Content-Oblivious Leader Election on Rings

Fabian Frei
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Ran Gelles
Bar-Ilan University, Ramat Gan, Israel

Ahmed Ghazy
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
Saarland University, Saarbrücken, Germany

Alexandre Nolin
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

In *content-oblivious computation*, n nodes wish to compute a given task over an asynchronous network that suffers from an extremely harsh type of noise, which corrupts the content of all messages across all channels. In a recent work, Censor-Hillel, Cohen, Gelles, and Sela (Distributed Computing, 2023) showed how to perform arbitrary computations in a content-oblivious way in 2-edge connected networks but only if the network has a distinguished node (called *root*) to initiate the computation.

Our goal is to remove this assumption, which was conjectured to be necessary. Achieving this goal essentially reduces to performing a content-oblivious leader election since an elected leader can then serve as the root required to perform arbitrary content-oblivious computations. We focus on ring networks, which are the simplest 2-edge connected graphs. On *oriented* rings, we obtain a leader election algorithm with message complexity $O(n \cdot \mathsf{ID}_{\max})$, where $\mathsf{ID}_{\max}$ is the maximal assigned ID. As it turns out, this dependency on $\mathsf{ID}_{\max}$ is inherent: we show a lower bound of $\Omega(n \log(\mathsf{ID}_{\max}/n))$ messages for content-oblivious leader election algorithms. We also extend our results to *non-oriented* rings, where nodes cannot tell which channel leads to which neighbor. In this case, however, the algorithm does not terminate but only reaches quiescence.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Content-Oblivious Computation, Faulty Communication, Leader Election, Ring Networks, Ring Orientation

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.26

Related Version *Full Version*: `https://arxiv.org/abs/2405.03646` [22]

Funding *Ran Gelles*: Partially supported by a grant from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel, Grant No. 2020277.

Acknowledgements R. Gelles would like to thank CISPA – Helmholtz Center for Information Security and MPI – Max Planck Institute for Informatics for hosting him while part of this work was done.

1 Introduction

The field of distributed computing is rich with models helping us understand different types of architectures and computational hardness by making different kinds of assumptions about how computations are carried out. A recent work by Censor-Hillel, Cohen, Gelles, and Sela [8] introduced a particularly weak computational model coined *fully defective networks*. This model considers an asynchronous network in which messages may be fully corrupted and thus not carry any information beyond their sheer existence. Algorithms designed for this model cannot rely in any way on possible contents of messages, and thus are named *content-oblivious*. Instead of relying on the content of messages or on their time of arrival (since arbitrary delays may occur in asynchronous networks), such algorithms depend solely on the *order* in which messages arrive from different neighbors.

The aforementioned work by Censor-Hillel et al. [8] showed that any computation possible in the asynchronous setting with reliable content-carrying messages can be simulated in the fully defective setting under two assumptions: that the network is 2-edge connected and that there is a distinguished leader (which they call the *root node*) in the network. While the same paper showed 2-edge connectivity to be essential for any kind of nontrivial computation in fully defective networks, the question of the necessity of a pre-existing leader was not settled. Censor-Hillel et al. conjectured that general computations in fully defective networks do indeed require the existence of such a pre-elected leader.

In this paper, we *disprove* this conjecture, at least for the most fundamental type of 2-edge connected topologies, namely, rings. We design a content-oblivious algorithm that successfully performs a leader election in oriented rings.

▶ **Theorem 1.** *There is a* quiescently terminating *content-oblivious algorithm of message complexity* $n(2 \cdot \mathsf{ID}_{\max} + 1)$ *that elects a leader in oriented rings of* n *nodes with unique IDs.*

Here, $\mathsf{ID}_{\max}$ denotes the maximal ID assigned to a node in the network, and *quiescent termination* refers to the valuable property of our algorithm that nodes do not receive messages after termination. We discuss quiescent computations and the issue of composability in more detail in Section 1.1.

We further extend our result to the case where the ring is non-oriented, albeit under a weaker definition of computation. Instead of termination, we only require *stabilization*, which means that every node eventually settles on a decision (to be or not to be a leader, in our case) that is never revised again. However, the nodes might not know whether they have achieved their stable output already and remain ready to receive and potentially send messages forever. This potential is not actualized for a *quiescently* stabilizing algorithm, where all messaging activity ceases after finite time. Our algorithm not only elects a leader in such a manner but also orients the ring.

▶ **Theorem 2.** *There is a quiescently stabilizing content-oblivious algorithm of message complexity* $n(2 \cdot \mathsf{ID}_{\max} + 1)$ *that elects a leader and orients a non-oriented ring of* n *nodes with unique IDs.*

Considering this weaker definition of computation also allows us to perform the same tasks on the *anonymous ring*. In this setting, nodes are not given identifiers and they all have the same initial state, but each of them has access to an independent source of randomness. Furthermore, the algorithm is allowed to reach an incorrect result as long as it does so with sufficiently small probability (below an arbitrary constant negative power of n). It follows from prior work that there is no terminating algorithm for electing a leader in this setting. However, if we require only quiescent stabilization instead of termination, both electing a leader and orienting a ring are possible.

▶ **Theorem 3.** *There is a content-oblivious algorithm of complexity* $n^{O(1)}$ *that elects a leader and orients an anonymous ring of* n *nodes, each with access to its own source of randomness, with high probability. The algorithm reaches quiescence but does not terminate.*

Both of our deterministic leader election algorithms, for oriented and non-oriented rings, have message complexity $O(n \cdot \mathsf{ID}_{\max})$. Since $\mathsf{ID}_{\max} \geq n$, this implies at least cn^2 pulses, for some constant c. The term $\mathsf{ID}_{\max}$ is not very common in message complexities of algorithms, despite a few exceptions [21, 27].[1] Somewhat surprisingly, our analysis shows that this term is inherent to content-oblivious algorithms. We prove the following lower bound.

[1] Some algorithms depend on the IDs assigned to nodes. However, it is common for the complexity analysis to assume that all IDs are of length $O(\log n)$ bits, making the dependence on the IDs implicit.

▶ **Theorem 4.** *Any deterministic terminating content-oblivious algorithm for leader election in rings with unique IDs sends at least* $n\lfloor\log(\mathsf{ID}_{\max}/n)\rfloor$ *messages.*

Note that this $\Omega(n\log(\mathsf{ID}_{\max}/n))$ lower bound implies that the number of messages in a ring of size n is unbounded – we can always increase the message complexity by assigning larger IDs, even when n is a small constant.

1.1 Quiescence and Composability

As mentioned above, we concatenate our content-oblivious leader election algorithm (Theorem 1) with the root-dependent universal content-oblivious algorithm [8, Thm. 1] to obtain the following powerful corollary.

▶ **Corollary 5.** *Assuming unique IDs, any asynchronous algorithm on rings can be simulated in a fully defective oriented ring.*

However, some subtleties arise when concatenating algorithms in the content-oblivious setting. With reliable message content, each message could be tagged, indicating the algorithm it belongs to. However, when concatenating content-oblivious algorithms, messages sent by the first algorithm may be mistaken for ones sent by the second algorithm, and vice versa.

To make this composition work, we require two properties: (1) *termination*, i.e., nodes should have a distinct point in time where they end the first algorithm and switch to the second one, and (2) *message-algorithm attribution*, i.e., while a node executes an algorithm, it only ever receives messages generated by nodes while they were executing the same algorithm.

Our algorithm for oriented rings (Theorem 1) achieves correct message-algorithm attribution (when composed with the scheme of [8]) by combining two mechanisms. First, it features a *quiescent* termination. In particular, any node terminates eventually, and at that time no messages are in delivery towards that node anymore, nor will any message be sent to that node after its termination. Secondly, the nodes terminate in order, so that the leader is the last to terminate. This makes our algorithm easy to compose with the algorithm of [8], by replacing the act of termination with the act of switching to the second algorithm. The leader, which is the last to terminate the first algorithm, is the node that initiates the computation of the scheme in [8] (i.e., it acts as the root), and at the time it sends its first message, we are guaranteed that all other nodes have already switched to that algorithm.

As a matter of fact, quiescent termination could be relaxed when composing general algorithms: If we have a bound r on the number of messages of the first algorithm that might reach a node after it transitions to the second algorithm, we could still concatenate any algorithm in an *altered form* where nodes send $r+1$ copies of each message, and process arriving messages in groups of $r+1$ messages as well. However, this clearly leads to an undesired r-fold increase in the message complexity of the composed algorithm.

On the other hand, our algorithm for non-oriented rings (Theorem 2) does not terminate and cannot be composed with other algorithms. As mentioned above, we only require the algorithm to reach quiescence in this case.

