The Power of Abstract MAC Layer: A Fault-tolerance Perspective

Qinzi Zhang
Boston University, USA

Lewis Tseng
UMass-Lowell, USA

Abstract

This paper studies the power of the "*abstract MAC layer*" model in a single-hop asynchronous network. The model captures primitive properties of modern wireless MAC protocols. In this model, Newport [PODC '14] proves that it is impossible to achieve deterministic consensus when nodes may crash. Subsequently, Newport and Robinson [DISC '18] present randomized consensus algorithms that terminate with $O(n^3 \log n)$ expected broadcasts in a system of n nodes. We are not aware of any results on other fault-tolerant distributed tasks in this model.

We first study the computability aspect of the abstract MAC layer. We present a *wait-free* algorithm that implements an atomic register. Furthermore, we show that in general, k-set consensus is impossible. Second, we aim to minimize storage complexity. Existing algorithms require $\Omega(n \log n)$ bits. We propose two *wait-free* approximate consensus and two *wait-free* randomized binary consensus algorithms that only need *constant* storage complexity (except for the phase index). One randomized algorithm terminates with $O(n \log n)$ expected broadcasts. All our algorithms are *anonymous*, meaning that at the algorithm level, nodes do not need to have a unique identifier.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Abstract MAC Layer, Computation Power, Consensus

Funding *Lewis Tseng*: This material is based upon work partially supported by the National Science Foundation under Grant CNS-2334021.

Acknowledgements The authors want to thank the anonymous reviewers for insightful comments. The work was partially done when Lewis Tseng was affiliated with Boston College and Clark University.

1 Introduction

This paper studies fault-tolerant primitives, with the focus on the aspect of wireless links in a single-hop asynchronous network. We adopt the "*abstract MAC layer*" model [33, 34, 25], which captures the basic properties guaranteed by existing wireless MAC (medium access control) layers such as TDMA (time-division multiple access) or CSMA (carrier-sense multiple access). Even though the abstraction does not model after any specific existing MAC protocol, the abstract MAC layer still serves an important goal – the separation of high-level algorithm design and low-level logic of handling the wireless medium and managing participating nodes. This separation helps identify principles that fills the gap between theory and practice in designing algorithms that can be readily deployed onto existing MAC protocols [33, 34]. In fact, recent works in the networking community propose approaches to implement the abstract MAC layer in more realistic network conditions, e.g., dynamic systems [41], dynamic SINR channels [40], and Rayleigh-Fading channels [39].

Consider an asynchronous network [10, 31] in which messages may suffer an arbitrary delay. Compared to conventional message-passing models [10, 31], the abstract MAC layer has two key characteristics (formal definition in Section 2):

- Nodes use a broadcast primitive which sends a message to all nodes that have *not* crashed yet, and triggers an acknowledgement upon the completion of the broadcast.
- Nodes do *not* have a priori information on other participating nodes.

The second characteristic is inspired by the observation that in a practical large-scale deployment, it is difficult to configure and manage all the connected devices so that they have the necessary information about other nodes. This assumption makes it difficult to port algorithms from conventional message-passing models to the abstract MAC layer, as these algorithms typically require the knowledge of other nodes and/or the size of the system. In fact, Newport and Robinson prove [33] that in message-passing models it is impossible to solve deterministic and randomized consensus, even if there is no fault, and nodes are assumed to have a constant-factor approximation of the network size.

In the abstract MAC layer model, Newport [34] proves that deterministic consensus is impossible when nodes may crash. Subsequently, Newport and Robinson [33] propose randomized consensus algorithms. We are not aware of any study on other fault-tolerant primitives. This paper answers the following two fundamental problems:

- Can we implement other fault-tolerant primitives?
- How do we minimize storage complexity when designing fault-tolerant primitives?

First Contribution. In Herlihy's wait-free hierarchy [23], the consensus number defines the "power" of a shared object (or primitive). An object has a consensus number c, if it is possible for $\leq c$ nodes to achieve consensus using the object and atomic registers, *and* it is not possible for $c+1$ nodes to do so. For example, an atomic register has consensus number 1, whereas consensus and compare-and-swap have consensus number ∞. The proof in [34] implies that any objects with consensus number ≥ 2 cannot be implemented in the abstract MAC layer. The natural next step is to understand whether objects with consensus number 1 can be implemented in the abstract MAC layer.

We first show that the abstract MAC layer is fundamentally related to the *store-collect object* [7, 8] by presenting a simple *wait-free* algorithm to implement the object in the abstract MAC layer. "Stacking" the constructions in [8] on top of our store-collect object solves many well-known computation tasks, e.g., registers, counters, atomic snapshot objects, and approximate, and randomized consensus. That is, we provide a wait-free approach to implement *some primitives with consensus number 1 in the abstract MAC layer.*

Next, we identify that *not all* primitives with consensus number 1 can be implemented. In particular, we prove that in a system of n nodes, $(n-1)$-set consensus is impossible to achieve in the abstract MAC layer model. This implies that other similar objects, like write-and-read-next objects [16], cannot be implemented as well.

Second Contribution. From a more practical perspective, we study *anonymous* and *storage-efficient* fault-tolerant primitives. First, *anonymous* algorithms do not assume unique node identity, and thus lower efforts in device configuration and deployment. Second, most wireless devices are made small; hence, naturally, they are not equipped with abundant storage capacity, and storage-efficiency is an important factor in practical deployment.

Table 1 compares the state-of-the-art algorithm NR18 [33] and our algorithms. All the randomized consensus algorithms work for binary inputs and all algorithms are *wait-free*. The time complexity is measured as the expected number of broadcasts needed for all fault-free nodes to output a value. For our algorithms, "values" can be implemented using an integer or a float data type in practice. The exact size of the values will become clear later.

Due to space limits, we focus only on our randomized algorithms and present our approximate agreement algorithms along with the full analysis in the technical report [42].

	consensus	storage complexity	note
NR18 [33]	randomized	$\Theta(n \log n)$ bits	$O(n^3 \log n)$ expected broadcasts
MAC-RBC	randomized	8 values, 4 Booleans	$O(2^n)$ expected broadcasts
MAC-RBC2	randomized	12 values, 5 Booleans	$O(n \log n)$ expected broadcasts
MAC-AC	approximate	4 values, 1 Boolean	convergence rate $1/2$
MAC-AC2	approximate	2 values, 1 Boolean	convergence rate $1 - 2^{-n}$

Table 1 Consensus in abstract MAC layer.
• The bottom four rows present our algorithms, whereas NR18 is the algorithm from DISC '18 [33].
• For approximate consensus, the convergence rate identifies the ratio that the range of fault-free nodes' states decreases after each asynchronous round. The smaller the ratio, the faster the convergence.

2 Preliminary

Related Work. We first discuss prior works in the abstract MAC layer model. The model is proposed by Kuhn, Lynch and Newport [25]. They present algorithms for multi-message broadcast, in which multiple messages may be sent at different times and locations in a multi-hop network communicating using the abstract MAC layer. Subsequent works [32, 24, 20] focus on *non-fault-tolerant* tasks, including leader election and MIS.

The closest works are by Newport [34] and Newport and Robinson [33]. Newport presents several impossibilities for achieving *deterministic* consensus when nodes may crash [34]. Newport and Robinson [33] present a randomized consensus algorithm that terminates after $O(n^3 \log n)$ broadcasts w.h.p. In their algorithm, nodes need to count the number of acknowledgements received from unique nodes and determine when to safely output a value. As a result, their algorithm requires storage space $\Theta(n \log n)$ bits and the knowledge of identities to keep track of unique messages. An accompanied (randomized) approach of assigning node identities with high probability is also proposed in [33]. Tseng and Sardina [36] present Byzantine consensus algorithms in the abstract MAC layer model, but they assume the knowledge of an upper bound on n and unique identities. Our consensus algorithms do not rely on identities; hence, fundamentally use different techniques.

Fault-tolerant consensus has been studied in various models that assume message-passing communication links [10, 31]. We consider a different communication model; hence, the techniques are quite different. An important distinction is that with asynchronous message-passing, it is impossible to implement a *wait-free* algorithm [23]. Furthermore, nodes require accurate information on the network size [33].

Di Luna et al. have a series of works on anonymous dynamic network [27, 28, 30, 29, 18, 17]. They do not assume any failures. A series of papers [21, 2, 13] study a related problem, called consensus with unknown participants (CUPs), where nodes are only allowed to communicate with other nodes whose identities have been provided by some external mechanism. Our consensus algorithms do not need unique node identities. Failure detectors are used in [1, 35, 12] to solve consensus with anonymous nodes. We do not assume a failure detector.

Model. We consider a static asynchronous system consisting of n nodes, i.e., we do not consider node churn. Each node is assumed to have a unique identifier; hence, the set of nodes is also denoted as the set of their identifiers, i.e., $\{1, \ldots, n\}$. For brevity, we often denote it by $[n]$. Our construction of store-collect requires identifiers due to its semantics. Our approximate and randomized algorithms are anonymous, and do *not* assume node identifiers. The identifiers are used only for analysis.

We consider the crash fault model in which any number of nodes may fail. A faulty node may crash and stop execution at any point of time. The adversary may control faulty behaviors and the message delays. Nodes that are not faulty are called *fault-free* nodes.

