Beyond 2-Edge-Connectivity: Algorithms and Impossibility for Content-Oblivious Leader Election

Yi-Jun Chang
National University of Singapore, Singapore

Lyuting Chen
National University of Singapore, Singapore

Haoran Zhou
National University of Singapore, Singapore

Abstract

The *content-oblivious model*, introduced by Censor-Hillel, Cohen, Gelles, and Sela (PODC 2022; Distributed Computing 2023), captures an extremely weak form of communication where nodes can only send asynchronous, content-less pulses. They showed that in 2-edge-connected networks, any distributed algorithm can be simulated in the content-oblivious model, provided that a unique leader is designated *a priori*. Subsequent works of Frei, Gelles, Ghazy, and Nolin (DISC 2024) and Chalopin et al. (DISC 2025) developed content-oblivious leader election algorithms, first for unoriented rings and then for general 2-edge-connected graphs. These results establish that all graph problems are solvable in content-oblivious, 2-edge-connected networks.

Much less is known about networks that are *not* 2-edge-connected. Censor-Hillel, Cohen, Gelles, and Sela showed that no non-constant function $f(x, y)$ can be computed correctly by two parties using content-oblivious communication over a single edge, where one party holds x and the other holds y. This seemingly ruled out many natural graph problems on non-2-edge-connected graphs.

In this work, we show that, with the knowledge of network topology G, leader election is possible in a wide range of graphs. Our main contributions are as follows:

Impossibility: *Graphs* symmetric about an edge admit no randomized terminating leader election algorithm, even when nodes have unique identifiers and full knowledge of G.

Leader election algorithms: *Trees* that are not symmetric about any edge admit a quiescently terminating leader election algorithm with topology knowledge, even in anonymous networks, using $O(n^2)$ messages, where n is the number of nodes. Moreover, even-diameter trees admit a terminating leader election given only the knowledge of the network diameter $D = 2r$, with message complexity $O(nr)$.

Necessity of topology knowledge: In the family of graphs $\mathcal{G} = \{P_3, P_5\}$, both the 3-path P_3 and the 5-path P_5 admit a quiescently terminating leader election if nodes know the topology exactly. However, if nodes only know that the underlying topology belongs to $\mathcal{G}$, then terminating leader election is impossible.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Asynchronous model, fault tolerance, quiescent termination

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.36

Related Version *Full Version*: `https://arxiv.org/abs/2511.23297` [10]

Funding The research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (24-1323-A0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.

1 Introduction

Distributed systems often operate under unreliable communication, where messages may be corrupted. In many networks, silicon-based or natural, the inherent presence of *alteration noise* means any *fault-tolerant* algorithm must take received messages "with a grain of salt", instead of relying on their content verbatim, whose integrity could have been compromised in transmission.

Content-oblivious model. To design a *fault-tolerant* algorithm that operates correctly even when facing the most extreme form of alteration noise, the work of Censor-Hillel, Cohen, Gelles, and Sela [7] adopted the idea of using *the sheer existence* of messages to encode information. To demonstrate the full power of their results, they first proposed the extremely weak model of *fully-defective* networks, where message content is subject to arbitrary, unbounded alteration; therefore, such content-less messages are aliased *pulses*, and any correct algorithm must completely ignore message content, hence *content-oblivious*. Also, the network is *asynchronous*, meaning that nodes lack access to a global clock; each pulse delivery takes an unpredictable time, thus forbidding the use of the presence/absence of pulses to encode information.

Universal solvability from 2-edge-connectivity. Naturally, one might conjecture that any non-trivial computational task is impossible when all messages are fully corrupted and arbitrarily delayed. However, Censor-Hillel, Cohen, Gelles, and Sela [7] showed otherwise: Assuming the presence of a *pre-elected* leader, any noiseless algorithm can be simulated in the content-oblivious setting, provided the network is *2-edge-connected*. The key idea is that the leader communicates with a recipient using two disjoint paths – one to transmit the message in unary, and the other to signal the end of the transmission. This ensures that the recipient never waits indefinitely for the message to conclude, allowing the computation to make progress. Next, the work of Frei, Gelles, Ghazy, and Nolin [18] showed how to elect a leader from scratch in an oriented ring as the simplest form of 2-edge-connected networks, while [8] removed the orientation requirement. Moreover, the latter [8] showed that even for general 2-edge-connected networks, with an upper bound on the network size N known, non-uniform leader election exists. Combined with the simulation result of Censor-Hillel, Cohen, Gelles, and Sela [7], their findings imply the solvability of all graph problems in a content-oblivious, 2-edge-connected network.

Beyond 2-edge-connectivity. On graphs that are not 2-edge-connected, Censor-Hillel, Cohen, Gelles, and Sela [7] showed the first negative result in the *vanilla* content-oblivious model. They showed that no non-constant function $f(x, y)$ can be computed correctly and deterministically by two parties using content-oblivious communication over a single edge, where one party holds x and the other holds y. This two-party impossibility immediately yields a general impossibility result for any network containing a bridge e, since we may let the two parties simulate the two connected components of $G - e$.

While this observation suggests that many natural graph problems – including leader election – might be impossible to solve on non-2-edge-connected graphs, it is not immediately clear what *specific* impossibility results can be derived directly from the two-party impossibility. In this work, we take a different perspective and investigate which tasks *can* still be performed on non-2-edge-connected graphs. Indeed, it may well be that many non-trivial tasks remain feasible in ways that are fully consistent with the two-party impossibility result.

1.1 Contribution and Roadmap

In this work, we show a *complete characterization* of content-oblivious leader election on trees with topology knowledge: Terminating leader election is possible *if and only if* the tree is not symmetric about an edge.

▶ **Definition 1** (Edge symmetry). *A graph G is symmetric about an edge $e = \{u, v\}$ if $G - e$ has exactly two connected components, H and K, containing u and v respectively, and there is an isomorphism $f : V(H) \to V(K)$ with $f(u) = v$.*

If a graph G is symmetric about an edge e, then e is necessarily a bridge, so G is non-2-edge-connected. On the feasibility side, in Section 3, we first design a content-oblivious leader election algorithm for even-diameter trees only (Theorem 2); following that, we extend the result to all trees that are not symmetric about any edge (Theorem 3).

▶ **Theorem 2** (Leader election on even-diameter trees). *There exists a terminating, anonymous content-oblivious leader election algorithm with message complexity $O(nr)$ for a tree with even diameter $D = 2r$ and n nodes, provided that each node knows the diameter D of the tree* a priori.

▶ **Theorem 3** (Leader election on general asymmetric trees). *There exists a quiescently terminating, anonymous content-oblivious leader election algorithm with message complexity $O(n^2)$ for a tree with n nodes, provided that each node knows tree topology G* a priori, *and the tree is not symmetric about any edge.*

Turning to the impossibility side, in Section 4, we show that for *graphs* that are symmetric about an edge, topology knowledge does not result in any randomized algorithm that significantly outperforms trivial guesswork.

▶ **Theorem 4** (Edge symmetry implies impossibility, randomized). *Let G be a network that is symmetric about an edge, where nodes are anonymous and know the network topology G. Then there exists no terminating, randomized leader-election algorithm which succeeds with probability greater than $\frac{1}{2}$.*

As a corollary, we obtain the analogous impossibility result in the deterministic setting where nodes have unique IDs. The proof of the corollary is in the full version [10] of the paper.

▶ **Corollary 5** (Edge symmetry implies impossibility, deterministic). *Let G be a network that is symmetric about an edge, and nodes are equipped with unique* IDs *and know the network topology G. Then there exists no terminating, deterministic content-oblivious leader election algorithm.*

In Section 5, we show that the *exact* knowledge of the network topology is indeed a necessary condition for terminating leader election on asymmetric trees.

▶ **Theorem 6** (Necessity of topology knowledge). *There exists no deterministic, terminating content-oblivious leader election algorithm if the underlying topology is drawn from the family $\mathcal{G} = \{P_3, P_5\}$, and $\mathcal{G}$ is known to the nodes* a priori, *even if the nodes have unique* IDs.

Finally, in the full version [10] of the paper, we present a simple *stabilizing* leader election algorithm that works for all trees, showing that the *termination* requirement is indeed *necessary* for the impossibility results.

▶ **Theorem 7** (Stabilizing leader election). *There exists a stabilizing content-oblivious leader election algorithm of message complexity $n + 2 \cdot \mathsf{ID}_{\max} - 1$ on a tree with n nodes with unique* IDs.

Technical overview. We briefly outline the main techniques underlying our leader election algorithms. Let D denote the diameter of the tree. In an even-diameter tree, if we repeatedly remove the leaves of the remaining tree, then after $D/2$ steps, only a single node remains. Our algorithm for even-diameter trees (Theorem 2) can be viewed as an implementation of this *leaf-peeling* process in the content-oblivious model, where the main challenge is handling asynchrony using only pulses. The key idea is to use the number of pulses sent to encode the height of a node.

For odd-diameter trees, the above leaf-peeling process leaves exactly two nodes. To elect a leader between them, we exploit the assumption that the tree is asymmetric. Our algorithm for general asymmetric trees (Theorem 3) uses the number of pulses to encode the topology of a subtree, thereby breaking symmetry between the two remaining nodes.

Finally, for our stabilizing algorithm for all trees (Theorem 7), we simply trim leaves in an arbitrary order until only two nodes remain, and then break symmetry between them by comparing their identifiers.