1.2 Related Work

Leader election is a fundamental task that has been studied by the distributed computing community since the 1970s. Simple leader election algorithms for asynchronous rings were proposed by Le Lann [28] and by Chang and Roberts [10]. These algorithms employ $O(n^2)$ messages to ensure that all nodes yield to the node with the maximal ID, which becomes the

leader. Later work [25, 14, 29] improved this down to $O(n \log n)$ messages. This complexity is tight in asynchronous rings [7, 21]. In *synchronous* rings, leader election can be performed by communicating only $O(n)$ messages [21, 17].

For the case of anonymous rings, i.e., with identical nodes without IDs, Angluin [2] proved that symmetry cannot be broken, and thus no leader election algorithm exists. Attiya, Snir, and Warmuth [5] examined anonymous asynchronous rings further and characterized computable tasks. In particular, they showed that many tasks, including ring orientation, require communicating $\Omega(n^2)$ messages. Attiya and Snir [4] gave tight bounds on the complexities of randomized algorithms. Relaxed notions of ring orientation were given by Attiya, Snir, and Warmuth [5], where the orientation is either consistent with all nodes or alternating between any two neighbors, and by Syrotiuk and Pachel [31], where nodes determine whether a majority agrees on the same orientation or not. Orienting a ring and leader election when n is known to the nodes is presented by Flocchini et al. [19].

The question of termination in anonymous rings was explored by Itai and Rodeh [26], who discovered that a network cannot compute its size n by a terminating algorithm. As a consequence, a leader election algorithm that terminates is impossible, too, since it is trivial to learn n once a leader is given. On the other hand, if the nodes know n or even an upper bound on n, a randomized leader election algorithm that terminates exists [26]. Afek and Matias [1] give various leader election algorithms that trade off knowledge, termination, and number of sent messages.

Censor-Hillel, Gelles, and Haeupler [9] designed a content-oblivious BFS algorithm as a pre-processing step for their distributed interactive coding scheme [23]. Building on that idea, Censor-Hillel et al. [8] introduced the concept of content-oblivious computation and proved that general computations are only possible over 2-edge connected networks. They designed a compiler that converts any asynchronous algorithm into a content-oblivious one, assuming a pre-existing leader. They also gave explicit algorithms for content-oblivious DFS, ear decomposition, and (generalized) Hamiltonian cycle construction.

Computation with fully corrupted messages has been comparatively more studied in the synchronous setting, where the presence or absence of a message in a given round can still be used to send one bit of information [30]. A notable example is the *Beeping* model, introduced by Cornejo and Kuhn [11], in which each message sent by a node is received by all its neighbors, and nodes can only distinguish between receiving no message and receiving at least one message. Among other problems, leader election has been studied in this setting in a sequence of works [24, 20, 15, 12]. Similar to the fully defective setting, compilers were devised for transforming algorithms with stronger communication primitives into algorithms for the Beeping model [16, 3, 13].

1.3 Organization

Section 2 formally introduces the content-oblivious setting and some other notions and notations. In Section 3 we give our leader election algorithm for oriented rings, deferring some proofs to the full version of our paper [22]. Section 4 discusses non-oriented rings. In Section 5, we discuss anonymous rings and design a randomized quiescently stabilizing algorithm with a high probability of success for both electing a leader and orienting a non-oriented ring, with some proofs deferred to the full version of our paper [22]. Our lower bound on the number of messages sent by a content-oblivious leader election algorithm is presented in Section 6. Section 7 concludes our work with a brief summary and suggests a few follow-up questions.

2 Preliminaries

Notations. For an integer n, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. All logarithms are binary. For a variable *var* and a node v, we let $var[v]$ denote the value the variable *var* holds at the node v. By the term *with high probability* we mean a probability of at least $1 - n^{-O(1)}$.

The content-oblivious computation model. Consider a distributed network $G = (V, E)$ with $n = |V|$ nodes. We usually assume that each node $v \in V$ is assigned a positive integer as its (unless otherwise stated) unique ID, usually denoted by $\mathsf{ID}_v \in \mathbb{N}$. We denote the largest ID assigned to a node in the network by $\mathsf{ID}_{\max}$, that is, $\mathsf{ID}_{\max} := \max_{v \in V} \mathsf{ID}_v$, and by ℓ the node possessing this ID. Note that the subset of natural numbers that can be assigned to the nodes is not restricted to $[n]$; it can be chosen arbitrarily, as long as it contains n distinct IDs. Apart from the IDs, the nodes are identical. Algorithms are *uniform* by default, i.e., the nodes neither know the size of the network n nor any bound on it; for non-uniform algorithms, the nodes may be equipped with such knowledge.

In this paper, we consider rings, i.e., connected graphs where all nodes have degree 2. Neighboring nodes communicate by sending messages to each other. We assume that all messages are subject to corruption: the content of any message is completely erased by noise, resulting in an empty message of length 0, which we call a *pulse*. Computations that ignore any potential content of a message and thus work only with pulses are called *content-oblivious*.

Further, the network is *asynchronous*: the time it takes a pulse to travel through a channel and arrive at its end is unpredictable; the delays are unbounded but always finite. Pulses cannot be dropped or injected by the channel. In such an asynchronous network, the nodes possess neither a common clock nor any notion of time, and they are assumed to be *event-driven*. This means that a node may act once right in the beginning of the computation and from then on only upon receiving a pulse. As a function of its own ID, the previously received pulses, and possibly its own source of randomness, it can then change its state and, for each connected channel, send any number of pulses. Most of our algorithms are deterministic, except for the ones in Section 5, where we consider a special case where nodes do not have IDs and use randomness in order to generate them. We say that an algorithm *terminates* (sometimes also referred to as process termination, explicit termination, or termination detection in the literature) if for each node there is a time at which it has decided on an output and entered a terminating state. Once in a terminating state, a node ignores all incoming pulses and does not send any new ones. We say that an algorithm has *quiescent termination* if at the time the last node terminates it is guaranteed that no pulses are still in transit. An algorithm's *message complexity* is the total number of messages (pulses) it sends during a worst-case computation until all nodes have terminated or until the network has reached quiescence.

Ring's orientation. In a ring, each node communicates with its two neighbors via Port_0 and Port_1. Consider a pulse re-sent from Port_1 by every node receiving it. If such a pulse passes through all edges (i.e., is never reflected by a node with misaligned ports), we call the ring *oriented* and the pulse *clockwise* (CW). Port_1 of each node is then called its CW port, leading to its CW neighbor. Counterclockwise (CCW) is defined analogously via Port_0. Note that CW pulses are sent from CW ports but arrive at CCW ports, and vice versa.

However, rings may not be oriented to begin with. In a *non-oriented* ring, there is no guarantee that Port_0 and Port_1 of a node are aligned with a clockwise or counterclockwise walk on the ring. See Fig. 1 for a demonstration. Algorithms for non-oriented rings must work correctly for all assignments of the nodes' ports. In this case the CW/CCW direction is local per node, and we will use the notion of Port_0 and Port_1 to avoid confusion.

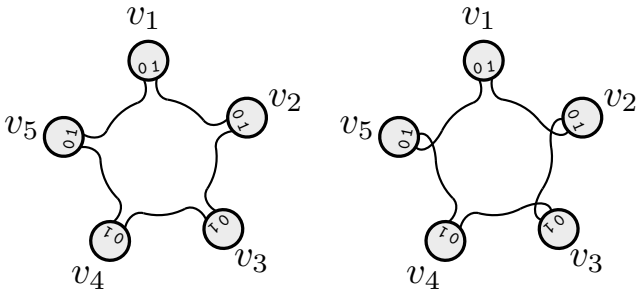

Figure 1 An oriented ring (left) and a non-oriented one (right).

3 Leader Election in an Oriented Ring

In this section, we consider the problem of electing a leader in an oriented ring of size unknown to the nodes. Recall that in the *leader election task*, nodes are required to terminate with an output: Leader or Non-Leader, where a single node $\ell \in V$ outputs Leader, while all other nodes must output Non-Leader. We design an algorithm to elect a leader that features quiescent termination and prove the following.