In a single-hop network with abstract MAC layer [25, 34, 33], nodes communicate using the `mac-broadcast` primitive, which eventually delivers the message to all the nodes that

have not crashed yet, including i itself. Moreover, at some point after the `mac-broadcast` has succeeded in delivering the message, the broadcaster receives an acknowledgement, representing that the `mac-broadcast` is complete. The broadcaster *cannot* infer any other information from the acknowledgment, not the system size n, nor the identities of other nodes. A crash may occur during the `mac-broadcast`, which leads to inconsistency. That is, if a broadcaster crashes, then some nodes might receive the message while others do not.

The key difference between message-passing and abstract MAC layer models is that in the message-passing model, sender requires explicit responses, which is the main reason that this model does not support wait-free algorithms and requires a priori information on participating nodes. In other words, abstract MAC layer allows us to design primitives with stronger properties due to the implication from the acknowledgement. This paper is the first to identify how to use its "power" to implement (some) primitives with consensus number 1.

For our store-collect and approximate consensus algorithms, the adversary may enforce an arbitrary schedule for message delivery and crashes. For our randomized algorithms, we assume the *message oblivious (or value oblivious)* adversary as in [33]. That is, the adversary does *not* know the private states of each node (process states or message content).

Algorithm Presentation and Message Processing. Each node is assumed to take steps sequentially (a single-thread process). Each line of the pseudo-code is executed *atomically*, except when calling `mac-broadcast`, since this primitive is handled by the underlying abstract MAC layer. Each algorithm also has a message handler that processes incoming messages. Our algorithms assume that (i) the message handler is triggered whenever the underlying layer receives a message and sends an interrupt; (ii) there is only one message handler thread, which processes messages one by one, i.e., the underlying layer has a queue of pending messages; and (iii) the handler has a priority over the execution of the main thread. The third assumption implies the following observation, which is important for ensuring the correctness of our algorithms:

▶ Remark 1. At the point of time when the main thread starts executing a line of the pseudo-code, there is *no* pending message to be processed by the handler.[1]

It is possible that *during* the execution of a line in the main thread, the underlying layer sends an interrupt. The message handler will process these messages *after* the completion of that particular line of code due to the assumption of atomic execution. The only exception is the call to `mac-broadcast`. Messages can still be received and processed when a broadcaster is waiting for the acknowledgement from the abstract MAC layer.

3 Abstract MAC Layer: Computability

From the perspective of computability, asynchronous point-to-point message-passing model is fundamentally related to linearizable shared objects [22]. However, it was pointed out that register simulation in conventional point-to-point models like ABD [6] is "thwarted" [33]. In other words, this observation indicated that the computability of the abstract MAC layer remained an *open* problem. We fill the gap by presenting a framework of implementing some linearizable shared objects with consensus number 1.

[1] This assumption is not needed in prior works [34, 33], because their algorithm design is fundamentally different from ours. On a high-level, their algorithms proceed in an atomic block, whereas our algorithms have shared variables between the main thread and multiple message handlers. The assumption captures the subtle interaction between them.

Our Insight. In the point-to-point model, ABD requires the communication among a quorum, because "information kept by a quorum" ensures that the information is *durable* and *timely* in quorum-based fault-tolerant designs. Durable information means that the information is not lost, even after node crashes. Timely information means that the information satisfies the real-time constraint, i.e., after the communication with a quorum is completed, others can learn the information by contacting any quorum of nodes.

Our important observation is that the `mac-broadcast` achieves *both* goals upon learning the acknowledgement. That is, after the broadcaster learns that the broadcast is completed, it can infer that the message is both durable and timely.

Durability and timeliness are indeed sufficient for ensuring "regularity," which can then be used to implement linearizable shared objects (as will be seen in Theorem 3). We next present a construction of store-collect objects.

Store-Collect Object. A store-collect object [7, 8] provides two operations (or interfaces) at node i: (i) $\textsc{Store}_i(v)$: store value v into the object; and (ii) $\textsc{Collect}_i()$: collect the set of "most recent" values (of the object) from each node. The returned value is a *view* V – a set of (v_j, j) tuples where j is a node identity and v_j is its most recent stored value. For each j, there is at most one tuple of $(*, j)$ in V. With a slight abuse of notation, $V(j) = v_j$ if $(v_j, j) \in V$; otherwise, $V(j) = \perp$.

To formally define store-collect, we first discuss a useful notion. A *history* is an execution of the store-collect object, which can be represented using a partially ordered set $(H, <_H)$. Here, H is the set of invocation (inv) and response ($resp$) events of the Store and Collect operations, and $<_H$ is an irreflexive transitive relation that captures the real-time "occur-before" relation of events in H. Formally, for any two events e and f, we say $e <_H f$ if e occurs before f in the execution. For two operations op_1 and op_2, we say that op_1 precedes op_2 if $resp(op_1) <_H inv(op_2)$.

Every value in Store is assumed to be unique (this can be achieved using sequence numbers and node identifiers). A node can have at most one pending operation. Given views V_1 and V_2 returned by two Collect operations, we denote $V_1 \preceq V_2$, if for every $(v_1, j) \in V_1$, there exists a v_2 such that (i) $(v_2, j) \in V_2$; and (ii) either $v_1 = v_2$ or the invocation of $\textsc{Store}_j(v_2)$ occurs after the response of $\textsc{Store}_j(v_1)$. That is, from the perspective of node j, v_2 is more recent than v_1. We then say that a history σ satisfies *regularity* if:

- For each $\textsc{Collect}()$ $c \in \sigma$ that returns V and for each node j, (i) if $V(j) = \perp$, then no Store by j precedes c in σ; and (ii) if $V(j) = v$, then $\textsc{Store}_j(v)$'s invocation precedes c's response, and there does not exist $\textsc{Store}_j(v')$ such that $v' \neq v$, and $\textsc{Store}_j(v')$'s response occurs after $\textsc{Store}_j(v)$'s response and before c's invocation.
- Consider any pair of two Collect's in history σ, c_1 and c_2, which return views V_1 and V_2, respectively. If c_1 precedes c_2, then $V_1 \preceq V_2$.

▶ **Definition 2** (Store-Collect). *An algorithm correctly implements the store-collect object if every execution of the algorithm results into a history that satisfies regularity.*

Our Wait-free Construction of Store-Collect. To achieve regularity, each stored value has to be durable and timely. If a value is not durable, then the first condition for regularity may be violated. If a value is not timely, then the second condition may be violated. Moreover, any current information needs to be known by subsequent Collect's, potentially at other nodes. These observations together with the aforementioned insight of the `mac-broadcast` primitive give us a surprisingly straightforward construction. Our algorithm MAC-SC is presented in Algorithm 1.

Each node i keeps a local variable $view_i$, which is a set of values – one value for each node (that is known to node i so far). With a slight abuse of terminology, we use $C = A \cup B$ to denote the merge operation of two views A and B, which returns a view C that contains the newer value from each node. Since each node can have at most one pending operation and each value is unique, the notion of "newer" is well-defined. For brevity, the sequence number is omitted in the notation.

For $\text{STORE}_i(v)$, node i first adds the value v to form a new view, and uses `mac-broadcast` to inform others about the new view. Because this information is both durable and timely upon the completion of the broadcast, regularity is satisfied. Since the broadcast delivers the message to the broadcaster as well, (v, i) is added to $view_i$ at line 8. For $\text{COLLECT}_i()$, it is similar except that the broadcast view is the current local view at node i. Upon receiving a new view (from the incoming message with the STORE tag), i simply merges the new view and its local view $view_i$.

Algorithm 1 MAC-SC: Steps at each node i

Local Variable: /* It can be accessed by any thread at i. */
$view_i$ ▷`view, initialized to` $\emptyset$

When $Store_i(v)$ **is invoked**:
1: $currentView_i \leftarrow view_i \cup \{(v, i)\}$
2: `mac-broadcast`(`STORE`, $currentView_i$)
3: **return** ACK ▷STORE `is completed`

When $Collect_i()$ **is invoked**:
4: $currentView \leftarrow view_i$
5: `mac-broadcast`(`STORE`, $currentView$)
6: **return** $currentView$

// Background message handler
7: **Upon** receive(`STORE`, $view$) **do**
8: $view_i \leftarrow view_i \cup view$

▶ **Theorem 3.** *MAC-SC implements the store-collect object.*

Proof Sketch. ***Property I***: Consider a $\text{COLLECT}_i()$ operation c that returns V. For each node j, consider two cases:

- $V(j) = \perp$: this means that node i has not received any message from j's `mac-broadcast`. This implies that either `mac-broadcast` by j is not yet completed, or node j has not invoked any `mac-broadcast`. In both cases, no STORE by j precedes c.
- $V(j) = v$: by construction, v is in $V(j)$ because $\text{STORE}_j(v)$ is invoked before c completes. Next we show that there is no other STORE by node j that completes between two events: the response event of $\text{STORE}_j(v)$ and the invocation of c. Assume by way of contradiction that $\text{STORE}_j(v')$ completes between these two events. Now observe that: (i) By definition, $\text{STORE}_j(v)$ precedes $\text{STORE}_j(v')$, so v' is more recent than v from the perspective of node j; and (ii) By the assumption of the abstract MAC layer, when $\text{STORE}_j(v')$ completes, node i must have received the value v'. These two observations together imply that node i will add v' into its view at line 8 before the invocation of c. Consequently, $V(j) = v'$ in the view returned by c, a contradiction.

Property II: Suppose c_1 and c_2 are two COLLECT's such that c_1 returns view V_1, c_2 returns view V_2, and c_1 precedes c_2. By assumption, when `mac-broadcast` completes, all the nodes that have not crashed yet have received the broadcast message. Therefore, $V_1 \preceq V_2$. ◀

From Store-Collect to Linearizable Objects. Constructions of several linearizable shared objects over store-collect are presented in [8]. These constructions only use STORE and COLLECT without relying on other assumptions; hence, can be directly applied on top of MAC-SC. More concretely, Attiya et al. [8] consider a dynamic message-passing system,

where nodes continually enter and leave. Similar to our model, their constructions do not assume any information on other participating nodes. All the necessary coordination is through the store-collect object.

