contention managers that are leader election services. $MAXLS_P$ is the $P$-contention manager described by the set $\{t_{CM} | \exists S \in CM_P \text{ s.t. } t_{CM} \in S\}$.

*Property 4* (*Completeness*) A given $P$-collision detector, $Q$, satisfies completeness if and only if for all pairs $(t_T, t_{CD})$—where $t_T$ is an $P$-transmission trace, $t_{CD}$ is an $P$-CD trace, and $t_{CD} \in Q(t_T)$—and for all $r > 0$ and $i \in P$, the following holds: if $t_T(r) = (c, T)$ and $T(i) < c$, then $t_{CD}(r)(i) = \pm$. That is, if a process loses any message then that process detects a collision.

*Property 5* (*Majority completeness*) A given $P$-collision detector, $Q$, satisfies majority completeness if and only if for all pairs $(t_T, t_{CD})$—where $t_T$ is an $P$-transmission trace, $t_{CD}$ is a $P$-CD trace, and $t_{CD} \in Q(t_T)$—and for all $r > 0$ and $i \in P$, the following holds: if $t_T(r) = (c, T)$ and $c > 0$ and $T(i)/c \leq 0.5$, then $t_{CD}(r)(i) = \pm$. That is, if a process loses half or more of the messages then that process detects a collision.

*Property 6* (*Half completeness*) A given $P$-collision detector, $Q$, satisfies half completeness if and only if for all pairs $(t_T, t_{CD})$—where $t_T$ is an $P$-transmission trace, $t_{CD}$ is a $P$-CD trace, and $t_{CD} \in Q(t_T)$—and for all $r > 0$ and $i \in P$, the following holds: if $t_T(r) = (c, T)$ and $c > 0$ and $T(i)/c < 0.5$, then $t_{CD}(r)(i) = \pm$. That is, if a process loses more than half of the messages then that process detects a collision.

*Property 7* (*Zero completeness*) A given $P$-collision detector $Q$, satisfies zero completeness if and only if for all pairs $(t_T, t_{CD})$—where $t_T$ is an $P$-transmission trace, $t_{CD}$ is an $P$-CD trace, and $t_{CD} \in Q(t_T)$—and for all $r > 0$ and $i \in P$, the following holds: if $t_T(r) = (c, T)$ and $c > 0$ and $T(i) = 0$, then $t_{CD}(r)(i) = \pm$. That is, if a process loses every message then that process detects a collision.

*Property 8* (*Accuracy*) A given $P$-collision detector, $Q$, satisfies accuracy if and only if for all pairs $(t_T, t_{CD})$—where $t_T$ is an $P$-transmission trace, $t_{CD}$ is an $P$-CD trace, and $t_{CD} \in Q(t_T)$—and for all $r > 0$ and $i \in P$, the following holds: if $t_T(r) = (c, T)$ and $T(i) = c$, then $t_{CD}(r)(i) = null$. That is, if a process receives all messages then that process does not detect a collision.

*Property 9* (*Eventual accuracy*) A given $P$-collision detector $Q$, satisfies eventual accuracy if and only if there exists

a round $r_{\text{acc}}$ such that for all pairs $(t_T, t_{CD})$—where $t_T$ is an $P$-transmission trace, $t_{CD}$ is a $P$-CD trace, and $t_{CD} \in Q(t_T)$—and for all $r > 0$ and $i \in P$, the following holds: if $t_T(r) = (c, T)$ and $r \geq r_{\text{acc}}$ and $T(i) = c$, then $t_{CD}(r)(i) = null$. That is, starting at some round $r_{\text{acc}}$, if a process receives all messages than that process does not detect a collision.

**Definition 23** ($MAXCD_P(C)$) Let $P$ be any non-empty subset of $I$, and let $C$ be a set of collision detectors that includes at least one $P$-collision detector. Then $MAXCD_P(C)$ is a $P$-collision detector defined as follows: For any $P$-trans mission trace $t$, $MAXCD_P(C)(t) = \bigcup_{Q \in C, Q \text{ is a } P-CD} Q(t)$.