▶ **Theorem 1.** *There is a* quiescently terminating *content-oblivious algorithm of message complexity* $n(2 \cdot \mathsf{ID}_{\max} + 1)$ *that elects a leader in oriented rings of* n *nodes with unique IDs.*

For the algorithms in this section, we will use the following methods for sending and receiving pulses. The method sendCW() sends one pulse over the CW channel. The method recvCW() checks whether pulses are waiting in the CW incoming queue. If no pulse is in the queue, the method returns 0. Otherwise, it consumes a single pulse from the queue and returns 1. The methods sendCCW() and recvCCW() are the analogous CCW versions. Moreover, for each node v, we introduce counters ρ_{cw} and σ_{cw} for the total number of received and sent CW pulses, respectively. Likewise, we introduce ρ_{ccw} and σ_{ccw} for CCW pulses. We assume each of the four methods above updates those counters with every received or sent pulse. More precisely, every node v has variables $\rho_{\mathsf{cw}}[v]$ and $\sigma_{\mathsf{cw}}[v]$ initially set to 0. Every time recvCW() processes a pulse from the queue of incoming pulses or sendCW() sends a pulse, the counters increase according to $\rho_{\mathsf{cw}}[v] \leftarrow \rho_{\mathsf{cw}}[v] + 1$ and $\sigma_{\mathsf{cw}}[v] \leftarrow \sigma_{\mathsf{cw}}[v] + 1$, respectively.

3.1 Warm-up: Leader Election Without Termination

To demonstrate some of the main ideas of our algorithm, let us begin with a simple quiescently stabilizing algorithm that uses only clockwise pulses and elects a leader in an oriented ring. We emphasize that this algorithm is non-terminating. In this algorithm (see Algorithm 1), each node starts by sending one pulse and then relays every received pulse in the same direction, except for the single time when the number of received pulses reaches its own ID. In this event, the node does not relay this one pulse and assigns itself the state of Leader, at least temporarily. However, any pulses received after this are relayed again, and revert the node to being a Non-Leader.

The main intuition behind this algorithm is that each node will eventually have sent and received exactly $\mathsf{ID}_{\max}$ pulses: n pulses are being generated at the initialization, and these pulses keep circulating in the ring, increasing the counter of received pulses until some node has received as many pulses as its ID. At this point, the node removes one pulse, and we are left with $n - 1$ pulses in circulation. Except for the node with $\mathsf{ID}_{\max}$, every node will receive

more pulses than its ID, ensuring it eventually becomes a follower. This continues until all nodes, except the one with $\mathsf{ID}_{\max}$, have removed a single pulse from the circulation and declared themselves followers. The last remaining pulse keeps circulating until all nodes have received exactly $\mathsf{ID}_{\max}$ pulses. As soon as the last node receives its $\mathsf{ID}_{\max}$-th pulse, it sets itself as a leader and removes the last remaining pulse. Since the network no longer contains any pulse, which we call quiescence, the states of the event-driven nodes remain unchanged as well. Note, however, that nodes do not terminate since they do not know whether the ring has achieved this quiescent state or not, i.e., whether some pulses are still in transit.

Algorithm 1 Quiescently Stabilizing Leader Election for Node v.

```
1: sendCW()
2: while true do
3:     if recvCW() returns 1 then
4:         if ρ_cw = ID_v then
5:             state ← Leader
6:         else                                   ▷ v acts as a relay unless ρ_cw = ID_v
7:             state ← Non-Leader
8:             sendCW()
```

Even though Algorithm 1 does not terminate, analyzing it will help us design our terminating leader election algorithm (Algorithm 2) later on. We begin by stating some key invariants of Algorithm 1, used to show the algorithm's correctness. Their proofs can be found in the full version of the paper [22].

▶ **Lemma 6.** *For every node v running Algorithm 1, the following invariants hold at the end of each iteration of the main loop:*

1. *If $\rho_{\mathsf{cw}} < \mathsf{ID}_v$, then $\sigma_{\mathsf{cw}} = \rho_{\mathsf{cw}} + 1$, i.e., v has sent exactly one pulse more than it has received.*
2. *If $\rho_{\mathsf{cw}} \geq \mathsf{ID}_v$, then $\sigma_{\mathsf{cw}} = \rho_{\mathsf{cw}}$, i.e., v has sent exactly as many pulses as it has received.*

▶ **Lemma 7.** *Let ℓ be the node with $\mathsf{ID}_\ell = \mathsf{ID}_{\max}$. If $\rho_{\mathsf{cw}}[\ell] \geq \mathsf{ID}_\ell$ at some point, then $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds for every node v at this point. That is, ℓ is the last node to satisfy $\rho_{\mathsf{cw}}[\ell] \geq \mathsf{ID}_\ell$.*

We now show that quiescence has been reached when $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds for all nodes.

▶ **Lemma 8.** *If $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds at every node v, then the network is in quiescence.*

Proof. By Lemma 2, we get that $\sigma_{\mathsf{cw}}[v] = \rho_{\mathsf{cw}}[v]$ holds for all v; hence, the total number of sent pulses is equal to the total number of received pulses. In particular, no pulses are in transit (sent but not received). ◀

In fact, the converse is also true; that is, a necessary condition for quiescence is that $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds at every node v and becomes a relay.

▶ **Lemma 9.** *If the network is in quiescence, then $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds at every node v.*

Proof. If there is quiescence, then $\sum_v \sigma_{\mathsf{cw}}[v] = \sum_v \rho_{\mathsf{cw}}[v]$.

By Lemma 6, every node v has $\sigma_{\mathsf{cw}}[v] \geq \rho_{\mathsf{cw}}[v]$. Assuming there is some bad node b with $\rho_{\mathsf{cw}}[b] < \mathsf{ID}_b$, then by Lemma 1, $\sigma_{\mathsf{cw}}[b] = \rho_{\mathsf{cw}}[b] + 1$. Therefore, $\sum_v \sigma_{\mathsf{cw}}[v] = \sum_{v \neq b} \sigma_{\mathsf{cw}}[v] + \sigma_{\mathsf{cw}}[b] \geq \sum_v \rho_{\mathsf{cw}}[v] + 1$, so this cannot occur. ◀

▶ **Corollary 10.** *There is quiescence at some point of time if and only if each node v has $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ at that point of time.*

Proof. The corollary follows directly from Lemmas 8 and 9. ◀

Another equivalent statement to the ones in Corollary 10 is that every node has sent and received exactly $\mathsf{ID}_{\max}$ pulses.

▶ **Lemma 11.** *In any execution of Algorithm 1, at any point of time, the following statements are equivalent:*

1. *The network is in quiescence,*
2. $\forall v : \rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$*, and*
3. $\forall v : \rho_{\mathsf{cw}}[v] = \sigma_{\mathsf{cw}}[v] = \mathsf{ID}_{\max}$.

Proof. The first two statements are equivalent by Corollary 10.

Now, assuming $\rho_{\mathsf{cw}}[v] = \mathsf{ID}_{\max}$ holds for every v, then so does $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$. Conversely, assume $\forall v : \rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$. By Lemma 7, the node ℓ with the largest ID is the last to satisfy that inequality. After the iteration where this occurs for the first time, we have

$$\rho_{\mathsf{cw}}[\ell] = \mathsf{ID}_\ell = \mathsf{ID}_{\max}.$$

By the first equivalence, there is quiescence, and no more pulses are sent or received. Any pulses that have been sent by a node u have been received by its neighbor v, that is, over all CW edges (u, v), it holds that $\sigma_{\mathsf{cw}}[u] = \rho_{\mathsf{cw}}[v]$.

Also, by Lemma 2, every node v has $\sigma_{\mathsf{cw}}[v] = \rho_{\mathsf{cw}}[v]$. As the network forms a ring, combining those equations yields that $\rho_{\mathsf{cw}}[v] = \sigma_{\mathsf{cw}}[v] = \mathsf{ID}_{\max}$ holds for every v. ◀

Given the above properties, we are ready to prove that certain interesting events occur in every execution of Algorithm 1.

First, we show that the inequality $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ is eventually satisfied by every v. Due to the equivalences given by Lemma 11, this leads to a setting where quiescence is achieved, and every node has sent and received exactly $\mathsf{ID}_{\max}$ pulses.

▶ **Lemma 12.** *In any execution of Algorithm 1, for every node v, there is some iteration, where $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds.*

Proof. Let us track the evolution through time of B, the set of nodes that have not met the condition yet; that is, at every point of time, for all $b \in B$, $\rho_{\mathsf{cw}}[b] < \mathsf{ID}_v$.

Initially, B contains all nodes, each of which are removed upon meeting the condition. Since a removed node never enters B again, $|B|$ is monotonically decreasing.

Consider the point of time where $|B|$ reaches a minimum, so B remains fixed. If $|B| = 0$, then there is nothing to prove, so assume $|B| > 0$. At that time, maintain values $\Delta_b := \mathsf{ID}_b - \rho_{\mathsf{cw}}[b] > 0$ for every $b \in B$.

By Lemma 1, for all $b \in B$, we have that $\sigma_{\mathsf{cw}}[b] = \rho_{\mathsf{cw}}[b] + 1$ always holds. Also, from that point on, for every $v \notin B$, we have $\sigma_{\mathsf{cw}}[v] = \rho_{\mathsf{cw}}[v]$ by Lemma 2.

