Consensus and Collision Detectors in Wireless Ad Hoc Networks

Calvin Newport

July 10, 2006

Abstract

In this study, we consider the fault-tolerant consensus problem in wireless ad hoc networks with crash-prone nodes. Specifically, we develop lower bounds and matching upper bounds for this problem in single-hop wireless networks, where all nodes are located within broadcast range of each other. In a novel break from existing work, we introduce a highly unpredictable communication model in which each node may lose an arbitrary subset of the messages sent by its neighbors during each round. We argue that this model better matches 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 ten 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.

Our contention is that the introduction of (possibly weak) receiver-side collision detection is an important approach to reliably solving problems in unreliable networks. Our results, derived in a realistic network model, provide important feedback to ad hoc network practitioners regarding what hardware (and low-layer software) collision detection capability is sufficient to facilitate the construction of reliable and fault-tolerant agreement protocols for use in real-world deployments.

Contents

1 Introduction

1.1 Wireless Ad Hoc Networks

Properties of Wireless Ad Hoc Networks. Wireless ad hoc networks are an important platform for bringing computational resources to diverse contexts. These networks are characterized by limited devices, deployed in novel environments in an ad hoc fashion (that is, typically, no *a priori* knowledge of the environment or connection topology is assumed). Direct communication is possible only with neighbors through the use of local radio broadcast. It is often the case, though not always, that the devices have limited computational capability, local memory, and power. Depending on the context, location information is sometimes available; perhaps derived from a GPS unit or through the use of special ranging hardware (e.g. [62]) coupled with distance-based localization schemes; c.f. [55, 65].

Because devices in ad hoc networks are commonly low-cost (to ease the expense of large, rapid, and temporary deployments), they are prone to unpredictable crash failures. Similarly, their local clocks can operate at varying rates depending on temporal environmental effects such as temperature; complicating the task of maintaining synchronized clocks. See [24] for a more extensive discussion of clock behavior and expected skew under different conditions.

GPS units, on the other hand, can be used to provide high precision time values. In practice, however, the rate at which these values are obtained from the unit is reduced by the demands of the device driver and operating system. The delay between timer updates can therefore be sufficiently large for the intervening clock drift to cause non-trivial skew. Gray et al. encountered this problem when trying to calculate message latency values from a mobile ad hoc network deployment [31]. Here, the skew accumulated between GPS time updates was sufficient to require the use of an alternative clock synchronization scheme based on the approach presented in [25]

There exists, however, a strong body of both experimental and theoretical research on protocols that overcome these timing-related difficulties to achieve reasonably close clock synchronization; c.f. [4, 25, 26, 66]. For example, in [25], clock synchronization within $3.68 \pm 2.57 \mu sec$ was achieved for a multihop network deployed over 4 communication hops.

In many networks, devices have unique identifiers, derived through randomization or provided in advance (such as a MAC Address read from a wireless adapter). These identifiers, however, are not always present. For example, in an extremely dense network of tiny devices—such as the cubic millimeter sized motes envisioned for "smart dust" deployments [35, 60]—the size of the random numbers needed to ensure uniqueness, with high probability, or the effort required to provide identifiers in advance, might be prohibitive. Also, in some scenarios, the use of unique identifiers might induce privacy concerns. Consider, for example, a wearable wireless device that interacts with static devices, with known positions, deployed throughout a hospital. Perhaps the device provides its user with an interactive map of the building or monitors his vital signs so that it can report an medical emergency to the hospital staff. If this wearable device made use of a unique identifier during these interactions, it would, in effect, be leaving a trace of the user's movement through the hospital; potentially revealing private information about the owner's health status. This type of concern motivated the design of the identifier-free location service in [62].

Finally, we note that radio broadcast communication, the only means of communication available to devices in wireless ad hoc networks, is inherently unreliable. Two (or more) nearby radios broadcasting at the same time can interfere with each others' transmissions. This could lead to the loss of all messages at a given receiver as the signal-to-noise ratio grows too large to distinguish one transmission from another.

It's also likely, however, as a result of the well-know capture effect [71], that in this scenario one of the messages is successfully received while the others are lost. This capture behavior is unpredictable and can lead, in practice, to non-uniform receive sets among multiple receivers within range of multiple simultaneous transmissions. For example, assume, in an area contained within a single broadcast radius, that two devices, A and B, broadcast a message at the same time, while two devices, C and D, are listening. Multiple outcomes are possible: perhaps both C and D receive no message, or C receives A's message and D receives B's message, or both C and D receive A's message, or C receives nothing and D receives B's message, *etc.*

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; c.f. [1, 61, 68, 72]. 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 [9] 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. And 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 [30, 38, 70, 73], which show that even with sophisticated collision avoidance mechanisms (e.g., 802.11 [1], B-MAC [61], S-MAC [72], and T-MAC [68]), 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. Either they feature a built-in resiliency to lost communication, or expend the computational and time resources required to build a higher-level solution; such as constructing a global TDMA schedule that prevents nearby nodes from broadcasting during the same slot; c.f. [7, 8, 10, 12, 43, 51]. Notice, however, that the TDMA approach incurs a heavy static overhead, relies on knowing the local topology and membership information, and therefore, does not scale. This makes it inappropriate for many scenarios.

Mobile Ad Hoc Networks. An important subclass of wireless ad hoc networks are *Mobile Ad Hoc Networks*. In such networks, the devices are assumed to be attached to mobile agents whose movements patterns cannot be controlled or predicted. Clearly, this situation introduces new problems for coordination as the topology of the underlying connection graph is constantly changing. The point-to-point routing problem—where a named source needs to route a message to a named destination—is the most widely studied problem in these networks; c.f. [29, 34, 36, 58, 59]. This is perhaps a reflection of the difficulty of performing more complicated coordination under such dynamic conditions. Recent work, however, such as the virtual infrastructure systems developed at MIT [20–22]—which makes use of the underlying mobile devices to emulate arbitrary automaton at fixed locations or following well-defined movement patterns—and the NASCENT system developed by Luo and Hubaux [52]—which provides several group-management primitives for small networks of mobile devices—facilitate the design of more complex coordination algorithms for this challenging environment.

Static Ad Hoc Networks. Among the different static wireless ad hoc networks discussed in the literature, perhaps the most widely cited are so-called "sensor networks." These networks, typically consisting of small devices running Berkley's TinyOS [32] operating system and equipped with some manner of environmental sensing equipment, are used to gather, analyze, and aggregate data from the environment in which they are deployed. For example, in [67] a dense sensor network was used to monitor climate conditions on a remote island off the coast of Maine.

Research involving static ad hoc networks, such as sensor networks, can be, roughly speaking, divided into three main categories. The first is *information dissemination*. Protocols such as TRICKLE [48] (and a similar scheme proposed by Lynch and Livadas [50])—which first flood a message through the network and then later have devices "gossip" with their neighbors to see if they missed any recent messages—and GLIDER [27]—which first builds up a synthetic coordinate system based upon distances to pre-determined "landmarks" and then uses greedy geographic routing techniques to route messages—are among many that have been proposed as a practical method for delivering a message to an entire network or specific destination. Of course, the point-to-point routing algorithms developed for mobile ad hoc networks can also be used in these static networks. But their mechanisms for coping with mobility tend to produce an unnecessary degree of overhead.

Starting with a paper by Bar-Yehuda et. al. [7], and followed by many others (e.g., [6, 39, 41]), there have also been many strictly theoretical examinations of the broadcast problem in such static networks; with a focus on producing lower bounds. These studies describe, for example, a logarithmic, in the number of devices, deterministic lower bound on the time required to broadcast a message under certain conditions [39]. And a randomized lower bound, in terms of the expected number of rounds to complete a broadcast, of $\Omega(D \log(\frac{N}{D}))$ [46] (where D is the maximum minimum hop-count between two devices—sometimes called the *network diameter*—and N is an upper bound on the number of devices).

The second category is *data aggregation*. Almost all of the original uses of sensor networks involved gathering data over time and aggregating it at a central source. Systems such as Madden's TinyDB [54] focus on efficient structures for accomplishing this task with a minimum of energy expenditure. More recently, some attention has been diverted toward more responsive data gathering applications, such as the tracking of a mobile agent through a field of sensor-equipped devices; c.f. [17, 45]

The final category is *local coordination*. To facilitate the achievement of higher-level goals, such as information dissemination or data aggregation, it is often helpful to first tame some of the unpredictability introduced by an ad hoc deployment. For example, there has been much work on the topology control problem (e.g. [5, 49]), which attempts to have nodes reduce their transmit power to a minimum level that still provides sufficient connectivity throughout the network. By reducing transmit power one can reduce the number of devices within range of each other's radio. This, in turn, reduces the overall contention in the network. It also preserves energy, which, as mentioned, is an omnipresent goal in resource-constrained networks.

Another local coordination problem of interest is the construction of clusters, such that each device ends up belonging to a single cluster with a well-defined "clusterhead." This goal is considered useful for coordinating both local and global communication. Early work focused on clusters that represented *dominating sets*—a collection (preferably minimal) of "clusterheads," such that each device in the network is either a clusterhead or within communication range of a clusterhead; c.f. [2, 33, 42]. More recent research (e.g. [56]) considers *maximal independent sets*, which add the additional restriction that the cluster heads themselves are not within communication range of each other. This extra property is advantageous as it allows these cluterheads to communicate with their respective clusters while minimizing interference with the transmissions at neighboring clusters.

Examples of other local coordination problems include leader election in a single-hop radio network (e.g. [57]), in which a single device from among many competing declares itself a "leader," and the k-selection problem (e.g. [16, 40]), also considered in single-hop regions, in which k active devices coordinate such that each gets a time slot to broadcast its message.

1.2 The Total Collision Model

A claim we first made in [13] and expanded upon in [14, 15], is that there exists a considerable gap between theory and reality when it comes to the study of wireless ad hoc networks. This gap is caused, in our opinion, by differing treatments of message loss. As mentioned in the preceding discussion of ad hoc networks, radio behavior in these settings is inherently unpredictable. When producing theoretical results for these networks, however, precise communication models are required. These models, in the interest of clarity and simplicity, often replace the unpredictable behavior of real networks with a set of well-defined rules. Perhaps the most widely-used communication model, which we refer to as the *total collision model*, specifies:

1. *If no neighbor of device* d *broadcasts, then* d *receives nothing.*

2. *If two or more neighbors of* d *broadcast, then* d *receives nothing.*

3. *If a single neighbor of* d *broadcasts, then* d *receives its message.*

This model was first introduced, in the context of wireless ad hoc networks, with the Bar-Yehuda et al. [7] broadcast paper mentioned previously. It was later adopted in almost every subsequent theoretical study of the broadcast problem, as well as in most theoretical studies of local coordination problems. A variant on this model, sometimes referenced, is to provide the devices with strong receiver-side collision detection. Here, it is possible for a device to distinguish cases 1 and 2. The introduction of this strong collision detection can, in some instances, significantly change the costs of basic operations. For example, in [19] it is shown that, under certain assumptions, a $\Omega(n)$ lower bound for broadcast in a network of n nodes and diameter D can be reduced to $\Omega(D + \log n)$ with the availability of collision detection.

