Counting on Anonymous Dynamic Networks: Bounds and Algorithms *

Roberto Baldoni, Giuseppe A. Di Luna
University of Rome "La Sapienza"

Abstract

In this paper we consider a static set of anonymous processes, i.e., they do not have distinguished IDs, that communicate with neighbours using a local broadcast primitive. The underlying communication system is dynamic, i.e., the communication graph might change at each computational round, changing thus neigbors of a process at each round.

In this challenging environment we investigate the problem of counting the number of processes in the network. We study the problem by analysing increasingly complex dynamic graphs. We provide tight bounds for networks where the distance between the leader process and a process is persistent across rounds. Then we use tools developed in such *persistent distance networks* to design upper bound for counting in dynamic networks whose only constraint is to remain connected across rounds, namely 1-interval connected networks.

1 Introduction

We consider a static set of anonymous process $|V|$ which exchange messages over a communication network. Processes execute a distributed computation which proceeds by rounds. At each round a fictional omniscient entity, namely *the adversary*, can change the topology of the network graph. The adversary is able to read the local memory of each process in order to deploy the worst possible network graph at each round to challenge the computation. The only restriction imposed to the adversary is that the graph has to be connected. This corresponds to the 1-interval connected model proposed by [19]. In this setting, the paper investigates one of the fundamental problem in distributed computing, namely counting the number of processes participating to the computation [18, 19, 23, 28] when considering each process communicates with its neighbours in the graph using a local broadcast communication primitive without bandwidth constraint.

Under this anonymous and dynamic system model, it has been proved that the presence of a leader process is *necessary* in order to compute non trivial tasks [25]. If we consider realistic computing settings a leader is indeed present. Such as a base station in a mobile network, a gateway in a sensors network. Also in other dynamic system models, such as population protocols [5], the use of a leader process has been largely employed for solving non-trivial computations.

Nevertheless, anonymity and the dynamic network graph make the system model very challenging despite of the presence of a leader. It has been indeed conjectured in [24,25] that the presence of a leader is not sufficient to count. Moreover the only known bound on counting is the trivial $\Omega(D)$ where D is the dynamic diameter of the network graph, that is informally the maximum time needed by a process to disseminate a message in the whole networks.

Due to the difficulty of the environment, we studied the problem on increasingly complex dynamic networks, by progressively relaxing some hypothesis on the relative distance between a node and the leader across rounds. We started developing our basic tools (techniques and algorithms) by studying anonymous dynamic networks where each node has a "Persistent Distance" (PD) from the leader, and this distance is at most 2. We denote such networks as $\mathcal{G}(\mathrm{PD})_2$. Then we applied previous results to analyse networks where this distance can be initially arbitrary but it is still persistent across rounds, $\mathcal{G}(\mathrm{PD}) \supseteq \mathcal{G}(\mathrm{PD})_2$. Finally, we used the body of results developed in both $\mathcal{G}(\mathrm{PD})$ and $\mathcal{G}(\mathrm{PD})_2$ to tackle

*A preliminary version of this work has been published in PODC 15 and OPODIS 15

1-interval connected networks, denoted as $\mathcal{G}$(1-IC) which are a superset of $\mathcal{G}$(PD) where the distance between a leader and a node might change from round to round.

Below you find the summary of the results presented in this paper[1]:

$\mathcal{G}$(PD) networks We introduce a counting algorithm, namely OPT, for $\mathcal{G}(\mathrm{PD})_2$. OPT terminates in $\mathcal{O}(\log |V|)$ rounds. Secondly, we prove that counting in $\mathcal{G}(\mathrm{PD})_2$ networks requires $\mathcal{O}(\log |V|)$ rounds showing that OPT is optimal. We show that the bound for $\mathcal{G}(\mathrm{PD})_2$ is robust by using smooth analysis [12]. Moreover, in order to reason on how to build an algorithm working on $\mathcal{G}$(1-IC), we provide a deterministic counting algorithm on $\mathcal{G}$(PD) when process can leave the computation. Detecting such condition helps recognising that a node changes the distance from the leader. This is a necessary step for counting in $\mathcal{G}$(1-IC). We extend OPT for $\mathcal{G}(\mathrm{PD})_h$ networks obtaining an algorithm that terminates in $\mathcal{O}(|V|)$ rounds.

$\mathcal{G}$(1-IC) networks We first show that the distinguished leader process is necessary and sufficient condition to implement counting on anonymous 1-interval connected networks. This closes an open question posed in [24, 25]. The proof is carried out by constructing a deterministic terminating counting algorithm, namely EXT. Unfortunately, EXT has en exponential complexity both in messages and in the number of rounds.

To derive counting algorithms with polynomial execution time, we introduce a reduction from a certain class of average consensus specification and counting. Thanks to this reduction we prove that when an upper bound on node degree is known, counting is polynomial. Interestingly, this polynomial time is also achieved when the the network contains a stable spanning tree, $\mathcal{G}(\infty$-IC). This algorithm is presented in the Appendix.

Outline In Section 2 relevant related work is investigated. In Sections 3 and 4 we formally define our system model and the problems that we want to investigate. In Section 5 we begin the investigation of $\mathcal{G}$(PD) networks. We show the optimal algorithm for counting in $\mathcal{G}(\mathrm{PD})_2$, then we analyse lower bounds on such networks (Sections 5.2, 5.3, 5.4). Section 5 terminates with the study of counting when processes "halts" (Section 5.5). Then, we show a counting algorithm for $\mathcal{G}$(PD) networks (Section 5.6), In Section 6 we show the counting algorithm for $\mathcal{G}$(1-IC) and in Section 6.4 we study the relationship between average consensus and terminating counting. Conclusions are reported in Section 7.

In the Appendix we show a polynomial counting algorithm for $\mathcal{G}(\infty$-IC).

2 Related Work

The question concerning what can be computed on top of static anonymous networks, has been pioneered by Angluin in [2] and it has been the further investigated in many papers [6, 7, 14, 37, 38]. In a static anonymous network with broadcast, the presence of a leader is enough to have a terminating counting algorithm as shown in [24].

Considering dynamic non anonymous networks, counting has been studied under several dynamicity models. In [3, 4], dynamicity corresponds to processes churn where processes leave and join the system. In [23, 34] dynamicity is governed by a random adversary to model peer-to-peer networks. Finally considering the dynamicity model employed in this paper (worst-case adversary), in [19], a counting algorithm for 1-interval connectivity has been proposed. Other results related to counting can be found in [30] where a model similar to 1-interval connected is considered, moreover [30] contains a conjecture similar to the one of [24] regarding the impossibility of terminating broadcast in the anonymous version of their model. In the context of possibly disconnected adversarial network, counting has been studied in [26]. The approaches followed by the latter works are not suitable in the model proposed by this paper, they use the asymmetry introduced by IDs.