The number of pulses in transit[2] at any given time is the difference between the total number of sent and received pulses across all nodes. Therefore, since $\sum_v \sigma_{\mathsf{cw}}[v] = \sum_v \rho_{\mathsf{cw}}[v] + |B|$, there are still $|B|$ pulses in the network. Since the nodes outside B act as relays, they maintain the number of pulses in transit and, in particular, never remove a pulse from the network. Eventually, some node $b \in B$ must receive a pulse, which decreases the difference Δ_b by one. If $\Delta_b = 0$, then b is removed from B, and $|B|$ decreases beyond its minimum, a contradiction. Otherwise, b forwards a pulse and the number of pulses in transit remains $|B|$, so we can re-apply this argument. At some point, Δ_b reaches 0 for some $b \in B$. Thus, $\rho_{\mathsf{cw}}[b] \geq \mathsf{ID}_b$ holds for b, and b is removed from B, again, a contradiction. ◀

[2] Including pulses that are in some node's queue, but were not processed yet.

A result of Lemma 12, is that, eventually, all nodes send and receive exactly $\mathsf{ID}_{\max}$ pulses, and no further activity occurs on the network.

▶ **Corollary 13.** *In any execution of Algorithm 1, at some point, every node has sent and received exactly* $\mathsf{ID}_{\max}$ *pulses, and the network reached quiescence.*

Proof. By Lemma 12, there is an iteration, after which $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds for all nodes v. The statement then follows directly from Lemma 11. ◀

We also have the following trivial upper bound as a direct corollary.

▶ **Corollary 14.** *In any execution of Algorithm 1, for every node* v*, at every point of time* $\rho_{\mathsf{cw}}[v] \leq \mathsf{ID}_{\max}$.

Proof. Immediate from Corollary 13 and the monotonicity of $\rho_{\mathsf{cw}}[v]$, which is initially 0. ◀

3.2 Leader Election With Quiescent Termination

Although Algorithm 1 is a quiescently stabilizing algorithm for leader election, more ideas are needed in order to achieve this with quiescent termination. The main idea is to utilize the CCW channel to notify all nodes when the leader is elected. The immediate approach would be to leverage the event $\rho_{\mathsf{cw}}[v] = \mathsf{ID}_v$, which signifies the successful election once it happens at the node with maximal ID. However, this is impossible, since the same event also occurs for every other node before the election process has finished.

To be able to detect termination, we require an event that occurs *uniquely* for the leader, and never for other nodes. If we managed to run Algorithm 1 over the CCW channel after a full execution of the same algorithm over the CW channel, then, by symmetry, all nodes would eventually receive $\mathsf{ID}_{\max}$ many CCW pulses. In this scenario, the event $\rho_{\mathsf{cw}}[v] = \mathsf{ID}_v = \rho_{\mathsf{ccw}}[v]$ would, in fact, be unique to the node with the largest ID. Indeed, an initial full execution of Algorithm 1 over the CW channel guarantees that all other nodes have $\rho_{\mathsf{cw}}[v] > \mathsf{ID}_v$ prior to exchanging CCW pulses, so the above event occurs uniquely at the leader and could be used as the trigger for termination.

The main remaining difficulty is that we *cannot* start the CCW algorithm after the CW one is done since neither the leader nor any other node can infer that the CW algorithm has stabilized purely from the number of CW pulses received. We overcome this obstacle by running both algorithms in parallel, ensuring that the CCW one lags behind the CW one. By subtly prioritizing the execution of the CW algorithm over that of the CCW one, we enforce that once $\rho_{\mathsf{ccw}}[v] = \mathsf{ID}_v$ occurs for some Non-Leader node, we already have $\rho_{\mathsf{cw}}[v] > \mathsf{ID}_v$. Then, the only node v satisfying $\rho_{\mathsf{cw}}[v] = \mathsf{ID}_v = \rho_{\mathsf{ccw}}[v]$ is the elected leader. It is the uniqueness of all IDs, crucially including $\mathsf{ID}_{\max}$, that enables this approach.

The complete algorithm with quiescent termination is given in Algorithm 2. It contains two instances of Algorithm 1: one over the CW channel (lines 3–8) and one over the CCW channel (lines 9–13). The CW instance starts, as before, upon initialization, while the CCW instance starts at node v once it reaches the $\rho_{\mathsf{cw}}[v] = \mathsf{ID}_v$ event in the CW instance. This guarantees that the CCW instance lags behind the CW one. Finally, the last part of the algorithm (lines 14–17) is executed once the condition $\rho_{\mathsf{cw}}[v] = \mathsf{ID}_v = \rho_{\mathsf{ccw}}[v]$ is observed to be true, which happens only for the leader. At this point, all the nodes have ρ_{cw} and ρ_{ccw} set to $\mathsf{ID}_{\max}$, and the leader was the last node for which this event occurred, triggering the following termination process: the leader sends a single CCW pulse. Any node receiving this extra pulse sees, for the first time, $\rho_{\mathsf{ccw}} > \rho_{\mathsf{cw}}$, forwards the pulse and terminates (Algorithm 2). The extra pulse is forwarded until it returns to the leader, causing it to terminate without forwarding the pulse. The analysis of Algorithm 2 is deferred to the full version of the paper [22] due to the space constraints.

Algorithm 2 Quiescently Terminating Leader Election for Node v.

```
1: sendCW()
2: repeat
3:     if recvCW() returns 1 then                ▷ Run Algorithm 1 over the CW channel
4:         if ρ_cw = ID_v then
5:             state ← Leader
6:         else
7:             state ← Non-Leader
8:             sendCW()
9:     if ρ_cw ≥ ID_v then        ▷ Run Algorithm 1 over the CCW channel, once ρ_cw ≥ ID_v
10:        if σ_ccw = 0 then sendCCW() end if
11:        if recvCCW() returns 1 then
12:            if ρ_ccw ≠ ID_v then
13:                sendCCW()
14:    if ρ_cw = ID_v = ρ_ccw then                ▷ Initiate a termination pulse
15:        sendCCW()
16:        while recvCCW() returns 0 do
17:            pass                               ▷ Wait for return of termination pulse
18: until ρ_ccw > ρ_cw
19: output state
```

4 Leader Election in Non-Oriented Rings

A natural follow-up question to the above leader election algorithm in oriented rings is whether the same holds for *non-oriented* rings. In this setting, nodes do not possess a predefined CW channel and CCW channel anymore. Instead, they have two ports, Port_0 and Port_1, connecting them to their two neighbors in an arbitrary order.

The straightforward approach would be to first orient the ring with a quiescently terminating algorithm and then use our leader election algorithm from Section 3. Since orienting the ring is easy with leader elected by quiescent termination, orientation and leader election are essentially equivalent tasks from the perspective of quiescently terminating algorithms.

In this section, we instead present a quiescently *stabilizing* algorithm for non-oriented rings, which both elects a leader and orients the ring. Recall that the difference from quiescent termination is that the nodes do not need to terminate explicitly; it suffices for all pulse activity to cease with the correctly computed output still present. Recall our main theorem for this part.

▶ **Theorem 2.** *There is a quiescently stabilizing content-oblivious algorithm of message complexity $n(2 \cdot \mathsf{ID}_{\max} + 1)$ that elects a leader and orients a non-oriented ring of n nodes with unique IDs.*

We emphasize again that this algorithm does not terminate in the usual sense. Instead, its success is defined as reaching quiescence while guaranteeing that at that time only a single node has set its internal state to Leader, while all other nodes have set their state to Non-Leader. Additionally, we require that nodes achieve a consistent orientation of the ring as follows: each node has to label exactly one of its two ports as the port leading to the CW neighbor such that starting at some node and repeatedly moving to node connected to CW port lets us pass through all edges in the ring.

For ease of exposure, we first present an algorithm of slightly worse complexity but whose analysis is almost fully captured by results from earlier sections. We then improve its complexity by proving an additional property about Algorithm 1, our main building block for our algorithm for non-oriented rings (Algorithm 3). Namely, we show that executing Algorithm 1 on a ring with non-unique IDs essentially achieves the same guarantees as when used on a ring with unique IDs (Lemma 16). This observation allows us to halve the complexity of Algorithm 3, as we shall see, and also has implications for solving the same tasks on anonymous rings with access to randomness. As this extension to anonymous rings is a minor effort and follows from standard techniques, we defer it to Section 5.