1.2 Additional Related Works

Leader election is one of the most fundamental problems in distributed systems, and has therefore enjoyed a long history of study (e.g., [9, 23, 26, 13, 24]). The works of [5, 31] study the conditions and impossibilities of breaking symmetry in anonymous networks, while [30, 12] examine symmetry breaking when nodes are labeled but ID collisions occur. A more recent result closely related to our work is that of Glacet, Miller, and Pelc [21], who studied the role of uniform advice in leader election. However, since their setting was the LOCAL *message-passing* model, advice served only to accelerate leader election rather than enable it, leading to a series of trade-offs between locality and advice size. By contrast, if advice is non-uniform, leader election becomes trivial when only a simple "yes/no" answer is required. For stronger definitions of leader election [22] and for a broader range of information dissemination problems, non-uniform advice, also known as informative labeling schemes [25], has been studied.

Similar to the content-oblivious setting, various other weak-communication models have been studied to capture systems with constrained communication capabilities. A notable example is the synchronous *beeping model* introduced by Cornejo and Kuhn [11], where during each round, nodes can broadcast a beep to their neighbors and can distinguish between silence and the presence of at least one beep from their neighbors.

In the beeping model, the first positive result for leader election was shown by Ghaffari and Haeupler [19], who also established an $\Omega(D + \log n)$-round lower bound. Subsequent works [17, 15, 16] gradually approached this lower bound. In parallel, the works of Gilbert and Newport [20] and Vacus and Ziccardi [29] focused on a different direction: Designing algorithms that operate on a constant number of states, at a cost of a slightly higher round complexity.

The *population protocol* introduced by Angluin et al. [3] studies massive systems of state machines under random pairwise interactions. Assuming each pair of agents has equal probability of interaction, a series of results [14, 1, 27] explores the trade-off between the number of states and the number of expected interactions for electing a leader. The works of Berenbrink, Giakkoupis, and Kling [4] and Sudo et al. [28] proposed optimal leader elections under different state-size constraints. A recent work of Alistarh, Rybicki, and Voitovych [2] extended the study to cover the setting where only a subset of agent pairs may interact.

The *leaf-peeling* technique is well known and has been used in the design of various tree algorithms. For example, Bruell, Ghosh, Karaata, and Pemmaraju [6] employed this approach to develop distributed algorithms for finding the centers and medians of a tree network.

2 Preliminaries

Content-oblivious model. Let $G = (V, E)$ be a communication network, where each node $v \in V$ is a computing device of unbounded computation power, and each edge $e \in E$ is a communication link. Messages exchanged by nodes are subject to unbounded alteration noise, therefore, effectively contentless. Equivalently, we say computing devices only exchange *pulses*. All pulses are subject to *unpredictable, unbounded delay*, despite that an eventual delivery is always guaranteed, and nodes are aware of the incoming port of a pulse received. Therefore, a notion of shared time *does not* provide additional power to the network: the content-oblivious network is *asynchronous*, meaning a computing device may only perform actions upon initiation of any algorithm, or upon receiving a pulse. Without loss of generality, we assume all local computations to be instantaneous. When measuring the message complexity of an algorithm, we consider the total number of pulses sent.

Model variants. Nodes in a content-oblivious network may either have unique positive integer IDs or be anonymous. In the latter case, the behavior of a node is completely determined by the pulses it has received so far. The algorithm itself may be deterministic or randomized; in the randomized setting, each node is given an *infinite* and *independent* stream of random bits.

Topology knowledge. We say that nodes are provided with topology knowledge of G if every node receives the same encoding of the topology of G *without* any ID information, so that no node can *immediately infer its position* in G from identifiers alone. However, a node may still deduce partial information about its position from its degree. For example, if G contains exactly two nodes of degree 3, then those two nodes can infer from the topology description that they must be among these two positions. When G is a tree, the underlying topology can be represented succinctly in $O(n)$ bits (e.g., using a balanced-parentheses encoding), where n is the number of nodes.

Leader election. We study the *leader election* problem, in which each node must decide between the outputs Leader and Non-Leader, and an execution of a leader election algorithm is successful if eventually exactly one node outputs Leader while all others output Non-Leader. We say that an algorithm *stabilizes* if, after some finite time from the initiation of the algorithm, the outputs of all nodes remain fixed. This notion of stabilization is purely analytic and global: nodes are not required to detect that they have stabilized.

A stronger guarantee is *termination*, which means that, after some finite time, each node *commits* to an output and halts. Termination is said to be *quiescent* if any node that has halted no longer receives pulses from its neighbors.

3 Terminating Anonymous Leader Election on Trees

In this section, we first prove Theorem 2 by providing a terminating algorithm for even-diameter trees, where network diameter D *is provided to nodes a priori*. Following that, we generalize the above result to prove Theorem 3 by providing a quiescently terminating algorithm where the underlying graph topology G *is provided to nodes a priori*.

3.1 Terminating Anonymous Leader Election on Even-Diameter Trees

In this section, we show a terminating leader election in anonymous, even-diameter trees. This algorithm requires each node to know the diameter $D = 2r$.

▶ **Theorem 2** (Leader election on even-diameter trees). *There exists a terminating, anonymous content-oblivious leader election algorithm with message complexity $O(nr)$ for a tree with even diameter $D = 2r$ and n nodes, provided that each node knows the diameter D of the tree* a priori.

We first provide a high-level overview of the algorithm. Each node performs a deterministic preprocessing phase locally using the information of D to derive a set of rules that the node uses for the upcoming election phase. Nodes hence monitor the triggering conditions of all rules, and perform (sending pulses and outputting leader election results) accordingly as the rules dictate. The pulse movement follows a bottom-top-bottom pattern. Leaf nodes initiate the election by sending pulses to their neighbors, and each node monitors the number of pulses it receives from each edge, constantly checking them against the set of rules derived. The center, which is the node with the minimum eccentricity, is unique for even-diameter trees. It triggers a special **Leader** rule and terminates as leader. It then broadcasts a pulse so that every other node terminates as non-leader.

In the above high-level description, we mentioned nothing about what a rule looks like or how to interpret a rule. To understand that, let us first provide an intuition of what pulses encode in our leader election.

A certain number of pulses sent from node u to node v along the edge $e = \{u, v\}$ encodes the *height of the subtree* rooted at u, that is, the connected component of u in $G - e$. A subtree with height i, where $0 \leq i \leq r - 1$, is encoded by $r - i$ pulses. To be elected as leader, a node must receive at least one pulse from each neighbor. This seemingly upside-down encoding is to ensure two important properties:

- **The center is the only node that can become a leader.** If we root the tree at any vertex $v \neq l$, then there exists a root-to-leaf path whose length exceeds r. Consequently, there is a subtree rooted at a child u of v whose height exceeds $r - 1$. In this case, v can never receive a valid pulse from u encoding that subtree, and thus can never claim leadership. This ensures the *correctness* of the leader election.
- **Before a leader is elected, all pulses travel in the direction from leaves to the center.** This property ensures that after a leader is elected, it can broadcast a pulse that unambiguously signals that a leader has been elected, hence making all other pulses terminate as non-leaders, ensuring the *termination* of all nodes.

We are now ready to formally describe the algorithm. Each node derives three categories of rules from the knowledge of $D = 2r$: **Upstream**, **Leader**, and **Downstream**. A node keeps track of the number of received pulses on each port and compares against the rules upon each update. Initially, only **Upstream** and **Leader** rules are active. After a node triggers at least one of **Upstream** and **Leader** rules, **Downstream** rules become active for that node.

- **Upstream**: There are r copies of **Upstream**, one for each $i \in [1, r]$. The ith rule is defined as follows:
  Whenever a node with d ports sees $d - 1$ of its ports (that is, all except one) each receive at least $i+1$ pulses while the remaining port receives *none*, mark the remaining port as Port$*$. Send pulses along Port$*$, until the *total number of pulses* sent through Port$*$, since the start of the algorithm, is exactly i. Add **Downstream** to the list of active rules, and remove the **Leader** rule.

Note that if a node has one port only, it automatically triggers the **Upstream** rule constructed for all $i \in [1, r]$ upon initiation, resulting in sending r pulses along its only port, which is marked Port$*$.

- **Leader**: Whenever a node with d ports sees each of its ports receive at least one pulse, send a pulse along all ports, declare Leader and terminate.
- **Downstream**: When a pulse is received from Port$*$, send a pulse along every port except for Port$*$, declare Non-Leader, and terminate.

To facilitate the analysis, we consider the *layer decomposition* resulting from the leaf-peeling process, defined as follows.

Algorithm 1 Layer decomposition.

```
i ← 0
G_0 ← G
while G_i is not empty do
    V_i ← set of leaf nodes of G_i
    G_{i+1} ← G_i \ V_i
    i ← i + 1
end
```

Since the radius of the network is r, the last non-empty layer V_r contains exactly one node (the center of the tree), which we denote by l in the subsequent discussion.