Counting in anonymous dynamic networks: In [18], the authors propose a gossip-based protocol to compute aggregation function. The network graph considered by [18] is governed by a fair random adversary, moreover the proposed approach converges to the actual count without having a terminating condition. A similar model and strategy is also used by [17]. In the context of population protocols and

[1] A structured summary can be found in the Table of Figure 11 in the Conclusion Section.

passive mobility, counting has been investigated in many papers [5, 8, 16], some sort of fairness in agent interactions is assumed. The first work investigating the problem of terminating counting in an anonymous network with worst-case adversary and a leader node is [24]. They show that when a process is able to send a different message to each neighbors, the presence of a leader is enough to have a terminating naming algorithm. For the broadcast case, under the assumption of a fixed known upper bound on the maximum process degree, they provided an algorithm that computes an upper bound on the network size. Building on this result, [10] proposes an exact counting algorithm under the same assumption. In the same model [27] proposes an algorithm that improves the performance of [10], even though the algorithm cost is still exponential. Finally, [11] provides a counting algorithm for 1-interval connected networks considering each process is equipped with a local degree detector, i.e. an oracle able to predict the degree of the process before exchanging messages. Other works [19, 28] have investigated leader-less randomized approaches to obtain approximated counting algorithms. We are interested in study how anonymity impacts the computational power of 1-interval connected networks with broadcast, for this reason we assume that processes do not have access to a source of randomness, e.g. they cannot break symmetry by using coin tosses.

Average Consensus in Distributed Control Theory. In the Average Consensus problem, each process v_i starts with an input value $x_i(0)$, collectively processes have to compute a variable that converges to the average of their initial inputs. An archetypal solution, proposed in [35], is to use a local averaging approach, where, at each round r, each process updates a local value $x_i(r)$ using a weighted average of its previous value and the values of its neighbours

$$x_i(r) = \sum_{\forall v_j \in N(r,v_i) \cup \{v_i\}} a_{ij}(r) \cdot x_j(r-1)$$

The ϵ-convergence for such algorithm is usually defined as the number of rounds needed to ensure an upper bound of ϵ on some function that defines the gap between $x(r)$ and the average, e.g. $\max_{v_i}(x_i(r) - \frac{\sum_{i \in V} x_i(0)}{|V|})$. Local averaging has attracted a lot of attentions in control theory: algorithm variations, value of convergence, and upper/lower bounds on ϵ-convergence have been deeply investigated in both the context of static and dynamic graphs [33].

Let us remark that given a leader and an averaging consensus algorithm with exact termination and bounded convergence error, it is possible to obtain an exact counting algorithm: the leader starts with input 1, the others with input 0, and the average will converge to $\frac{1}{|V|}$, this idea has been proposed in [17].

[29] shown that local averaging, converges to the average if at each round the matrix of weights $A(r)|(A(r))_{ij} = a(r)_{ij}$ is doubly stochastic (i.e. the sum of values of each row is 1, and the sum of values of each column is 1), and it ϵ-converge in $\mathcal{O}(|V|^3 \log(\frac{1}{\epsilon}))$ rounds. If the matrix is only stochastic (i.e., the sum of values for each row is 1) the algorithm eventually converges to a common value, solving the weaker *consensus* problem, where the common value is a convex combination of initial values of processes, this convex combination is not necessarily the average [32].

The possibility of ensuring a sequence of double stochastic weights matrix is intimately connected to the graph topology and the weights selection. In undirected dynamic networks where an upper bound U on node maximum degree in known, it is possible to design a fixed weight policy that leads to doubly stochastic weight matrices [9]. Such policy ϵ-converges to the average in $\mathcal{O}(|V|^3 \log(\frac{1}{\epsilon}))$ rounds. In the same context, if a node knows its degree before the send phase, i.e. it is a perfect local degree detector oracle as in [11], then it is possible to use Metropolis Weights [36] and this also leads to a sequence of doubly stochastic matrices and therefore to an algorithm that ϵ-converges in $\mathcal{O}(|V|^3 \log(\frac{1}{\epsilon}))$ rounds.
Always considering undirected dynamic networks, when the degree is unknown at the best of our knowledge there is no Average Consensus algorithm based on local averaging.

Other works show lower bounds for convergence the average consensus [31, 39], however such lower bound are specific for some class of local averaging algorithms.

Related Bounds: A fundamental problem that is correlated with counting in dynamic networks with IDs is the k tokens dissemination, defined as follows [19]: each token is initially owned by a node

belonging to V, then processes exchange tokens, the k tokens dissemination terminates when all k tokens have been received by each node in V. [19] proved that when each node may send only one token at each round any k token dissemination algorithm based on token forwarding [2] terminates in $\Omega(|V|\log k)$ round. In [13], the authors improved the bound to $\Omega(\frac{|V|k}{\log |V|})$. Starting from these results, [15] provides bounds for different adversarial-based dynamic networks. It is well known that in network with IDs n (all-to-all) token dissemination solves counting [1]. In the same paper, it is introduced a connection between two-party token dissemination, a variant of k-token dissemination, and the counting problem. The authors show a lower bound for the two-party problem is also a lower bound for counting. In anonymous dynamic networks considered in [1], k-token dissemination can be solved by a trivial flooding algorithm in $\mathcal{O}(D)$ rounds. Finally, [20] shows that in directed static network with IDs and limited bandwidth, the number of rounds needed to solve counting is function of the network size even when $D = 2$.

3 Model of the computation

We consider a synchronous distributed system composed by a finite static set of processes V. Processes in V are *anonymous*, they initially have no identifiers and execute a deterministic *round-based* computation. Processes communicate through a communication network which is *dynamic*. We assume at each round r the network is stable and represented by a graph $G_r = (V, E(r))$ where $E(r) \subseteq V \times V$ is the set of bidirectional links at round r connecting processes in V.

Definition 1. *A* dynamic graph $G = \{G_0, G_1, \ldots, G_r, \ldots\}$ *is an infinite sequence of graphs one at each round r of the computation.*

Definition 2. *[21] A* dynamic graph *is* 1*-interval connected, if, and only if, $G \in \mathcal{G}$(1-IC), if $\forall G_r \in G$ we have that G_r is connected.*

The neighborhood of a process v at round r is denoted by $N(v, r) = \{v' : (v', v) \in E(r)\}$. We say that v has *degree* d at round r iff $|N(v, r)| = d$. Given a round r we denote with $p_{v,v'}$ a path on G_r between v and v'. Moreover we denote as $P_r(v', v)$, the set of all paths between v, v' on graph G_r. The distance $d_r(v', v)$ is the minimum length among the lengths of the paths in $P_r(v', v)$, the length of the path is defined as the number of edges. We consider the computation proceed by exchanging messages through synchronous rounds.

Every round is divided in two phases: (i) *send* where processes send the messages for the current round, (ii) *receive* where processes elaborate received messages and prepare those that will be sent in the next round. Processes can communicate with its neighbors through an *anonymous broadcast* primitive. Such primitive ensures that a message m sent by process v_i at the beginning of a certain round r will be delivered to all its neighbors during round r. A process v floods message m by broadcasting it for each round. If process receives a flooded message m then it starts the flooding of m. The flood of m terminates when it has been received by all processes. We say that a network has dynamic diameter D if for any v and any round r the flood of a message that starts at round r from process v terminates by at most round $r + D$. Intuitively the dynamic diameter is the maximum time needed to disseminate messages to all processes in the network.

Due to the impossibility result shown in [24], we assume any counting algorithm that works over a dynamic graph has a leader process v_l starting with a different unique state w.r.t. all the other processes.

Persistent distance dynamic graphs Let us characterize dynamic graphs according to the distances among a process v and the leader v_l.

Definition 3. *(Persistent Distance over G) Consider a dynamic graph G. The distance between v and v_l over G, denoted $D(v, v_l) = d$, is defined as follow: $D(v, v_l) = d$ iff $\forall r, d_r(v, v_l) = d$.*

Let us now introduce a set of dynamic graphs based on the distance between the leader and the processes of a graph.