Algorithm overview. At a high level, Algorithm 3 consists of two parallels executions of Algorithm 1, each one using each channel in the ring in a single direction. For an intuition of how that is possible, consider a setting where the network has a single pulse in transit. Suppose that all nodes execute the same algorithm that sends a pulse on Port_1 whenever one is received on Port_0, and vice versa. Forwarding the pulse in this manner has it travel the entire ring, since every time a pulse is received by a given node v from one of its neighbors, it is sent to its other neighbor. If adding another pulse going in the other direction to the network, the two pulses independently travel around the ring in opposite directions. The nodes can thus effectively run two algorithms in parallel without them interfering with one another, as long as one only works with clockwise pulses and the other with counterclockwise pulses. As Algorithm 1 only uses pulses going in one direction, it satisfies this requirement. The major caveat is that the nodes cannot be certain which of the two algorithms they execute is working with clockwise pulses and which is working with counterclockwise pulses.

Our algorithm has essentially two parts: one in which the node reacts to pulses and possibly forwards them (Lines 5 to 7), and one in which it computes its output depending on the number of pulses it received from each port (Lines 8 to 16). From the point of view of analyzing how many pulses are eventually sent in the network, only the first part is relevant. It is the part that simulates two executions of Algorithm 1. For the nodes to settle on a consistent ring orientation in the second part, we break symmetry between the two options by having strictly more pulses sent in one orientation of the ring than in the other.

We distinguish the two parallel executions of Algorithm 1 by having each node v pick two distinct virtual IDs for itself, $\mathsf{ID}_v^{(0)}$ and $\mathsf{ID}_v^{(1)}$ (Algorithm 3). $\mathsf{ID}_v^{(1)}$ affects how v behaves regarding pulses received from its Port_0, and symmetrically for $\mathsf{ID}_v^{(0)}$ and Port_1. While nodes do not know which of their two virtual IDs is used in the clockwise or counterclockwise execution of Algorithm 1, they use a distinct ID in both executions. Eventually, in each direction, the number of pulses received by each node stabilizes to the largest ID in that direction. The choice of IDs ensures that the two parallel executions have distinct largest IDs, so eventually all nodes see strictly more pulses being sent in one direction than the other. This allows nodes to agree on a common orientation of the ring, and elect as leader the node who was the source of the largest ID.

The nodes use the following methods for sending and receiving pulses. Let $i \in \{0, 1\}$.

1. $\text{sendPort}_i()$: sends a pulse through Port_i,
2. $\text{recvPort}_i()$: check whether a pulse is waiting in the incoming queue of Port_i. If not, return 0. Otherwise, consume a single pulse from the queue and return 1.

▶ **Proposition 15.** *Algorithm 3 elects a leader and consistently orients the ring using* $n(4\mathsf{ID}_{\max} - 1)$ *pulses. It achieves quiescence but does not terminate.*

Algorithm 3 Quiescently Stabilizing Leader Election on Non-Oriented Rings for Node v.

```
1:  for i ∈ {0,1} do
2:      ID_v^(i) ← 2 · ID_v − 1 + i
3:      sendPort_i()
4:  while true do
5:      for i ∈ {0,1} do
6:          if recvPort_{1−i}() returns 1 and ρ_{1−i} ≠ ID_v^(i) then
7:              sendPort_i() ▷ Pulses received at one port are sent forward at the opposite one
8:      if max(ρ_0, ρ_1) ≥ ID_v^(1) then
9:          if ρ_0 = ID_v^(1) and ρ_1 < ID_v^(1) then
10:             state ← Leader
11:         else
12:             state ← Non-Leader
13:         if ρ_0 > ρ_1 then
14:             name Port_0 ≔ CCW and Port_1 ≔ CW    ▷ Port_0 connects the CCW neighbor
15:         else
16:             name Port_0 ≔ CW and Port_1 ≔ CCW
```

Proof. Consider ℓ, the node of largest ID. We define as clockwise the direction of a pulse sent from ℓ's Port_1 and forwarded by all other nodes (i.e., sent from Port_i after arriving at Port_{1-i}). We call the ports sending such a pulse clockwise, and those receiving it counterclockwise. We show the our algorithm elects ℓ as a leader and all nodes declare the correct port as clockwise.

We show that the network eventually achieves quiescence and all nodes receive the same number of clockwise and counterclockwise pulses. Let us argue this for clockwise pulses; the property for counterclockwise pulses will follow by symmetry. For every node v whose Port_1 connects it to its clockwise neighbor, let us rename $\text{sendPort}_1()$ to $\text{sendCW}()$ and $\text{recvPort}_0()$ to $\text{recvCW}()$ in its code. Let us also define $\mathsf{ID}_v^{\text{cw}} = \mathsf{ID}_v^1$ for such nodes. For the other nodes, which are connected to their clockwise neighbors through Port_0, rename $\text{sendPort}_0()$ to $\text{sendCW}()$ and $\text{recvPort}_1()$ to $\text{recvCW}()$ in their code, and let $\mathsf{ID}_v^{\text{cw}} = \mathsf{ID}_v^0$. We emphasize that this renaming is done purely for our analysis and is not an operation performed by the nodes, which are oblivious to what the clockwise direction is. Consider an execution of Algorithm 3. Whenever the scheduler delivers a clockwise pulse, this pulse is read by a $\text{recvCW}()$ method, and if forwarded (which depends on $\mathsf{ID}_v^{\text{cw}}$), it is re-sent by a $\text{sendCW}()$ method, according to our renaming. Other methods are never activated by clockwise pulses and never emit a clockwise pulse. As such, Algorithm 3 executes the exact same code on clockwise pulses as Algorithm 1 and thus has the same guarantees as Algorithm 1 regarding clockwise pulses. By Corollary 13, this means that we achieve quiescence for clockwise pulses, with all nodes eventually having sent and received exactly the same number $\max_v \mathsf{ID}_v^{\text{cw}}$ of clockwise pulses. The same holds symmetrically for $\max_v \mathsf{ID}_v^{\text{ccw}}$ pulses counterclockwise pulses per node, where $\mathsf{ID}_v^{\text{ccw}}$ is defined similarly to $\mathsf{ID}_v^{\text{cw}}$.

Since ℓ picks as identifiers $2 \cdot \mathsf{ID}_{\max}$ and $2 \cdot \mathsf{ID}_{\max} - 1$ for the two directions, we have $\max_v \mathsf{ID}_v^{\text{cw}} = 2 \cdot \mathsf{ID}_{\max}$ and $\max_v \mathsf{ID}_v^{\text{ccw}} = 2 \cdot \mathsf{ID}_{\max} - 1$. Hence, all nodes have sent and received $2 \cdot \mathsf{ID}_{\max}$ clockwise pulses and $2 \cdot \mathsf{ID}_{\max} - 1$ counterclockwise pulses. This yields a bound of $n(4 \cdot \mathsf{ID}_{\max} - 1)$ pulses, ensuring a consistent orientation according to the test on Algorithm 3. ◀

Improving the message complexity. We now show how to improve the complexity of Algorithm 3 to $n(2 \cdot \mathsf{ID}_{\max} + 1)$. Since nodes send pulses according to their IDs, Algorithm 3 effectively doubles the number of pulses sent by the algorithm. To avoid this doubling, one can generate the two IDs in a different manner; for instance, $\mathsf{ID}_v^{(1)} \leftarrow \mathsf{ID}_v + 1$, and $\mathsf{ID}_v^{(0)} \leftarrow \mathsf{ID}_v$. However, this leads to assigning the same ID to multiple nodes. We argue that Algorithm 3 works correctly even when IDs are not unique as long as the largest clockwise and counterclockwise IDs are different. To that goal, we need to re-analyze Algorithm 1 in such case, which we do in the next two technical lemmas.

▶ **Lemma 16.** *Corollary 13 still holds if nodes run Algorithm 1 with non-unique IDs. This includes the case in which multiple nodes v have $\mathsf{ID}_v = \mathsf{ID}_{\max}$.*

Note that Lemma 16 is about Algorithm 1, the implicit main subroutine of Algorithm 3, not Algorithm 3 itself. Most elements of the proof of Corollary 13 make no reference to the node of largest ID. Most importantly, the invariants of Lemma 6 (that $\sigma_{\mathsf{cw}}[v] = \rho_{\mathsf{cw}}[v] + 1$ while $\rho_{\mathsf{cw}}[v] < \mathsf{ID}_v$, and $\sigma_{\mathsf{cw}}[v] = \rho_{\mathsf{cw}}[v]$ once $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$) are consequences of how each node reacts to the pulses it receives, and changing the distribution of IDs does not change that. Lemma 7, however, makes an explicit reference to a node of largest ID, which requires us to change the argument somewhat. Lemma 17, which we shall now prove, generalizes Lemma 7 to the setting with non-unique IDs.