The problem with the total collision model is that it is unrealistic. As we described previously, it is not true that two or more neighbors of device d, broadcasting at the same time, will *always* lead to d losing *all* messages. It's certainly possible that d, due to the capture effect [71], receives one of these messages. Furthermore, though synchronized broadcast rounds can be a reasonable assumption (as clock synchronization is, as mentioned, a well-studied problem in practice), it's not always reasonable to imagine that these rounds are tightly tuned to the exact time required to broadcast a single packet. Such a goal might require a degree of synchrony that defeats what can actually be achieved. It also neglects the sometimes significant degree of non-determinism that exists in the time between an application deciding to broadcast a message and a packet actually being transmitted. It is reasonable, therefore, to expect that communication rounds are large relative to the time required to send a single packet. In this case, d might receive more than one, but perhaps not all, of the many messages sent during the same round.

Clearly, the total collision model failures to capture these possibilities; and this failure has significant implications. For example, Kowalski and Pelc [39], using the total collision model, construct a broadcast algorithm that operates in $O(\log n)$ rounds in small diameter networks of n devices. They also provide a lower bound that shows this result to be tight in this context. Their algorithm, however, fails in a slightly less predictable variant of this model where, in the case of two or more neighbors of d broadcasting, d might receive no message *or* one message. In fact, Bar Yehuda et al. [7] show that in this new model the lower bound on broadcasting is increased to $\Omega(n)$ rounds.[1]

We claim that an important first step toward closing the gap between theory and practice with regard to wireless ad hoc networks is to replace the total collision model with one that better captures the unpredictability of this setting. In the next sub-section we describe a network model, inspired by the weaker model introduced (somewhat unintentionally) by Bar Yehuda et al. in [7], that we feel achieves this goal.

1.3 Our Network Model

Here we present an overview of our network model and justifications for its constituent assumptions. Because this study focuses on fault-tolerant consensus—a local coordination problem—our model captures only a single-hop network of static nodes. Other local coordination problems—such as leader election [57] and k-selection [16,40]—have also been studied mainly in the context of a single-hop static network. As we describe in Section 1.4, local consensus provides a fundamental building block for building reliable services at a network-wide scale. This study, therefore, represents an important first step toward understanding the necessary conditions for bringing reliability to this unreliable setting.

Basic Assumptions. We model a fully-connected single-hop collection of n crash-prone wireless devices running deterministic protocols. By "single-hop," we mean that every device is within communication range of every other device. We assume no mobility. To match the realities of ad hoc deployment, we assume the value n is *a priori* unknown to the devices. And, as both are common, we will consider the case where devices have access to unique identifiers and the case where they do not. Indeed, one of the questions we investigate in this study is the advantage of identifiers when attempting to coordinate in such a network.

Synchronized Rounds. We assume synchronized rounds with all devices starting during the same round. These rounds could be implemented with a well-known clock synchronization algorithm such as RBS [25];

[1]Note, in the original version of [7] Bar Yehuda et al. mistakenly specified that they were, in fact, working in the total collision model. As pointed out in [39], and in an errata published later, these results require the ability of a single message to be occasionally received in the case of two or more neighbors of a single device broadcasting during the same round.

which has proved to work well in practice. For the sake of theoretical consistency, however, we also describe, in [14], a fault-tolerant round synchronization algorithm that is provably correct in a partially synchronous variant of our model. In other words, we show how, starting with drifting clocks, wireless devices can efficiently build and maintain synchronized broadcast rounds under the various realistic communication restraints assumed in our model. [2]

Message Loss. Communication in our model is unpredictable. Specifically, in any round, any device can lose any subset of the messages broadcast by other devices during the round. Of course, in real networks, it is usually the case that if a *single* device broadcasts, then *all* devices should receive its message. To capture this reality, we introduce a property called *eventual collision freedom*, which states that there exists some round in every execution after which if a single device broadcasts then all devices receive its message. The reason we don't always assume this property to hold from the first round is that our single-hop network might be a clique in the middle of a larger multi-hop network. In this case, interference, in the form of broadcasts from neighboring regions, can cause a single message to be lost. If one assumes eventual collision freedom, then one is assuming that eventually, through some sort of higher-level coordination, that neighboring regions will be quiet long enough for the region of interest to accomplish what it needs to accomplish without outside interference. We study coordination both in executions that satisfy this property and those that do not.

Collision Detectors. To help mitigate the complications introduced by our communication model, we also assume receiver-side collision detectors. These detectors are binary. Each round they return to each device either $null$—a rough indication that the receiver didn't lose any messages this round—or $\pm$—a rough indication that the receiver lost a message during the round. Notice, these detectors offer no information concerning the number, content, or source of lost messages.

In a novel break from past work, we do not necessarily assume that these detectors are "perfect." (that is, return $\pm$ if and only if that device lost a message). Though such perfect detectors might be useful in

[2]The algorithm described in [14] works for an arbitrary multi-hop network of diameter D. It requires a $\Theta(D)$ delay to resynchronize every $\Theta(1)$ time. For the special case of a single-hop network, however, where $D = 1$, this is quite reasonable, especially considering the constant factor within the $\Theta(D)$ term is less than one round length, and the constant factor in the $\Theta(1)$ term is, for reasonable values of round length and clock drift rates, around 1000.

theory, they might also be more difficult to realize in practice. Accordingly, we consider many variants of collision detectors. Specifically, 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. We define them as follows:

- **Completeness:** A detector satisfies completeness if it guarantees to return $\pm$ to a device if that device lost one or more messages during the round.

- **Majority Completeness:** A detector satisfies majority completeness if it guarantees to return $\pm$ to a device if that device didn't receive a strict majority of the messages sent during that round. This property corresponds to the practical reality that often, when many messages are sent, it is possible for a *small* number of these messages to be lost in the clutter without detection, but, if too many are lost, the detector will be able to detect some noise on the channel indicative of this loss.

- **Half Completeness:** Similar to majority completeness, a detector satisfies half completeness if it guarantees to return $\pm$ to a device if that device didn't receive half or more of the messages sent during that round. The difference between this property and the last appears to be slight. We introduce them both, however, because we are able to find a significant complexity gap between them concerning the number of rounds required to solve consensus.

- **Zero Completeness:** A detector satisfies zero completeness if it guarantees to return $\pm$ to a device if that device lost all of the messages sent during that round. This property is particularly appealing because of its practicality. A zero complete detector is required only to distinguish between silence and the loss of all messages. In other words, it need only conduct physical carrier sensing, a process already well studied and commonly implemented as part of most CSMA protocols used in many wireless MAC layers; c.f. [1, 61, 68, 72]. In fact, in a study by Deng et al. [18], it is suggested that there currently exists no technical obstacle to adding carrier-sensing based collision detection support to the current 802.11 protocol.

- **Accuracy:** A detector satisfies accuracy if it guarantees to return $null$ to a device if that device received all messages sent during the round.

- **Eventual Accuracy:** A detector satisfies eventual accuracy if there exists a round in every execution after which it guarantees to be accurate. This weaker property is meant to capture the possibility of the occasional false positive that might be generated by practical collision detection schemes.

We have begun to explore implementations of collision detectors that match these properties. Early experiments have shown that simple detection 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. See [14] for a more detailed discussion of the techniques used in these early detector implementations.

Contention Managers. We also introduce a service, which we call a contention manager, that 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 manager properties should eventually stabilize on only a small number of devices (namely, 1) being labeled as *active*, thus allowing, in executions satisfying eventual collision freedom, for messages to be delivered without collision. One could imagine, for example, such a service being implemented in a real system by a backoff protocol. Such protocols have been studied extensively; cf. [16, 69].

Our motivation behind encapsulating this task into an abstract service is to free both the designer of algorithms and the designer of lower bounds from the concerns specific to contention management. As mentioned, much work has already been done in this field, and we don't desire, for example, to re-prove the properties of various backoff protocols for each problem we consider. Instead, we specify time bounds *relative* to stabilization points of the contention manager. For example, we show that, using certain types of collision detectors, consensus can be solved within a constant number of rounds after the contention manager stabilizes to a single broadcaster, while, using different types of collision detectors, consensus requires an additional $\Theta(\log |V|)$ rounds after this stabilization point (where V is the set of possible initial values for consensus).

Exactly *when* this stabilization point occurs is a property of a specific contention manager implementation, and it is a detail we do not concern ourselves with in this study. In a sense, by encapsulating contention management in an abstract service we make it easier to focus on the complexity unique to specific problems separate from the complexity of reducing contention.

Furthermore, this encapsulation provides an important separation between safety and liveness. That is, if one relies on the contention manager only to ensure liveness (as is the case for all protocols described in this study), then, even if, in practice, the contention manager satisfies its property only with high probability, only the liveness of the protocol becomes probabilistic in nature. This separation, between a guaranteed safety property and a (potentially) probabilistic liveness property is important for the design of *robust applications*—such as coordinating actuator-equipped wireless devices to reconfigure a factor assembly line, or using a sensor network to aim a missile strike—where the violation of certain safety properties, even with only a low probability of occurrence, is unacceptable. See [14] for a more detailed discussion of such applications.

Of course, for the designer who is specifically interested in constructing exact contention management bounds in our model, one can simply disregard the contention manager, and handle this problem of contention explicitly in their protocol design. We introduce this abstraction only to simplify the examination of problems, such as consensus, for which the reduction of contention is not the most important issue.

1.4 The Consensus Problem In Wireless Ad Hoc Networks

The focus of this paper is the fault-tolerant consensus problem. In this problem, all devices in a single-hop network are provided with some initial value from a known value set V. They then execute a protocol that results in each device deciding some $v \in V$. This protocol must satisfy three properties:

1. **Agreement:** No two devices decide a different value.
2. **Strong Validity:** If a device decides value v, then v is the initial value of some device. A variant to this property is **Uniform Validity**, which requires that if all devices 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 composing our lower bounds, and strong validity when composing our matching upper bounds.
3. **Termination:** All devices that do not crash eventually decide.

Fault-tolerant consensus is an important building block for wireless ad hoc networks, as it is a fundamental primitive for many local coordination activities. For example, devices within a single region of a sensor network may need to decide on a new offset parameter to calibrate their sensors. It is important that all devices agree on the same parameter, as, otherwise, some device might produce sensor readings that are incomparable with the others, destroying attempts to perform meaningful data aggregation.