Definition 4. *(Persistent Distance set) A graph G belongs to Persistent Distance set, denoted $\mathcal{G}(PD)$, iff $\forall v \in G$, $\exists d \in \mathbb{N}^+ :: D(v, v_l) = d$*

[2] Token-forwarding algorithms are not allowed to combine, split, or change tokens in any way [15].

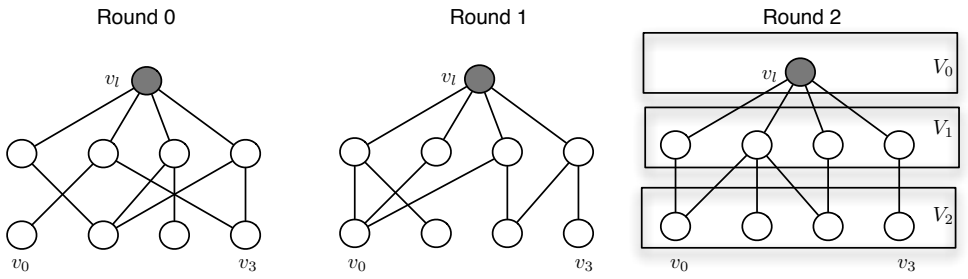

Figure 1: An example of a graph belonging to $\mathcal{G}(\text{PD})_2$ along three rounds

Graphs in $\mathcal{G}(\text{PD})_2$ Among the dynamic graphs belonging to $\mathcal{G}(\text{PD})$ we can further consider the set of graphs, denoted $\mathcal{G}(\text{PD})_h$, whose nodes have maximum distance h from the leader with $1 < h \leq |V|$. Thus, given a graph in $\mathcal{G}(\text{PD})_h$ we can partition its nodes in h sets, $\{V_0, V_1, \ldots, V_h\}$, according to their distance from the leader. The focus of this paper is on dynamic graphs belonging to $\mathcal{G}(\text{PD})_2$. As an example, figure 1 depicts a graph belonging to $\mathcal{G}(\text{PD})_2$ at round $0, 1$ and 2 whose dynamic diameter is $D = 4$. If node v_0 starts a flood at round 0, this flood will indeed reach node v_3 at round 3. The task of the leader node v_l is to count nodes in V_2.

4 Problems definition

Let us start by defining the convergent counting problem:

Definition 5. Convergent Counting: *Given a dynamic network G with $|V|$ processes, a distributed algorithm $\mathcal{A}$ solves terminating counting on G if it exists a round r such that for each round $r' > r$ the leader outputs $|V|$.*

The previous definition can be strengthen by requiring termination:

Definition 6. Terminating Counting: *Given a dynamic network G with $|V|$ processes, a distributed algorithm $\mathcal{A}$ solves terminating counting on G if it exists a round r at which the leader outputs $|V|$ and terminates.*

5 Algorithms and Bounds for $\mathcal{G}(\text{PD})$ Networks

In this section we first design OPT counting algorithm for $\mathcal{G}(\text{PD})_2$ networks. The corresponding lower bound for $\mathcal{G}(\text{PD})_2$ network is proved in Subsection 5.2, and in Subsection 5.3 we show its robustness. In Subsection 5.5 we analyse the counting problem when processes can leave the system. We study this specific setting since it will allow us to abstract and solve some problems that arise when we design counting algorithms for $\mathcal{G}(1\text{-IC})$. Finally, an optimal algorithm OPT_h for $\mathcal{G}(\text{PD})$ is proposed in Subsection 5.6.

5.1 OPT algorithm for $\mathcal{G}(\text{PD})_2$

OPT initially starts a *get_distance* phase. At the end of this phase each process is aware of its distance from the leader. In $\mathcal{G}(\text{PD})_2$ this phase takes one round and it works as follow: Each process knows if it is the leader or not. This information is broadcast by each process (including the leader) to its neighbors at the beginning of round 0. Thus, at the end of round 0, each process knows if it belongs either to V_1 or to V_2.

Non-leader process behavior Starting from round 1, a process broadcasts its distance from the leader (i.e., 1 or 2) and each process v in V_2 builds its *degree history* $v.H(r)$ with $r \geq 0$ where $v.H(r)$ is an ordered list containing the number of neighbors of v belonging to V_1 at rounds $[0, \ldots, r-1]$. Thus $v.H(r) = [\perp, |N(v,1) \cap V_1|, \ldots, |N(v,r-1) \cap V_1|]$.

Starting from round $r > 0$, each $v \in V_2$ broadcasts $v.H(r)$. These histories are collected by each process $v' \in V_1$ and sent to the leader at the beginning of round $r+1$.

Leader behavior Starting from the beginning of round $r \geq 2$ the leader receives degree histories from each process in V_1. The leader merges histories in a multiset denoted $v_l.M(r)$. Let us remark that $v_l.M(r)$ may contain the same history multiple times.

Data structure: The leader uses $v_l.M(r)$ to build a tree data structure T whose aim is to obtain $|V_2|$. For each distinct history $[A] \in v_l.M(r)$ the leader creates a node $t \in T$ with label $[A]$ and two variables $< m_{[A]}, n_{[A]} >$. $m_{[A]}$ denotes the number of histories $[A]$ in $v_l.M(r)$ and $n_{[A]}$ is the number of processes in V_2 that have sent $[A]$. Following the information flow, at round 2, $v_l.M(2)$ will be formed by a single history $[\perp]$ with multiplicity $m_{[\perp]}$. The leader creates the root of T with label $[\perp]$, value $m_{[\perp]}$, and $n_{[\perp]} =?$ (where ? means unknown value). It is important to remark that m values are directly computed from $v_l.M(r)$ while n values are set by the leader at a round $r' \geq r$ through a counting rule that will be explained later. The leader final target is to compute $n_{[\perp]}$ which corresponds to the number of processes in V_2.

At round $r+2$ if the leader receives a history $h = [\perp, x_0, \ldots, x_{r-2}, x_{r-1}]$ and $n_{[\perp,x_0,\ldots,x_{r-2}]} =?$, then it creates a node in $t \in T$ with label h and value m_h, this node is a child of the node with label $[\perp, x_0, \ldots, x_{r-2}]$. Otherwise the leader ignores h. It is straightforward to see that the following equations hold:

$$\begin{cases} m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1}]} = \sum_{i=1}^{|V_1|} i \cdot n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},i]} \\ n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1}]} = \sum_{i=1}^{|V_1|} n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},i]} \end{cases} \tag{1}$$

Where $i \cdot n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},i]}$ means that the leader received i copies of history $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}]$, one for each process in V_2 that at round $r+1$ had history $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, i]$.

Counting Rule: When in T there is a non-leaf node with label $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ such that the leader knows the number of processes (i.e., $n_{[A]}$), for each of its children but one (i.e., $n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]} =$?). Then the leader computes $n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]}$ using $m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r]} = \sum_{i=1}^{|V_1|} i \cdot n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r,i]}$.

When the leader knows the values n for each of the children of a non leaf-node t, it sums the children values and sets the n_t (see the second equation of Eq. 1).

Due to the fact that the number of processes is finite, eventually there will be a non-leaf node in T with only one child (a leaf). Thanks to the counting rule, the n variables of the child and of the father will be set. This will start a recursive procedure that will eventually set $n_{[\perp]}$ terminating the counting.

In Figure 2 is depicted an example run of the algorithm. In Figure 1 the detailed pseudocode for T is provided.