## References


1. IEEE 802.11. Wireless lan mac and physical layer specifications, June 1999
2. Abramson, N.: Development of the alohanet. IEEE Tran. Inform. Theor **31**, 119–123 (1985)
3. Aspnes, J., Fich, F., Ruppert, E.: Relationships between broadcast and shared memory in reliable anonymous distributed systems. In: 18th International Symposium on Distributed Computing, pp. 260–274 (2004)
4. Bender, M.A., Farach-Colton, M., He, S., Kuszmaul, B.C., Leiserson, C.E.: Adversarial contention resolution for simple channels. In: Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 325–332 (2005)
5. Bharghavan, V., Demers, A., Shenker, S., Zhang, L.: Macaw: A media access protocol for wireless lans. In: Proceedings of the ACM SIGCOMM '94 Conference on Communications Architectures, Protocols, and Applications (1994)
6. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
7. Bogdan, S.C., Dariusz, R.K., Mariusz, A.R.: Adversarial queuing on the multiple-access channel. In: PODC '06: Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing, pp. 92–101. ACM Press, New York (2006)
8. Chockler, G., Demirbas, M., Gilbert, S., Lynch, N., Newport, C., Nolte, T.: Reconciling the theory and practice of (un)reliable wireless broadcast. International Workshop on Assurance in Distributed Systems and Networks (ADSN) (to appear) (2005)
9. Chockler, G., Demirbas, M., Gilbert, S., Newport, C.: A middleware framework for robust applications in wireless ad hoc networks. In: Proceedings of the 43rd Allerton Conference on Communication, Control, and Computing (2005)
10. Chockler, G., Demirbas, M., Gilbert, S., Newport, C., Nolte, T: Consensus and collision detectors in wireless ad hoc networks. In: PODC '05: Proceedings of the Twenty-fourth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 197–206. ACM Press, New York (2005)
11. Chockler, G., Demirbas, M., Gilbert, S., Newport, C., Nolte, T.: Consensus and collision detectors in wireless ad hoc networks. In: Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing. ACM Press, New York (2005)
12. Clementi, A.E.F., Monti, A., Silvestri, R.: Selective families, superimposed codes, and broadcasting on unknown radio networks. In: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 709–718. Society for Industrial and Applied Mathematics, Philadelphia (2001)
13. Deng, J., Varshney, P.K., Haas, Z.J.: A new backoff algorithm for the IEEE 802.11 distributed coordination function. In: Communication Networks and Distributed Systems Modeling and Simulation (CNDS '04) (2004)
14. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
15. Farach-Colton, M., Fernandes, R.J., Mosteiro, M.A.: Lower bounds for clear transmissions in radio networks. In: Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN), pp. 447–454 (2006)
16. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
17. Ganesan, D., Krishnamachari, B., Woo, A., Culler, D., Estrin, D., Wicker, S.: Complex behavior at scale: an experimental study of low-power wireless sensor networks. UCLA Computer Science Technical Report UCLA/CSD-TR (2003)
18. Goldberg, L.A., Jerrum, M., Kannan, S., Paterson, M.: A bound on the capacity of backoff and acknowledgment-based protocols. SIAM J. Comput. **33**(2), 313–331 (2004)
19. Goldberg, L.A., Mackenzie, P.D., Paterson, M., Srinivasan, A.: Contention resolution with constant expected delay. J. ACM **47**(6), 1048–1096 (2000)
20. Haas, Z.J., Deng, J.: Dual busy tone multiple access (dbtma)-a multiple access control scheme for ad hoc networks. IEEE Trans. Comm. **50**(6), 975–985 (2002)
21. Haringstad, J., Leighton, T., Rogoff, B.: Analysis of backoff protocols for mulitiple access channels. SIAM J. Comput. **25**(4), 740–774 (1996)
22. Jurdzinski, T., Stachowiak, G.: Probabilistic algorithms for the wake-up problem in single-hop radio networks. Theor. Comput. Syst. **38**(3), 347–367 (2005)
23. Koo, C.-Y.: Broadcast in radio networks tolerating byzantine adversarial behavior. ACM Symposium on Principles of Distributed Computing (PODC), pp. 275–282 (2004)
24. Kotz, D., Newport, C., Gray, R.S., Liu, J., Yuan, Y., Elliott, C.: Experimental evaluation of wireless simulation assumptions. In: Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, pp. 78–82 (2004)
25. Kowalski, D.R.: On selection problem in radio networks. In: Proceedings of the Twenty-fourth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 158–166, ACM Press, New York (2005)
26. Kumar, M.: A consensus protocol for wireless sensor networks. Master's thesis, Wayne State University (2003)
27. Lamport, L.: Paxos made simple. ACM SIGACT News **32**(4), 18–25 (2001)
28. Lynch, N.: Distributed Algorithms. Morgan Kaufman, San Francisco (1996)
29. Metcalfe, R.M., Boggs, D.R.: Ethernet: distributed packet switching for local computer networks. Commun. ACM **19**(7), 395–404 (1976)
30. Moscibroda, T., Wattenhofer, R.: Maximal independent sets in radio networks. In: Proceedings of the 24th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 148–157, ACM, New York (2005)
31. Newport, C.: Consensus and Collision Detectors in Wireless Ad Hoc Networks. Master's thesis, MIT, Cambridge (2006)
32. Polastre, J., Culler, D.: Versatile low power media access for wireless sensor networks. The Second ACM Conference on Embedded Networked Sensor Systems (SENSYS), pp. 95–107 (2004)

33. Raghavan, P., Upfal, E.: Stochastic contention resolution with short delays. SIAM J. Comput. **28**(2), 709–719 (1999)
34. Santoro, N., Widmayer, P.: Time is not a healer. In: Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science, pp. 304–313. Springer, Heidelberg (1989)
35. Santoro, N., Widmayer, P.: Distributed function evaluation in presence of transmission faults. In: Proceedings of International Symposium on Algorithms (SIGAL), pp. 358–367 (1990)
36. van Dam, T., Langendoen, K.: An adaptive energy-efficient MAC protocol for wireless sensor networks. The First ACM Conference on Embedded Networked Sensor Systems (SENSYS), pp. 171–180 (2003)
37. Woo, A., Tong, T., Culler, D.: Taming the underlying challenges of multihop routing in sensor networks. The First ACM Conference on Embedded Networked Sensor Systems (SENSYS), pp. 14–27 (2003)
38. Woo, A., Whitehouse, K., Jiang, F., Polastre, J., Culler, D.: Exploiting the capture effect for collision detection and recovery. In: Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors, pp. 45–52 (2005)
39. Ye, W., Heidemann, J., Estrin, D.: An energy-efficient mac protocol for wireless sensor networks. In: Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM) (2002)
40. Zhao, J., Govindan, R.: Understanding packet delivery performance in dense wireless sensor networks. The First ACM Conference on Embedded Networked Sensor Systems (SENSYS), pp. 1–13 (2003)