Similarly, for many activities, such as the selection of a clusterhead for a network clustering scheme, leader election is necessary. Consensus run on unique identifiers is an obvious, reliable solution to this problem. Furthermore, many data aggregation systems (e.g. [54]) aggregate data by passing values up a spanning tree. Due to unreliable communication some values might get lost, weakening the guarantees that can be made about the final output of the aggregation. To help counter this unreliability, a consensus protocol can be run among the children of each parent in the tree to agree on the values to be disseminated.

And, as Kumar proposes in [44], consensus can be used to simplify the dissemination of information from a large sensor network to a common source. Specifically, he suggests that first the devices sub-divide themselves into non-overlapping clusters. Then, within each cluster, consensus is executed to decide on what information that cluster wants to return to the source. This process has the effect of reducing the number of messages traveling through the network while ensuring that all devices still have a "vote" in deciding what information is ultimately returned.

There has been extensive prior work on fault-tolerant consensus in synchronous [53], partially synchronous [23], asynchronous with failure detectors [11, 47] and fully asynchronous [28] message passing systems with reliable or eventually reliable point-to-point channels. In particular, to tolerate message loss the work of [23, 47] assumes 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 [63, 64] 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 [64] are not applicable in our setting since they rely on a priori known number of participants, and do not tolerate node failures.

In [44], Kumar presents a quorum-based solution to solving 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 [37] presents an (almost) tight lower bound for the minimum fraction of Byzantine neighbors that 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 [13]. Many of the algorithms and lower bounds examined in this study were first described in [15]. And, in [14], we discussed how to implement the elements of our model in practice.

1.5 Our Results

In this study we examine the fault-tolerant consensus problem under different conditions. We are interested in determining both how much collision detection information is necessary to solve the problem, and, for the cases where the problem *is* solvable, how many rounds are required. We also examine the effect of the eventual collision freedom property and the availability of unique identifiers on our results. Specifically, we produce the following:

Impossibility Results Under Eventual Collision Freedom Assumption.

- In Theorem 4 in Section 8.1 we show consensus cannot be solved with no collision detector, and in Theorem 5 in Section 8.2, we show that consensus cannot be solved with a collision detector that doesn't satisfy eventual accuracy. These results hold even if we assume a contention manager that eventually stabilizes to a single $active$ device, and the eventual collision freedom property. In other words, eventually electing a leader, and giving it the ability to communicate reliably, is not enough to solve consensus. The reason is that without a useful collision detector, one cannot tell when the system has stabilized to this good point.

Impossibility Result Under No Eventual Collision Freedom Assumption.

- In Theorem 8 in Section 8.4, we show that for executions that do not satisfy eventual collision freedom, consensus cannot be solved with a collision detector that satisfies only eventual accuracy. This holds even if the detector also satisfies completeness and we assume a contention manager that eventually stabilizes to a single $active$ device. In other words, having a collision detector that is always complete and eventually accurate is not enough to solve consensus in an environment with no message delivery guarantees, as, in this context, collision notifications are the only way to communicate, and the eventual accuracy conditions makes it difficult to tell whether a notification is real or a false positive.

Round Complexity Lower Bounds Under Eventual Collision Freedom Assumption.

- In Theorem 6 in Section 8.3.3, we show that, using a collision detector that satisfies half completeness and accuracy, no anonymous algorithm can guarantee to solve consensus in less than $\Theta(\log |V|)$ rounds[3] for all initial value assignments from value set V. This holds even if we assume a contention manager that eventually stabilizes to a single $active$ device and the eventual collision freedom property. In other words, if devices are equipped with detectors that can allow half of the messages in a round to be lost without notification, then they are reduced to transmitting their values at a rate of one bit per round. Roughly speaking, this is due to the fact that such a detector can allow the network to partition into two equal-sized groups that will remain unaware of each other unless their exists a round in which processes from one group broadcast while processes from the other are silent. The only way for anonymous processes to generate such an asymmetry is to use the bits of their initial values as a broadcast pattern.

- In Theorem 7 and Corollary 3 in Section 8.3.4, we show that, for the case of non-anonymous algorithms, the previous half completeness bound can be refined to $\Omega(min\{\log |V|, \log \frac{|I|}{n}\})$ rounds,

[3]All bounds described in this sub-section are relative to the first round after which the contention manager has stabilized to a single $active$ process and the eventual collision freedom property holds.

where I is the set of all possible identifiers, and n is the number of nodes participating. Once again, this holds even if we assume a contention manager that eventually stabilizes to a single $active$ device and the eventual collision freedom property. This indicates the perhaps surprising reality that unique identifiers, roughly speaking, do not help solve consensus faster. That is, if I is large relative to V (as is often the case, because identifiers in most real networks either consist of many randomly chosen bits or a long MAC address), then the lower bound is asymptotically the same for both the anonymous and non-anonymous case.

Round Complexity Lower Bound Under No Eventual Collision Freedom Assumption.

- In Theorem 9 in Section 8.5, we show that, for executions that do not satisfy eventual collision freedom, no anonymous protocol that does not use a contention manager can solve consensus in less than $\Theta(\log |V|)$ rounds, even if we assume a perfect detector (e.g. complete and accurate). In other words, for an environment that never guarantees the successful transmission of a message, processes are reduced to spelling out their value bit-by-bit (i.e., a silent round indicates 0, a collision notification indicates 1). We conjecture that this bound holds even if we assume a leader election service and unique identifiers, as neither helps processes communicate a value faster than one bit per round.

Upper Bounds Under Eventual Collision Freedom Assumption

- In Section 7.1 we present an *anonymous* protocol (Algorithm 1) that solves consensus in $O(1)$ rounds if: (1) each process has access to a collision detector that is majority complete and eventually accurate, and a contention manager that eventually stabilizes to no more than one $active$ process per round; (2) the execution satisfies eventual collision freedom.[4]

- In Section 7.2 we present an *anonymous* protocol (Algorithm 2) that solves consensus in $\Theta(\log |V|)$ rounds if: (1) each process has access to a collision detector that is zero complete and eventually accurate, and a contention manager that eventually stabilizes to no more than one $active$ process per round; (2) the execution satisfies eventual collision freedom. This algorithm matches the $\Theta(\log |V|)$ lower bound for collision detectors that are half-complete or weaker.

[4]As with the lower bounds, all upper bounds are relative to the first round after which the contention manager has stabilized to a single $active$ process and the eventual collision freedom property holds.

- In Section 7.3 we describe, informally, a *non-anonymous* protocol that solves consensus in $\Theta(min\{\log|V|, \log|I|\})$ rounds, where I is the size of the ID space, if: (1) each process has access to a collision detector that is zero complete and eventually accurate, and a contention manager that eventually stabilizes to no more than one $active$ process per round; (2) the execution satisfies eventual collision freedom. This protocol is a simple variant of Algorithm 2, and, for the case of I being large relative to V (which is typically true in real deployments), matches our non-anonymous lower bound of $\Omega(min\{\log|V|, \log\frac{|I|}{n}\})$. For the case where I is small, this algorithm comes within a factor of $\frac{1}{n}$ of this bound. Note, however, that n describes only the number of nodes in a single-hop area of a network—n is, in this respect, a constant, as only so many devices can physically be fit into a single broadcast radius (V and I, on the other hand, can be arbitrarily large).

Upper Bounds Under No Eventual Collision Freedom Assumption

- In Section 7.4, we present an *anonymous* protocol (Algorithm 3) that solves consensus in $\Theta(\log|V|)$ rounds if the process has access to a collision detector that is zero complete and accurate. This algorithm matches the $\Theta(\log|V|)$ lower bound for collision detectors that are accurate and executions that do not satisfy eventual collision freedom.

2 Preliminaries

- 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 unique value that appears in M.

3 The System Model

3.1 Model Definitions

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 components:

1. $states_A$, a potentially infinite set of *states*. It describes all possible states of A.

2. $start_A$, a non-empty subset of $states_A$ known as the *start states*. It describes the states in which A can begin an execution.

3. $fail_A$, a single state from $states_A$ known as the *fail state*. We will use this state to model crash failures in our model.

4. msg_A, a message generation function that maps $states_A \times \{active, passive\}$ to $M \bigcup \{null\}$, where M is our fixed message alphabet and $null$ is a placeholder indicating no message. We assume $msg_A(fail_A, *) = null$. This function describes what message (or $null$ if no message) is generated by A for each combination of a state and advice from a contention manager. As we will soon describe, the advice $active$ indicates that a process should try to send a message, while $passive$ indicates that it should not (due to contention). As is made obvious by this definition, the process is under no obligation to follow this advice. For the special case of the fail state, we constrain the function to always return $null$ regardless of the contention manager advice.

5. $trans_A$, a state transition function mapping $states_A \times Multi(M) \times \{\pm, null\} \times \{active, passive\}$ to $states_A$, where $Multi(M)$ is the set of all possible finite multisets defined over M. We assume $trans_A(fail_A, *, *, *) = fail_A$. This function describes the evolution of the states of A based on the current state, the received messages, the collision detector advice, and the contention manager advice. For the special case of the fail state, we force the process to stay in the fail state. This models a process crash failure (from which there is not restarting).

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

Notice, by this definition, it is perfectly valid for some algorithm $\mathcal{A}$ to encode i in the state of automaton $\mathcal{A}(i)$, for all $i \in I$. In some scenarios, however—especially those involving ad hoc wireless networks consisting of a large number of small, low-cost devices—it might be useful to consider only algorithms that provide *no* differentiation among the processes. This corresponds to the practical case where devices are assumed to have no unique IDs. We capture this possibility with the following algorithm property:

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* and a P*-CD trace*, each defined over a non-empty subset P of I. The former will be used to describe, for a given execution involving the indices in P, how many processes broadcast a message and how many receive a message, at each round. The latter will be used to describe, for a given execution also involving processes in P, what collision detector advice each process receives at each round.

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

Definition 5 (P-CD trace). A P-CD trace, where P is a non-empty subset of I, is an infinite sequence of mappings, $CD_1, CD_2, \ldots$ 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 message were sent in each round, and how many messages each process received 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. This captures our practically motivated ideal of a receiver-side device that only attempts to distinguish whether or not some messages broadcast during the round were lost.

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 given set, P, of indexes, as a set of P-CM traces. That is, a contention manager is simply defined by the full set of possible advice sequences that it might return. Notice, this separates the contention manager from the communication behavior occurring during the execution. We do not mean to imply that our model captures only oblivious contention management schemes. The separation of the formal contention manager definition from other aspects of the execution was enacted to promote clarity in our theoretical model. We assume, in practice, that a contention manager might be actively monitoring the channel and, perhaps, even generating control messages of its own. For the purposes of this framework, however, we are concerned only with the eventual guarantees of a contention manager (i.e., it eventually stabilizes to a single $active$ process) not the details of how these guarantees are met. As we described in the introduction, this latter point is already well-studied and can obscure other aspects of the problem at hand that might be interesting in their own right.