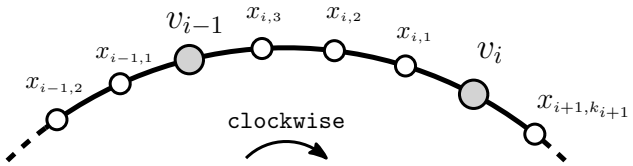

Figure 2 The naming of nodes between nodes of largest ID in the proof of Lemma 17.

▶ **Lemma 17.** *Consider the set of nodes of largest ID, $V_{\max} = \{v : \mathsf{ID}_v = \mathsf{ID}_{\max}\}$. In Algorithm 1, if $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ for all $v \in V_{\max}$ at some point, then $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds for every node v at this point. That is, one of the nodes in $V_{\max}$ is the last node to satisfy $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$.*

Proof. Let $m = |V_{\max}|$ be the number of nodes that hold $\mathsf{ID}_{\max}$. Denote these nodes $v_1, \ldots, v_m$, ordered clockwise from an arbitrary one of them. Let us identify $v_{m+1} = v_1$ for ease of notation. For each $i \in [m]$, let $k_i \in \{0, \ldots, n-1\}$ be the number of consecutive nodes with $\mathsf{ID}_v < \mathsf{ID}_{\max}$ preceding v_i in the ring. For each $j \in [k_i]$, let $x_{i,j}$ be the node j counterclockwise hops from v_i in the ring. See Figure 2 for an illustration.

We show that if $\rho_{\mathsf{cw}}[v_i] \geq \mathsf{ID}_{v_i}$ holds at v_i, then it also holds at all $x_{i,j}$, $j \in [k_i]$. Since every node $v \notin V_{\max}$ has a node of largest ID in its clockwise direction later in the ring, this implies that when $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ holds at all $v_i \in V_{\max}$, it also holds at every node $v \in V$.

Consider $v_i \in V_{\max}$ s.t. $\rho_{\mathsf{cw}}[v_i] \geq \mathsf{ID}_{v_i} = \mathsf{ID}_{\max}$. Suppose $k_i > 0$, as otherwise the result is trivial. Let $x_{i,0} = v_i$. For each $j \in [0, k_i)$, we show that $\rho_{\mathsf{cw}}[x_{i,j}] \geq \mathsf{ID}_{\max}$ implies $\rho_{\mathsf{cw}}[x_{i,j+1}] \geq \mathsf{ID}_{\max}$. As the base case $\rho_{\mathsf{cw}}[x_{i,0}] = \rho_{\mathsf{cw}}[v_i] \geq \mathsf{ID}_{\max}$ holds, we get our result by induction. Let $j \in [0, k_i)$ and suppose $\rho_{\mathsf{cw}}[x_{i,j}] \geq \mathsf{ID}_{\max}$. Since $\rho_{\mathsf{cw}}[x_{i,j}] \leq \sigma_{\mathsf{cw}}[x_{i,j+1}]$, we have that $\sigma_{\mathsf{cw}}[x_{i,j+1}] \geq \mathsf{ID}_{\max}$. Since $\mathsf{ID}_{\max} > \mathsf{ID}_{x_{i,j+1}}$, by Lemma 6, it needs to hold that $\rho_{\mathsf{cw}}[x_{i,j+1}] = \sigma_{\mathsf{cw}}[x_{i,j+1}]$. Therefore, $\rho_{\mathsf{cw}}[x_{i,j+1}] \geq \mathsf{ID}_{\max}$. ◀

Equipped with Lemma 17, the proof of Lemma 16 follows quite naturally.

Proof of Lemma 16. Let us review the proof of Corollary 13, and see which elements of it could be affected by multiple nodes having the same ID. Let $V_{\max} = \{v : \mathsf{ID}_v = \mathsf{ID}_{\max}\}$ be the set of nodes of largest ID.

As already stated, the invariants of Lemma 6 still hold, and arguments relying on Lemma 7 must be amended to rely on Lemma 17 instead. Lemmas 8 and 9 and Corollary 10, about how the network is in quiescence if and only if each node v has received (and, by Lemma 6, also sent) at least ID_v pulses, are immediate consequences of Lemma 6, and hence still hold.

We now get to Lemmas 11 and 12, which are two lemmas from which Corollary 13 follows most directly. Lemma 11 shows equivalences between three properties: quiescence, that $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ at each v, and that $\rho_{\mathsf{cw}}[v] = \sigma_{\mathsf{cw}}[v] = \mathsf{ID}_{\max}$ at each v. Again, the arguments still hold if IDs are not unique: the connection between quiescence and all nodes satisfying $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$ was shown in Corollary 10; while some node $v \in V_{\max}$ has received less than $\mathsf{ID}_{\max}$ pulses, the network is not in quiescence; for all nodes in $V_{\max}$ to have received $\mathsf{ID}_{\max}$ pulses, other nodes in the network need to have sent this many pulses, which implies they have also received $\mathsf{ID}_{\max}$ pulses.

Lemma 12 shows that the network cannot permanently remain in a state in which some nodes have $\rho_{\mathsf{cw}}[v] < \mathsf{ID}_v$. The argument again does not rely on the uniqueness of the IDs, and only uses that if some non-empty set B of nodes satisfies $\rho_{\mathsf{cw}}[v] < \mathsf{ID}_v$ for each $v \in B$, then since for those nodes $\sigma_{\mathsf{cw}}[v] = \rho_{\mathsf{cw}}[v] + 1$ (Lemma 6) the network has pulses still in transit. These pulses must eventually reach nodes in B, increase the number of received pulses ρ_{cw} of nodes in it, and eventually to the point that for a node in B, $\rho_{\mathsf{cw}}[v] \geq \mathsf{ID}_v$.

Put together, the whole argument still holds with non-unique IDs. ◀

The proof of our second main theorem is a corollary of the above.

Proof of Theorem 2. Let us modify Algorithm 3 as follows:

- In Algorithm 3, we set as IDs $\mathsf{ID}_v^{(1)} \leftarrow \mathsf{ID}_v + 1$, and $\mathsf{ID}_v^{(0)} \leftarrow \mathsf{ID}_v$.

Similar to the argument in the proof of Proposition 15, this amounts to running two parallel instances of Algorithm 1 over each channel. As in that proof, consider the maximal clockwise and counterclockwise IDs, $\max_v \mathsf{ID}_v^{\mathrm{cw}} = \mathsf{ID}_{\max} + 1$ and $\max_v \mathsf{ID}_v^{\mathrm{ccw}} = \mathsf{ID}_{\max}$. From Lemma 16, we know that the number of clockwise pulses sent and received by each node eventually stabilizes at $\mathsf{ID}_{\max} + 1$, and similarly stabilizes at $\mathsf{ID}_{\max}$ for the counterclockwise direction. This results in a single leader being elected and a consistent orientation as before. The upper bound of $n(2\mathsf{ID}_{\max} + 1)$ follows from $n(\mathsf{ID}_{\max} + 1)$ pulses being exchanged in one direction, and $n \cdot \mathsf{ID}_{\max}$ in the other. ◀

5 Anonymous Rings

In this section, we consider the setting where a ring consists of n identical nodes without IDs, each with access to its own independent source of randomness. We call such a ring *anonymous*. As is standard in the literature about randomized algorithms, we aim to solve our computational task *with high probability*, defined as bounding the probability of failure by an arbitrary negative power of the network size n. That is, we present an algorithm parameterized by a value $c > 0$, such that for any n, the algorithm correctly elects a leader in rings of size n with probability at least $1 - O(n^{-c})$. Our algorithm also correctly orients a non-oriented ring with high probability.

Similar to other sections of this paper, proofs omitted from this section can be found in the full version of this paper [22].

As alluded to in the introduction, electing a leader with quiescent termination is impossible under the assumptions of this section, even if we relax the objective to succeed with only some arbitrary small constant probability. This follows from a negative result by Itai and Rodeh [26, Thm. 4.1] about counting the number of nodes in an anonymous ring because computing this number is easy in a ring with an elected leader. Consequently, we only aim for a quiescently stabilizing algorithm.

As in Section 4, observe that despite assuming the uniqueness of all IDs in Section 3.1 to elect the node with the largest ID, the task remains well-defined even if only the largest ID is unique. Lemma 16 showed that on a ring with possibly non-unique IDs, Algorithm 1 elects as leader all nodes v such that $\mathsf{ID}_v = \mathsf{ID}_{\max}$. Hence, if a single node satisfies $\mathsf{ID}_v = \mathsf{ID}_{\max}$, a single leader is elected. Algorithm 3 is simply two parallel executions of Algorithm 1. As we already showed in the proof of Theorem 2, those two parallel executions still yield the desired result as long as the maximal ID in the two executions is unique. Hence, providing an algorithm for sampling IDs with the guarantee that the maximal ID is unique with high probability is sufficient to obtain a variant of Theorem 2 for anonymous rings.