As a concrete example, consider a complete binary tree of radius r. The leader election proceeds as follows: First, by the **Upstream** rule, each node in V_0 (i.e., leaves) sends r pulses to its parent, who is in V_1. Each node in V_1 therefore receives r pulses from its neighbors in V_0, and by the **Upstream** rule, sends $r-1$ pulses to its parent, etc. Performing induction upwards, eventually the root receives at least one pulse from each of its neighbors, so it becomes the leader by triggering the **Leader** rule. After that, the **Downstream** rule will be triggered top-down: the now elected leader sends a pulse to each of its children, and then each node that receives a pulse from its parent triggers the **Downstream** rule, sending a pulse to each of its children, until the entire tree is reached.

While the preceding example provides intuition for the correctness of the algorithm on all even-diameter trees, we now present a formal proof that the constructed rules indeed yield a terminating leader election procedure.

▶ **Observation 8.** *For each $v \in V_i$ with $0 \le i \le r-1$, it has exactly one neighbor u in G that belongs to $\bigcup_{j>i} V_j$, while all other neighbors are in $\bigcup_{j<i} V_j$. For the unique node $l \in V_r$, it has all neighbors in $\bigcup_{j<r} V_j$.*

▶ **Definition 9** (Parent and children)**.** *For each $v \in V_i$, define the (possible) neighbor of v in G belonging to $\bigcup_{j>i} V_j$ as the parent of v. Define the neighbors of v in G belonging to $\bigcup_{j<i} V_j$ as the children of v.*

Clearly, the unique node $l \in V_r$ has no parent, and all nodes $v \in V_0$ have no children.

▶ **Observation 10.** *For each $v \in V_i$ with $1 \le i \le r$, it has at least one child that belongs to V_{i-1}.*

▶ **Lemma 11.** *Node l has at least two children in V_{r-1}.*

▶ **Observation 12.** *Once a node triggers an* ***Upstream*** *rule, it never changes* Port∗.

Proof. To assign a port (without loss of generality, port 0) as Port∗, a node must trigger an **Upstream** rule at a time when port 0 has not yet received any pulses. Hence, no other port can have been assigned Port∗ earlier, since doing so would require port 0 to have already received a nonzero number of pulses in order to trigger some **Upstream** rule. ◀

▶ **Lemma 13.** *If a node u is in V_i, $i \in [0, r-1]$, then u sends at most $r-i$ pulses to its parent.*

Proof. We proceed by induction over the layers from V_0 to V_r, i.e., in a bottom-up manner.

Base case. Let u be a leaf node in V_0. By the definition of **Upstream**, u sends r pulses to its parent immediately upon initialization and never sends any more thereafter. Thus, the lemma holds for u.

Induction step. Assume the lemma holds for all nodes in layers up to V_s for some $s \in [0, r-2]$. Let u be a node in V_{s+1}, and let v be its parent.

Suppose that u sets Port∗ to point toward v (otherwise, the lemma holds trivially for u). By Observation 10, u has exactly one child m in layer V_s. By the induction hypothesis, m sends at most $r-s$ pulses to u. By the definition of the **Upstream** rules, u can therefore send at most $r-s-1$ pulses through Port∗ to its parent v. ◀

▶ **Lemma 14.** *Node l cannot trigger any* **Upstream** *rule.*

Proof. By Lemma 11, node l has at least two children in V_{r-1}, each sending at most one pulse to l by Lemma 13. By the definition of **Upstream**, l cannot trigger any **Upstream** rule. ◀

▶ **Lemma 15.** *If a node u is in V_i with $i \in [0, r-1]$, then u does not receive any pulses from its parent v before sending one to v.*

Proof. We proceed by induction over the layers from V_{r-1} to V_0, i.e., in a top-down manner.

Base case. Consider any $v \in V_{r-1}$, whose parent is l. Suppose, for the sake of contradiction, that v receives a pulse from l before sending any pulse to l. Such a pulse cannot result from a triggered **Upstream** rule at l, by Lemma 14. It also cannot arise from a triggered **Leader** rule, since that rule requires l to have received pulses from *all* of its ports, yet v has not sent any pulse to l. This shows that v cannot receive a pulse from l before sending one.

Induction step. Assume that the lemma holds for all nodes in V_s and higher layers for some $s \in [1, r-1]$. Let u be a node in V_{s-1} and v its parent. Suppose, for the sake of contradiction, that u receives a pulse from v before sending one to v. Similar to the base case, v cannot trigger the **Leader** rule. Thus, it remains to rule out the possibility that v triggers an **Upstream** rule and sets the port pointing to u as Port∗.

For v to trigger an **Upstream** rule, it must have received a nonzero number of pulses from its parent w. By the induction hypothesis, v must have sent pulses to w before receiving any, and such pulses can only result from an **Upstream** rule with the port pointing to w designated as Port∗. This, however, contradicts Observation 12. Hence, the assumed scenario is impossible, and u also satisfies the lemma. ◀

The following three lemmas follow directly from Lemma 15.

▶ **Lemma 16.** *The only node that can trigger the* **Leader** *rule is l.*

Proof. To trigger the **Leader** rule, a node v must receive pulses from all neighbors before sending any. For any node v with a parent, the above is impossible due to Lemma 15, so the only node that can trigger the **Leader** rule is l, as it is the only node without a parent. ◀

▶ **Lemma 17.** *For each node v that has a parent, its* Port$*$ *(if any) may only point to its parent.*

Proof. Suppose Port$*$ points to a child. For this to occur, v must have triggered an **Upstream** rule in which it received pulses from its parent. By Lemma 15, v must then have previously sent a pulse to its parent, and this pulse cannot be the result of the **Leader** rule by Lemma 16. Thus, v must have triggered an **Upstream** rule and sent pulses to its parent earlier, which contradicts Observation 12. ◀

▶ **Lemma 18.** *No* **Downstream** *rule is triggered before l triggers the* **Leader** *rule.*

Proof. We proceed by induction over the layers from V_r to V_0, i.e., in a top-down manner.

Base case. The unique node l in V_r never triggers any **Upstream** rule (Lemma 14), and therefore **Downstream** is never activated at l.

Induction step. Assume that the lemma holds for all nodes in V_s and higher layers for some $s \in [1, r]$. Let u be a node in V_{s-1} and let v be its parent. For u to trigger **Downstream**, it must receive a pulse from Port$*$, which, by Lemma 17, means a pulse from its parent v. Such a pulse cannot result from v triggering **Upstream** or **Leader**, by Lemma 17 and Lemma 16, respectively. Hence, v must have triggered **Downstream**, and by the induction hypothesis, this implies that **Leader** has already been triggered by l. ◀

Combining the above three lemmas, we see that *the direction of pulse movement is always from children to parents*, before l triggers the **Leader** rule.

▶ **Lemma 19.** *The direction of pulse movement is always from children to parents, before l triggers the* **Leader** *rule.*

Proof. By Lemma 16, no node other than l ever triggers the **Leader** rule. Moreover, by Lemma 18, no node other than l triggers the **Downstream** rule before l triggers the **Leader** rule. Thus, the **Upstream** rules are the only rules that nodes other than l may trigger before l triggers the **Leader** rule. By Lemma 17, any pulse generated by such a **Upstream** rule always travels toward the parent of the sender. ◀

▶ **Lemma 20.** *Node l eventually terminates as* Leader *by triggering the* **Leader** *rule, after every other node sets* Port$*$ *to its parent.*

Proof. We again apply induction over the layers from V_0 to V_r. Our goal is to show that every $v \in V_i$ sends *at least* $r - i$ pulses to its parent. In particular, this implies that each child of l sends at least one pulse to l, since all of them belong to V_{r-1} or a lower layer. Consequently, l will trigger the **Leader** rule.

Base case. It is immediate that $u \in V_0$ sends r pulses to its parent, by the definition of **Upstream**.

Induction step. Assume the lemma holds for all nodes up to V_s for some $s \in [0, r-1]$. Let u be a node in V_{s+1}. By the induction hypothesis, every child of u, all of which lie in V_s or a lower layer, sends at least $r - s$ pulses to u. By the definition of the **Upstream** rules, u will therefore eventually send $r - s - 1$ pulses to its parent. We emphasize that the correctness of this induction step relies on Lemma 19, which ensures that we need only consider pulses traveling from children to parents. ◀

The above proof, together with Lemma 13, implies that every $v \in V_i$ sends *exactly* $r - i$ pulses to its parent. The argument is divided into two lemmas rather than one because the induction in the proof of Lemma 20 relies crucially on Lemma 19, which itself depends on Lemma 13.