Definition 8 (P-Contention Manager). A P-contention manager, where P is a non-empty subset of I, is 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 indexes, and an algorithm is a mapping from process indexes 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.

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

3.2 Executions and Indistinguishability

Given a system $(E, \mathcal{A})$, we introduce the following definitions:

- 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.

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

Definition 11 (Execution). An *execution* 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 $E.P$. 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. 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. Let t_T be the P-transmission trace $(c_1, T_1)(c_2, T_2), ...$ where for all $i > 0$: $c_i = |\{j | j \in P \; and \; M_i[j] \neq null\}|$; and, for all $i > 0$ and $j \in P$: $T_i[j] = |N_i[j]|$. That is, t_T is the unique P-transmission trace described by the message assignments in this execution. Let t_{CD} be the P-CD trace $CD_1, CD_2, ...$ where for all $i > 0$ and for all $j \in P$: $CD_i[j] = D_i[j]$. That is, t_{CD} is the unique P-CD trace described by the collision advice assignments. Then $t_{CD} \in E.CD(t_T)$.

7. Let t_{CM} be the P-CM trace $CM_1, CM_2, ...$ where for all $i > 0$ and for all $j \in P$: $CM_i[j] = W_i[j]$. That is, t_{CM} is the unique P-CM trace described by the contention advice assignments. Then $t_{CM} \in E.CM$.

Informally, constraints 1 and 2 require that each process start from an initial state and subsequently evolve its state according to its transition function. Notice, in constraint 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 3 requires that processes broadcast according to their message transition function. Constraint 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 mutliset defined by the union of all messages broadcast that round. Constraint 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 6 and 7 require the collision advice and contention advice to conform to the definitions of the environment's 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).

Definition 12 (Indistinguishability). Let α and α' 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 α is indistinguishable from α', 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 α and α', $\mathcal{A}(i)$ 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.

3.3 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 formal 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 our process, it can never leave the fail state and never broadcast any message. We use the following definition to distinguish crashed processes from non-crashed processes:

Definition 13 (Correct). Let α be an execution of system $(E, \mathcal{A})$. For a given $i \in E.P$, we say process $\mathcal{A}(i)$ is correct in α 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 α.

Message Loss As described above, our execution formalism places no explicit limit on message loss. 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*.[5] Accordingly, we define a communication property, which we refer to as the *eventual collision freedom (ECF)* property, that captures this behavior.

Property 1 (Eventual Collision Freedom).
Let α be an execution of system $(E, \mathcal{A})$, and let t_T be the unique P-transmission trace described by α. We say α satisfies the eventual collision freedom property if there exists a round r_{cf} such that for all $r \geq r_{cf}$

[5] 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.

and all $i \in E.P$: if $t_T(r) = (c, T)$ and $c = 1$, then $T(i) = 1$. That is, there exists a round r_{cf} such that for any round greater than or equal to r_{cf}, if only a single process broadcasts then all processes receive its message.

4 Contention Managers

As described in the introduction, 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 manager properties should eventually stabilize on only a small number of processes (namely, 1) being labeled as *active* in each round, thus allowing, in executions satisfying eventual collision freedom, for messages to be delivered without collisions.

4.1 The Wake-up and Leader Election Services

A natural contention manager property can be defined as follows:

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

Notice, however, that this property maintains no fairness conditions. That it is, it only specifies *how many* processes will eventually be active in a given round, not *which* processes these will be. A reasonable extension of this property might guarantee stabilization to a single leader:

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

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.

To solve other interesting problems, one could might imagine a more expansive property that includes, for example, the guarantee that *all* processes get a chance to be the single *active* process. For example, one might describe a k-wake-up service that guarantees *all* processes k rounds of being the only $active$ process in the system. There exist simple problems, such as counting the number of anonymous processes in the system, that can easily be shown to be solvable with a k-wake-up service, but impossible with a leader election service (and, thus, wake-up service as well).

4.2 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. The first is the **WS** class which we define to include all wake-up services. The second is the **LS** class which we define to include all leader-election services. 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 $P \subseteq I$.

4.3 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, and provide the following definition:

Definition 14 ($MAXLS_P$). 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 $\{t_{CM} | \exists S \in CM_P \; s.t. \; t_{CM} \in S\}$.

5 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.

5.1 Completeness Properties

We say that a collision detector satisfies *completeness* if it guarantees to report a collision at any process that lost a message. We formalize this property as follows:

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 fails to receive all messages then that process detects a collision.*

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. To this end, it is reasonable 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 did not receive a majority of the messages sent during the round. We formalize this property as follows:

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 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 $T(i)/c \leq 0.5$, then $t_{CD}(r)(i) = \pm$. That is, if a process fails to receive a strict majority of the messages then that process detects a collision.

A collision detector satisfies *half completeness* if it guarantees to report a collision at any process that receives less than half of the messages sent during the round. Notice the close similarity between this property and majority completeness. The two properties differ only by a single message. That is, the half completeness property allows a process to lose one more message than the majority completeness property before guaranteeing to report a collision. We formalize this property as follows:

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 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 $T(i)/c < 0.5$, then $t_{CD}(r)(i) = \pm$. That is, if a process fails to receive half of the messages then that process detects a collision.

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. This final definition is appealing because of its practicality. It requires only the ability to distinguish silence from noise (a problem already solved by the carrier sensing capabilities integrated into many existing wireless MAC layers). We formalize this property as follows:

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 fails to receive any message then that process detects a collision.

5.2 Accuracy Properties

A collision detector satisfies *accuracy* if it guarantees to report a collision to a process only if that process failed to receive a message. We formalize this property as follows:

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.

	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-$\Diamond\mathcal{AC}$

Figure 1: A summary of collision detector classes.

In order to account for the situation in which arbitrary noise can be 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 after which the detector becomes accurate. 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.

Property 9 (Eventual Accuracy).
A given P-collision detector, Q, satisfies eventual accuracy if and only if there exists a round r_{acc} such that 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 $r \geq r_{acc}$ and $T(i) = c$, then $t_{CD}(r)(i) = null$. That is, starting at some round r_{acc}, if a process receives all messages than that process does not detect a collision.

Notice that we don't 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. 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?

5.3 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 Table 1. 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 only that we have some detector from a given class. When we derive lower bounds for a given class, we, as the lower bound designer, are free to choose any detector from this class.

Before continuing, we introduce two special collision detection classes for which notation is not included in Figure 1. The first is the **NoACC** class, which we define to include all collision detectors that satisfy completeness.

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 useful lemma which will aid our lower bound construction:

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

Proof. Follows directly from the definitions. □

5.4 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. Formally:

Definition 15 ($MAXCD_P(C)$)**.** 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-transmission trace t, $MAXCD_P(C)(t) = \bigcup_{Q \in C, Q \text{ } is \text{ } a \text{ } P-CD} Q(t)$.

5.5 The Noise Lemma

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

Lemma 2. *For any execution α of system $(E, \mathcal{A})$, where $E.CD$ satisfies zero completeness, and t_T and t_{CD} are the unique transmission and collision advice traces described by α, respectively, the following guarantee is satisfied: For all $r > 0$ and $i \in E.P$, if $t_T(r) = (c, T)$ and $c > 0$, then either $T(i) > 0$ or $t_{CD}(r)(i) = \pm$. That is, if one more or 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 (Lemma 2). *For any execution α of system $(E, \mathcal{A})$, where $E.CD$ satisfies zero completeness, and t_T and t_{CD} are the unique transmission and collision advice traces described by α, respectively, the following guarantee is satisfied: For all $r > 0$ and $i \in E.P$, if $t_T(r) = (c, T)$ and $T(i) = 0$ and $t_{CD}(r)(i) = null$, then $c = 0$. That is, if any process receives nothing and detects no collision, then no process broadcast.*

Proof. Follows directly from Lemma 2. □

6 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.[6] 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 formal notation for describing exactly the conditions under which the algorithm guarantees to solve the consensus problem. To accomplish this, we first offer the following two definitions that describe large classes of environments that share similar properties:

Definition 16 ($\mathcal{E}(D, M)$). 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 17 ($\mathcal{E}^n(D, M)$). 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 $|E.P| = n\}$.

To obtain the strongest possible results, we use the first definition when proving upper bounds and the second when proving lower bounds. We now offer two different notations for describing the guarantees of an algorithm. The first specifies correctness only for executions that satisfy eventual collision freedom, the second requires correctness for all executions.

[6]To capture the notion of an "input value" in our formal 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.

Definition 18 (($\mathcal{E}$,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 α of system $(E, \mathcal{A})$, where $E \in \mathcal{E}$, initial values are assigned from V, and α satisfies eventual collision freedom, α solves consensus.

Definition 19 (($\mathcal{E}$,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 α of system $(E, \mathcal{A})$, where $E \in \mathcal{E}$ and initial values are assigned from V, α solves consensus.

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

Definition 20 (Communication Stabilization Time (CST)). Let α be an execution of system $(E, \mathcal{A})$, where α satisfies eventual collision freedom, $E.CM$ is a wake-up service, and $E.CD$ satisfies eventual accuracy. The *Communication Stabilization Time* of α (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 α be an execution of system $(E, \mathcal{A})$, where α 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 returned $passive$ by the contention manager broadcasts, the following conditions are true:*

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

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

7 Consensus Algorithms

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 Section 2, 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, and $halt_i$ to indicated that process p_i halts.

Roadmap. We start in Section 7.1 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 Section 7.2 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 subset 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 Section 7.3 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 Section 7.4 we describe an anonymous algorithm that solves consensus, even in executions that don't satisfy eventual collision freedom, using any collision detector from 0-$\mathcal{AC}$. The algorithm terminates in $O(\lg(|V|)$ rounds after failures cease.

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

```
Process P_i:
  estimate_i ∈ V, initially set to the initial value of process P_i
  phase_i ∈ {proposal, veto}, initially proposal
  For each round r, r ≥ 1 do:
    if (phase_i = proposal) then
      if CM()_i = active then
        bcast(estimate_i)_i
      messages_i ← SET(recv()_i)
      CD-advice_i ← CD()_i
      if (CD-advice_i ≠ ±) and (|messages_i| > 0) then
        estimate_i ← min{messages_i}
      phase_i ← veto
    else if (phase_i = veto) then
      if (CD-advice_i = ±) or (|messages_i| > 1) then
        bcast(veto)_i
      veto-messages_i ← recv()_i
      CD-advice_i ← CD()_i
      if (veto-messages_i = ∅) and (CD-advice_i = null) and (|messages_i| = 1) then
        decide(estimate_i)_i and halt_i
      phase_i ← proposal
```

7.1 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 guarantees to solve consensus in any execution, satisfying eventual collision freedom, of 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 $CST + 2$.