This stacked approach sheds light on the computability of the abstract MAC layer. We can use the approach in [8] to implement an atomic register on top of MAC-SC in abstract MAC layer in a *wait-free* manner. Consequently, MAC-SC opens the door for the implementation of many shared objects with consensus number 1. In particular, any implementation on atomic register that does not require a priori information on participating nodes can be immediately applied, e.g., linearizable abort flags, sets, and max registers [26, 8].

Interestingly, despite the strong guarantee, *not all* objects with consensus number 1 can be implemented in the abstract MAC layer. In particular, we prove that $(n-1)$-set consensus is impossible to achieve in our technical report [42]. Our proof follows the structure of the counting-based argument developed by Attiya and Paz (for the shared memory model) [9].

4 Anonymous Storage-Efficient Randomized Binary Consensus

While general, the stacked approach comes with two drawbacks in practice – assumption of unique identities and high storage complexity. Stacking prior shared-memory algorithms on top of MAC-SC requires $\Omega(n \log n)$ due to the usage of store-collect. Prior message-passing algorithms (e.g., [19, 38, 11]) usually require the assumption of unique identities.

This section considers anonymous storage-efficient randomized binary consensus. Recall that deterministic consensus is impossible under our assumptions [34], so the randomized version is the best we can achieve. As shown in Table 1, the state-of-the-art algorithm NR18 [33] requires $O(n^3 \log n)$ time complexity w.h.p. and $\Theta(n \log n)$ storage complexity. We present two *anonymous wait-free* algorithms using only constant storage complexity.

Our Techniques. Our algorithms are inspired by Aspnes's framework [3] of alternating adopt-commit objects and conciliator objects. The framework is designed for the shared memory model, requiring *both* node identity and the knowledge of n. Moreover, it requires $O(\log n)$ atomic multi-writer registers in expectation.

To address these limitations, we have two key technical contributions. First, we replace atomic multi-writer registers by `mac-broadcast`, while using only constant storage complexity. Second, we integrate the "doubling technique," for estimating the system size n, with the framework and present an accompanied analysis to bound the expected round complexity.

More concretely, we combine the technique from [37] and Aspnes's framework to avoid using new objects in a new phase. More precisely, we borrow the "jump" technique from [37], which allows nodes to skip phases (and related messages), to reduce storage complexity. This comes with two technical challenges. First, our proofs are quite different from the one in [37], because nodes progress in a different dynamic due to the characteristics of the abstract MAC layer. In particular, we need to carefully analyze which broadcast message has been processed to ensure that the nodes are in the right phase in our proof. This is also where we need to rely on Remark 1, which is usually not needed in the proofs for point-to-point message-passing models. Second, compared to [3], our proofs are more subtle in the sense that we need to make sure that concurrent broadcast events and "jumps" do not affect the probability analysis. The proof in [3] mainly relies on the atomicity of the underlying shared memory, whereas our proofs need to carefully analyze the timing of broadcast events. (Recall that we choose not to use MAC-SC, since it requires nodes to have unique identities.)

Prior solutions rely on the knowledge of network size n [14, 15, 3] or an estimation of n [33] to improve time complexity. For anonymous storage-efficient algorithms, nodes do not

Algorithm 2 MAC-AdoptCommit Algorithm: Steps at each node i with input v_i

Local Variables: /* These can be accessed by any thread at i. */
$seen_i[0]$ ▷`Boolean, initialized to` $false$
$seen_i[1]$ ▷`Boolean, initialized to` $false$
$proposal_i$ ▷`value, initialized to` $\perp$

1: `mac-broadcast(VALUE,` v_i`)`
2: **if** $proposal_i \neq \perp$ **then**
3: $v_i \leftarrow proposal_i$
4: `mac-broadcast(PROPOSAL,` v_i`)`
5: **if** $seen_i[-v_i] =$ false **then**
6: **return** $(commit, v_i)$
7: **else**
8: **return** $(adopt, v_i)$

// Background message handler
9: **Upon** receive(`VALUE`, v) **do**
10: $seen_i[v] \leftarrow true$
11: **Upon** receive(`PROPOSAL`, v) **do**
12: $proposal_i \leftarrow v$

know n, and there is no unique node identity. The solution for estimating the network size in the abstract MAC layer in [33] only works correctly with a large n (i.e., with high probability). We integrate a "doubling technique" to *locally* estimate n which does not require any message exchange. For our second randomized binary consensus algorithm MAC-RBC2, nodes double the estimated system size n' every c phases for some constant c, if they have not terminated yet. We identify a proper value of c so that n' is within a constant factor of n, and nodes achieve agreement using $O(n \log n)$ broadcasts on expectation.

Randomized Binary Consensus and Adopt-Commit.

▶ **Definition 4** (Randomized Binary Consensus). *A correct randomized binary consensus algorithm satisfies: (i)* ***Probabilistic Termination****: Each fault-free node decides an output value with probability 1 in the limit; (ii)* ***Validity****: Each output is some input value; and (iii)* ***Agreement****: The outputs are identical.*

▶ **Definition 5** (Adopt-Commit Object). *A correct adopt-commit algorithm satisfies: (i)* ***Termination****: Each fault-free node outputs either $(commit, v)$ or $(adopt, v)$ within a finite amount of time; (ii)* ***Validity****: The v in the output tuple must be an input value; (iii)* ***Coherence****: If a node outputs $(commit, v)$, then any output is either $(adopt, v)$ or $(commit, v)$; and (iv)* ***Convergence****: If all inputs are v, then all fault-free nodes output $(commit, v)$.*

4.1 Algorithm MAC-AdoptCommit

We present MAC-AdoptCommit, which implements a *wait-free* adopt-commit object for binary inputs in the abstract MAC layer model. The pseudo-code is presented in Algorithm 2, and the algorithm is inspired by the construction in shared memory [4]. Following the convention, we will use $-v$ to denote the opposite (or complement) value of value v.

Each node i has two Booleans, $seen_i[0]$ and $seen_i[1]$, and a value $proposal_i$. The former variables are initialized to $false$, and used to denote whether a node i has seen input value 0 and 1, respectively. The last variable $proposal_i$ is initialized to $\perp$, and used to record the "proposed" output from some node. The algorithm has two types of messages:

- A `VALUE` type message (`VALUE`, v_i) that is used to exchange input values.
- A `PROPOSAL` type message (`PROPOSAL`, v_i) that is to announce a proposed value.

Upon receiving the message (`VALUE`, v), node i updates $seen_i[v]$ to $true$ (line 11), denoting that it has seen the value v. Upon receiving the message (`PROPOSAL`, v), i updates $proposal_i$ to v (line 13), denoting that it has recorded the proposed value, by either itself or another node. Due to concurrency and asynchrony, it is possible that there are multiple proposal messages; thus, node i may overwrite existing value in $proposal_i$ with an opposite value.

Node i first broadcasts input v_i. After `mac-broadcast` completes (line 1), i checks whether it has received any `PROPOSAL` message. If so, it updates its state v_i to the value (line 5). Otherwise, it becomes a proposer and broadcast `PROPOSAL` message with its own input v_i (line 3). After Line 5, the state v_i could be i's original input, or a state copied from the proposed value (from another node). Finally, if node i has not observed any `VALUE` message with the opposite state ($-v_i$), then it outputs $(commit, v_i)$; otherwise, it outputs $(adopt, v_i)$.

Correctness. Validity, termination and convergence are obvious. To see how MAC-AdoptCommit achieves coherence, first observe that it is impossible for some node to output $(commit, v)$, and the others to output $(commit, -v)$. It is due to the property of `mac-broadcast`: if some node outputs $(commit, v)$, then every node must observe $seen[v] = true$ when executing line 6. Second, if a node outputs $(commit, v)$, then it must be the case that there has already been a proposer that has broadcast both message $(\texttt{VALUE}, v)$ and message $(\texttt{PROPOSAL}, v)$. Therefore, it is impossible for a node to output $(adopt, -v)$. For completeness, we present the proof of correctness in Appendix A.