▶ **Lemma 21.** *Each node other than l eventually terminates as* Non-Leader*.*

Proof. By Lemma 19, every non-l node sets Port$*$ to point to its parent. Immediately after l triggers **Leader** (which occurs by Lemma 20), each child of l receives a pulse through its Port$*$ and therefore triggers **Downstream**. Any node that triggers **Downstream** sends a pulse along every port except Port$*$, so each of its children likewise receives a pulse through its own Port$*$ and triggers **Downstream**. This recursive propagation ensures that every non-l node eventually triggers **Downstream** and terminates as Non-Leader. ◀

▶ **Lemma 22.** *The total number of pulses sent by all nodes in this algorithm is $O(nr)$.*

Proof. Each node other than l sends at most r pulses to its parent via the **Upstream** rules (see Lemma 19), contributing at most $(n-1)r$ pulses in total. In addition, the nodes send exactly $n-1$ pulses due to the **Leader** and **Downstream** rules, since exactly one such pulse is sent across each edge and a tree has $n-1$ edges (see the proof of Lemma 21). Therefore, the total number of pulses sent is at most $(n-1)r + (n-1) = O(nr)$. ◀

▶ **Theorem 2** (Leader election on even-diameter trees)**.** *There exists a terminating, anonymous content-oblivious leader election algorithm with message complexity $O(nr)$ for a tree with even diameter $D = 2r$ and n nodes, provided that each node knows the diameter D of the tree* a priori*.*

Proof of Theorem 2. The correctness and termination of the leader election follow from Lemmas 20 and 21, while the message complexity follows from Lemma 22. ◀

Failing the quiescence guarantee. The algorithm above does not guarantee quiescent termination. Recall from the proofs of Lemmas 13 and 20 that any node $u \in V_{s+1}$, for $s \in [0, r-1]$, will eventually send exactly $r - s - 1$ pulses to its parent. For this to occur, each child of u must have sent u $r - s$ pulses. However, some children of u may lie in layers strictly below V_s, and thus will ultimately send u more than $r - s$ pulses. Crucially, u cannot determine whether such additional pulses will arrive in the future. Consequently, some pulses from the children of u may reach u after u has already terminated, and therefore quiescence cannot be guaranteed.

3.2 Quiescently Terminating Anonymous Leader Election on Asymmetric Trees

In this section, we show a quiescently terminating leader election in anonymous trees, unless they are symmetric about an edge. This algorithm, however, requires each node to know the full topology of G. In the remainder of the section, we will make the distinction between nodes, which are real computing devices performing content-oblivious leader election, and vertices of the topology knowledge G.

In contrast with the algorithm for even-diameter trees, the main challenge here is to break symmetry within layer V_r, which contains two vertices when the tree diameter is odd. To resolve this symmetry, we use the assumption that G is not symmetric about an edge, implying that the two nodes in V_r have distinct views of the lower-layer nodes. Rather than using the number of pulses to encode only the *height* of a subtree, our algorithm uses the number of pulses to encode the full *shape* of each subtree. A pleasant consequence of this more refined encoding is that the algorithm now achieves quiescent termination.

▶ **Theorem 3** (Leader election on general asymmetric trees). *There exists a quiescently terminating, anonymous content-oblivious leader election algorithm with message complexity $O(n^2)$ for a tree with n nodes, provided that each node knows tree topology G* a priori*, and the tree is not symmetric about any edge.*

The remainder of this section is organized as follows. In Section 3.2.1, we detail the layer decomposition, enumeration of distinct subtrees, and construction of rules. In Section 3.2.2, we prove that the above-constructed rules indeed result in a quiescently terminating leader election.

3.2.1 Algorithm Description

Each node first identifies the center(s) of the advised topology G. To do so, every node locally performs the layer decomposition (Algorithm 1) using the advice G, thereby partitioning the vertex set V into disjoint layers $V_0, V_1, \ldots, V_r$. We emphasize that, unlike in the even-diameter algorithm, where the layer decomposition is used only in the analysis, here *each node actually performs the layer decomposition locally.*

▶ **Definition 23** (Subtrees). *For each vertex $v \in V_i$, define $G_{\le v}$ as the induced subgraph of G over the vertex set $\{v\} \cup (\bigcup_{j<i} V_j)$. Let T^v be the connected component of $G_{\le v}$ that contains v. We call T^v the subtree rooted at v.*

We first enumerate all distinct subtrees in an "increasing" order, and denote them by $T_1, T_2, \ldots$ in the order in which they are listed. We stress the distinction between subscripts and superscripts: we use a superscript v to denote the subtree rooted at v, i.e., T^v, and a subscript i to index the enumeration, i.e., T_i.

Each (rooted) subtree isomorphic class is enumerated only once. Note that if $v \in V_i$, then T^v has height i; hence, two subtrees can be isomorphic only if their roots lie in the same layer. Conceptually, to carry out the enumeration, it suffices to define a comparator on subtrees. The rules of this comparator are specified as follows.

Subtree comparator. The subtree comparator takes two non-isomorphic subtrees T^v and T^u as input and determines whether $T^v \prec T^u$ or $T^u \prec T^v$. The enumeration of subtrees is required to be consistent with this comparator: if $T^v \prec T^u$, then T^v must be listed before T^u. We will later show that the comparator is transitive, ensuring that a valid enumeration indeed exists. For isomorphic subtrees, we write $T^v = T^u$, and we use $T^v \preceq T^u$ to denote either $T^v = T^u$ or $T^v \prec T^u$.

Given inputs T^v and T^u, the comparator follows the three rules below.

Low layer first: Assume v and u are not of the same layer. If v is from a lower layer, then the comparator outputs $T^v \prec T^u$, and vice versa.

Fewer children first: Assume v and u are of the same layer. Let $N_{T^v}(v)$ (resp., $N_{T^u}(u)$) be the set of neighbors of v (resp., u) in T^v (resp., T^u). If $|N_{T^v}(v)| < |N_{T^u}(u)|$, then the comparator outputs $T^v \prec T^u$, and vice versa.

Recursion rule: Assume v and u lie in the same layer and have the same number of children in their respective rooted subtrees, i.e., $|N_{T^v}(v)| = |N_{T^u}(u)|$. Let the children of v be $v_1, v_2, \ldots, v_{d-1}$, ordered so that

$$T^{v_1} \preceq T^{v_2} \preceq \cdots \preceq T^{v_{d-1}}.$$

Similarly, list the children of u as $u_1, u_2, \ldots, u_{d-1}$, ordered according to $\preceq$.
Now consider the smallest index i for which T^{v_i} and T^{u_i} are not isomorphic. If $T^{v_i} \prec T^{u_i}$, then comparator outputs $T^v \prec T^u$, and vice versa.

▶ **Lemma 24.** *The subtree comparator is well-defined in the following sense: (1) the comparator does not recurse infinitely and (2) the comparator is transitive.*

Proof. The proof is deferred to the full version [10] of the paper due to the page limit. ◀

Subtree enumeration. Now all nodes obtain identical enumerations: $T_1, T_2, \ldots T_k$, where k is the number of distinct (non-isomorphic) subtrees appearing in G.

For convenience of presentation, we define a helper function that maps each vertex v to the index of the subtree rooted at v in the enumerated list.

▶ **Definition 25.** *For a vertex v, define $\tau(v) = i$ such that T^v is isomorphic to T_i.*

We also define a helper function that maps the index i to the integer $k - i$.

▶ **Definition 26.** *Define $\lambda(i) = k - i$.*

Write $\lambda\tau(v)$ as shorthand for $\lambda(\tau(v))$. By definition, we have $\lambda\tau(v) < \lambda\tau(u)$ if and only if $T^v \prec T^u$, and $\lambda\tau(v) = \lambda\tau(u)$ if and only if T^v is isomorphic to T^u.

We are now ready to present the rules for our algorithm. Each node derives three categories of rules from the subtree enumeration: **Upstream**, **Leader**, and **Downstream**. Initially, only **Upstream** and **Leader** rules are active. After a node triggers at least one of **Upstream** and **Leader** rules, **Downstream** rules become active for that node.

- **Upstream**: One rule is constructed for each enumerated subtree, *except for the last one*. For each T_i with $i \leq k - 1$, let the neighbors of the root be $m_1, m_2, \ldots, m_{d-1}$. Create the rule

  $$[\lambda\tau(m_1), \lambda\tau(m_2), \ldots, \lambda\tau(m_{d-1})] \;\to\; \lambda(i).$$

  This rule is interpreted as follows: whenever a node with d ports observes that $d - 1$ of its ports have respectively received

  $$\lambda\tau(m_1), \lambda\tau(m_2), \ldots, \lambda\tau(m_{d-1})$$

  pulses, while the remaining port has received *none*, it marks that remaining port as Port$*$ and sends pulses along Port$*$ until the total number of pulses sent through it is exactly $\lambda(i)$. The node then adds **Downstream** to its list of active rules and disables the **Leader** rule.
  Note that in the special case of T_1, the subtree consists of a single leaf node. The rule generated by T_1 is therefore equivalent to the following: if a node has only one port, it sends $\lambda(1) = k - 1$ pulses along that port *immediately upon initialization.*

- **Leader** (Odd diameter): (This rule is added if G has odd diameter.) For T_k, consider the neighbors of the root $m_1, m_2, \ldots, m_{d-1}$. The rule is as follows: whenever a node with d ports sees that $d-1$ of its ports have respectively received

  $$\lambda\tau(m_1),\ \lambda\tau(m_2),\ \ldots,\ \lambda\tau(m_{d-1})$$

  pulses, *while the remaining port has received one pulse*, the node sends a pulse along all ports, declares Leader, and terminates.
- **Leader** (Even diameter): (This rule is added if G has even diameter.) For T_k, consider the neighbors of the root $m_1, m_2, \ldots, m_d$. The rule is as follows: whenever a node with d ports sees *all of its ports* receive, respectively,

  $$\lambda\tau(m_1),\ \lambda\tau(m_2),\ \ldots,\ \lambda\tau(m_d)$$

  pulses, the node sends a pulse along all ports, declares Leader, and terminates.
- **Downstream**: When a pulse is received from Port$*$, the node sends a pulse along every port except Port$*$, declares Non-Leader, and terminates.