We outline in Algorithm 4 a process by which nodes can utilize their access to randomness to sample random IDs with the guarantee that, with high probability, the maximal ID is unique and of order $n^{O(1)}$. The algorithm terminates quiescently – in fact, it uses no communication – enabling composition: any algorithm can be performed afterwards with the sampled IDs. As a result, with high probability, the anonymous setting reduces to the one considered in Lemma 16. As explained previously, such an algorithm for sampling IDs immediately implies the following result.

▶ **Theorem 3.** *There is a content-oblivious algorithm of complexity $n^{O(1)}$ that elects a leader and orients an anonymous ring of n nodes, each with access to its own source of randomness, with high probability. The algorithm reaches quiescence but does not terminate.*

At a high level, the algorithm for sampling IDs has each node first sample the number of bits in its ID from a geometric distribution with parameter $p = 2^{-1/\Theta(c)}$ before sampling said bits uniformly at random. While each ID is of expected length $\Theta(c)$ and value $2^{\Theta(c)}$ at the end of this process, the maximal ID is of length $\Theta(c^2 \log n)$ and value $n^{\Theta(c^2)}$, with high probability. See [6]. Intuitively, while sampling a large ID is an unlikely event, this unlikely event becomes more and more likely to occur somewhere in the network as the network grows. Similar ideas have previously appeared in the literature, e.g., in [1, 18].

Algorithm 4 Message-Free Algorithm for Sampling an ID of Order $n^{O(c^2)}$ for Node v, $c > 0$.

1: $p \leftarrow 2^{-1/(c+2)}$
2: Sample $BitCount \sim Geo(1-p)$, i.e., according to the geometric distribution with parameter $1-p$
3: Sample ID_v uniformly at random from $\{0,1\}^{BitCount}$

▶ **Lemma 18.** *For any constant $c > 0$, with high probability, running Algorithm 4 in an anonymous ring of n nodes assigns each one an ID of size $n^{O(c^2)}$ such that the maximal ID is attained uniquely by one node and is at least $n^{\Omega(c)}$.*

Proof sketch. Each node has a probability of p^x to have their $BitCount$ variable exceed some value x. With $x \in \Theta(\log_p n)$, this probability is of order $1/\operatorname{poly}(n)$. Doing a union bound over all nodes, we get that with high probability, all nodes' $BitCount$ variables stay below some value $x \in \Theta(\log_p n)$. It follows that the largest ID is of order at most $\operatorname{poly}(n)$. Since

nodes do their sampling independently, the probability that all nodes' *BitCount* variables stay below some value x is $(1-p^x)^n$. For $x \in \Theta(\log_p n)$, this is $1/\operatorname{poly}(n)$. As a result, at least one node's *BitCount* variable exceeds $\Theta(\log_p n)$, with high probability. Such nodes then sample their random bits uniformly at random from $\operatorname{poly}(n)$ choices, which makes a collision at the highest value very unlikely. ◀

The proof of Theorem 3 follows immediately from Lemma 18.

Finally, we note that a slight modification of Algorithm 3 would also allow us to sample a unique ID for each node (Proposition 19), with high probability. As a result, the three settings of (1) the anonymous ring, (2) the ring with a leader, and (3) the ring with unique IDs and an orientation, are crucially separated by the possibility of quiescent termination. Leader election, orienting the ring, and assigning unique IDs can all be computed in setting (1). However, they can only be done without termination, while in settings (2) and (3), the same tasks and more can be performed with quiescent termination.

▶ **Proposition 19.** *Let ID_v be the ID sampled by nodes in Algorithm 4, before running Algorithm 3. Modify Algorithm 3 so that whenever a node receives a pulse, if $\min(\rho_0, \rho_1) > \mathsf{ID}_v$, node v updates its ID to a new ID sampled uniformly at random between 1 and $\min(\rho_0, \rho_1) - 1$. Then, with high probability, all nodes have distinct IDs when reaching quiescence.*

6 Lower Bound on Message Complexity in Content-Oblivious Rings

In this section, we show that the dependency of the message complexity of our algorithms on $\mathsf{ID}_{\max}$ is natural and inevitable by providing a lower bound showing that the number of pulses sent increases indefinitely with the number of available IDs.

▶ **Theorem 20.** *Let k and n be arbitrary positive integers, $k \geq n$. If k distinct IDs are assignable to the n nodes of the ring, at least $n\lfloor \log(k/n) \rfloor$ pulses are sent by any leader election algorithm for some assignment of IDs. In particular, an unbounded number of pulses is sent for an infinite supply of IDs even on rings with just a single node.*

The proof of this theorem makes use of the following definition.

▶ **Definition 21** (Solitude pattern)**.** *Consider a ring with a single node ($n = 1$), and fix a specific algorithm. Assume a scheduler that delivers pulses one by one, keeping the order in which they were sent (breaking ties by prioritizing* CW *pulses). Define the* solitude pattern *as the sequence of incoming pulses observed by the node, encoded as a binary string where* 0 *and* 1 *encode* CW *and* CCW *pulses, respectively. We denote the solitude pattern of a node with $\mathsf{ID} = i$ by p_i.*

Besides this crucial definition, we make use of the following lemma telling us that each ID has its own, unique solitude pattern. Essentially, the proof relies on matching all possible pairs of IDs against each other in a ring of two nodes. If any two IDs had the same solitude pattern, they would send and receive pulses in this ring ($n = 2$) exactly as they would in solitude ($n = 1$). Thus, in one of these execution they give an invalid output.

▶ **Lemma 22.** *For any uniform content-oblivious leader election algorithm, each solitude pattern is unique. In other words, for any pair of distinct IDs $i \neq j$, we have $p_i \neq p_j$.*

Proof. Fix a content-oblivious leader election algorithm on uniform rings. Seeking contradiction, assume that two nodes with distinct IDs, $i \neq j$, have the same solitude pattern, $p_i = p_j$. Note that each of these nodes outputs Leader when run in isolation ($n = 1$).

Consider a ring with $n = 2$ nodes, which are assigned the IDs i and j, respectively. Assume a scheduler behaving the same way as in the definition of a solitude pattern for each node individually. That is, pulses arrive one by one in the order they were sent out. Moreover, we assume that the same delay is applied to all pulses. Since the two nodes' solitude patterns are identical, and the scheduler maintains order, each node will receive (and thus generate) exactly its solitude pattern. Indeed, since $p_i = p_j$, both nodes send their first pulse in the same direction, and hence both receive it from the same direction as they would when alone and then send their next pulse accordingly; thus both receive and send exactly the pattern $p_i = p_j$. This means that both nodes output Leader as they must in their solitude situation, contradicting the guarantees of the leader election task. ◀

Having established that each ID has its own unique solitude pattern, a lower bound arises from properties of binary strings, namely the length required to avoid repeating patterns. We begin with the following simple property following from the pigeon-hole principle.

▶ **Lemma 23.** *For any $s, n \geq 1$, any set of $n2^s$ distinct binary strings contains n strings sharing a common prefix of length at least s.*

Proof. Let s and n be positive integers. There are only $2^s - 1$ distinct binary strings shorter than s. Therefore, in a set of $n2^s$ distinct binary strings at least $n2^s - 2^s - 1 = (n-1)2^s + 1$ of them have length at least s, implying that they contain a prefix string of length s. There are only 2^s distinct binary prefixes of length s, thus the pigeon-hole principle implies that at least one prefix of length s is shared by at least $\lceil ((n-1)2^s + 1)/2^s \rceil = \lceil n - 1 + 2^{-s} \rceil = n$ of the strings in the set. ◀

After deducing another corollary, we are ready to prove the lower bound of Theorem 20.