Correctness proof

Lemma 1. *Let us consider the algorithm OPT. Eventually v_l sets a value for $n_{[\perp]}$ and this value is $|V_2|$.*

Proof. We first prove that eventually we reach a round in which the counting rule can be applied for any leaf of T. Let us consider the subtree of T rooted in the node with label $[A]$, if there is only one child then the counting rule can be applied and $n_{[A]}$ can be computed. Thus let us suppose that $[A]$ has at least two children with labels $[A,x], [A,x']$ with $x \neq x'$. We have that $n_{[A,x]} \geq 1$ and $n_{[A,x']} \geq 1$ because there must be at least one sending process for each degree-history. Considering that $n_{[A]} = \sum_{j=1}^{k} n_{[A,j]}$, it follows that $n_{[A,x]} \leq n_{[A]} - 1$. Iterating this reasoning we have that when the height of the subtree rooted in $[A]$ is greater than $n_{[A]} - 1$, then each leaf has no sibling: when there is a single process sending a certain degree history H, in the next round there will be only one degree history with H as suffix. As a consequence, after at most $n_{[A]}$ rounds, we may apply the counting rule for any leaf of the subtree rooted in $[A]$.

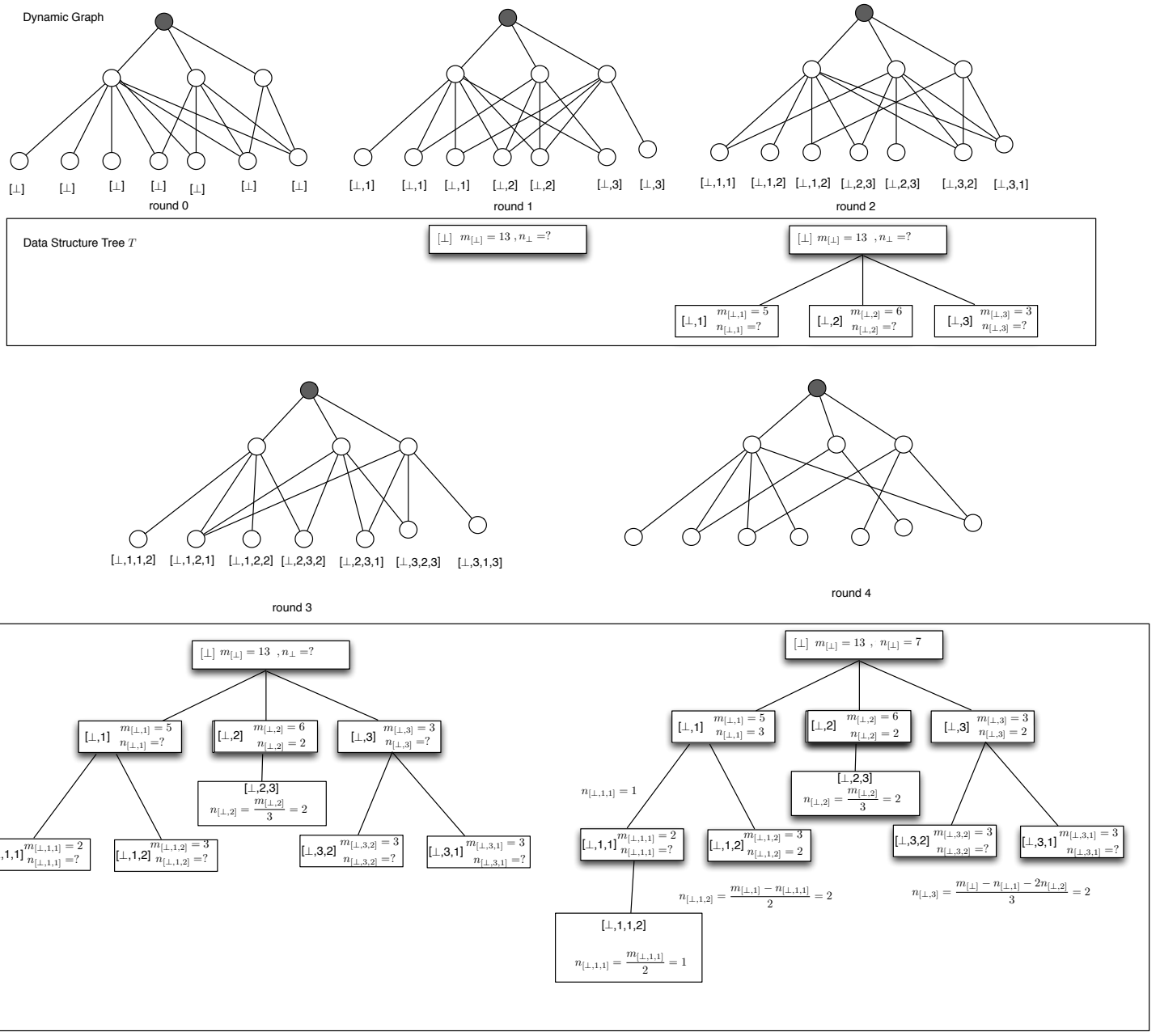

Figure 2: A run of OPT algorithm

algorithm 1: Handling of T by the Leader in OPT algorithm

$T = \perp$;
Function BUILDT($MultiSet\ M_r$)
  if $r == 2$ **then**
    Assert($\exists[\perp] \in M_1$);
    T.setRoot($[\perp] :< m_{[\perp]}, \perp >$);
  end
  forall the $[x_0, \ldots, x_{r-2}, x_{r-1}] \in M_r$ **do**
    v_l creates a node $[x_0, \ldots, x_{r-2}, x_r - 1] :< m_{[x_0,\ldots,x_{r-1}]}, n_{|[x_0,\ldots,x_{r-1}]|} :? >$;
    $t : T$.FINDNODE($[x_0, \ldots, x_{r-2}]$);
    if $n_t \neq ?||t = null$ **then**
      continue;
    end
    T.ADDCHILD($[x_0, \ldots, x_{r-2}, x_r - 1]$) ;
  end
  COMPUTE(T);
  if $T.root.n_{[\perp]} \neq ?$ **then**
    output(T.root.$\mathrm{n}_{[\perp]}$)
  end
  return;
Function COMPUTE($Tree\ T$)
  forall the $t \in T$ *starting from the level of the leaves until the root* **do**
    $C : T.findChildren(t)$;
    $X \subseteq C$ such that $[x_0, \ldots, x_k] \in X$ iff $n_{[x_0,\ldots,x_k]} \neq ?$;
    if $\exists! c : [y_0, \ldots, y_k] \in C \setminus X$ **then**
      $n_c : \frac{m_t - \sum_{\forall [x_0,\ldots,x_k] \in X} (x_k \cdot (n_{[x_0,\ldots,x_k]}))}{y_k}$;
    end
    if $X = C$ **then**
      $n_t : \sum_{\forall [x_0,\ldots,x_k] \in X} n_{[x_0,\ldots,x_k]}$;
    end
  end
  return;

Now we prove by induction that: for each node $v \in T$ if $n_v \neq ?$, then n_v is equal to the number of processes in V_2 that had degree history equal to v at a given round.

Base case, leaf without siblings: Let $v_1 : [x_0, \ldots, x_{r+1}]$ be a leaf without siblings and $v_0 : [x_0, \ldots, x_r]$ its father. v_l sets, according to the counting rule, $n_{v_0} = n_{v_1} = \frac{m_{v_0}}{x_{r+1}}$. From Eq 1 we have $n_{v_0} = n_{v_1}$ which is equal to the number of processes in V_2 that had degree history $[x_0, \ldots, x_r]$.