Manual parent assignment. Recall that we defined parent and children in Definition 9. If the advised topology G has even diameter, then $V_r = \{l\}$ is a singleton set; apart from l, each vertex has its parent assigned, and we aim to elect the node corresponding to l as the leader. If G has odd diameter, V_r contains two vertices l_1 and l_2, which both have no parents. The fact that G is not symmetric over any edge enables us to decide a leader between l_1 and l_2: in fact, T^{l_1} must not be isomorphic to T^{l_2}, and they are exactly the last two trees enumerated due to belonging to the highest layer r. Without loss of generality, we assume $T^{l_1} = T_{k-1}$, $T^{l_2} = T_k$ in the remainder of this section. To make proofs more concise, we manually assign l_2 as the parent of l_1, and l_1 as a child of l_2. At no risk of confusion, if the diameter is odd, define $l = l_2$. In either case, the only vertex *without a parent* is l, and we call it the *root* of the tree. Our goal is to elect the node corresponding to the root vertex l as leader.

3.2.2 Proof of Correctness

We now prove the correctness of the algorithm.

▶ **Lemma 27 (Upstream rules are well-defined).** *Consider any two distinct **Upstream** rules, generated by T_j and $T_{j'}$, whose roots each have the same number of neighbors:*

$$[\lambda\tau(m_1), \lambda\tau(m_2), \ldots, \lambda\tau(m_{d-1})] \to \lambda(j) \quad \textit{and} \quad [\lambda\tau(m'_1), \lambda\tau(m'_2), \ldots, \lambda\tau(m'_{d-1})] \to \lambda(j').$$

Without loss of generality, assume each list of $\lambda\tau$-values is sorted in non-increasing order. If

$$\lambda\tau(m_1) \geq \lambda\tau(m'_1),\ \lambda\tau(m_2) \geq \lambda\tau(m'_2),\ \ldots,\ \lambda\tau(m_{d-1}) \geq \lambda\tau(m'_{d-1}),$$

then $\lambda(j) > \lambda(j')$.

Proof. The definition of λ implies that

$$\tau(m_1) \leq \tau(m'_1),\ \tau(m_2) \leq \tau(m'_2),\ \ldots,\ \tau(m_{d-1}) \leq \tau(m'_{d-1}).$$

To prove $\lambda(j) > \lambda(j')$, it suffices to show that $j < j'$, i.e., that T_j is enumerated before $T_{j'}$. Since the respective roots of T_j and $T_{j'}$ have the same degree, the **Fewer children first** rule does not apply. Thus, the comparison must be determined by either the **Low layer first** rule or the **Recursion rule**.

If the root of T_j lies in a lower layer than the root of $T_{j'}$, then $T_j \prec T_{j'}$ immediately by the **Low layer first** rule.

Now consider the case where the two roots lie in the same layer. Since the **Upstream** rules under consideration are distinct, there exists a smallest index i such that $\tau(m_i) < \tau(m'_i)$, meaning that T^{m_i} is enumerated before $T^{m'_i}$, and hence $T^{m_i} \prec T^{m'_i}$. By the **Recursion rule**, we then obtain $T_j \prec T_{j'}$, so T_j is enumerated first.

It remains to show that it is impossible for the root of T_j to lie in a *higher* layer than the root of $T_{j'}$. Since each list of $\lambda\tau$-values is sorted in non-increasing order, we have

$$\tau(m_1) \leq \tau(m_2) \leq \cdots \leq \tau(m_{d-1}).$$

Thus $T^{m_{d-1}}$ is the last among these subtrees to be enumerated, so m_{d-1} is a highest-layer child of the root of T_j. By Observation 10, m_{d-1} must lie exactly one layer below the root of T_j, and therefore must lie in a higher layer than m'_{d-1}. Hence $\tau(m_{d-1}) > \tau(m'_{d-1})$, contradicting the fact that $\tau(m_{d-1}) \leq \tau(m'_{d-1})$. ◀

The following observation still applies.

▶ **Observation 12.** *Once a node triggers an **Upstream** rule, it never changes* Port$*$.

Lemma 27 and Observation 12 show that our **Upstream** rules are *well-defined* in the following sense: While a node can trigger multiple **Upstream** rules throughout the election process, an **Upstream** rule triggered earlier does not send more pulses along Port$*$ than an **Upstream** rule triggered later.

▶ **Lemma 28.** *Let u be a node with $u \neq l$. Then u sends at most $\lambda\tau(u)$ pulses to its parent.*

Proof. Recall from the definition of **Upstream** that if the rule constructed from the $\tau(u)$th enumerated tree $T_{\tau(u)} = T^u$ is triggered, $\lambda\tau(u)$ pulses are sent to Port$*$. The only possible way for u to send additional pulses would be to trigger another **Upstream** rule and send $\lambda\tau(u') > \lambda\tau(u)$ pulses to v. We will show that this cannot occur. We proceed by induction over layers, from V_0 to V_r, i.e., in a bottom-up manner.

Base case. Let u be a leaf node in V_0. Then T^u is a single-node tree and must be the first to be enumerated, i.e., $\tau(u) = 1$. By the **Upstream** rule constructed from it, u sends $k - 1 = \lambda\tau(u)$ pulses to its parent immediately upon initialization and never sends any additional pulses. Hence, the lemma holds for u.

Induction step. Assume the lemma holds for all nodes in V_s and lower layers for some $s \in [0, r-1]$. Let u be a node in V_{s+1} with $u \neq l$, and let v be its parent. Note that $\tau(u) \leq k - 1$, so $\lambda\tau(u) \geq 1$.

Suppose u sets Port$*$ to point to v (otherwise the lemma follows immediately for u). Let the children of u be $\{m_1, m_2, \ldots, m_{d-1}\}$ with

$$\lambda\tau(m_1) \geq \lambda\tau(m_2) \geq \cdots \geq \lambda\tau(m_{d-1}).$$

Assume, for the sake of contradiction, that u eventually triggers some **Upstream** rule sending $\lambda\tau(u') > \lambda\tau(u)$ pulses to v. Let the children of u' be $\{m'_1, m'_2, \ldots, m'_{d-1}\}$, ordered so that

$$\lambda\tau(m'_1) \geq \lambda\tau(m'_2) \geq \cdots \geq \lambda\tau(m'_{d-1}).$$

Since each child of u lies in a layer below V_{s+1}, the induction hypothesis implies that each m_i sends at most $\lambda\tau(m_i)$ pulses to u. Consequently,

$$\lambda\tau(m_1) \geq \lambda\tau(m'_1),\ \lambda\tau(m_2) \geq \lambda\tau(m'_2),\ \ldots,\ \lambda\tau(m_{d-1}) \geq \lambda\tau(m'_{d-1}).$$

By Lemma 27, we have $\lambda\tau(u') < \lambda\tau(u)$, contradicting the assumption that $\lambda\tau(u') > \lambda\tau(u)$. ◀

▶ **Lemma 29.** *Let u be a node with $u \neq l$. Then u does not receive any pulses from its parent v before sending one to v.*

Proof. We first show that l cannot trigger any **Upstream** rule. Recall that $T^l = T_k$ is the last enumerated subtree; let the children of l be $\{m_1, m_2, \ldots, m_d\}$, ordered so that $\lambda\tau(m_1) \geq \lambda\tau(m_2) \geq \cdots \geq \lambda\tau(m_d)$. By Lemma 28, node l receives at most $\lambda\tau(m_1), \lambda\tau(m_2), \ldots, \lambda\tau(m_d)$ pulses from its children. Consider any **Upstream** rule associated with some node u' of degree d, whose children are $\{m'_1, m'_2, \ldots, m'_{d-1}\}$, with $\lambda\tau(m'_1) \geq \lambda\tau(m'_2) \geq \cdots \geq \lambda\tau(m'_{d-1})$.

Suppose first that u' lies in a layer lower than or equal to $r-1$. Then, by Observation 10, the child m'_{d-1} must lie in a layer at least as high as $r-2$. Meanwhile, regardless of whether the diameter is even or odd, the two children m_{d-1} and m_d of l lie in layers at least as high as $r-1$. Hence, $\lambda\tau(m'_{d-1}) > \lambda\tau(m_{d-1}) \geq \lambda\tau(m_d)$, which shows that l cannot trigger the **Upstream** rule for the subtree rooted at u'.

Now suppose instead that u' also lies in layer r. Then $T^{u'} = T_{k-1}$, and moreover the last child of l, namely m_d, is exactly u'. By the **Recursion rule** of the subtree comparator, $T^{u'}$ is ordered before T^l, which implies there exists a smallest index i such that $\lambda\tau(m_i) < \lambda\tau(m'_i)$. Thus l again cannot satisfy the **Upstream** rule for the subtree rooted at u'.

Next, we proceed by induction over the layers to prove the lemma, from V_r to V_0, i.e., in a top-down manner.

Base case. In V_r, the (possibly) only node we need to consider is l_1, whose parent is l. Suppose, for the sake of contradiction, that l_1 receives a pulse from l before sending one to l. Such a pulse cannot result from a triggered **Upstream** rule, as established above. It also cannot arise from a triggered **Leader** rule, since that would require l to have received pulses from *all* its ports, yet l_1 has not sent any pulses to l. This yields a contradiction.

Induction step. Assume that the lemma holds for all nodes in V_s and higher layers for some $s \in [1, r]$. Let u be a node in V_{s-1} and let v be its parent. Suppose, for the sake of contradiction, that u receives a pulse from v before sending one to v. As in the base case, v cannot trigger the **Leader** rule. Thus, it remains to rule out the possibility that v triggers an **Upstream** rule and sets the port pointing to u as Port$*$.