4.2 Algorithm MAC-RBC

Algorithm 3 MAC-RBC Algorithm: Steps at each node i with input x_i

Local Variables: /* These variables can be accessed and modified by any thread at node i. */

$seen_i[0]$ ▷`(Boolean, phase), initialized to` $(false, 0)$
$seen_i[1]$ ▷`(Boolean, phase), initialized to` $(false, 0)$
$seen_i^2[0]$ ▷`(Boolean, phase), initialized to` $(false, 0)$
$seen_i^2[1]$ ▷`(Boolean, phase), initialized to` $(false, 0)$
v_i ▷`state, initialized to` x_i`, the input at node` i
p_i ▷`phase, initialized to` 0
$proposal_i$ ▷`(value, phase), initialized to` $(\perp, 0)$

1: **while** true **do**
2: $p_{old} \leftarrow p_i$
3: `mac-broadcast`$(\texttt{VALUE}, v_i, p_i)$
4: **if** $proposal_i.phase \geq p_i$ **then**
5: $(v_i, p_i) \leftarrow proposal_i$
6: `mac-broadcast`$(\texttt{PROPOSAL}, v_i, p_i)$
7: **if** $p_{old} \neq p_i$ **then**
8: **go to** line 2 in p_i ▷`"Jump" to` p_i
9: **else if** $seen_i[-v_i].phase < p_i$ **then**
10: **output** v_i
11: **else**
12: `mac-broadcast`$(\texttt{VALUE}^2, v_i, p_i)$
13: **if** $seen_i^2[-v_i].phase > p_i$ **then**
14: $(v_i, p_i) \leftarrow seen_i^2[-v_i]$
15: **go to** line 2 in p_i ▷`"Jump" to` p_i
16: **else if** $seen_i^2[-v_i] = (true, p_i)$ **then**
17: $v_i \leftarrow$FlipLocalCoin()
18: $p_i \leftarrow p_i + 1$ ▷`"Move" to` p_i

// Background message handler
19: **Upon** receive$(\texttt{VALUE}, v, p)$ **do**
20: **if** $p \geq p_i$ **then**
21: $seen_i[v] \leftarrow (true, p)$

22: **Upon** receive$(\texttt{VALUE}^2, v, p)$ **do**
23: **if** $p \geq p_i$ **then**
24: $seen_i^2[v] \leftarrow (true, p)$

25: **Upon** receive$(\texttt{PROPOSAL}, v, p)$ **do**
26: **if** $p \geq p_i$ **then**
27: $proposal_i \leftarrow (v, p)$

We present MAC-RBC in Algorithm 3. The algorithm uses a sequence of adopt-commit and conciliator objects. A conciliator object helps nodes to reach the same state, and an adopt-commit is used to determine whether it is safe to output a value, and choose a value for the next phase when one cannot "commit" to an output. We adapt MAC-AdoptCommit to store phase index, which allows nodes to jump to a higher phase. Effectively, any adopt-commit object with a phase $< p$ can be interpreted as having $\perp$ in phase p. This also allows us to "reuse" the object. For the conciliator object, we use Ben-Or's local coin [11], which achieves expected exponential time complexity.

In Algorithm 3, line 3 to line 10 effectively implement a reusable adopt-commit object using `VALUE` and `PROPOSAL` messages. Line 12 to line 17 implement a conciliator object using the $\texttt{VALUE}^2$ message. The *seen* variables store both a Boolean and a phase index. Nodes only update these variables when receiving a corresponding message from the same or a higher phase. A node i flips a local coin to decide the state for the next phase at line 17 *only if* it can safely infer that both 0 and 1 are some node's state at the beginning of the phase, i.e., it flips a coin when it has *not* seen a $\texttt{VALUE}^2$ message from a higher phase (line 13), and it has observed a $\texttt{VALUE}^2$ message with value $-v_i$ from the same phase (line 16).

Correctness Proof. It is straightforward to see that MAC-RBC satisfies validity, since the state is either one's input or a value learned from received messages (which must be an input value) and there is no Byzantine fault. We then prove the agreement property.

▶ **Lemma 6.** *Suppose node i is the first node that makes an output and it outputs v in phase p, then all the other nodes either output v in phase p or phase $p+1$.*

Proof. Suppose node i outputs v in phase p at time T_3. Then it must have $seen_i[-v].phase < p$. Assume this holds true at time T_2. Furthermore, assume line 6 was executed at time T_1 by node i at time T_1 such that $T_1 < T_2 < T_3$.

We first make the observation, namely *Obs1*, no node with $-v$ in phase $p' \geq p$ has completed line 3 at time $\leq T_2$. Suppose node j has state $-v$ in some phase $p' \geq p$. By assumption (in Section 2), before node i starts to execute line 9 at time T_2, its message handler has processed all the messages received by the abstract MAC layer. Therefore, the fact that $seen_i[-v].phase < p_i$ at time T_2 implies that node i has *not* receive any message of the form $(\texttt{VALUE}, -v, p')$ at time T_2. Consequently, node j has not completed `mac-broadcast`(`VALUE`, $-v, p'$) (line 1) at time T_2.

Consider the time T when the first `mac-broadcast`(`VALUE`, $-v, p$) is completed (if there is any). At time T, there are two cases for node k that has not crashed yet:

- Node k has *not* moved beyond phase p:
  k must have already received (`PROPOSAL`, v) at some earlier time than T, because (i) *Obs1* implies that $T > T_2$; and (ii) by time T, node i has already completed line 6 (which occurred at time T_1). Consider two scenarios: (s1) k executes line 4 after receiving (`PROPOSAL`, v): k sets $proposal_k$ to value v before executing line 6 (potentially at some later point than T); and (s2) k executes line 4 before receiving (`PROPOSAL`, v): in this case: k's input at phase p must be v; otherwise, T cannot be the first `mac-broadcast`(`VALUE`, $-v, p'$) that is completed. (Observe that by assumption of this case, k executes line 4 before node i completes its line 6 at time T_1.)
- Node k has moved beyond phase p:
  By assumption, time T is the time that the first `mac-broadcast`(`VALUE`, $-v, p$) is completed. Thus, it is impossible for node k to have set (v_k, p_k) to $(-v, p')$ for some $p' \geq p$.

In both cases, right before executing line 6, node k can only `mac-broadcast`(`PROPOSAL`, v, p'), for $p' \geq p$, i.e., no `mac-broadcast`(`PROPOSAL`, $-v, p'$) is possible. Consequently, the lemma then follows by a simple induction on the order of nodes moving to phase $p+1$. ◀

Since we assume a message oblivious adversary, the termination and exponential time complexity follow the standard argument of using local coins [11]. In particular, we have the following Theorem, which implies that MAC-RBC requires, on expectation, an exponential number of broadcasts. The proof is deferred to Appendix B.

▶ **Theorem 7.** *For any $\delta \in (0,1)$, let $p = \lceil 2^{n-1} \ln(1/\delta) \rceil$. Then with probability at least $1-\delta$, MAC-RBC terminates within p phases. (In other words, all nodes have phase $\leq p$.)*

Algorithm 4 MAC-FirstMover Algorithm: Steps at each node i with input v_i

Local Variables: /* These variables can be accessed by any thread at node i. */
$coin_i$ ▷value, initialized to $\perp$
Input: n' ▷estimated system size, given as an input to MAC-FirstMover

1: $k \leftarrow 0$
2: **while** $coin_i = \perp$ **do**
3: **if** a local random number $< \frac{2^k}{2n'}$ **then**
4: mac-broadcast(COIN, v_i)
5: **else**
6: mac-broadcast(DUMMY)
7: $k \leftarrow k + 1$
8: mac-broadcast(COIN, $coin_i$)
9: **return** $coin_i$

// Background message handler
10: **Upon** receive(COIN, v) **do**
11: **if** $coin_i = \perp$ **then**
12: $coin_i \leftarrow v$
13: **Upon** receive(DUMMY) **do**
14: **do** nothing

4.3 MAC-RBC2: Improving Time Complexity

There are several solutions for an efficient conciliator object, such as a shared coin [5] and the "first-mover-win" strategy [14, 15, 3]. The first-mover-win strategy was developed for a single multi-writer register in shared memory such that agreement is achieved when only one winning node (the first mover) successfully writes to the register. If there are concurrent operations, then agreement might be violated. On a high-level, this strategy translates to the "first-broadcaster-win" design in the abstract MAC layer. One challenge in our analysis is the lack of the atomicity of the register. We need to ensure that even in the presence of concurrent broadcast and failure events, there is still a constant probability for achieving agreement, after nodes have a "good enough" estimated system size n'.

Conciliator and Integration. Our conciliator object is presented in Algorithm 4, which is inspired by the ImpatientFirstMover strategy [3]. The key difference from [3] is that MAC-FirstMover uses an estimated size n', instead of the actual network size n (as in [3]), which makes the analysis more complicated, as our analysis depends on both n and n'. Algorithm 4 presents a standalone conciliator implementation. We will later describe how to integrate it with Algorithm 3 by adding the field of phase index and extra message handlers.

In our design, each node proceeds in rounds and increases the probability of revealing their coin-flip after each round k, if it has not learned any coin flip at line 2. To prevent the message adversary from scheduling concurrent messages with conflicting values, nodes have two types of messages: COIN and DUMMY. The first message is used to reveal node's input v_i, whereas the second is used as a "decoy" that has no real effect. At line 3, node i draws a local random number between $[0, 1)$ to decide which message to broadcast. Since the adversary is oblivious, it cannot choose its scheduling based on the message type.

MAC-RBC2 can be obtained by integrating Alg. 4 (MAC-FirstMover) with Alg. 3 (MAC-RBC) with the changes below. The complete algorithm is presented in Appendix C.

- FlipLocalCoin() is replaced by MAC-FirstMover($2^{\lfloor \frac{p_i}{c} \rfloor} n_0$), where c is a constant to be defined later and n_0 is a constant that denotes the initial guess of the system size. All nodes have an identical information of c and n_0 in advance. Therefore, nodes in the same phase call MAC-FirstMover with the same estimated system size n'. Recall that p_i is the phase index local at node i. Hence, effectively in our design, each node i is doubling the estimated size n' every c phases.
- To save space, $coin_i$ consists of two fields $(value, phase)$, and is used in a fashion similar to how $proposal_i$ is used in MAC-RBC. That is, if $coin_i$ has a phase field lower than the current phase p_i, then the value field is treated as $\perp$.

- The messages by node i are tagged with its current phase p_i. That is, the two messages in Algorithm 4 have the following form: $(\texttt{COIN}, v, p_i)$ and $(\texttt{DUMMY}, p_i)$.
- MAC-RBC2 needs to have two extra message handlers to process `DUMMY` and `COIN` messages. The `COIN` message handler only considers messages with phase $\leq p_i$.
- In MAC-RBC2, nodes jump to a higher phase upon receiving a coin broadcast. More precisely, if a node i receives a coin broadcast m from a phase $p > p_i$, then i updates v_i to the value in m and jumps to phase $p + 1$.