Inductive case: Let us consider $v_0 : [x_0, \ldots, x_r]$ and the set of its children C_{v_0} with $|C_{v_0}| > 1$. Let introduce a set X_{v_0} formed by the children for which n is known and set, formally: $X_{v_0} : \{x \in C_{v_0} | n_x \neq ?\}$. If $\exists! v_1 : [x_0, \ldots, x_{r+1}] \in C_{v_0} \setminus X_{v_0}$, the leader sets (according to the counting rule) $n_{v_1} = \frac{m_{v_0} - \sum_{\forall [x_0,\ldots,x_k] \in X_{v_0}} (x_k \cdot n_{[x_0,\ldots,x_k]})}{x_{r+1}}$. By inductive hypothesis we have $\forall x \in X_{v_1}$, n_x is equal to the number of processes in V_2 with degree history equal to x. Due to Eq. 1, we have both n_{v_1} and n_{v_0} will be set to the correct value.

From the previous arguments we have that after at most $|V_2|$ rounds all the leaves of $[\perp]$ have no siblings, thus the counting rule will be applied recursively until the value $n_{[\perp]}$ is set to $|V_2|$. □

Theorem 1. *Let G be a dynamic graph of size $|V|$ belonging to $\mathcal{G}(PD)_2$. A run of* OPT *on G terminates in at most $\lceil log_2 |V| \rceil + 3$ rounds.*

Proof. Let consider the algorithm OPT. The latter counts processes in V_2, since the number of processes in V_1 is immediately known by v_l at round 0, thus let us suppose that we are in the worst case i.e., $|V_2| = \mathcal{O}(|V|)$. Let us consider the tree T, given a node $[A]$ the maximum height of the subtree rooted in $[A]$ is a function $h_{max}(n_{[A]})$. We have that h_{max} is non decreasing, $h_{max}(n_{[A]} - 1) \leq h_{max}(n_{[A]})$: let us consider the worst scheduling that the adversary uses with $n_{[A]} - 1$ processes in order to obtain the maximum height. It easy to show that the same scheduling can be created with $n_{[A]}$ processes, the adversary will simply force two processes to follow the behavior of a single process in the old schedule. Let us restrict to the case when $[A]$ has only two children: $[A, x], [A, x']$, for the counting rule $h_{max}(n_{[A]}) = min(h_{max}(n_{[A,x]}), h_{max}(n_{[A,x']}))+1$. Considering the second equation of Eq. 1, $h_{max}(n_{[A]})$ can be rewritten as follows: $h_{max}(n_{[A]}) = 1 + min(h_{max}(\frac{n_{[A]}}{2} - \delta), h_{max}(\frac{n_{[A]}}{2} + \delta)) \leq 1 + min(h_{max}(\frac{n_{[A]}}{2}), h_{max}(\frac{n_{[A]}}{2}))$ with $\delta \in [0, \frac{n_{[A]}}{2}]$. Thus, the optimal height can be reached by having $n_{[A,x]} = n_{[A,x']} = \frac{n_{[A]}}{2}$. Let us

notice that when $[A]$ has more than two children, the maximum height of the subtree rooted in $[A]$ cannot be greater than the one obtained when $[A]$ has two children. Iterating this reasoning, in the worst case T is a balanced tree with degree at most 2 for each non leaf node and with exactly $|V|$ leaves. The height of this tree is $\lceil \log_2(|V|) \rceil$. Each level of T corresponds to one round of OPT, this completes the proof. □

Given the $\Omega(\log |V|)$ bound on $\mathcal{G}(\mathrm{PD})_2$, that we will show in Section 5.2, we have that OPT is asymptotically optimal.

5.2 Lower bound for $\mathcal{G}(\mathbf{PD})_2$

In this section we consider the $\mathcal{G}(\mathrm{PD})_2$ set and compute a lower bound for counting time. This is done by introducing a *Dynamic Bipartite Labeled k-Multigraphs* ($\mathcal{M}(\mathrm{DBL}_k)$), by showing that counting on $\mathcal{G}(\mathrm{PD})_2$ requires at least the same number of rounds as counting over $\mathcal{M}(\mathrm{DBL}_k)$ and by finally showing a lower bound on the number of rounds needed to count over $\mathcal{M}(\mathrm{DBL}_k)$.

5.2.1 Counting in Dynamic Bipartite Labeled k-Multigraphs ($\mathcal{M}(\mathbf{DBL}_k)$)

Let consider a dynamic connected multigraph M defined as follows $M = \cup_{r=0}^{\infty}\{(\{v_l\} \cup W, E(r), f_r, l_r)\}$ where $E(r)$ is a set of edges at round r, W a set of nodes, $f_r : E(r) \to \{v_l\} \times W$ a function that maps each edge to the endpoints nodes and $l_r : E(r) \to \{1, 2, \ldots, k\}$ a function labeling edges. M belongs to $\mathcal{M}(\mathrm{DBL}_k)$ if for each round r the number of edges connecting a node $v \in W$ to v_l is less than $k+1$, more formally $\forall r, \forall v \in W, E^v(r) = f_r^{-1}(v_l, v) :: 1 \leq |E^v(r)| \leq k$; Given $e', e'' \in E^v(r)$ we have $l_r(e') \neq l_r(e'')$, that is if two edges e', e'' share the same non leader node as endpoint they must have different label, as example see edges that involve node v in Figure 3. For simplicity we will refer as M_r the instance of M at round r. Figure 3 shows an example of a dynamic connected multigraph M at round r belonging to $\mathcal{M}(\mathrm{DBL}_3)$. We assume that when a node $v \in \{v_l\} \cup W$ receives a message from a node w at round r by edge e, it also obtains the label $l_r(e)$.

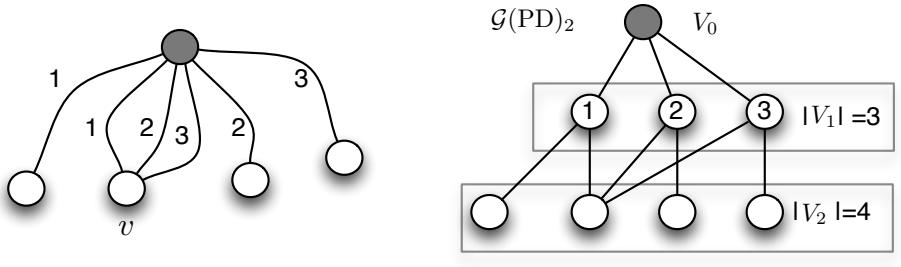

Figure 3: Trasformation, at round r, from $\mathcal{M}(DBL_3)$ multigraph to $\mathcal{G}(\mathrm{PD})_2$.