If $v = l$, then the same argument as in the base case yields a contradiction. If instead v has a parent w, then for v to trigger **Upstream**, it must have received a non-zero number of pulses from w. By the induction hypothesis, v must have sent pulses to w before receiving any, and such pulses can only result from an **Upstream** rule with the port pointing to w designated as Port$*$. This contradicts Observation 12. Hence, u also satisfies the lemma. ◀

Observe that the following four lemmas continue to apply. We omit the proofs, as they follow from arguments nearly identical to those used previously.

▶ **Lemma 16.** *The only node that can trigger the* **Leader** *rule is l.*

▶ **Lemma 17.** *For each node v that has a parent, its* Port$*$ *(if any) may only point to its parent.*

▶ **Lemma 18.** *No* **Downstream** *rule is triggered before l triggers the* **Leader** *rule.*

▶ **Lemma 19.** *The direction of pulse movement is always from children to parents, before l triggers the* **Leader** *rule.*

The next lemma follows from the way we constructed **Upstream** and **Leader** rules.

▶ **Lemma 30.** *Node l may trigger the* **Leader** *rule only after every node $v \neq l$ has triggered the rule corresponding to its subtree $T_{\tau(v)} = T^v$ and has set* Port$*$ *to point to its parent.*

Proof. Lemma 19 establishes that, before l triggers **Leader**, pulses travel from children to parents. Therefore, we only need to consider pulses directed as such, as we are reasoning about a necessary condition for l to trigger **Leader**.

Base case. First, let the children of l be $\{m_1, m_2, \ldots, m_d\}$, ordered so that

$$\lambda\tau(m_1) \geq \lambda\tau(m_2) \geq \cdots \geq \lambda\tau(m_d).$$

By the definition of the **Leader** rule, l must receive

$$\lambda\tau(m_1),\ \lambda\tau(m_2),\ \ldots,\ \lambda\tau(m_d)$$

pulses from its children, respectively, in order to trigger **Leader**. Since Lemma 28 shows that each child m_i can send at most $\lambda\tau(m_i)$ pulses to l, it follows (by an induction on i from d down to 1) that l *may* trigger the **Leader** rule *only after* each child m_i has triggered the **Upstream** rule corresponding to its own subtree $T_{\tau(m_i)} = T^{m_i}$.

Induction step. Let the children of a node $v \neq l$ be $\{m_1, m_2, \ldots, m_{d-1}\}$, ordered so that

$$\lambda\tau(m_1) \geq \lambda\tau(m_2) \geq \cdots \geq \lambda\tau(m_{d-1}).$$

Assume it has been established that l *may* trigger the **Leader** rule *only after* v triggers the **Upstream** rule defined by $T_{\tau(v)} = T^v$.

By the definition of the **Upstream** rule constructed from $T_{\tau(v)} = T^v$, node v must receive

$$\lambda\tau(m_1),\ \lambda\tau(m_2),\ \ldots,\ \lambda\tau(m_{d-1})$$

pulses from its $d-1$ children in order to trigger the **Upstream** rule of $T_{\tau(v)} = T^v$. Meanwhile, by Lemma 28, each child m_i can send *at most* $\lambda\tau(m_i)$ pulses to v. Similarly, it follows that v *may* trigger the **Upstream** rule corresponding to $T_{\tau(v)} = T^v$ *only after* each child m_i has triggered its own **Upstream** rule associated with $T_{\tau(m_i)} = T^{m_i}$.

Therefore, we have extended the necessary condition for l to trigger **Leader** from node $v \neq l$ to all children of v. ◀

The above lemma does not guarantee that l *will* trigger the **Leader** rule. It only points out a condition for l to trigger the **Leader** rule. To show that this condition is eventually satisfied and that l indeed triggers the **Leader** rule, we present the next lemma.

▶ **Lemma 31.** *Every node v other than l eventually triggers the* **Upstream** *rule corresponding to its subtree $T_{\tau(v)} = T^v$ and sets* Port$*$ *to point to its parent. After this has occurred for all nodes $v \neq l$, node l will trigger the* **Leader** *rule. At the moment* **Leader** *is triggered, the tree is in quiescence, i.e., no pulses are in transit.*

Proof. We again apply induction over layers, from V_0 to V_r, to show that every node u other than l eventually triggers the **Upstream** rule corresponding to its subtree $T_{\tau(u)} = T^u$ and sets Port$*$ to point to its parent. To do so, it suffices to show that u eventually sends $\lambda\tau(u)$ pulses to its parent, as the **Upstream** rules are well-defined (Lemma 27 and Observation 12).

Indeed, once every node $v \neq l$ has sent $\lambda\tau(v)$ pulses to its parent, node l receives enough pulses from its neighbors to trigger the **Leader** rule, as required.

Base case. It is immediate that any node $u \in V_0$ eventually sends $\lambda\tau(u) = \lambda(1) = k - 1$ pulses to its parent, as this follows directly from the definition of the **Upstream** rule applied to degree-1 nodes.

Induction step. Assume the lemma holds for all nodes up to V_s for some $s \in [0, r-1]$. Let u be a node in V_{s+1}. By the induction hypothesis, the children of u, denoted $\{m_1, m_2, \ldots, m_{d-1}\}$, eventually send $\lambda\tau(m_1), \lambda\tau(m_2), \ldots, \lambda\tau(m_{d-1})$ pulses to u, respectively. This causes the **Upstream** rule corresponding to the subtree $T_{\tau(u)} = T^u$ to trigger, so u sends $\lambda\tau(u)$ pulses through Port$*$ to its parent.

During this process, u does not receive any unexpected pulses from its parent. The possibilities of u getting such a pulse from its parent due to **Upstream**, **Leader**, or **Downstream** are ruled out by combining Lemma 16, Lemma 17, Lemma 18 (or equivalently, just Lemma 19), and Lemma 30.

Finally, once the **Upstream** rule for $T_{\tau(u)} = T^u$ has been triggered, the entire subtree rooted at u reaches quiescence: each child of u has already sent the maximum number of pulses permitted by Lemma 28, and all such pulses have been received. ◀

We remark that the above proof, combined with Lemma 28, shows that the number of pulses every node $v \neq l$ sends to its parent due to the **Upstream** rules is *exactly* $\lambda\tau(v)$.

▶ **Lemma 32.** *All nodes other than l eventually terminate quiescently as* Non-Leader *after l terminates quiescently as* Leader*.*

Proof. We first observe that by Lemma 31, the node l eventually terminates as Leader, and at that moment the tree is in quiescence. For all remaining nodes, we assume that l has already terminated as Leader and proceed by induction from V_r down to V_0.

Base case. For the layer V_r, the only node we may need to consider is l_1. By Lemma 31, the node l_1 has already triggered the **Upstream** rule corresponding to $T_{\tau(l_1)}$ and set Port$*$ to point to l before l terminates as Leader and sends a pulse to l_1. Consequently, l_1 will eventually receive a pulse from Port$*$, trigger the **Downstream** rule, and terminate as Non-Leader. At this point, l has already terminated and will not send further pulses to l_1, and by Lemma 28, l_1 will not receive additional pulses from its children. Thus, l_1 reaches quiescent termination.

Induction step. Assume the lemma holds for all nodes in V_s or higher layers, for some $s \in [1, r]$. Let u be a node in V_{s-1}. By Lemma 31, u must have triggered the **Upstream** rule corresponding to $T_{\tau(u)}$ and set Port$*$ to its parent v. Since v lies in a higher layer, the induction hypothesis ensures that v eventually triggers the **Downstream** rule and terminates as Non-Leader. Therefore, u will eventually receive a pulse from Port$*$, trigger the **Downstream** rule, and terminate as Non-Leader. The argument for quiescent termination follows exactly as in the base case. ◀

▶ **Lemma 33.** *The total number of pulses sent by all nodes in this algorithm is $O(n^2)$.*

Proof. For each node other than l, exactly $\lambda\tau(u) \leq k - 1 \leq n - 1$ pulses are sent to its parent due to the **Upstream** rules (see the proofs of Lemmas 28 and 31), contributing at most $(n-1)^2$ pulses in total. The **Leader** and **Downstream** rules cause exactly one pulse to be sent across each edge (see the proof of Lemma 32), contributing an additional $n - 1$ pulses. Therefore, the total number of pulses sent during the execution of the algorithm is at most $(n-1)^2 + (n-1) = O(n^2)$. ◀

▶ **Theorem 3** (Leader election on general asymmetric trees). *There exists a quiescently terminating, anonymous content-oblivious leader election algorithm with message complexity $O(n^2)$ for a tree with n nodes, provided that each node knows tree topology G* a priori*, and the tree is not symmetric about any edge.*

Proof of Theorem 3. The correctness and quiescent termination of the leader election algorithm follow from Lemmas 31 and 32, and the message complexity bound follows from Lemma 33. ◀

4 Randomized Impossibilities

In this section, we show the following impossibility result:

▶ **Theorem 4** (Edge symmetry implies impossibility, randomized). *Let G be a network that is symmetric about an edge, where nodes are anonymous and know the network topology G. Then there exists no terminating, randomized leader-election algorithm which succeeds with probability greater than $\frac{1}{2}$.*

Let us first consider a 2-node path P_2, where the two nodes are $\{u, v\}$. Consider the view of u (resp., v): since the node has only one communication channel, and the communication channel transmits only indistinguishable pulses, the algorithm restricted to u can be captured by an automaton with input alphabet size 1, i.e., a probabilistic unary automaton $\{Q, q_s, F_{\mathsf{Leader}}, F_{\mathsf{Non\text{-}Leader}}, \delta, \sigma\}$ as follows. Note that u must reach some accepting state to terminate, and the output of u, Leader or Non-Leader depends on the exact accepting state it reaches.