The algorithm consists of two alternating phases: a *proposal* phase and a *veto* phase. In the proposal phase, every process that was returned the advice $active$ from 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 or received more than one unique value in the preceding round. 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[7] or detecting a collision.

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

[7] 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.

the 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 it 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:

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

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. To prove this statement we demonstrate that $v \in E_r \Rightarrow v \in E_{r-1}$, for $r \geq 1$. By definition of Algorithm 1, $estimate$ can be altered only on line 11 of the $proposal$ phase, where it is assigned the value of a message received during a $proposal$-phase round. By line 7, only $estimate$ values are broadcast in these rounds. Therefore, if some process p_i ends round r with $estimate_i = v$, then only two cases are possible. (1) p_i ends round $r-1$ with $estimate_i = v$ and maintains it through r; or, (2) some other node p_j ends $r-1$ with $estimate_j = v$, and then broadcast the value to p_i in r. In either case: $v \in E_{r-1}$. □

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

Proof. By the lemma assumptions, each process receives exactly one value and no collision notification during round r. Assume, for the sake of contradiction, that some process p_i receives only the value v in r, and some other node p_j receives only the value v' in r ($v \neq v'$). Because neither p_i nor p_j receives a collision notification, by the definition of majority completeness each must receive a majority of the messages broadcast during r. Because p_i receives only value v, a majority of the messages broadcast in r must contain v. Similarly, because p_j receives only value v', a majority of the messages broadcast in r

must contain v'. This is, of course, impossible, as majority sets intersect. A contradiction. It follows that each process receives the same value. Furthermore, because no process, by assumption, receives a collision notification, then, by lines 10 and 11, all processes set $estimate$ to this single value during round r. □

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

Proof. A process decides only its $estimate$ value. Accordingly, if a process p decides in round r, then it decides a value from E_{r-1}. From Lemma 4, we know $E_{r-1} \subseteq E_0$, where E_0 is the set of initial values. □

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

Proof. Let r be the first round in which a process decides. Let p_i be a process that decides in r. By line 18, since p_i decides in r, then it receives exactly one unique value in $r-1$. It follows that at least one message is sent in $r-1$. Therefore, we can apply Lemma 2, which provides that *all* non-crashed processes must therefore receive at least one unique value or a collision notification in $r-1$.

Line 18 also provides that p_i receives no messages or collision notifications during $veto$-phase round r. By Corollary 1, it follows that no process broadcasts a veto in r. By line 14, a process vetos during round r if it receives more than one unique value or a collision notification in $r-1$. Therefore, we know that any process that is non-crashed though round r does not receive a collision notification or more than one unique value in $r-1$ (as they would have then send a veto at line 15 during r). We also know, from our proceeding observation, that each of these processes receive at least one unique value or a collision notification in $r-1$. Combined, this tells us that each of these processes receives exactly *one* unique value and no collision notifications during round $r-1$.

This matches the assumptions for Lemma 5, which provide that $|E_{r-1}| = 1$. Because p_i decides v in r, we further conclude $E_{r-1} = \{v\}$. By Lemma 4, we know for all $r' \geq r-1$, $E_{r'} \subseteq E_{r-1}$. Because processes only decide their $estimate$ value, any process that decides in round $r' \geq r-1$, must decide v.

□

Lemma 8 (Termination). *All correct processes decide and halt 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. By our algorithm, and the fact that $CST \geq r_{wake}$, we also know a single process broadcasts.

Every process receives the lone broadcaster's value (which we will call v_r) and no collision notification. By lines 10 and 11, every non-crashed process therefore adopts v_r as its $estimate$ during this round.

During the next round, $r+1$, no process sends a veto, as each non-crashed process receives one message and no collision notifications in r. Therefore, it is trivially true that no process that is returned $passive$ during the round broadcasts in $r+1$, as no process broadcasts in $r+1$. Thus, we can apply Lemma 3 once again, which provides that there are no collision notifications in $r+1$. Accordingly, every non-crashed process will pass the test on line 18 and decide.

In the worst case, CST is a $veto$-phase round. This means that $r = CST + 1$. Since all processes decide by $r+1$, we get the desired result that all processes decide by $CST + 2$. □

Proof (Theorem 1). Correctness follows from Lemmas 6, 7 and 8. □

7.2 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 guarantees to solve consensus in any execution, satisfying eventual collision freedom, of 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 returned $active$ from 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

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

```
Process $P_i$:
  $estimate_i \in V^{0,1}$, initially set to a binary rep. of $P_i$'s initial value
  $phase_i \in$ {prepare, propose, accept}, initially prepare
  $size \leftarrow \lceil \lg |V| \rceil$
  For each round $r$, $r \geq 1$ do:
    if ($phase_i$ = prepare) then
      if CM()$_i$ = active then
        bcast($estimate_i$)$_i$
      $messages_i \leftarrow$ SET(recv()$_i$)
      $CD\text{-}advice_i \leftarrow$ CD()$_i$
      if ($CD\text{-}advice_i \neq \pm$) and ($|messages_i| > 0$) then
        $estimate_i \leftarrow \min\{messages_i\}$
      $decide_i \leftarrow$ true
      $bit_i \leftarrow 1$
      $phase_i \leftarrow$ propose
    else if ($phase_i$ = propose) then
      if ($estimate_i[bit_i] = 1$) then
        bcast(veto)$_i$
      $votes_i \leftarrow$ recv()$_i$
      $CD\text{-}advice_i \leftarrow$ CD()$_i$
      if (($|votes_i| > 0$) or ($CD\text{-}advice_i = \pm$)) and ($estimate_i[bit_i] = 0$) then
        $decide_i \leftarrow$ false
      $bit_i \leftarrow bit_i + 1$
      if ($bit_i > size$) then
        $phase_i \leftarrow$ accept
    else if ($phase_i$ = accept) then
      if (not $decide_i$) then
        bcast(veto)$_i$
      $veto\text{-}messages_i \leftarrow$ recv()$_i$
      $CD\text{-}advice_i \leftarrow$ CD()$_i$
      if ($|veto\text{-}messages_i| = 0$) and ($CD\text{-}advice_i \neq \pm$) then
        decide($estimate_i$)$_i$ and halt$_i$
      $phase_i \leftarrow$ prepare
```

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 at this round.

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

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. The proof follows from the same logic as Lemma 4. As in Algorithm 1, processes can only alter their $estimate$ value to a value received in a round where only $estimate$ values are broadcast (see line 12). Therefore, if some process p_i ends round r with $estimate_i = v$, then only two cases are possible. (1) p_i ends round $r-1$ with $estimate_i = v$ and maintains it through r; or, (2) some other node p_j ended $r-1$ with $estimate_j = v$, and then broadcast the value to p_i in r. In either case: if $v \in E_r$, then $v \in E_{r-1}$ □

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. 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 broadcast. It follows from Corollary 1, which states that silence implies no one broadcast, that there was never a round during this phase where two (non-crashed) processes behaved differently (i.e., one broadcast, one did not). Therefore, all processes that make it through this $propose$-phase without failing must have started the phase with the same $estimate$ value. Because this value is only modified during the $prepare$-phase, these processes all begin the subsequent $accept$-phase with the same $estimate$. □

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

Proof. By the definition of Algorithm 2, processes only decide their $estimate$ value (line 32). Accordingly, if some process p decides in round r, then p decides a value from E_r. By Lemma 9, we know $E_r \subseteq E_0$, where E_0 is the set of initial values. □

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

Proof. 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. Line 31 provides that $|veto\text{-}messages_i| = 0$ and $CD\text{-}advice_i \neq \pm$ during this round, where $veto\text{-}messages_i$ and $CD\text{-}advice_i$ are the veto messages received and collision detector advice, respectively. By Corollary 1, we conclude that no process broadcasts a veto during r. Processes would broadcast a veto in r if their $decide$ value equals $false$. Therefore, all non-crashed processes start r with $decide$ equal to $true$. Lemma 10 provides that, in this case, all non-crashed processes also started round r with the same $estimate$ value. Because p_i decides v during this round, and processes decide their $estimate$ value, it follows that this common $estimate$ value is v. Thus $E_{r-1} = \{v\}$. By Lemma 9, for all $r' \geq r$, $E_{r'} \subseteq E_{r-1}$. Therefore, any process that decides in round $r' \geq r$, must also decides v. □

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

Proof. 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 receive every message broadcast; (2) no process receives a collision notification. By our algorithm, and the fact that $CST \geq r_{wake}$, we know that a single process will broadcast in r.

By our results from above, all non-crashed processes receive this process's value (which we will call v_r) and no collision notification. By lines 11 and 12, all non-crashed processes therefore adopt v_r as their $estimate$ during this round.

It follows that all processes start the $propose$ phase with the same $estimate$. This implies, by the definition of the algorithm, that all processes broadcast on the same schedule for the $size = \lceil \lg |V| \rceil$ rounds of this phase. We want to show that no process will set $decide \leftarrow false$ during this phase. To do so, we consider only rounds corresponding to a 0 bit in v_r, as, by the definition of the algorithm, these are the only rounds in which a process with estimate v_r can set $decide \leftarrow false$.

It is trivially true that no process returned $passive$ during one of these rounds broadcasts, as *no* process broadcast in these rounds. Thus, we can apply Lemma 3 once again, which provides that no collision notifications are received during these listening rounds.

Accordingly, all non-crashed processes begin the $accept$ phase with $decide$ still equal to $true$. Thus, no process broadcasts a $veto$. By the same logic used above to reason about the listening rounds during the $propose$ phase, no process will receive a collision notification during this $accept$-phase round. Therefore, all non-crashed processes pass the tests on line 31 and decide and halt.

In the worst case, CST occurs during the first round of the $propose$ phase. This means r would fall $\lceil \lg |V| \rceil + 1$ rounds after CST. Since all processes decide by $r + \lceil \log |V| \rceil + 1$, we get the desired result that all processes decide by $CST + 2(\lceil \lg |V| \rceil + 1)$. □

Proof (Theorem 2). Correctness follows from Lemmas 11, 12 and 13. □

7.3 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-$\Diamond\mathcal{AC}$,WS),V,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[8] our non-anonymous lower bound for this setting (Corollary 3 in Section 8).

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

[8] 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 $\frac{1}{n}$ between the two. As mentioned earlier, however, n is, practically speaking, a small constant, as it describes only the number of devices within a single broadcast radius. The values $|V|$ and $|I|$, on the other hand, can be arbitrarily large, and can easily swamp the $\frac{1}{n}$ factor. In Conjecture 1, we claim that $\Omega(min\{\lg |V|, \lg |I|\})$ is, in fact, the real lower bound.