Lemma 2. *Let us consider a dynamic connected multigraph M in $\mathcal{M}(DBL_k)$. If any counting algorithm based on message passing takes more than T rounds to complete on M, then there exists a graph G in $\mathcal{G}(PD)_2$ such that any counting algorithm based on message passing requires more than T rounds to complete on G.*

Proof. From $M_r = (\{v_l\} \cup W, E(r), f_r, l_r)$ we build an instance $G_r^{id} = (V = \{(V_0 = \{v_l\}) \cup V_1 \cup (V_2 = W)\}, E_{id}(r))$ belonging to $G^{id} \in \mathcal{G}(\mathrm{PD})_2$ such that V_1 contains k nodes having unique identifiers in $[1, \ldots, k]$, V_0 contains only the leader node v_l and the set of nodes $V_2 = W$. Additionally, at round r we have that $\exists e : (v, w) \in E_{id}(r)$ with $v \in V_1$ and $w \in V_2$ where $id(v) = j$ if and only if $\exists e' \in E(r)$ with $f_r(e') = (v_l, w), l_r(e') = j$ with $w \in W$. Figure 3 shows the transformation at round r between a dynamic graph in $\mathcal{M}(\mathrm{DBL}_3)$ and one in $\mathcal{G}(\mathrm{PD})_2$. Let us notice that the node with label 1 in V_1 at each G_r^{id} is connected to the nodes in V_2 that correspond to nodes in W that are connected in M_r to v_l by edges labeled with 1. As a consequence, the leader v_l in M is actually the union of local memories of processes in $\{v_l\} \cup V_1$ in G^{id}. Let us assume that there not exist a message passing algorithm solving the

counting problem in M with T rounds, then it is not possible to count nodes in V_2 on G^{id} in T rounds even by merging the memories of $\{v_l\} \cup V_1$, by knowing the size k of V_1 and by having unique IDs for nodes in V_1. Consider now the dynamic graph G derived by G^{id} removing the identifiers of nodes in V_1. Counting nodes in G is at least as hard as counting nodes in G^{id}. As an example, without identifiers the leader cannot realize if messages of two successive rounds arrive from the same node of V_1. Thus it is not possible for the leader to count the size of G in less than T rounds. □

From the lemma follows that a lower bound for counting on $\mathcal{M}(\mathrm{DBL}_k)$ holds also for graphs in $\mathcal{G}(\mathrm{PD})_2$. Now we introduce some definitions on M. Let consider an instance M of the family $\mathcal{M}(\mathrm{DBL}_k)$.

Definition 7. *(Set of edge labels of a node at round r) Given a node $v \in W$ at round r we define the set of edge labels $L(v,r) : \{l_1, \ldots, l_j\}$ with $l_i \in L(v,r)$ iff $\exists e \in E(r)$ and $l_r(e) = l_i$ and $f_r(e) = (v, v_l)$.*

As an example in Figure 3, the edge label set of node v at round r is $\{1, 2, 3\}$.

Definition 8. *(State of a non-leader process) Given a node $v \in W$ at round r, we define the state $S(v,r)$ as an ordered list $S(v,r) : [(\perp), L(v,0), \ldots, L(v,r-1)]$ where $(\perp)$ is the first state of any non-leader node[3].*

Given a list $A : [L_0, L_1, \ldots, .., L_{r-1}]$ we have that $|A|$ denotes the number of nodes with the same state $S(v,r) = A$ at round r. Ref. Figure 1: we have $S(v, r+1) = [\perp, \ldots, \{1,2,3\}]$ and $|S(v, r+1)| = 1$ since v is the only node connected to v_l by $\{1,2,3\}$ at round r.

Definition 9. *(State of a leader node at round r) Given the leader v_l at round r we define the leader state $S(v_l, r)$ as $[C(v_l, 0), \ldots, C(v_l, r-1)]$ where $C(v_l, i)$ with $i < r$ is a multiset of elements, $(j, S(v,i)) \in C(v_l, i)$ iff it exists a node v with state $S(v,i)$ connected to v_l by an edge with label j.*

As for states of local nodes, $|(j, S(v,r))|$ denotes the number of nodes with state equal to $S(v,r)$ connected to v_l by an edge with label j at round r. Let us remark that the state of the leader v_l can be constructed by a simple message passing protocol where at each round each node sends to the leader its own state and where the leader node sends just a dummy message.

5.2.2 Lower Bound for $\mathcal{M}(\mathbf{DBL}_k)$

We introduce some notation on vectors and matrices used in this section.

Linear algebra notation Given a vector $\mathbf{a} \in \mathbb{Z}^n$, we denote as $(\mathbf{a})_j$ the j-th component of $\mathbf{a}$ (with $1 \leq j \leq n$) and as $\sum \mathbf{a}$ the sum of all components of $\mathbf{a}$. Additionally, $\sum^+ \mathbf{a}$ (resp. $\sum^- \mathbf{a}$) denotes the sum of only the positive (resp. negative) components of $\mathbf{a}$. Given two vectors $\mathbf{a}, \mathbf{b}$ we have $\begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix}$ is the vector obtained by appending the elements of the vector $\mathbf{b}$ after the last element of vector $\mathbf{a}$. Given a matrix $\mathbf{M} \in \mathbb{Z}^{n,m}$ we denote with $(\mathbf{M})_j$ its j-th row (with $1 \leq j \leq n$) and we denote as $ker(\mathbf{M})$ the set of vectors $\mathbf{a} \in \mathbb{Z}^m$ such that $\mathbf{Ma} = \mathbf{0}$. We also consider the set of vectors $B = \{\mathbf{a}^1, \ldots, \mathbf{a}^\ell\}$ that form a basis for $ker(\mathbf{M})$, i.e., $ker(\mathbf{M}) = SPAN(B)$. Finally we denote as $\mathbf{a}_r$ the instance of vector $\mathbf{a}$ at round r.

We prove the bound for the family $\mathcal{M}(\mathrm{DBL}_k)$ by first proving the lower bound for $\mathcal{M}(\mathrm{DBL}_2)$. Considering the latter proof, we first introduce a system of equations that characterizes the states of the nodes of the multigraph at round zero. The lower bound for $\mathcal{M}(\mathrm{DBL}_2)$ is then proved by studying the evolution of this system of equations through the rounds.
Consider $M \in \mathcal{M}(\mathbf{DBL}_2)$ and $r = 0$: At the end of round 0 the leader has state $S(v_l, 0) : [\{(1, [\perp]), (2, [\perp])\}]$, this leader state can be generated by many different configurations of nodes and edges in M. Such configurations are determined by the number of non leader processes with states $[\{1\}], [\{2\}], [\{1,2\}]$ and they are solutions of the following system of equations at round 0:

$$\begin{cases} |(1, [\perp])| = |[\{1\}]| + |[\{1,2\}]| \\ |(2, [\perp])| = |[\{2\}]| + |[\{1,2\}]| \end{cases} \qquad (2)$$
$$r=0$$

[3]For simplicity whenever not necessary we omit the presence of $(\perp)$ as first element of $S(v,r)$.

with the additional constraint that any variable in the solution cannot assume a negative value. When the leader updates its state, in successive rounds, we have a new system of equations. The system of equations 2 can be written in a matrix form as follows:

$$\mathbf{m}_0 = \mathbf{M}_0 \mathbf{s}_0 \tag{3}$$

where $\mathbf{M}_0 : \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$ represents the matrix of coefficients of the system at round 0, $\mathbf{m}_0$ is the column vector of constant terms (each component of $\mathbf{m}_r$ represents the multiplicity of a certain element in the state of the leader at round r) and $\mathbf{s}_0$ is a solution vector. Let us remark that $\mathbf{M}_r$ depends of the round only while $\mathbf{m}_r$ depends of the leader state at round r. As a consequence $\mathbf{M}_r$ characterizes any multigraph of the family $\mathcal{M}(\mathrm{DBL}_2)$.