- Q is the state space.
- q_s is the initial state.
- $F_{\mathsf{Leader}} \subseteq Q$ is the set of accepting states labeled Leader.
- $F_{\mathsf{Non\text{-}Leader}} \subseteq Q$ is the set of accepting states labeled Non-Leader.
- $\delta : Q \times Q \to [0, 1]$ is the transition function. $\delta(q_1, q_2)$ denotes the probability that the automaton transitions into state q_2 after receiving a symbol when in state q_1. It holds that $\sum_{q_2 \in Q} \delta(q_1, q_2) = 1$. If $q_1 \in F_{\mathsf{Leader}}$, $\delta(q_1, q_2) > 0$ implies $q_2 \in F_{\mathsf{Leader}}$. If $q_1 \in F_{\mathsf{Non\text{-}Leader}}$, $\delta(q_1, q_2) > 0$ implies $q_2 \in F_{\mathsf{Non\text{-}Leader}}$. This requirement stems from termination of the algorithm: after outputting Leader or Non-Leader (denoted by transitioning into a state in F_{Leader} and $F_{\mathsf{Non\text{-}Leader}}$), a node cannot change its output.
- $\sigma : Q \to \mathbb{Z}_{\geq 0}$. When the automaton transitions into state q, the node sends $\sigma(q)$ pulses.

Let $P_{\mathsf{Leader}}(c)$ denote the probability that the automaton's state is in F_{Leader} after receiving c symbols, and define $P_{\mathsf{Non\text{-}Leader}}(c)$ in a similar fashion.

▶ **Lemma 34.** *For $c_1 \leq c_2$, the following hold.*

- $P_{\mathsf{Leader}}(c_1) \leq P_{\mathsf{Leader}}(c_2)$.
- $P_{\mathsf{Non\text{-}Leader}}(c_1) \leq P_{\mathsf{Non\text{-}Leader}}(c_2)$.

Proof. We only need to show that for $c_2 = c_1 + 1$, the statements are true. Since $F_{\mathsf{Leader}} \subseteq Q$ are the *accepting* states, upon receiving c_1 symbols, there are two cases.

- If the automaton is in a state in F_{Leader}, then upon receiving the next letter, the automaton is still in a state in F_{Leader}.

- If the automaton is not in a state in F_{Leader}, then upon receiving the next letter, the automaton may or may not make a transition to a state in F_{Leader}.

Therefore, $P_{\mathsf{Leader}}(c_1) \leq P_{\mathsf{Leader}}(c_2)$. The same argument works for the Non-Leader side. ◀

Now we define $P_{\mathsf{Leader}} = \sup_{c \in \mathbb{Z}_{\geq 0}} P_{\mathsf{Leader}}(c)$, and $P_{\mathsf{Non\text{-}Leader}}$ is defined similarly.

▶ **Lemma 35.** $P_{\mathsf{Leader}} + P_{\mathsf{Non\text{-}Leader}} \leq 1$.

Proof. Assume otherwise $P_{\mathsf{Leader}} + P_{\mathsf{Non\text{-}Leader}} > 1$. Then there exist $c_{\mathsf{Leader}}, c_{\mathsf{Non\text{-}Leader}}$ such that $P_{\mathsf{Leader}}(c_{\mathsf{Leader}}) + P_{\mathsf{Non\text{-}Leader}}(c_{\mathsf{Non\text{-}Leader}}) > 1$. Without loss of generality, let $c_{\mathsf{Non\text{-}Leader}} \geq c_{\mathsf{Leader}}$. By Lemma 34, $P_{\mathsf{Leader}}(c_{\mathsf{Non\text{-}Leader}}) + P_{\mathsf{Non\text{-}Leader}}(c_{\mathsf{Non\text{-}Leader}}) > 1$, which is impossible. ◀

Proof for Theorem 4. We first consider the two-party case, where two nodes $\{u, v\}$ performing some leader election algorithm via content-oblivious communication along an edge. To upper bound the success probability, it suffices to assume that the communication between $\{u, v\}$ stabilize at some moment (i.e., there are no more pulses traveling between u and v), for otherwise u and v will never terminate, failing the leader election task. Let $Q(c_u, c_v)$ denote the probability that at stabilization, u received c_u pulses, and v received c_v pulses. The probability that the leader election is successful can be obtained as follows: For all possibilities of (c_u, c_v) pair, sum up the probability that the outputs of u and v constitute a valid leader election outcome, weighted by $Q(c_u, c_v)$.

$$\begin{aligned}\Pr[\text{Correct}] &= \sum_{c_v, c_u} Q(c_u, c_v)(P_{\mathsf{Leader}}(c_v)P_{\mathsf{Non\text{-}Leader}}(c_u) + P_{\mathsf{Non\text{-}Leader}}(c_v)P_{\mathsf{Leader}}(c_u)) \\ &\leq P_{\mathsf{Leader}}P_{\mathsf{Non\text{-}Leader}} + P_{\mathsf{Non\text{-}Leader}}P_{\mathsf{Leader}} \leq \frac{1}{2}\end{aligned}$$

We have thus proven that for an anonymous edge, content-oblivious leader election terminates correctly with probability at most $\frac{1}{2}$. We are now ready to extend the argument to all graphs G symmetric about an edge $\{u, v\}$ by a simple simulation argument.

Let graph G be symmetric about edge e, so $G - e$ has two isomorphic connected components, H and K. We claim G does not admit a leader election algorithm that succeeds with probability greater than $\frac{1}{2}$, even if all nodes know the topology G. Otherwise, on a 2-node path P_2, let the two nodes u and v respectively simulate H and K, and perform the algorithm on G, which leads to contradiction. Since H and K are isomorphic, the simulation does not require first breaking the symmetry between the two nodes u and v. ◀

The proof of our impossibility result is similar to the two-party impossibility in [18, Theorem 20]. In both proofs, the argument uses the contentless nature of pulses to model the behavior of degree-1 nodes as a unary automaton. In our case, we allow this automaton to be probabilistic, which yields a slightly stronger impossibility. The key difference between the two results lies in the reduction that extends the impossibility from a single edge to graphs containing bridges.

5 Necessity of Knowledge on Underlying Topology

In this section, we show the necessity of exact topology knowledge for leader election on trees. We show a family $\mathcal{G}$ of two graphs where each $G \in \mathcal{G}$ by itself grants a quiescently terminating algorithm, but no algorithm achieves termination if the graph topology is drawn from $\mathcal{G}$.

We write P_n to denote a path of n nodes. We prove the following:

▶ **Theorem 6** (Necessity of topology knowledge)**.** *There exists no deterministic, terminating content-oblivious leader election algorithm if the underlying topology is drawn from the family* $\mathcal{G} = \{P_3, P_5\}$*, and* $\mathcal{G}$ *is known to the nodes* a priori*, even if the nodes have unique* IDs*.*

Since P_3 and P_5 are not edge-symmetric, if the graph topology is known, then we do have a quiescently terminating algorithm for them, as shown in Theorem 3.

Proof. For the sake of contradiction, we assume such a deterministic, terminating algorithm $\mathcal{A}$ exists. We first define a labeling over the nodes in P_n.

- Assign label L to the nodes with degree 1 (leaf nodes).
- Assign label M to the nodes with degree 2 (middle nodes).

Since a node can count the number of its ports, it can label itself at the initiation of any algorithm, so we may assume that the labels are provided as input to the nodes. Therefore, $\mathcal{A}$ can be viewed as a deterministic, terminating leader election algorithm $\mathcal{A}'$ for the family of labeled graphs:

$$\{\mathsf{L}-\mathsf{M}-\mathsf{L}\ ,\ \ \mathsf{L}-\mathsf{M}-\mathsf{M}-\mathsf{M}-\mathsf{L}\}$$

Now we define a new type of labeled node X. Observe that the algorithm $\mathcal{A}'$ can be turned into an algorithm $\mathcal{A}''$ that elects a leader for the following family of labeled graphs:

$$\{\mathsf{L}-\mathsf{M}-\mathsf{L}\ ,\ \ \mathsf{L}-\mathsf{M}-\mathsf{M}-\mathsf{M}-\mathsf{L}\ ,\ \ \mathsf{X}-\mathsf{L}\ ,\ \ \mathsf{X}-\mathsf{M}-\mathsf{X}\}$$

To see this, we simply let $\mathcal{A}''$ on all nodes labeled X simulate $\mathsf{L}-\mathsf{M}-$, and then perform $\mathcal{A}'$. A subtle issue is that since one X node needs to simulate two nodes, X nodes must ensure the simulated nodes have IDs that do not conflict with the rest of the graph. To achieve this, when performing $\mathcal{A}''$, L and M nodes with assigned ID k runs A' with ID $2k$, while a X simulates an L node of ID $2k$ and an M node of ID $2k+1$.