Correctness and Time Complexity. Correctness follows from the prior correctness proof, as MAC-FirstMover is a valid conciliator object that returns only 0 or 1. To analyze time complexity, we start with several useful notions.

▶ **Definition 8** (Active Nodes). *We say a node is an active node in phase p if it ever executes MAC-FirstMover in phase p. Let $\mathcal{A}_p$ denote the set of all active nodes in phase p.*

Due to asynchrony, different nodes might execute MAC-FirstMover in phase p at different times. Moreover, nodes may "jump" to a higher phase in our design. Consequently, not all nodes would execute MAC-FirstMover in phase p for every p.

▶ **Definition 9** (Broadcast). *We distinguish different types of broadcasts, which will later be useful for our probability analysis:*

- *A broadcast is a phase-p broadcast if it is tagged with phase p. By definition, only active nodes in phase p (i.e., nodes in $\mathcal{A}_p$) make phase-p broadcasts.*
- *A broadcast made in MAC-FirstMover (Algorithm 4) is a coin broadcast if its message has the `COIN` tag; otherwise, it is a dummy broadcast.*
- *A broadcast is an original broadcast if it is made in the while loop (Line 4 and Line 6 in Algorithm 4). It is a follow-up broadcast if it is made after $coin_i$ becomes non-empty (line 8 of Algorithm 4). By design, a follow-up broadcast must be a coin broadcast.*
- *Consider an original broadcast $m = (\texttt{COIN}, v, p)$ by node i. The broadcast is said to be successful in phase p if there exists a node j that completes a follow-up broadcast with $coin_j = v$ in phase p, i.e., node j receives the acknowledgement for its broadcast at line 8 of Algorithm 4. Note that i may not equal to j, and both i and j might be faulty (potentially crash at a future point of time).*

Recall that we define a broadcast to be "*completed*" if a node making the broadcast receives the acknowledge from the abstract MAC layer. This notion should not be confused with the notion of "successful." In particular, we have (i) a broadcast might be completed, but not successful – this is possible if there are multiple original coin broadcasts with different v; (ii) a broadcast might be successful, but not completed – this is possible if a node j receives an original coin broadcast by a faulty node and node j completes the follow-up broadcast.

We will apply the following important observation in our proofs. The observation directly follows from our definition of different broadcasts.

▶ Remark 10. If there is a completed original coin broadcast in phase p, then there must be at least one successful original coin broadcast in phase p.

▶ **Definition 11.** *A node completes MAC-FirstMover of phase p if it receives a coin broadcast of the form $(\texttt{COIN}, *, p')$ with $p' \geq p$.*[2] *Let $\mathcal{C}_p$ denote the set of all nodes that complete MAC-FirstMover of phase p.*

[2] This coin broadcast can be an original or a follow-up coin broadcast.

By definition, a node *not* in $\mathcal{A}_p$ can still complete MAC-FirstMover of phase p, if it receives a coin broadcast from a higher phase.

We first bound the number of expected original broadcasts in order for nodes to complete MAC-FirstMover. Recall that k in Algorithm 4 denotes the round index. In our analysis below, we only bound the number of broadcasts made by *fault-free* nodes.

▶ **Lemma 12.** *With probability* $\geq 1-\delta$*, all fault-free nodes complete MAC-FirstMover in phase* p*, after* $\leq 2n' \ln(1/\delta)$ *original broadcasts are made by fault-free nodes in phase* p*.*

Proof. We begin with the following claim. It follows from the definition of successful coin broadcasts. For completeness, we include the proof in Appendix D.

▷ Claim 13. All fault-free nodes complete MAC-FirstMover of phase p if there exists *at least one* successful coin broadcast in phase p.

Every broadcast in phase p has probability $\geq \frac{1}{2n'}$ to be a coin broadcast (by line 3 of Algorithm 4). Since we only care about the number of original broadcasts made by fault-free nodes, all these broadcasts must be eventually completed. Consequently, for all the fault-free nodes in $\mathcal{A}_p$, we have the probability that *first* t *completed broadcasts by any fault-free node in* $\mathcal{A}_p$ *are all dummy*, denoted by P, bounded by

$$P \leq \prod_{i=1}^{t} \left(1 - \frac{1}{2n'}\right) \leq \exp\left(-\frac{t}{2n'}\right).$$

Equivalently, for any $t \geq 2n' \ln(1/\delta)$, with probability at least $1-\delta$, there exists at least one completed coin broadcast among the first t completed broadcasts in phase p, which further implies the existence of at least one successful broadcast by Remark 10. This, together with Claim 13, conclude the proof. (Note that there could be a successful coin broadcast by a faulty node in $\mathcal{A}_p$, but this does not affect the lower bound we derived.) ◀

▶ **Lemma 14.** *Consider the case when all active nodes in phase* p *(i.e., nodes in* $\mathcal{A}_p$*) execute MAC-FirstMover of phase* p *with parameter* $n' \geq n$*. With probability* ≥ 0.05*, each node* $j \in \mathcal{C}_p$ *must reach the same state* v_j *in either phase* p *or phase* $p+1$*.*

Proof. We begin with the following claim. The proof is presented in Appendix E.

▷ Claim 15. If there is exactly one successful original coin broadcast in phase p, then all nodes in $\mathcal{C}_p$ must achieve the same state in either phase p or phase $p+1$.

The analysis below aims to identify the lower bound on the probability of the event that there exists exactly one successful original coin broadcast in phase p.

Consider any message scheduling by the adversary. Since we assume it is oblivious, we can define r_i as the probability that the i-th completed original broadcast in phase p, across the entire set of nodes in $\mathcal{A}_p$, is a coin broadcast given this unknown message scheduling. That is, since the schedule by the adversary is chosen a priori, r_i is a fixed number. Next, we introduce two variables:

- Let $T-1$ denote the number of completed original dummy broadcasts in phase p before the first *completed* original coin broadcasts in phase p, given the message scheduling; and
- Let k_j denote the number of completed original dummy broadcasts by a node $j \in \mathcal{A}$, among these $T-1$ broadcasts. Note that only k_j is defined with respect to a single node.

The first definition implies that the T-th completed original broadcast is a coin broadcast.

Without loss of generality, assume that in the given schedule, the i-th completed original broadcasts across the entire set of nodes in $\mathcal{A}_p$ is the k-th completed original broadcast made by node j. Then by Line 3 of Algorithm 4, we can quantify r_i as follows:

$$r_i = \frac{2^{k-1}}{2n'} \tag{1}$$

Observe that if some node $j \in \mathcal{A}_p$ fails to complete an original broadcast, then it cannot make any further broadcasts. This is because if j is not able to complete a broadcast, then it must be a faulty node. Consequently, the k-th "*completed*" original broadcast made by node j must also be the k-th original broadcast by j. Hence, Equation (1) still holds for a faulty j.

Define $t^* = \min\{t : \sum_{i=1}^{t} r_i \geq \frac{1}{4}\}$. Then we have

$$\mathbb{P}\{T > t^*\} = \prod_{i=1}^{t^*}(1 - r_i) \leq \exp\left(-\sum_{i=1}^{t^*} r_i\right) \leq \exp(-1/4). \tag{2}$$

Define $\mathcal{A}'_p$ as the nodes in $\mathcal{A}_p$ that have completed at least one original dummy broadcast among the first $T-1$ completed original dummy broadcasts in phase p. In other words, $j \in \mathcal{A}'_p$ iff $k_j \geq 1$. Then we can derive the following equality, based on the nodes that have made the completed original broadcast(s):

$$\sum_{i=1}^{T-1} r_i = \sum_{j \in \mathcal{A}'_p} \sum_{k=1}^{k_j} \frac{2^{k-1}}{2n'} = \sum_{j \in \mathcal{A}'_p} \frac{2^{k_j} - 1}{2n'}. \tag{3}$$

The first equality follows from the definition that the first $T-1$ broadcasts are all dummy, and thus r_i must "correspond" to the k-th completed original broadcast (for some $1 \leq k \leq k_j$) by some node j, whose prior broadcasts are all dummy as well. Furthermore, the k_j-th completed original dummy broadcast is the last one by node j (among the first $T-1$ broadcasts across the system). Note that by definition, r_i is a constant for all i. However, the summation $\sum_{i=1}^{T-1} r_i$ is indeed a random variable whose randomness comes from each coin flip. This explains why the first equality is valid.

Next, we upper bound the probability that there are multiple original *coin* broadcasts in one phase. Note that every active node in $\mathcal{A}_p$ can make *at most one* original coin broadcast in phase p because a node that makes a original coin broadcast must receive that coin broadcast from itself and thus terminate Algorithm 4. Since by definition, the T-th completed original broadcast is the first completed original coin broadcast in the entire system, any original coin broadcast made by some node $j \in \mathcal{A}_p$ must be the (k_j+1)-th original broadcast by node j. Equation (1) implies that the probability of the (k_j+1)-th original broadcast being a coin broadcast is $\frac{2^{k_j}}{2n'}$.