The matrix $\mathbf{M}_0$ is characterized by $ker(\mathbf{M}_0) = SPAN(\mathbf{k}_0 : \begin{bmatrix} 1 & 1 & -1 \end{bmatrix}^\intercal)$. Solutions of the matrix equation 3 are related by the following linear combination with the kernel vector $\mathbf{k}_0$: $\mathbf{s}'_0 = \mathbf{s}_0 + t\mathbf{k}_0$ with $t \in \mathbb{N}$ and such that each component of $\mathbf{s}'_0$ is non negative. As a consequence, given $\mathbf{m}_0$ the possible solutions of (2) are restricted to a finite discrete set of points over a segment with direction $\mathbf{k}_0$. From the point of view of the leader each solution represents a distinct graph belonging to $\mathcal{M}(\mathrm{DBL}_2)$ with a different number of processes: $\sum \mathbf{s}'_0 - \sum \mathbf{s}_0 = t \sum \mathbf{k}_0 = t$.

Considering the example of Figure 4, the system of equations at round 0 for the multigraph M is the following

$$\begin{cases} 2 = |[\{1\}]| + |[\{1,2\}]| \\ 2 = |[\{2\}]| + |[\{1,2\}]| \end{cases}_{r=0} \tag{4}$$

where $\mathbf{m}_0 : \begin{bmatrix} 2 & 2 \end{bmatrix}^\intercal$. For such system of equations a solution is $\mathbf{s}_0 : \begin{bmatrix} 0 & 0 & 2 \end{bmatrix}^\intercal$, then using the kernel transformation another solution is $\mathbf{s}'_0 = \mathbf{s}_0 + 2\mathbf{k_0} : \begin{bmatrix} 2 & 2 & 0 \end{bmatrix}^\intercal$, these two solutions correspond to two $M, M' \in \mathcal{M}(\mathrm{DBL}_2)$ of different size that generate the same state $S(v_l, 0)$ as depicted in Figure 4. These two graphs are indistinguishable from the leader at round 0 thus the leader is not able to output a correct count.

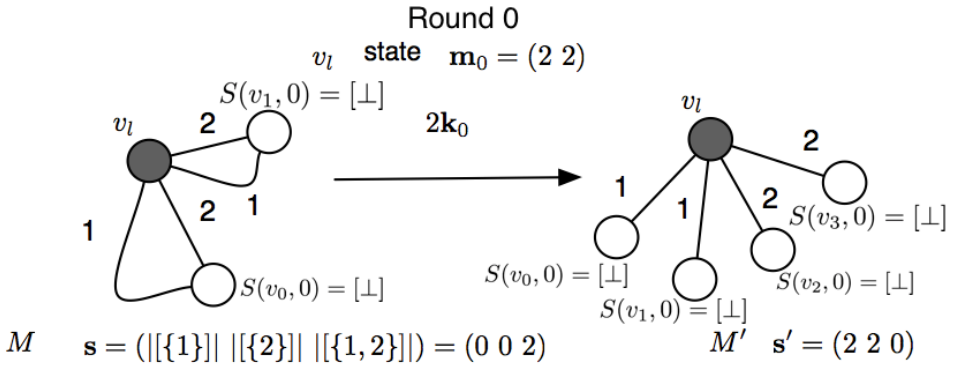

Figure 4: Two dynamic multigraphs $M, M' \in \mathcal{M}(\mathrm{DBL}_2)$ of different size that are indistinguishable at round $r = 0$, the relationship among M, M' is given by the kernel vector $\mathbf{k}_0$

The idea that we will use to show the lower bound is to characterize how the kernel space of $\mathbf{M}_r$ evolves and under which condition of the kernel space we have an unique solution, that corresponds to an unique size $|W|$.

General structure of $\mathbf{M}_r$ At round r, the system of equations becomes $\mathbf{m}_r = \mathbf{M}_r \mathbf{s}_r$. The number of columns of $\mathbf{M}_r$ is equal to the number of all possible states of non-leader nodes, at round $r+1$, which is $column(r) = 3^{r+1}$. Each row of $\mathbf{M}_r$ corresponds to a possible connection $(j, S(v, r'))$ of v_l at some round $0 \leq r' \leq r$. Thus the number of rows at round r is two times the number of existent states in $[0, r]$: $row(r) = 2 \sum_{k=0}^{r} 3^k$.

As an example, at the end of round 1, the system contains 8 equations ($2 \cdot 3^0 + 2 \cdot 3^1$) and 9 variables (i.e. 3^2 rows) and the associated matrix $\mathbf{M}_1$ are:

$$\begin{cases} |(1,[\perp])| = \sum_{\forall j \in \{\{1\},\{2\},\{1,2\}\}} |[\{1\},j]| + \sum_{\forall j \in \{\{1\},\{2\},\{1,2\}\}} |[\{1,2\},j]| \\ |(2,[\perp])| = \sum_{\forall j \in \{\{1\},\{2\},\{1,2\}\}} |[\{2\},j]| + \sum_{\forall j \in \{\{1\},\{2\},\{1,2\}\}} |[\{1,2\},j]| \\ |(1,[\{1\}])| = |[\{1\},\{1\}]| + |[\{1\},\{1,2\}]| \\ |(1,[\{2\}])| = |[\{2\},\{1\}]| + |[\{2\},\{1,2\}]| \\ |(1,[\{1,2\}])| = |[\{1,2\},\{1\}]| + |[\{1,2\},\{1,2\}]| \\ |(2,[\{1\}])| = |[\{1\},\{2\}]| + |[\{1\},\{1,2\}]| \\ |(2,[\{2\}])| = |[\{2\},\{2\}]| + |[\{2\},\{1,2\}]| \\ |(2,[\{1,2\}])| = |[\{1,2\},\{2\}]| + |[\{1,2\},\{1,2\}]| \end{cases}_{r=1} \tag{5}$$

$$\mathbf{M}_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{6}$$

Let us now consider how it is built the equation at round r derived from the generic leader connection $(j,[x_0,\ldots,x_{r'-1}])$ with $j \in \{1,2\}$ introduced at round r' in the system of equations, i.e.,$|(j,[x_0,\ldots,x_{r'-1}])| = |[x_0,\ldots,x_{r'-1},\{j\}]| + |[x_0,\ldots,x_{r'-1},\{1,2\}]$. This equation at round r becomes:

$$|(j,[x_0,\ldots,x_{r'}])| = \sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'}} |[x_0,\ldots,x_{r'-1},\{j\},s]| + \\ + \sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'}} [x_0,\ldots,x_{r'-1},\{1,2\},s]| \tag{7}$$

where $(\{1\}|\{2\}|\{1,2\})^{r-r'}$ is the set of all possible lists with elements in $\{\{1\},\{2\},\{1,2\}\}$ and size $r-r'$. As an example see the equation associated with $|(1,[\perp])|$ at round 0 (see Equation 2) and the equation associated with $|(1,[\perp])|$ at round 1 (see Equation 5).

Let notice $ker(\mathbf{M}_1) = \{\mathbf{k}_1 = \begin{bmatrix} 1 & 1 & -1 & 1 & 1 & -1 & -1 & -1 & 1 \end{bmatrix}^\intercal\}$, thus we have $< \mathbf{k}_1 >= 1$ with $< \mathbf{k}_1 >^+ = 5$, $< \mathbf{k}_1 >^- = 4$. Now let us consider a solution $\mathbf{s}_1$ with $< \mathbf{s}_1 > \leq 3$. It is easy to see that $\mathbf{s'}_1 = \mathbf{s}_1 + t\mathbf{k}_1$ has at least one negative component for any $t \neq 0$: since $< \mathbf{k}_1 >^- = 4$ is not possible to have a solution $\mathbf{s}_1$ that as at least one unitary component for each negative component of $\mathbf{k}_1$. Thus $\mathbf{s'}_1$ cannot be a solution that represents a dynamic multigraph.