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 2 rounds after whichever comes later: leader election or CST. This provides a worse 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 criteria 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. This ensures that the second run of consensus cannot terminate until all non-crashed processes have detected the current leader's failure. 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.

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

```
Process P_i:
  estimate_i ∈ V, initially set to the initial value of process P_i
  phase_i ∈ {vote-val, vote-left, vote-right, recurse}, initially vote-val
  curr_i,  A node pointer, initially set to the root of a balanced binary search tree representation of V
  For each round r, r ≥ 1 do:
    if (phase_i = vote-val) then
      if (estimate_i = val[curr_i ]) then
        bcast(‘‘vote′′)_i
      msgs(1)_i ← recv()_i
      CD(1)_i  ← CD()_i
      phase_i ← vote-left
    else if (phase_i = vote-left) then
      if (estimate_i ∈ left[curr_i ]) then
        bcast(‘‘vote′′)_i
      msgs(2)_i ← recv()_i
      CD(2)_i ← CD()_i
      phase_i ← vote-right
    else if (phase_i = vote-right) then
      if (estimate_i ∈ right[curr_i ]) then
        bcast(‘‘vote′′)_i
      msgs(3)_i ← recv()_i
      CD(3)_i ← CD()_i
      phase_i ← recurse
    else if (phase_i = recurse) then
      if (|msgs(1)_i| > 0) or (CD(1)_i = ±) then
        decide(val[curr_i ])_i
        halt_i
      else if (|msgs(2)_i| > 0) or (CD(2)_i = ±) then
        curr_i ← left[curr_i ]
      else if ((|msgs(3)| > 0) or (CD(3)_i = ±)) then
        curr_i ← right[curr_i ]
      else
        curr_i ← parent[curr_i ]
      phase_i ← vote-val
```

7.4 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 $vote\text{-}val$, processes can vote for the value represented by the current node in the tree by broadcasting. A process will vote in this phase if and only if this value is its initial value. In the second phase, called $vote\text{-}left$, 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 $vote\text{-}right$, processes behave symmetrically to $vote\text{-}left$. 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 $vote\text{-}val$ phase, they will decide the current value and halt. 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 $vote\text{-}right$ 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 so 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, by voting, 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. For simplicity, we give our termination time relative to failures ceasing—preventing the need to introduce a term, f, describing the total number of failures, into our termination bound.

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

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). □

To simplify the discussion of this proof, we introduce the following terminology which succinctly captures the state of the several important variables at the beginning of a $recurse$-phase round.

Definition 21 (Navigation Advice). For any process p_i and $recurse$-phase round r, the navigation advice for p_i at r is described by the binary 3-vector nav_i, where, for j, $1 \leq j \leq 3$, $nav[j]_i = 1$ if and only if, at the beginning of round r, $|msgs(j)_i| > 0$ or $CD(j)_i = \pm$.

Lemma 15. *For any $recurse$-phase round r, all non-crashed processes start r with the same navigation advice.*

Proof. By the definition of navigation advice and Algorithm 3, for any non-crashed process p_i, and integer

j, $1 \leq j \leq 3$, $nav[j]_i = 1$ if and only if p_i received a message or collision notification in round $r - 4 + i$. By Lemma 14, which states all processes receive something or all processes receive silence, if p_i sets $nav[j]_i \leftarrow 1$, then all other non-crashed processes do the same. □

Lemma 16. *For any round r, all non-crashed processes start r with $curr$ pointing to the same node in the binary search tree.*

Proof. The result follows from a simple inductive argument on the number of rounds. All processes are initialized with $curr$ pointing to the root of the tree. Processes update $curr$ during each $recurse$-phase round based only upon their navigation advice during that round. By Lemma 15, all process therefore update their $curr$ pointer in the same manner each time it is updated. □

Lemma 17 (Validity). *If some process decides value v, then v is the initial value of some process.*

Proof. A process decides in $recurse$-phase round r if and only if it receives a message or a collision notification during the $vote\text{-}val$-phase round $r - 3$. It it received a message, then, by the definition of an execution, some process sent a message. If it received a collision notification, then, by accuracy, some process sent a message that was lost. Either way, a process sent a message in $r - 3$, which, by line 7, occurs only if the value associated with $curr$ is the broadcaster's initial value. Because our decider decided the value associated with $curr$ (line 26), then it follows that it decided some process's initial value. □

Lemma 18 (Agreement). *No two processes decide different values.*

Proof. Nodes can decide only on line 23 of the $recurse$-phase. The decision to decide and the choice of value is entirely a function of their navigation advice and the $curr$ pointer at the start of this round. By Lemma 15, all non-crashed processes start each $recurse$-phase round with the same navigation advice, and by Lemma 16 all non-crashed processes start each $recurse$-phase round with the same $curr$ pointer. Therefore, if any process decides in r, then all non-crashed processes decide in r and decide the same value. □

Lemma 19 (Termination). *All correct processes decide and halt within $8 \lg |V|$ rounds after failures cease.*

Proof. By Lemmas 15 and 16, processes move through the binary tree together. In the worst-case, the last process to fail first brought all correct process to a leaf before crashing, and, now, all processes must ascend all the way back to the root before hearing another vote. This ascension requires up to $4\lg|V|$ rounds (the height of the tree is $\lg|V|$, and there are 4 rounds per movement in the tree). From here, it is at most another $4\lg|V|$ rounds for processes to arrive at a node in the tree corresponding to a correct process's value. $\square$

Proof (Theorem 3). Correctness follows from Lemmas 17, 18 and 19. $\square$

8 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 Section 8.1, by examining systems with collision detectors from the NoCD class. 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 Section 5)—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. This is formalized with Theorem 5 in Section 8.2.

Next, in Section 8.3.3, 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 $CST + \Theta(\log |V|)$.

We continue, in Section 8.3.4, 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 $CST + \lg\left(\frac{|V||I|}{n|V|+|I|}\right)\frac{1}{2}$. With Corollary 3 we simplify this expression to obtain the cleaner asymptotic result: $CST + \Omega(min\{\log |V|, \log \frac{|I|}{n}\})$. 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 Section 7, and the non-anonymous bound is (almost) matched by the variant of Algorithm 2 described in Section 7.3. Note: because we demonstrated in Section 7 a constant-round solution that uses a detector from the maj-$\Diamond\mathcal{AC}$ class, these result demonstrates 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 Section 8.4, 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 Section 8.5, 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 Section 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.

8.1 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 α, of the system $(A, \mathcal{A})$, and an execution β, of the system $(B, \mathcal{A})$, as follows:

1. Fix the executions so there is no message loss in either α or β.

2. In α, fix the contention manager, starting with round 1, to return $active$ only to the process described by $min(P_a)$. In β, 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 α, have all process start with initial value v, and in β 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 α and β.

We next construct an execution γ, 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 γ.

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

8.2 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 Section 5, 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. $\square$

8.3 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 Section 8.3.3, 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 Section 8.3.4, we show that for any non-anonymous ($\mathcal{E}$(half-$\mathcal{AC}$,LS),V,ECF)-consensus algorithm, there exists an execution that doesn't terminate before $CST + \Omega(min\{\log|V|, \log\frac{|I|}{n}\})$.

We start, however, with some general defintions and lemmas, presented in Section 8.3.1 and Section 8.3.2, which aid the discussion to follow.

8.3.1 Definitions

Definition 22 (Basic Broadcast Count Sequence). The Basic Broadcast Count Sequence of an execution α 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 α,
- 1 if and only if exactly one process broadcasts during round r of α,
- $2+$ if and only if two or more processes broadcast during round r of α.

We say two executions, α and β, 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 23 (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 $|V|$ initial states described by the set $\{init_i(v)|v \in V\}$.

Notice that any algorithm that solves consensus over a value set V is, by definition, a V-start algorithm. This holds because a consensus algorithm must have a unique initial state for each possible initial value. For simplicity of presentation, throughout this section, whenever we discuss a V-start algorithm, $\mathcal{A}$, that happens solves consensus for value set V, we assume for all $i \in I$ and $v \in V$, that initial state $init_i(v)$ 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 24 ($\alpha_P(v)$ (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 $init_i(v)$,
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, and
4. Fix $E_P.CD$ to satisfy completeness and accuracy (as it must by the definition of E_P).

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). 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.

8.3.2 Key Lemmas

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

Lemma 20. *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 $f : P \rightarrow P'$ 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. We prove this lemma by induction on the round number, r, showing that after r rounds, for every $i \in P$, the state, messages received, contention manager advice, and collison detector advice, for $\mathcal{A}(i)$ in round r of $\alpha_P(v)$, is the same as for $\mathcal{A}(f(i))$ in $\alpha_{P'}(v)$.

Basis ($r = 0$): Because $\mathcal{A}$ is anonymous, all processes start with the same initial state in both $\alpha_P(v)$ and $\alpha_{P'}(v)$.

Inductive Step ($r > 0$): Here we show, for every $i \in P$, that for $\mathcal{A}(i)$ in $\alpha_P(v)$ and $\mathcal{A}(f(i))$ in $\alpha_{P'}(v)$:

1. $\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ receive the same contention manager advice in round r

2. $\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ receive the same messages in round r.

3. $\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ receive the same collision detector advice in round r.

4. $\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ have the same state after r rounds.

(1) *$\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ receive the same contention manager advice in round r.*

If $i = min(P)$, then, by the definition of an alpha execution, $\mathcal{A}(i)$ will receive $active$ from its contention manager in round r of $\alpha_P(v)$. By definition of f, if $i = min(P)$, then $f(i) = min(P')$, meaning that $\mathcal{A}(f(i))$ will also receive $active$ during this round in its execution; keeping the contention manager advice the same for both. If, on the other hand, $i \neq min(P)$ then $\mathcal{A}(i)$ will receive the advice $passive$ from its contention manager in round r of $\alpha_P(v)$. By definition of f, if $i \neq min(P)$, then $f(i) \neq min(P')$, meaning that $\mathcal{A}(f(i))$ will also receive $passive$ during this round of its execution; once again keeping the advice the same for both.

(2) *$\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ receive the same messages in round r.*

The decision to broadcast (and what message to broadcast) in round r is a function of the state after round $r-1$ and the contention manager advice in r. By our inductive hypothesis, $\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ have the same state after $r-1$. By our above discussion (element (1)), they will also have the same contention manager advice. Therefore, a process $\mathcal{A}(i)$ in $\alpha_P(v)$ broadcasts in this round if and only if process $\mathcal{A}(f(i))$ broadcasts the same message in this round of $\alpha_{P'}(v)$. Thus, we know there are the same number of broadcasters and the same messages sent in both executions. This leaves three cases to consider regarding the common broadcast behavior in both executions in this round:

Case 1: If there is a single broadcaster in each execution, then, by the definition of alpha executions, every process receives the message; keeping element (2) the same for every process in both.