Similar to our approach in Theorem 4 and [18, Theorem 20], the behavior of an L node with assigned ID i performing algorithm $\mathcal{A}''$ can be described by an unary automaton ${\mathcal{A}''}_{\mathsf{L}}(i)$. The state of the unary automaton ${\mathcal{A}''}_{\mathsf{L}}(i)$ is entirely a deterministic function of the number of symbols received, while the terminating requirement of $\mathcal{A}''$ forbids it from changing output (from Leader to Non-Leader or the opposite direction). Therefore, the output of ${\mathcal{A}''}_{\mathsf{L}}(i)$ is entirely a function of i. Now consider all such unary automata: ${\mathcal{A}''}_{\mathsf{L}}(1), {\mathcal{A}''}_{\mathsf{L}}(2), {\mathcal{A}''}_{\mathsf{L}}(3), \ldots$ for all IDs. There must *not* be 2 IDs i and j where ${\mathcal{A}''}_{\mathsf{L}}(i)$ and ${\mathcal{A}''}_{\mathsf{L}}(j)$ both output Leader, since otherwise we have a contradiction by assigning IDs i and j to the two L nodes in the graph $\mathsf{L}-\mathsf{M}-\mathsf{L}$. Hence, there exists an ID k_{L} such that all L nodes with ID $\geq k_{\mathsf{L}}$ output Non-Leader. Similarly, we can define k_{X} due to the presence of the labeled graph $\mathsf{X}-\mathsf{M}-\mathsf{X}$.

Now for the graph $\mathsf{X}-\mathsf{L}$ which $\mathcal{A}''$ claims to solve with termination, assign IDs $\max(k_{\mathsf{L}}, k_{\mathsf{X}})$ and $\max(k_{\mathsf{L}}, k_{\mathsf{X}}) + 1$ to the two nodes respectively. Both nodes must output Non-Leader by our choice of k_{L} and k_{X}, but they are the only nodes in the graph $\mathsf{X}-\mathsf{L}$. Hence, we arrive at a contradiction. ◀

Similar to the proof of Theorem 4, we believe that the argument above can be extended to the randomized setting; however, we chose not to pursue this extension for the sake of simplicity.

There exist many graph families beyond $\mathcal{G} = \{P_3, P_5\}$ for which the same impossibility result can be established. Our argument does not depend on the internal structure of L; in fact, replacing L with any graph, *not limited to rooted trees*, does not affect the impossibility proof, thereby yielding infinitely many counterexamples. Consequently, our proof of the necessity of topology knowledge applies far more broadly than just to trees.

6 Conclusion and Open Questions

In this work, we fully characterize the solvability of content-oblivious leader election on trees with topology knowledge: Terminating leader election is possible *if and only if* the tree is not symmetric about an edge.

Being the first work to explore content-oblivious leader election beyond 2-edge-connected networks, our focus is primarily on *qualitative* results. It remains widely open whether the efficiency of our leader election algorithms can be improved, both in terms of message complexity and advice complexity. For odd-diameter trees, the advice complexity of our algorithm is $O(n)$, as the topology of an n-vertex tree can be encoded using $O(n)$ bits. For even-diameter trees, the advice complexity of our algorithm is $O(\log n)$, as the diameter of an n-vertex tree can be written using $O(\log n)$ bits.

Given that content-oblivious leader election in 2-edge-connected networks [18, 8] and in trees has been studied, a natural next step is to characterize the solvability of content-oblivious leader election in *general graphs*, since any connected graph decomposes into a tree of 2-vertex-connected components, and every 2-vertex-connected graph is also 2-edge-connected. On the impossibility side, we have shown in this work that if G is symmetric about an edge, then terminating leader election is not possible on G. We conjecture that, apart from the symmetric case, knowledge of the network topology G suffices to yield a terminating leader election algorithm.

Many graph problems remain unexplored in the content-oblivious model. For 2-edge-connected networks, a universal compiler transforms any distributed algorithm into one in the content-oblivious model, as shown by Censor-Hillel, Cohen, Gelles, and Sela [7]. Thus, the feasibility of leader election in 2-edge-connected networks implies that every other graph problem is also solvable. In contrast, due to the impossibility result of Censor-Hillel, Cohen, Gelles, and Sela [7], for trees and other non-2-edge-connected networks, such a universal simulation cannot exist, even with full topological knowledge. We leave open the question of whether other fundamental graph problems, such as 2-coloring, 3-coloring, and MIS, are solvable on trees in the content-oblivious model, provided that the network topology is known.

References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2560–2579, USA, 2017. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611974782.169.

2 Dan Alistarh, Joel Rybicki, and Sasha Voitovych. Near-optimal leader election in population protocols on graphs. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 246–256, 2022. doi:10.1145/3519270.3538435.

3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006. doi:10.1007/S00446-005-0138-3.

4 Petra Berenbrink, George Giakkoupis, and Peter Kling. Optimal time and space leader election in population protocols. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 119–129, 2020. doi:10.1145/3357713.3384312.

5 Paolo Boldi, Shella Shammah, Sebastiano Vigna, Bruno Codenotti, Peter Gemmell, and Janos Simon. Symmetry breaking in anonymous networks: Characterizations. In *Proceedings of the 4th Israel Symposium on Theory of Computing and Systems (ISTCS)*, pages 16–26, 1996.

6 Steven C. Bruell, Sukumar Ghosh, Mehmet Hakan Karaata, and Sriram V. Pemmaraju. Self-stabilizing algorithms for finding centers and medians of trees. *SIAM Journal on Computing*, 29(2):600–614, 1999. doi:10.1137/S0097539798427156.

7 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.

8 Jérémie Chalopin, Yi-Jun Chang, Lyuting Chen, Giuseppe A. Di Luna, and Haoran Zhou. Content-Oblivious Leader Election in 2-Edge-Connected Networks. In Dariusz R. Kowalski, editor, *39th International Symposium on Distributed Computing (DISC)*, volume 356 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:22, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2025.21.

9 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.

10 Yi-Jun Chang, Lyuting Chen, and Haoran Zhou. Beyond 2-edge-connectivity: Algorithms and impossibility for content-oblivious leader election, 2025. arXiv:2511.23297.

11 Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *Proceedings of the 24th International Conference on Distributed Computing (DISC)*, pages 148–162, Berlin, Heidelberg, 2010. Springer-Verlag. doi:10.1007/978-3-642-15763-9_15.

12 Stefan Dobrev and Andrzej Pelc. Leader election in rings with nonunique labels. *Fundamenta Informaticae*, 59(4):333–347, 2004. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi59-4-02.

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

14 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018. doi:10.1007/S00446-016-0281-Z.

15 Fabien Dufoulon, Janna Burman, and Joffroy Beauquier. Beeping a deterministic time-optimal leader election. In *32nd International Symposium on Distributed Computing (DISC 2018)*, pages 20:1–20:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.DISC.2018.20.

16 Yuval Emek and Eyal Keren. A thin self-stabilizing asynchronous unison algorithm with applications to fault tolerant biological networks. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 93–102, 2021. doi:10.1145/3465084.3467922.

17 Klaus-Tycho Förster, Jochen Seidel, and Roger Wattenhofer. Deterministic leader election in multi-hop beeping networks. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 212–226. Springer, 2014.

18 Fabian Frei, Ran Gelles, Ahmed Ghazy, and Alexandre Nolin. Content-Oblivious Leader Election on Rings. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.26.

19 Mohsen Ghaffari and Bernhard Haeupler. Near optimal leader election in multi-hop radio networks. In *Proceedings of the 24th annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 748–766. SIAM, 2013. doi:10.1137/1.9781611973105.54.

20 Seth Gilbert and Calvin Newport. The computational power of beeps. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC)*, pages 31–46, Berlin, Heidelberg, 2015. Springer-Verlag. doi:10.1007/978-3-662-48653-5_3.

21 Christian Glacet, Avery Miller, and Andrzej Pelc. *Time vs. Information Tradeoffs for Leader Election in Anonymous Trees*, pages 600–609. SIAM, 2016. doi:10.1137/1.9781611974331.ch44.

22 Barun Gorain, Avery Miller, and Andrzej Pelc. Four shades of deterministic leader election in anonymous networks. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 265–274, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3409964.3461794.

23 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.

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

25 David Peleg. Informative labeling schemes for graphs. *Theoretical Computer Science*, 340(3): 577–593, 2005. Mathematical Foundations of Computer Science 2000. doi:10.1016/j.tcs.2005.03.015.

26 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.

27 Yuichi Sudo and Toshimitsu Masuzawa. Leader election requires logarithmic time in population protocols. *Parallel Processing Letters*, 30(01):2050005, 2020.

28 Yuichi Sudo, Fukuhito Ooshita, Taisuke Izumi, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Time-optimal leader election in population protocols. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2620–2632, 2020. doi:10.1109/TPDS.2020.2991771.

29 Robin Vacus and Isabella Ziccardi. Minimalist leader election under weak communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 406–416, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3732772.3733559.

30 Masafumi Yamashita and Tiko Kameda. Electing a leader when processor identity numbers are not distinct. In *Proceedings of the 3rd International Workshop on Distributed Algorithms (WDAG)*, pages 303–314. Springer, 1989.

31 Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks. i. characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1):69–89, 1996.