Let E_p denote the event that there are strictly more than one original coin broadcast in phase p – these coin broadcasts may or may not be successful. Let E_p^c denote its complement. By union bound, we have

$$\mathbb{P}\{E_p\} \leq \sum_{j \in \mathcal{A}_p} \mathbb{P}\{\text{node } j \text{ makes an original coin broadcast}\} = \sum_{j \in \mathcal{A}_p} \frac{2^{k_j}}{2n'}.$$

Consequently, by Equation (3), the definition of t^* such that $\sum_{i=1}^{t} r_i < \frac{1}{4}$ for all $t < t^*$, and

the assumption that $n' \geq n \geq |\mathcal{A}_p|$, we have

$$\begin{aligned}
\mathbb{P}\{E_p \,|\, T \leq t^*\} &\leq \sum_{j \in \mathcal{A}_p} \frac{2^{k_j}}{2n'} = \sum_{j \in \mathcal{A}'_p} \frac{2^{k_j}}{2n'} + \sum_{j \in \mathcal{A}_p - \mathcal{A}'_p} \frac{1}{2n'} && (k_j = 0 \text{ for } j \notin \mathcal{A}'_p) \\
&= \left(\sum_{j \in \mathcal{A}'_p} \frac{2^{k_j} - 1}{2n'} + \sum_{j \in \mathcal{A}'_p} \frac{1}{2n'} \right) + \sum_{j \in \mathcal{A}_p - \mathcal{A}'_p} \frac{1}{2n'} \\
&= \sum_{i=1}^{T-1} r_i + \sum_{j \in \mathcal{A}_p} \frac{1}{2n'} = \sum_{i=1}^{T-1} r_i + \frac{|\mathcal{A}_p|}{2n'} \leq \frac{3}{4}.
\end{aligned}$$

By Remark 10, $T \leq t^*$, which denotes the event that there is at least one completed original coin broadcast in the first t^* completed original broadcasts, implies that there is at least one *successful* original broadcast in the first t^* completed original broadcasts. Therefore, the fact that $T \leq t^*$ together with E_p^c is a *subset* of the events that there is exactly one successful original coin broadcast in phase p. Consequently, together with Equation (2), we have

$$\begin{aligned}
&\mathbb{P}\{\text{exactly one successful original coin broadcast in phase } p\} \\
&\geq \mathbb{P}\{E_p^c, T \leq t^*\} \geq (1 - \exp(-1/4))(1 - 3/4) \geq 0.05.
\end{aligned}$$

This combined with Claim 15 prove the lemma. ◀

Define the constant c as follows: $c = \frac{\ln(2/\delta)}{0.05}$. Using c in MAC-FirstMover (Algorithm 4), we can derive the following theorem. The full proof is presented in Appendix F. Roughly speaking, nodes need $O(\log n)$ phases to have a large enough estimated system size n'. After that, nodes need a constant number of phases to reach agreement and terminate, due to Lemma 14. Next, Lemma 12 states that each phase needs $O(n)$ broadcasts on expectation. These give us the desired result.

▶ **Theorem 16.** *With probability $\geq 1 - \delta$, MAC-RBC2 terminates and achieves agreement using $O(n \log n)$ total broadcasts across the entire system.*

References

1 Mohssen Abboud, Carole Delporte-Gallet, and Hugues Fauconnier. Agreement without knowing everybody: a first step to dynamicity. In Djamal Benslimane and Aris M. Ouksel, editors, Proceedings of the 8th international conference on New technologies in distributed systems, NOTERE '08, Lyon, France, June 23-27, 2008, pages 49:1–49:5. ACM, 2008. doi:10.1145/1416729.1416792.

2 Eduardo Adílio Pelinson Alchieri, Alysson Neves Bessani, Joni da Silva Fraga, and Fabíola Greve. Byzantine consensus with unknown participants. In Theodore P. Baker, Alain Bui, and Sébastien Tixeuil, editors, Principles of Distributed Systems, 12th International Conference, OPODIS 2008, Luxor, Egypt, December 15-18, 2008. Proceedings, volume 5401 of Lecture Notes in Computer Science, pages 22–40. Springer, 2008. doi:10.1007/978-3-540-92221-6_4.

3 James Aspnes. A modular approach to shared-memory consensus, with applications to the probabilistic-write model. Distributed Comput., 25(2):179–188, 2012. doi:10.1007/s00446-011-0134-8.

4 James Aspnes and Faith Ellen. Tight bounds for anonymous adopt-commit objects. In Rajmohan Rajaraman and Friedhelm Meyer auf der Heide, editors, SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011), pages 317–324. ACM, 2011. doi:10.1145/1989493.1989548.

5 James Aspnes and Maurice Herlihy. Wait-free data structures in the asynchronous pram model. In Proceedings of the second annual ACM symposium on Parallel algorithms and architectures, pages 340–349, 1990.

6 Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. J. ACM, 42(1):124–142, 1995. doi:10.1145/200836.200869.

7 Hagit Attiya, Arie Fouren, and Eli Gafni. An adaptive collect algorithm with applications. Distributed Comput., 15(2):87–96, 2002. doi:10.1007/s004460100067.

8 Hagit Attiya, Sweta Kumari, Archit Somani, and Jennifer L. Welch. Store-collect in the presence of continuous churn with application to snapshots and lattice agreement. In Stéphane Devismes and Neeraj Mittal, editors, Stabilization, Safety, and Security of Distributed Systems - 22nd International Symposium, SSS 2020, Austin, TX, USA, November 18-21, 2020, Proceedings, volume 12514 of Lecture Notes in Computer Science, pages 1–15. Springer, 2020. doi:10.1007/978-3-030-64348-5_1.

9 Hagit Attiya and Ami Paz. Counting-based impossibility proofs for renaming and set agreement. In Marcos K. Aguilera, editor, Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings, volume 7611 of Lecture Notes in Computer Science, pages 356–370. Springer, 2012. doi:10.1007/978-3-642-33651-5_25.

10 Hagit Attiya and Jennifer Welch. Distributed Computing: Fundamentals, Simulations, and Advanced Topics. Wiley Series on Parallel and Distributed Computing, 2004.

11 Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, PODC '83, pages 27–30, New York, NY, USA, 1983. ACM. URL: http://doi.acm.org/10.1145/800221.806707, doi:10.1145/800221.806707.

12 François Bonnet and Michel Raynal. Anonymous asynchronous systems: the case of failure detectors. Distributed Comput., 26(3):141–158, 2013. doi:10.1007/s00446-012-0169-5.

13 David Cavin, Yoav Sasson, and André Schiper. Consensus with unknown participants or fundamental self-organization. In Ioanis Nikolaidis, Michel Barbeau, and Evangelos Kranakis, editors, Ad-Hoc, Mobile, and Wireless Networks: Third International Conference, ADHOC-NOW 2004, Vancouver, Canada, July 22-24, 2004. Proceedings, volume 3158 of Lecture Notes in Computer Science, pages 135–148. Springer, 2004. doi:10.1007/978-3-540-28634-9_11.

14 Ling Cheung. Randomized wait-free consensus using an atomicity assumption. In James H. Anderson, Giuseppe Prencipe, and Roger Wattenhofer, editors, Principles of Distributed Systems, 9th International Conference, OPODIS 2005, Pisa, Italy, December 12-14, 2005, Revised Selected Papers, volume 3974 of Lecture Notes in Computer Science, pages 47–60. Springer, 2005. doi:10.1007/11795490_6.

15 Benny Chor, Amos Israeli, and Ming Li. Wait-free consensus using asynchronous hardware. SIAM J. Comput., 23(4):701–712, 1994. doi:10.1137/S0097539790192635.

16 Eli Daian, Giuliano Losa, Yehuda Afek, and Eli Gafni. A wealth of sub-consensus deterministic objects. In Ulrich Schmid and Josef Widder, editors, 32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018, volume 121 of LIPIcs, pages 17:1–17:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. URL: https://doi.org/10.4230/LIPIcs.DISC.2018.17, doi:10.4230/LIPICS.DISC.2018.17.

17 Giuseppe Di Luna and Roberto Baldoni. Non Trivial Computations in Anonymous Dynamic Networks. In Emmanuelle Anceaume, Christian Cachin, and Maria Potop-Butucaru, editors, 19th International Conference on Principles of Distributed Systems (OPODIS 2015), volume 46 of Leibniz International Proceedings in Informatics (LIPIcs), pages 33:1–33:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.OPODIS.2015.33, doi:10.4230/LIPIcs.OPODIS.2015.33.

18 Giuseppe Antonio Di Luna and Roberto Baldoni. Brief announcement: Investigating the cost of anonymity on dynamic networks. In Proceedings of the 2015 ACM Symposium on

Principles of Distributed Computing, PODC '15, page 339–341, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2767386.2767442.

19 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. J. ACM, 33:499–516, May 1986. URL: http://doi.acm.org/10.1145/5925.5931, doi:http://doi.acm.org/10.1145/5925.5931.

20 Mohsen Ghaffari, Erez Kantor, Nancy A. Lynch, and Calvin C. Newport. Multi-message broadcast with abstract MAC layers and unreliable links. In Magnús M. Halldórsson and Shlomi Dolev, editors, ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014, pages 56–65. ACM, 2014. doi:10.1145/2611462.2611492.

21 Fabíola Greve and Sébastien Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25-28 June 2007, Edinburgh, UK, Proceedings, pages 82–91. IEEE Computer Society, 2007. doi:10.1109/DSN.2007.61.

22 D. Hendler, F. Fich, and N. Shavit. Linear lower bounds on real-world implementations of concurrent objects. In Proc. 46th Annual IEEE Symposium on Foundations of Computer Science, 2005.

23 M. Herlihy. Wait-free synchronization. ACM TOPLAS, 13(1), January 1991.

24 Majid Khabbazian, Dariusz R. Kowalski, Fabian Kuhn, and Nancy A. Lynch. Decomposing broadcast algorithms using abstract MAC layers. Ad Hoc Networks, 12:219–242, 2014. doi:10.1016/j.adhoc.2011.12.001.