Case 2: If there are no broadcasters in either execution, then every process receives nothing; again, keeping element (2) the same in both.

Case 3: If there is more than one broadcaster in each execution, then, by the definition of alpha executions, if $\mathcal{A}(i)$ broadcasts m in $\alpha_P(v)$, then it receives m and no other message, and $\mathcal{A}$ also sends and receives only m in this round. Otherwise, both processes receive no messages. Once again, element (2) is the same in both.

(3) *$\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ receive the same collision detector advice in round r .*

The equivalence of the collision detector advice in r follows from the argument presented for element (2). That is, processes receive $\pm$ during this round only in case 3 of the broadcast behaviors discussed above. As described, this case occurs in both executions or neither.

(4) *$\mathcal{A}(i)$ and $\mathcal{A}(f(i))$ have the same state after r rounds.*

The state of a process after r rounds is a function of the state of the process after $r-1$ rounds, the messages received during r, the collision detector advice in r, and the contention manager advice in r. For every $i \in P$, we know, by our hypothesis, that the state of $\mathcal{A}(i)$ in $\alpha_P(v)$ after $r-1$ rounds is the same as the state of $\mathcal{A}(f(i))$ in $\alpha_{P'}(v)$ after $r-1$ rounds. We also know, by our discussion of elements (1) – (3), that the same equivalence holds for the messages, collision detector advice, and contention manager advice received by these two processes in r. □

Corollary 2 (Lemma 20). *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 $r \in I^+$, $\alpha_P(v)$ and $\alpha_{P'}(v)$ have the same basic broadcast count sequence through the first r rounds, where both α 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. Therefore, by Lemma 20, 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. □

The next two lemmas are counting arguments that bound the number of basic broadcast sequences that can exist among pairs of executions over short execution prefixes. Lemma 21 considers anonymous algorithms, and Lemma 22 considers non-anonymous algorithms.

Lemma 21. *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. At each round of each execution three behaviors can occur that are relevant to the basic broadcast count: 1) no process broadcasts; 2) one process broadcasts; and 3) more than one process broadcasts. Therefore, for any sequence of k rounds, there are 3^k basic broadcast count sequences. 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:

$$3^{(\frac{\lg |V|}{2} - 1)}$$

$$< 3^{(\frac{\lg |V|}{\lg 3} - \log_3 2)}$$

$$= 3^{(\log_3 |V|)} 3^{(-log_3 2)}$$

$$= \frac{|V|}{2}$$

$$< |V|$$

□

Lemma 22. *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 $1 < n \leq \lfloor \frac{|I|}{2} \rfloor$ and $|I| = nk$ for some integer $k > 1$. There exist two alpha executions, $\alpha_P(v)$ and $\alpha_{P'}(v')$, defined over $\mathcal{A}$, where $P, P' \subseteq I, |P| = |P'| = n, P \cap P' = \phi, v, v' \in V$, $v \neq v'$, 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 Π 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 Π, 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 that, as described in the previous lemma, for any sequence of k rounds, there are 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 $|\Pi| = \frac{|I|}{n}$, and substitute accordingly:

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

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:

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

□

We conclude this sub-section with a general indistinguishability lemma, involving alpha executions with similar basic broadcast count sequences.

Lemma 23. *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 $R \cap R' = \phi$. 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, γ of system $(E_{R\cup R'}, \mathcal{A})$, that satisfies eventual collision freedom, such that γ is indistinguishable from $\alpha_R(v)$ (resp. $\alpha_{R'}(v')$), through round k, with respect to processes described by indices in R (resp. R').

Proof. We start by constructing an execution γ that satisfies our desired indistinguishabilities and eventual collision freedom. We then show that this execution satisfies the constraints of its environment. Specifically, let γ 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 a single process described by an index in R broadcasts, then all processes described by indices in R receive its message. If a single process described by an index in R' broadcasts, then all processes described by indices in R' receive its message. Broadcasters always receive their own message (as required by the model). All other messages are lost. 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 the two processes described by $min(R)$ and $min(R')$. Starting with round $k + 1$, it returns $active$ only to the process described by $min(R)$.

We constructed γ such that for every $i \in R$, $\alpha_R(v)$ is indistinguishable from γ, with respect to i, through round k, and for every $j \in R'$, $\alpha_{R'}(v')$ is indistinguishable from γ, 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 γ definition. First, notice that no process described by an index in R ever receives a message from a process described by an index in R', and vice versa. Second, a process described by an index in R (resp. R') only receives a message m if a single process described by an index in R (resp. 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 γ satisfies eventual collision freedom as message loss stops at round $k+1$.

We must next show that γ 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 $E_{R \cup R'}.CD = MAXCD_{R \cup R'}(\text{half-}\mathcal{AC})$ we must ensure that neither half-completeness nor accuracy is ever violated in γ. 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 γ 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 γ, then, by assumption 3 of our γ 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 γ; 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 γ. 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, then 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 γ, then, by assumption 3 of our γ 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 γ. Therefore, by assumption 2 of our γ 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.

$\square$

8.3.3 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 α of the system $(E, \mathcal{A})$, where α satisfies eventual collision freedom, $CST(\alpha) = 1$, and some process in α doesn't 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 21 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 Lemma 23 to $\alpha_P(v)$, $\alpha_{P'}(v')$, and $k = \frac{\lg|V|}{2} - 1$. This produces an execution γ 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 γ 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, γ 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 γ. 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 γ—violating agreement. A contradiction. $\square$

Making the Bound Tight We match this lower bound with Algorithm 2, described in Section 7, which is an anonymous ($\mathcal{E}$(0-$\Diamond\mathcal{AC}$,WS),V,ECF)-consensus algorithm that guarantees termination by $CST + \Theta(\lg|V|)$.

8.3.4 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 performs 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 $1 < n \leq \lfloor\frac{|I|}{2}\rfloor$ 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 α of the system $(E, \mathcal{A})$, where α satisfies eventual collision freedom, $CST(\alpha) = 1$, and some process in α doesn't decide until after round* $\lg\left(\frac{|V||I|}{n|V|+|I|}\right)\frac{1}{2}$.

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 doesn't decide for the desired number of rounds, our theorem will be proved.

First, we apply Lemma 22, which provides two such executions, $\alpha_P(v)$ and $\alpha_{P'}(v')$, where $|P| = |P'| = n$, $P \cap P' = \phi$, and both have the same basic broadcast count sequence through the first $\lg\left(\frac{|V||I|}{n|V|+|I|}\right)\frac{1}{2}$ rounds. We can now apply Lemma 23 to $\alpha_P(v)$, $\alpha_{P'}(v')$, and $k = \lg\left(\frac{|V||I|}{n|V|+|I|}\right)\frac{1}{2}$, which, as before, provides an execution γ 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 γ 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 $k = \lg\left(\frac{|V||I|}{n|V|+|I|}\right)\frac{1}{2}$ By the definition of an ($\mathcal{E}$(half-$\mathcal{AC}$,LS),V,ECF)-consensus algorithm, γ 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 γ. 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 γ—violating agreement. A contradiction. $\square$

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 α of the system $(E, \mathcal{A})$, where α satisfies eventual collision freedom, $CST(\alpha) = 1$, and some process in α doesn't 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:

$$k = \lg\left(\frac{|V||I|}{n|V| + |I|}\right)\frac{1}{2}$$

$$= \lg\left(\frac{|V|nc|V|}{n|V| + nc|V|}\right)\frac{1}{2}$$

$$= \lg\left(\frac{nc|V|^2}{(c+1)n|V|}\right)\frac{1}{2}$$

$$= \lg\left(\frac{c}{c+1}|V|\right)\frac{1}{2}$$

$$= \left(\lg\left(\frac{c}{c+1}\right) + \lg\left(|V|\right)\right)\frac{1}{2}$$

$$= \Omega(\lg|V|)$$

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:

$$k = \lg\left(\frac{|V||I|}{n|V| + |I|}\right)\frac{1}{2}$$

$$= \lg\left(\frac{\frac{c|I|}{n}|I|}{n\frac{c|I|}{n} + |I|}\right)\frac{1}{2}$$

$$= \lg\left(\frac{c|I|^2}{n(c+1)|I|}\right)\frac{1}{2}$$

$$= \lg\left(\frac{c|I|}{(c+1)n}\right)\frac{1}{2}$$

$$= \left(\lg\left(\frac{c}{c+1}\right) + \lg\left(\frac{|I|}{n}\right)\right)\frac{1}{2}$$

$$= \Omega(\lg\frac{|I|}{n})$$

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 $\Omega(\lg\frac{|I|}{n})$; meaning any tie-breaking criteria for the min function is fine.

$\square$

Making the Bound Tight To match this bound, we can use the algorithm informally described in Section 7.3. 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 $\frac{1}{n}$ 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 α of the system $(E, \mathcal{A})$, where α satisfies eventual collision freedom, $CST(\alpha) = 1$, and some process in α doesn't 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 22, 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 $\lg |I|$.

8.4 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 sets V, where $|V| > 1$, there exists no ($\mathcal{E}(\Diamond\mathcal{AC}$,LS),$V$,NOCF)-consensus algorithm.*

Proof. Assume by contradiction that an ($\mathcal{E}(\Diamond\mathcal{AC}$,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 γ, 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 γ 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 γ. Assume all processes decide by round k. Let $x \in \{v, v'\}$ be the single value decided.

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

1. All processes in α are initialized with v, and all processes in β are initialized with v'.

2. Fix the environments so there is no message loss in either execution.

3. In α, fix the contention manager to return $active$ only to the process described by $min(P_a)$, in β, 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 γ. 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 α and β satisfy the constraints of their respective environments. The contention manager in both executions stabilizes to a single $active$ process (Starting at round 1 in α, and round $k+1$ in β). 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 γ is indistinguishable from α, with respect to i, through round k. And for all j in P_b, the execution γ is indistinguishable from β, with respect to j, through round k. As noted above, all processes decide $x \in \{v, v'\}$, by round k in γ. Therefore, all processes also decide x in their respective α or β execution. Assume, without loss of generality, that $x = v$. This implies processes decide v in β—violating uniform validity. A contradiction. □

8.5 Impossibility of 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 Section 7, 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 α of the system $(E, \mathcal{A})$, where some process in α doesn't 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 $\beta(v)$.

By a simple counting argument (i.e., as we saw in Lemma 21), 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 β 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 Lemma 23). If we compose these two β executions into a larger execution, γ, 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 β executions by round k, as they would then decide the same values in γ—violating agreement. $\square$

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 $\Theta(\lg |V|)$.[9]

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 argument 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 told each process everyone who is in 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.

[9]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.