▶ **Corollary 24.** *For any integers $k \geq n \geq 1$, any set of k distinct binary strings contains n strings sharing a common prefix of length at least $\lfloor \log(k/n) \rfloor$.*

Proof. Apply Lemma 23 with the given n and $s = \lfloor \log(k/n) \rfloor$. Note that the given set is large enough since $n2^s \leq n2^{\log(k/n)} = n(k/n) = k$. ◀

Proof of Theorem 20. Assume that we have a uniform leader election algorithm, a ring with n nodes, and at least k assignable IDs for any positive integers $k \geq n$. Due to Lemma 22 we know that each ID has its own unique solitude pattern, which is just a binary string. By Corollary 24, there are at least n IDs whose solitude patterns share a common prefix of length $s = \lfloor \log(k/n) \rfloor$. Assume a scheduler that behaves as described in Lemma 22, for all n nodes. It follows that all nodes send and receive pulses in exactly the same way as in their respective solitude situations for the first $\lfloor \log(k/n) \rfloor$ time steps, sending one pulse in each time step. Consequently, at least $n \cdot s = n\lfloor \log(k/n) \rfloor$ pulses are sent in total, proving the first part of the theorem. The last statement in the theorem follows for an infinite set of assignable IDs because $n\lfloor \log(k/n) \rfloor$ grows indefinitely with increasing k, even for $n = 1$. ◀

Theorem 4 is then an immediate corollary of the above since the number of distinct IDs is bounded by $\mathsf{ID}_{\max}$. This lower bound complements the upper bound of Theorem 1 and proves that the $\mathsf{ID}_{\max}$ term is not an artifact of our analysis or algorithm design but, rather, an inherent property of the problem in this setting.

7 Conclusion and Open Questions

As our main result, we have presented a quiescently terminating algorithm for leader election in oriented rings with unique IDs that communicates $n(2 \cdot \mathsf{ID}_{\max} + 1)$ pulses. This implies that any content-oblivious computation can be performed on rings without assuming a pre-existing leader. We have also provided a lower bound showing that the message complexity depending on $\mathsf{ID}_{\max}$ is not a fluke but inherent to the problem.

An immediate candidate for future work is to extend our results from rings to general networks, i.e., to design a content-oblivious leader election algorithm in arbitrary 2-edge connected networks or, alternatively, prove this task impossible. Considering non-oriented rings may be useful towards that goal since there is no sense of direction in general networks. Our content-oblivious leader election for non-oriented rings does not terminate, and we conjecture that this is inherent to the model. It remains as an open task for future work to prove this or find a terminating algorithm.

References

1 Y. Afek and Y. Matias. Elections in anonymous networks. *Information and Computation*, 113(2):312–330, 1994. doi:10.1006/inco.1994.1075.

2 Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 82–93, New York, NY, USA, 1980. ACM. doi:10.1145/800141.804655.

3 Yagel Ashkenazi, Ran Gelles, and Amir Leshem. Noisy beeping networks. *Information and Computation*, 289(A):104925, 2022. doi:10.1016/J.IC.2022.104925.

4 Hagit Attiya and Marc Snir. Better computing on the anonymous ring. *Journal of Algorithms*, 12(2):204–238, 1991. doi:10.1016/0196-6774(91)90002-G.

5 Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. *J. ACM*, 35(4):845–875, October 1988. doi:10.1145/48014.48247.

6 F Thomas Bruss and Colm Art O'cinneide. On the maximum and its uniqueness for geometric random samples. *Journal of applied probability*, 27(3):598–610, 1990.

7 James E. Burns. A formal model for message passing systems. Technical report, Computer Science Dept., Indiana Univ., Bloomington, Ind., 1980. TR91.

8 Keren Censor-Hillel, Shir Cohen, Ran Gelles, and Gal Sela. Distributed computations in fully-defective networks. *Distributed Computing*, 36(4):501–528, 2023. doi:10.1007/s00446-023-00452-2.

9 Keren Censor-Hillel, Ran Gelles, and Bernhard Haeupler. Making asynchronous distributed computations robust to noise. *Distributed Computing*, 32(5):405–421, October 2019. doi:10.1007/s00446-018-0343-5.

10 Ernest Chang and Rosemary Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM*, 22(5):281–283, May 1979. doi:10.1145/359104.359108.

11 Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *Distributed Computing, 24th International Symposium, DISC 2010*, volume 6343 of *LNCS*, pages 148–162, 2010, Cambridge, MA, USA, September 13–15, 2010. Proceedings, 2010. Springer. doi:10.1007/978-3-642-15763-9_15.

12 Artur Czumaj and Peter Davies. Leader election in multi-hop radio networks. *Theoretical Computer Science*, 792:2–11, 2019. doi:10.1016/J.TCS.2019.02.027.

13 Peter Davies. Optimal message-passing with noisy beeps. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC*, pages 300–309, 2023, Orlando, FL, USA, June 19–23, 2023, 2023. ACM. doi:10.1145/3583668.3594594.

14 Danny Dolev, Maria Klawe, and Michael Rodeh. An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms*, 3(3):245–260, 1982. doi:10.1016/0196-6774(82)90023-2.

15 Fabien Dufoulon, Janna Burman, and Joffroy Beauquier. Beeping a deterministic time-optimal leader election. In *32nd International Symposium on Distributed Computing, DISC 2018*, volume 121 of *LIPIcs*, pages 20:1–20:17, New Orleans, LA, USA, October 15–19, 2018, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.DISC.2018.20.

16 Fabien Dufoulon, Janna Burman, and Joffroy Beauquier. Can uncoordinated beeps tell stories? In *PODC '20: ACM Symposium on Principles of Distributed Computing*, pages 408–417, Virtual Event, Italy, August 3–7, 2020, 2020. ACM. doi:10.1145/3382734.3405699.

17 M. El-Ruby, J. Kenevan, R. Carlson, and K. Khalil. A linear algorithm for election in ring configuration networks. In *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, volume i, pages 117–123 vol.1, Los Alamitos, CA, USA, 1991. IEEE Computer Society. doi:10.1109/HICSS.1991.183877.

18 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating amoebots via reconfigurable circuits. *Journal of Computational Biology*, 29(4):317–343, 2022. doi:10.1089/cmb.2021.0363.

19 Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Flaminia L. Luccio, and Nicola Santoro. Sorting and election in anonymous asynchronous rings. *Journal of Parallel and Distributed Computing*, 64(2):254–265, 2004. doi:10.1016/j.jpdc.2003.11.007.

20 Klaus-Tycho Förster, Jochen Seidel, and Roger Wattenhofer. Deterministic leader election in multi-hop beeping networks – (extended abstract). In *Distributed Computing – 28th International Symposium, DISC'14*, volume 8784 of *LNCS*, pages 212–226, 2014, Austin, TX, USA, October 12–15, 2014. Proceedings, 2014. Springer. doi:10.1007/978-3-662-45174-8_15.

21 Greg N. Frederickson and Nancy A. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1):98–115, January 1987. doi:10.1145/7531.7919.

22 Fabian Frei, Ran Gelles, Ahmed Ghazy, and Alexandre Nolin. Content-oblivious leader election on rings. *CoRR*, abs/2405.03646, 2024. URL: https://arxiv.org/abs/2405.03646, doi:10.48550/arXiv.2405.03646.

23 Ran Gelles. Coding for interactive communication: A survey. *Foundations and Trends® in Theoretical Computer Science*, 13(1–2):1–157, 2017. doi:10.1561/0400000079.

24 Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. In *SODA'13*, pages 748–766, New Orleans, Louisiana, USA, January 6–8, 2013, 2013. SIAM. doi:10.1137/1.9781611973105.54.

25 D. S. Hirschberg and J. B. Sinclair. Decentralized extrema-finding in circular configurations of processors. *Commun. ACM*, 23(11):627–628, November 1980. doi:10.1145/359024.359029.

26 Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990. doi:10.1016/0890-5401(90)90004-2.

27 Shay Kutten, Peter Robinson, Ming Ming Tan, and Xianbin Zhu. Improved tradeoffs for leader election. 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 355–365. ACM, 2023. doi:10.1145/3583668.3594576.

28 Gérard Le Lann. Distributed systems – towards a formal approach. In Bruce Gilchrist, editor, *Information Processing, Proceedings of the 7th IFIP Congress 1977*, pages 155–160, Toronto, Canada, August 8–12, 1977, 1977. North-Holland.

29 Gary L. Peterson. An $O(n \log n)$ unidirectional algorithm for the circular extrema problem. *ACM Trans. Program. Lang. Syst.*, 4(4):758–762, October 1982. doi:10.1145/69622.357194.

30 Nicola Santoro and Peter Widmayer. Distributed function evaluation in the presence of transmission faults. In Tetsuo Asano, Toshihide Ibaraki, Hiroshi Imai, and Takao Nishizeki, editors, *Algorithms, International Symposium SIGAL '90*, volume 450 of *Lecture Notes in*

Computer Science, pages 358–367, Tokyo, Japan, August 16–18, 1990, 1990. Springer. doi:10.1007/3-540-52921-7_85.

31 Violet Syrotiuk and Jan Pachl. A distributed ring orientation algorithm. In *International Workshop on Distributed Algorithms (WDAG'87)*, pages 332–336. Springer Berlin Heidelberg, 1988. doi:10.1007/BFb0019813.