25 Fabian Kuhn, Nancy A. Lynch, and Calvin C. Newport. The abstract MAC layer. Distributed Comput., 24(3-4):187–206, 2011. doi:10.1007/s00446-010-0118-0.

26 Petr Kuznetsov, Thibault Rieutord, and Sara Tucci Piergiovanni. Reconfigurable lattice agreement and applications. In Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller, editors, 23rd International Conference on Principles of Distributed Systems, OPODIS 2019, December 17-19, 2019, Neuchâtel, Switzerland, volume 153 of LIPIcs, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.OPODIS.2019.31.

27 Giuseppe Antonio Di Luna, Roberto Baldoni, Silvia Bonomi, and Ioannis Chatzigiannakis. Conscious and unconscious counting on anonymous dynamic networks. In Mainak Chatterjee, Jiannong Cao, Kishore Kothapalli, and Sergio Rajsbaum, editors, Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings, volume 8314 of Lecture Notes in Computer Science, pages 257–271. Springer, 2014. doi:10.1007/978-3-642-45249-9_17.

28 Giuseppe Antonio Di Luna, Roberto Baldoni, Silvia Bonomi, and Ioannis Chatzigiannakis. Counting in anonymous dynamic networks under worst-case adversary. In IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014, Madrid, Spain, June 30 - July 3, 2014, pages 338–347. IEEE Computer Society, 2014. doi:10.1109/ICDCS.2014.42.

29 Giuseppe Antonio Di Luna and Giovanni Viglietta. Brief announcement: Efficient computation in congested anonymous dynamic networks. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023, pages 176–179. ACM, 2023. doi:10.1145/3583668.3594590.

30 Giuseppe Antonio Di Luna and Giovanni Viglietta. Optimal computation in leaderless and multi-leader disconnected anonymous dynamic networks. In Rotem Oshman, editor, 37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy, volume 281 of LIPIcs, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.DISC.2023.18, doi:10.4230/LIPICS.DISC.2023.18.

31 Nancy A. Lynch. Distributed Algorithms. Morgan Kaufmann, 1996.

32 Nancy A. Lynch, Tsvetomira Radeva, and Srikanth Sastry. Asynchronous leader election and MIS using abstract MAC layer. In Fabian Kuhn and Calvin C. Newport, editors, FOMC'12, The Eighth ACM International Workshop on Foundations of Mobile Computing (part of PODC 2012), Funchal, Portugal, July 19, 2012, Proceedings, page 3. ACM, 2012. doi:10.1145/2335470.2335473.

33 Calvin Newport and Peter Robinson. Fault-tolerant consensus with an abstract MAC layer. In Ulrich Schmid and Josef Widder, editors, 32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018, volume 121 of LIPIcs, pages 38:1–38:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.38.

34 Calvin C. Newport. Consensus with an abstract MAC layer. In Magnús M. Halldórsson and Shlomi Dolev, editors, ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014, pages 66–75. ACM, 2014. doi:10.1145/2611462.2611479.

35 Eric Ruppert. The anonymous consensus hierarchy and naming problems. In Eduardo Tovar, Philippas Tsigas, and Hacène Fouchal, editors, Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings, volume 4878 of Lecture Notes in Computer Science, pages 386–400. Springer, 2007. doi:10.1007/978-3-540-77096-1_28.

36 Lewis Tseng and Callie Sardina. Byzantine Consensus in Abstract MAC Layer. In Alysson Bessani, Xavier Défago, Junya Nakamura, Koichi Wada, and Yukiko Yamauchi, editors, 27th International Conference on Principles of Distributed Systems (OPODIS 2023), volume 286 of Leibniz International Proceedings in Informatics (LIPIcs), pages 9:1–9:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.OPODIS.2023.9, doi:10.4230/LIPIcs.OPODIS.2023.9.

37 Lewis Tseng, Qinzi Zhang, and Yifan Zhang. Brief announcement: Reaching approximate consensus when everyone may crash. In Hagit Attiya, editor, 34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference, volume 179 of LIPIcs, pages 53:1–53:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.53.

38 Nitin H. Vaidya, Lewis Tseng, and Guanfeng Liang. Iterative approximate Byzantine consensus in arbitrary directed graphs. In Proceedings of the thirty-first annual ACM symposium on Principles of distributed computing, PODC '12. ACM, 2012.

39 Dongxiao Yu, Yifei Zou, Yuexuan Wang, Jiguo Yu, Xiuzhen Cheng, and Francis C. M. Lau. Implementing the abstract MAC layer via inductive coloring under the rayleigh-fading model. IEEE Trans. Wirel. Commun., 20(9):6167–6178, 2021. doi:10.1109/TWC.2021.3072236.

40 Dongxiao Yu, Yifei Zou, Jiguo Yu, Yong Zhang, Feng Li, Xiuzhen Cheng, Falko Dressler, and Francis C. M. Lau. Implementing the abstract MAC layer in dynamic networks. IEEE Trans. Mob. Comput., 20(5):1832–1845, 2021. doi:10.1109/TMC.2020.2971599.

41 Dongxiao Yu, Yifei Zou, Yong Zhang, Hao Sheng, Weifeng Lv, and Xiuzhen Cheng. An exact implementation of the abstract MAC layer via carrier sensing in dynamic networks. IEEE/ACM Trans. Netw., 29(3):994–1007, 2021. doi:10.1109/TNET.2021.3057890.

42 Qinzi Zhang and Lewis Tseng. The power of abstract mac layer: A fault-tolerance perspective, 2024. URL: https://arxiv.org/abs/2408.10779, arXiv:2408.10779.

A Correctness Proof of MAC-AdoptCommit

▶ **Theorem 17.** *MAC-AdoptCommit is correct for binary inputs.*

Proof. MAC-AdoptCommit satisfies validity, because v_i is either an input at node i or a value from $proposal_i$, which must be an input from another node.

MAC-AdoptCommit satisfies termination, because all the steps are non-blocking.

MAC-AdoptCommit satisfies coherence. Suppose node i outputs $(commit, v)$ at time T_3, and completes line 5 at T_2, and line 4 at T_1 such that $T_1 < T_2 < T_3$.

We first make the following observation, namely *Obs1*, no node with input $-v$ has completed line 1 at any time $\leq T_2$. Suppose node j has input $-v$. By Remark 1 in Section 2, before node i starts to execute line 5, its message handler has processed all the messages received by the abstract MAC layer. Therefore, the fact that $seen_i[-v] =$ false at time T_2 implies that node i has *not* receive any message of the form (`VALUE`, $-v$) at time T_2. Consequently, node j has not completed `mac-broadcast`(`VALUE`, $-v$) (line 1) at time T_2.

Consider the time T when the first `mac-broadcast`(`VALUE`, $-v$) is completed (if there is any). At time T, any node k that has not crashed yet must have already received (`PROPOSAL`, v) at some earlier time than T, because (i) *Obs1* implies that $T > T_2$; and (ii) by time T, node i has already completed line 4 (which occurred at time T_1). Consider two cases:

- k executes line 2 after receiving (`PROPOSAL`, v): in this case, k sets $proposal_k$ to value v before executing line 3 (potentially at some later point that T).
- k executes line 2 before receiving (`PROPOSAL`, v): in this case: k's input must be v; otherwise, T cannot be the first `mac-broadcast`(`VALUE`, $-v$) that is completed. (Observe that by assumption of this case, k executes line 2 before node i completes its line 4 at time T_1.)

In both cases, at line 4, node k can only `mac-broadcast`(`PROPOSAL`, v). That is, no `mac-broadcast`(`PROPOSAL`, $-v$) is possible. Consequently, coherence is satisfied.

MAC-AdoptCommit satisfies convergence. If all the inputs are v, then the only value that can appear in $proposal_i$ is v for each node i. Moreover, none of the nodes would broadcast $-v$; hence, $seen_i[-v]$ will always be false. Consequently, all nodes would output $(commit, v)$. ◀

B Proof of Theorem 7

Proof. Recall that we assume the message oblivious adversary; hence, termination proof is more straightforward. This is because if no node outputs a value, then all nodes rely on the conciliator (flipping a local coin) to reach the same states for the next phase. By construction, nodes may (i) jump to a higher phase with a copied state, (ii) obtain a state that is equivalent to the proposed value from a `PROPOSAL` message, or (iii) choose its new state randomly. Therefore, there is a non-zero probability that all of these random choices equal to the unique state value obtained using approach (i) or (ii). The reason that these obtained states are identical is due to the *coherence* property of the adopt-commit object (as proved in Appendix A).