9 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 (i.e., 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—detects all message loss—*does not* provide significant advantages over "pretty good" detection—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 their 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 can't do better than the detection of all messages being lost).

In the near future, 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 used simplified 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.

References

[1] IEEE 802.11. Wireless LAN MAC and physical layer specifications, June 1999.

[2] K. Alroubi, P. J. Wan, and 0. Frieder. Message-optimal connected dominating sets in mobile ad hoc networks. In *in Proceedingrs of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2002.

[3] J. Aspnes, F. Fich, and E. Ruppert. Relationships between broadcast and shared memory in reliable anonymous distributed systems. In *18th International Symposium on Distributed Computing*, pages 260–274, 2004.

[4] H. Attiya, D. Hay, and J. Welch. Optimal clock synchronization under energy constraints in wireless ad hoc networks. In *Proceedings of the ninth International conference on Principles of Distributed Systems*, 2005.

[5] M. Bahramgiri, M. T. Hajiaghayi, and V.S. Mirrokni. Fault-tolerant and three-dimensional distributed topology control algorithms in wireless multi-hop networks. In *Proceedings of the 11th IEEE International Conference on Computer Communications and Networks*, 2002.

[6] R. Bar-Yehuda, O. Goldreich, and A. Itai. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. *Distributed Computing*, 5:67–71, 1991.

[7] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992.

[8] R Bar-Yehuda, A Israeli, and A Itai. Multiple communication in multi-hop radio networks. *SIAM Journal on Computing*, 22(4):875–887, 1993.

[9] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. Macaw: A media access protocol for wireless lans. In *Proceedings of the ACM SIGCOMM '94 Conference on Communications Architectures, Protocols, and Applications*, 1994.

[10] Bluetooth. http://www.bluetooth.com.

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

[12] I. Chlamtac and S. Kutten. On broadcasting in radio networks - problem analysis and protocol design. *IEEE Transactions on Communications*, 33(12):1240–1246, 1985.

[13] G. Chockler, M. Demirbas, S. Gilbert, N. Lynch, C. Newport, and T. Nolte. Reconciling the theory and practice of (un)reliable wireless broadcast. *International Workshop on Assurance in Distributed Systems and Networks (ADSN)*, 2005. To appear.

[14] Gregory Chockler, Murat Demirbas, Seth Gilbert, and Calvin Newport. A middleware framework for robust applications in wireless ad hoc networks. In *Proceedings of the 43rd Allerton Conference on Communication, Control, and Computing*, 2005.

[15] Gregory Chockler, Murat Demirbas, Seth Gilbert, Calvin Newport, and Tina Nolte. 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, 2005.

[16] Andrea E. F. Clementi, Angelo Monti, and Riccardo Silvestri. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 709–718, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[17] M. Demirbas, A. Arora, T. Nolte, and N. Lynch. A hierarchy-based fault-local stabilizing algorithm for tracking in sensor network. In *Proceedings of the 8th International Conference on Principles of Distributed Systems*, Grenoble, France, dec 2004.

[18] J. Deng, P. K. Varshney, and Z. J. Haas. A new backoff algorithm for the IEEE 802.11 distributed coordination function. In *Communication Networks and Distributed Systems Modeling and Simulation (CNDS '04)*, 2004.

[19] Anders Dessmark and Andrzej Pelc. Tradeoffs between knowledge and time of communication in geometric radio networks. In *Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures*, pages 59–66, 2001.

[20] Shlomi Dolev, Seth Gilbert, Limor Lahiani, Nancy A. Lynch, and Tina Nolte. Timed virtual stationary automata for mobile networks. In *Proceedings of the 9th International Conference on Principles of Distributed Systems*, 2005.

[21] Shlomi Dolev, Seth Gilbert, Nancy A. Lynch, Elad Schiller, Alex A. Shvartsman, and Jennifer L. Welch. Virtual mobile nodes for mobile adhoc networks. In *Proceeding of the 18th International Conference on Distributed Computing*, 2004.

[22] Shlomi Dolev, Seth Gilbert, Elad Schiller, Alex A. Shvartsman, and Jennifer L. Welch. Autonomous virtual mobile nodes. In *Proceedings of the 3rd Workshop on Foundations of Mobile Computing*, 2005.

[23] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[24] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, 2001.

[25] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Symposium on Operating System Design and Implementation*, 2002.

[26] R. Fan, I. Chakraborty, and N. Lynch. Clock synchronization for wireless networks. In *Proceedings of the 8th International Conference on Principles of Distributed Systems*, Grenoble, France, dec 2004.

[27] Q. Fang, J. Gao, L. Guibas, V. de Silva, and L. Zhang. Glider: Gradient landmark-based distributed routing for sensor networks. In *Proceedings of the 24th Annual INFOCOM Conference*, 2005.

[28] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[29] E. Gafni and D. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE transactions on communications*, 1981.

[30] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. *UCLA Computer Science Technical Report UCLA/CSD-TR*, 2003.

[31] R. S. Gray, D. Kotz, C. Newport, N. Dubrovsky, A. Fiske, J. Liu, C. Masone, S. McGrath, and Y. Yuan. Outdoor experimental comparison of four ad hoc routing algorithms. In *Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 220–229, October 2004. Finalist for Best Paper award.

[32] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. *ASPLOS*, pages 93–104, 2000.

[33] L. Jia, R. Rajaruman, and R. Suel. An efficient distributed algorithm for constructing small dominating sets. In *Proceedings of the 2Oth ACM Symposium on Principles of Distributed Computing*, 2001.

[34] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless network. *Mobile Computing*, 5:153–181, 1996.

[35] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Mobile networking for smart dust. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, 1999.

[36] B. Karp and H. T. Kung. Greedy perimeter stateless routing for wireless networks. In *Proceedings of the sixth International conference on Mobile Computing and Networking*, 2000.

[37] C-Y. Koo. Broadcast in radio networks tolerating byzantine adversarial behavior. *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 275–282, 2004.

[38] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 78–82, 2004.

[39] D. Kowalski and A. Pelc. Time of deterministic broadcasting in radio networks with local knowledge. *SIAM Journal on Computing*, 33(4):870–891, 2004.

[40] Dariusz R. Kowalski. On selection problem in radio networks. In *Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 158–166, New York, NY, USA, 2005. ACM Press.

[41] E. Kranakis, D. Krizanc, and A. Pelc. Fault-tolerant broadcasting in radio networks. In *Proceedings of the 6th Annual European Symposium on Algorithms*, pages 283–294, 1998.

[42] E. Kuhn and K. Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings of 22nd ACM International Symposium on the Principles of Distributed Computing*, 2003.

[43] S. S. Kulkarni and U. Arumugam. Tdma service for sensor networks. *In Proceedings of the Third International Workshop on Assurance in Distributed Systems and Networks (ADSN)*, March 2004.

[44] M. Kumar. A consensus protocol for wireless sensor networks. Master's thesis, Wayne State University, 2003.

[45] H.T. Kung and D. Vlah. Efficient location tracking using sensor networks,. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, mar 2003.

[46] E. Kushelevitz and Y. Mansour. An omega(d log(n/d)) lower bound for broadcast in radio networks. In *Proceedings of the Twelth Annual ACM Symposium on Principles of Distributed Computing*, 1993.

[47] L. Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):18–25, 2001.

[48] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. *First USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.

[49] L. Li, J. Halpern, V. Bahl, M. Wang, and R. Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In *Proceedings of the Twentieth ACM Symposium on Principles of Distributed Computing*, 2001.

[50] C. Livadas and N. Lynch. A reliable broadcast scheme for sensor networks. Technical Report MIT-LCS-TR-915, MIT CSAIL, 2003.

[51] E. L. Lloyd. Broadcast scheduling for tdma in wireless multihop networks. pages 347–370, 2002.

[52] Jun Luo and Jean-Pierre Hubaux. Nascent: Network layer service for vicinity ad-hoc groups. In *Proceedings of the 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[53] N. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.

[54] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tinydb: An acqusitional query processing system for sensor networks. *ACM TODS*, 2005.

[55] D. Moore, J. Leonard D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *Proceedings of ACM Sensys'04*, 2004.

[56] T. Moscibroda and R. Wattenhofer. Efficient computation of maximal independent sets in unstructured multi-hop radio networks. In *Proceedings of the first IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, 2004.

[57] K. Nakano and S. Olariu. Uniform leader election protocols in radio networks. In *ICPP '02: Proceedings of the 2001 International Conference on Parallel Processing*, pages 240–250. IEEE Computer Society, 2001.

[58] V. Park and M. Corson. A highly adaptive distributed routing algorithm for mobile ad hoc networks. In *Proceedings of the sixteenth annual joint conference of the IEEE Computer and Communications Societies, Driving the Information Revolution*, 1997.

[59] C. Perkins and E. Royer. Ad hoc on-demand distance-vector routing. In *Proceedings of the 2nd workshop on Mobile Computing Systems and Applications*, 1999.

[60] K. S. J. Pister, J. M. Kahn, and B. E. Boser. Smart dust: Wireless networks of millimeter-scale sensor nodes. In *Highlight Article in 1999 Electronics Research Laboratory Research Summary*, 1999.

[61] J. Polastre and D. Culler. Versatile low power media access for wireless sensor networks. *The Second ACM Conference on Embedded Networked Sensor Systems (SENSYS)*, pages 95–107, 2004.

[62] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of the sixth International conference on Mobile Computing and Networking*, 2000.

[63] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, pages 304–313. Springer-Verlag, 1989.

[64] N. Santoro and P. Widmayer. Distributed function evaluation in presence of transmission faults. *Proc. Int. Symp. on Algorithms (SIGAL)*, pages 358–367, 1990.

[65] A. Savvides, C. Han, and M. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proceedings of the seventh annual International conference on Mobile Computing and Networking*, 2001.

[66] W. Su and I. F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 13(2):384–397, 2005.

[67] Robert Szewczyk, Joseph Polastre, Alan Mainwaring, and David Culler. Lessons from a sensor network expedition. *Lecture Notes in Computer Science*, 2920:307–322, 2004.

[68] T. van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. *The First ACM Conference on Embedded Networked Sensor Systems (SENSYS)*, pages 171–180, 2003.

[69] D. E. Willard. Log-logarithmic selection resolution protocols in a multiple access channel. *SIAM Journal of Computing*, 15(2):468–477, 1986.

[70] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of multihop routing in sensor networks. *The First ACM Conference on Embedded Networked Sensor Systems (SENSYS)*, pages 14–27, 2003.

[71] A. Woo, K. Whitehouse, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors*, pages 45–52, May 2005.

[72] W. Ye, J. Heidemann, and D. Estrin. 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.

[73] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. *The First ACM Conference on Embedded Networked Sensor Systems (SENSYS)*, pages 1–13, 2003.