In the worst case, all nodes "move in sync," i.e., they enter the same phase concurrently without using the jump, and have their states randomly generated. Otherwise if there is some "fast" node that is in a higher phase, it may force all other nodes to jump to its state after it becomes the "proposer" at line 6. We denote the probability that all states are equal after flipping a local coin by r^*. Clearly, $r^* = 2^{-(n-1)} > 0$. Let P be the random variable that denotes the termination phase of MAC-RBC, and note that $P > p$ only if the states are

not equal in the first p rounds. Therefore, $\mathbb{P}\{P > p\} \leq (1 - r^*)^p$. Finally, we conclude the proof by showing that for all $p \geq \ln(1/\delta)/r^* = 2^{n-1}\ln(1/\delta)$,

$$(1 - r^*)^p \leq (1 - r^*)^{\ln(1/\delta)/r^*} \leq \exp(-\ln(1/\delta)) = \delta.$$

The inequality follows from the identity that $1 - x \leq \exp(-x)$ for all $x > 0$. ◀

C MAC-RBC2

Algorithm 5 MAC-RBC2 Algorithm: Steps at each node i with input x_i

Local Variables: /* These variables can be accessed and modified by any thread at node i. */

$seen_i[0]$ ▷(Boolean, phase), initialized to $(false, 0)$
$seen_i[1]$ ▷(Boolean, phase), initialized to $(false, 0)$
$seen_i^2[0]$ ▷(Boolean, phase), initialized to $(false, 0)$
$seen_i^2[1]$ ▷(Boolean, phase), initialized to $(false, 0)$
v_i ▷state, initialized to x_i, the input at node i
p_i ▷phase, initialized to 0
$proposal_i$ ▷(value, phase), initialized to $(\perp, 0)$
n_0 ▷an initial guess of system size, initialized to some constant natural number
n' ▷estimated system size, initialized to 1
c ▷a constant defined as $c = \frac{\ln(2/\delta)}{0.05}$
$coin_i$ ▷(Boolean, phase), initialized to $(\perp, -1)$

```
1: mac-broadcast(ID, i)
2: while true do
3:     p_old ← p_i
4:     mac-broadcast(VALUE, v_i, p_i)
5:     if proposal_i.phase ≥ p_i then
6:         (v_i, p_i) ← proposal_i
7:     mac-broadcast(PROPOSAL, v_i, p_i)
8:     if p_old ≠ p_i then
9:         go to line 2                    ▷"Jump" to p_i
10:    else if seen_i[−v_i].phase < p_i then
11:        output v_i
12:        mac-broadcast(VALUE^2, v_i, p_i)
13:        if seen_i^2[−v_i].phase > p_i then
14:            (v_i, p_i) ← (−v_i, seen_i^2[−v_i].phase)
15:            go to line 2                ▷"Jump" to p_i
16:        else if seen_i^2[−v_i] = (true, p_i) then
17:            // MAC-FirstMover
18:            n' ← 2^⌊p_i/c⌋ n_0
19:            k ← 0
20:            while coin_i.phase < p_i do
21:                if a local random number < 2^k/(2n') then
22:                    mac-broadcast(COIN, v_i, p_i)
23:                else
24:                    mac-broadcast(DUMMY)
25:                k ← k + 1
26:            mac-broadcast(COIN, v, p)
27:            (v_i, p_i) ← coin_i
28:        p_i ← p_i + 1                   ▷"Move" to p_i
```

```
// Background message handlers
29: Upon receive(VALUE, v, p) do
30:     if p ≥ seen_i[v].phase then
31:         seen_i[v] ← (true, p)

32: Upon receive(VALUE^2, v, p) do
33:     if p ≥ seen_i^2[v].phase then
34:         seen_i^2[v] ← (true, p)

35: Upon receive(PROPOSAL, v, p) do
36:     if p ≥ proposal_i.phase then
37:         proposal_i ← (v, p)

// Message handlers for MAC-FirstMover
38: Upon receive(COIN, v, p) do
39:     if p = p_i  and  p > coin_i.phase then
40:         coin_i ← (v, p)
41:     else if p > p_i then
42:         (v_i, p_i) ← (v, p + 1)
43:         go to line 2                    ▷"Jump" to p_i

44: Upon receive(DUMMY) do
45:     do nothing
```

We can get rid of the *coin* variable and directly use v_i and p_i. However, we choose to reserve the variable so that it is more obvious how MAC-RBC2 utilizes MAC-FirstMover.

The reasons that we need to have the condition $p > coin_i.phase$ are: (i) $coin_i.phase$ may be decoupled from p_i; and (ii) each node i has at most two coin broadcasts for a phase p.

D Proof of Claim 13

Proof of Claim 13. Let $m = (\texttt{COIN}, v, p)$ be a successful coin broadcast in phase p. Recall that m is successful because there exists a node j that completes the follow-up broadcast with $(\texttt{COIN}, v, p)$ at some time t. Now, consider three groups of nodes:

- For any node i that was in $\mathcal{A}_p$ before time t: i completes MAC-FirstMover for phase p after receiving and processing m or j's follow-up broadcast.
- For any node i that has not executed MAC-FirstMover of phase p by time t: i would "jump" to phase p after receiving and processing m or j's follow-up broadcast.
- For any node i that has already completed MAC-FirstMover of phase p before time t: this is trivial. Note that this is possible if i processes message(s) faster than j does, or there is a coin broadcast other than m. ◀

E Proof of Claim 15

Proof of Claim 15. In the framework of [3], if every node that has not crashed obtains the same output from the conciliator object, then all the fault-free nodes are guaranteed to terminate in the next phase. This design, the definition of a successful coin broadcast, and the ability to jump to a higher phase in MAC-RBC2 imply the claim. This is because for all nodes that update its state v_i in phase p, they must use the same outcome from the conciliator object (the value field of the successful coin broadcast). For the other nodes that jump to phase $p+1$ (from a phase $< p$), they must either receive phase-p coin broadcast(s) or receive the messages from the adopt-commit object in phase $p+1$. These messages and phase-p coin broadcasts (both the one and only original coin broadcast and follow-up coin broadcasts) must contain exactly the same value. ◀

F Proof of Theorem 16

Proof. First, we can decompose the total number of broadcasts by all fault-free nodes, denoted by N, into three components $N = N^{RBC} + N^O + N^F$, where (i) N^{RBC} denotes the number of broadcasts required by the part of adopt-commit (i.e., all the communication in Algorithm 3); (ii) N^O denotes the number of original broadcasts used in MAC-FirstMover for all phases; and (iii) N^F denotes the number of follow-up broadcasts used in MAC-FirstMover for all phases.

Let P denote the random variable of the first phase index in which the agreement is achieved, i.e., all nodes that have not crashed begin with same v in this phase.

First observe that in each phase, each node makes $O(1)$ broadcasts for adopt-commit and one follow-up broadcast in for MAC-FirstMover. Therefore, $N^{RBC} + N^F = O(nP)$. The rest of the proof focuses on bounding N^O.

Let $n'_p = 2^{\lfloor p/c \rfloor} n_0$ denote the input to MAC-FirstMover, namely the estimated system size in phase p. Then $n'_p \geq n$ for all $p \geq c(1 + \log_2(n/n_0))$. Therefore, Lemma 14 implies that the event E_p of no agreement in phase p has bounded probability $\mathbb{P}\{E_p\} \leq 1 - 0.05$ for all $p \geq c(1 + \log_2(n/n_0))$. Consequently,

$$\mathbb{P}\{P > c(1 + \log_2(n/n_0)) + q\} \leq \prod_{i=1}^{q} \mathbb{P}\{E_{\lfloor c(1+\log_2(n/n_0)) \rfloor + i}\} \leq \prod_{i=1}^{q} (1 - 0.05) \leq \exp(-0.05q).$$

Consequently, let $p^* = c(1 + \log_2(n/n_0)) + \frac{\ln(2/\delta)}{0.05}$. Upon substituting $q = \frac{\ln(2/\delta)}{0.05}$ into the previous bound, we have

$$\mathbb{P}\{P > p^*\} \leq \delta/2. \tag{4}$$

Note that with $c = \frac{\ln(2/\delta)}{0.05}$, $p^* = \frac{\ln(2/\delta)}{0.05}(2 + \log_2(n/n_0)) = O(\ln(n)\ln(1/\delta))$.

Let N_p^O denote the number of original broadcasts made byfault-free nodes in MAC-FirstMover of phase p. Lemma 12 implies that with probability $\geq 1 - \delta$, $N_p^O \leq 2n'_p \ln(1/\delta)$. Therefore, upon applying union bound, we have with probability $\geq 1 - \delta/2$,

$$\begin{aligned}
\sum_{p=1}^{p^*} N_p^O &\leq \sum_{p=1}^{p^*} 2n'_p \ln(2p^*/\delta) && \text{(recall that } n'_p = 2^{\lfloor p/c \rfloor} n_0\text{)} \\
&\leq 2cn_0 \ln(2p^*/\delta) \sum_{q=1}^{\lceil p^*/c \rceil} 2^p \\
&\leq 4cn_0 \ln(2p^*/\delta) 2^{\lceil p^*/c \rceil} && \text{(substitute definition of } p^*\text{)} \\
&= 4n_0 \frac{\ln(2/\delta)}{0.05} \ln\left(\frac{2\ln(2/\delta)(2 + \log_2(n/n_0))}{0.05\delta}\right) \exp_2(2 + \log_2(n/n_0)) \\
&= 320n \ln(2/\delta) \ln\left(\frac{2\ln(2/\delta)(2 + \log_2(n/n_0))}{0.05\delta}\right) \\
&= O\left(n \ln(1/\delta) \ln\left(\frac{\ln(n)\ln(1/\delta)}{\delta}\right)\right).
\end{aligned}$$

Equivalently, we have

$$\mathbb{P}\left\{\sum_{p=1}^{p^*} N_p^O > 320n \ln(2/\delta) \ln\left(\frac{2\ln(2/\delta)(2 + \log_2(n/n_0))}{0.05\delta}\right)\right\} \leq \delta/2. \tag{5}$$

Upon combining Equations (4), (5) and applying union bound, we have with probability $\geq 1 - \delta$, MAC-RBC2 achieves agreement (and thus termination) with

$$N = N^{RBC} + N^O + N^F = O\left(n \ln(n) \ln(1/\delta)\right) \quad \blacktriangleleft$$