This means that if $n \leq 3$ is possible to obtain the count in 2 rounds, since there is only one possible solution of the system of equations for any $\mathbf{m}_1$ generated by a multigraph with $n \leq 3$. For $n \geq 4$ we have at least two possible solutions of different size, i.e. we have $\mathbf{s}_1 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}^\intercal$ with $n = 4$ processes and $\mathbf{s'} : \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}^\intercal = \mathbf{s}_1 + \mathbf{k}_1$ with $n = 5$. It is easy to check that $\mathbf{m_1} = \mathbf{M}_1\mathbf{s}_1 = \mathbf{M}_1\mathbf{s}'_1$, thus we have two multigraphs of different sizes that generate the same state $\mathbf{m}_1$ at v_l, see Figure 5.

In order to simplify the proofs, we order columns of $\mathbf{M}_r$ lexicographically with respect to the state of a process. We consider the following order among elements $\{1\} < \{2\} < \{1,2\}$. As a consequence, the first column of $\mathbf{M}_r$ will correspond to state: $|[\{1\},\ldots\{1\}]|$, the second column $|[\{1\},\ldots\{1\},\{2\}]|$ and the last one $|[\{1,2\},\ldots,\{1,2\}]|$. Rows are ordered in the same way. This ordering has been used in Equation 3 and Equation 6. Fixed this ordering we can use a connection $(j,[x_0,\ldots,x_{r'-1}])$ to denote a row $\mathbf{v} = (\mathbf{M}_r)_{(j,[x_0,\ldots,x_{r'-1}])}$ and a node state to denote a single component of a vector, i.e. $(\mathbf{v})_{[x_0,\ldots,x_{r-1}]}$.

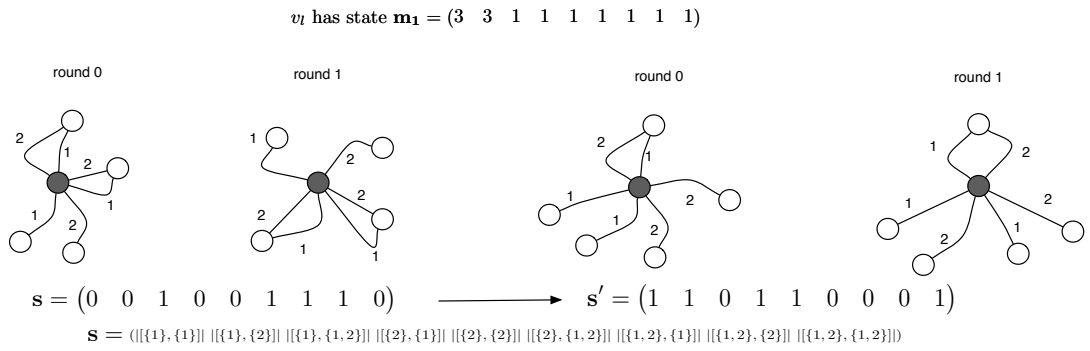

Figure 5: Two dynamic multigraph $M, M' \in \mathcal{M}(\mathrm{DBL}_2)$ of different size that are indistinguishable at round $r = 1$, they induce the same leader state $S(v_l, 1) = \mathbf{m}_1$, the relationship among the two is given by the kernel vector $\mathbf{k}_1$

Moreover we have that the row vector $(\mathbf{M}_r)_{(j,[x_0,\ldots,x_{r'-1}])}$ will have two trails of ones, with length $3^{r-r'}$, for all columns in the form $|[x_0, \ldots, x_{r'-1}, \{j\}, s]|, |[x_0, \ldots, x_{r'-1}, \{1,2\}, s]|$ with $s \in (\{1\}|\{2\}|\{1,2\})^{r-r'}$, and zero for all the other columns (as reference see Eq. 6).

In the following lemmas we specifically characterize the structure of the kernel space of $\mathbf{M}_r$ in order to identify at which round there is an unique solution.

Lemma 3. *Let us consider the matrix $\mathbf{M}_r$ of the family $\mathcal{M}(DBL_2)$ at round r. The dimension of the kernel space of $\mathbf{M}_r$ is one (i.e., $ker(\mathbf{M}_r) = SPAN(\mathbf{k}_r)$).*

Proof. We first show that rows of $\mathbf{M}_r$ are linearly independent, thus that the rank of the matrix is equal to the number of rows. The proof is by induction:

- Base Case $r = 0$: $\mathbf{M}_0 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$, $det(\mathbf{M}_0) = 1$ thus the rows are linearly independent.
- Inductive Case r: $\mathbf{M}_r$ can be written as $\mathbf{M}_r = \begin{bmatrix} \mathbf{M}'_{r-1} \\ \mathbf{U} \end{bmatrix}$, where $\mathbf{M}'_{r-1}$ is the matrix obtained by $\mathbf{M}_{r-1}$ substituting each element $1/(0)$ of $\mathbf{M}_{r-1}$ with a row vector $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} / (\begin{bmatrix} 0 & 0 & 0 \end{bmatrix})$. Now by inductive hyp. we have that all rows of $\mathbf{M}_{r-1}$ are linearly independent. This implies that also the rows of $\mathbf{M}'_{r-1}$ are linearly independent, this can be easily shown by contradiction, let us suppose that we have $(\mathbf{M}'_{r-1})_s = x_a(\mathbf{M}'_{r-1})_a + x_b(\mathbf{M}'_{r-1})_b$ for some rows s, a, b and two coefficient x_a, x_b, this means that also if we take the subvectors $\mathbf{v}^1, \mathbf{v}^2, \mathbf{v}^3$ of $(\mathbf{M}'_{r-1})_s, (\mathbf{M}'_{r-1})_a, (\mathbf{M}'_{r-1})_b$, obtained by taking the components in position j such that $j mod 3 = 0$, we must have $\mathbf{v}^1 = x_a\mathbf{v}^2 + x_b\mathbf{v}^3$ but this could be also written as $(\mathbf{M}_{r-1})_s = x_a(\mathbf{M}_{r-1})_a + x_b(\mathbf{M}_{r-1})_b$ that is clearly a contradiction since the rows of $\mathbf{M}_{r-1}$ are linearly independent. We now show that a row of $\mathbf{U}$ cannot be expressed as linear combination of rows of $\mathbf{M}'_{r-1}$, we have that the row $\mathbf{U}_c$ corresponding to connection $c : (j, [x_0, \ldots, x_{r-1}])$, has only two elements different from zero contained in the subvector $\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$ (if $j = 1$) or $\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$ (if $j = 2$) positioned in the columns with the form $[x_0, \ldots, x_{r-1}, (\{1\}|\{2\}|\{1,2\})^1]$. Now, for each row $(\mathbf{M}'_{r-1})_i$ considering only the values of columns $[x_0, \ldots, x_{r-1}, (\{1\}|\{2\}|\{1,2\})^1]$, we get a subvector that is either $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ or $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$. Therefore it follows that $\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$ or $\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$ cannot be expressed as linear combination of the row vectors of $\mathbf{M}'_{r-1}$.

  We have to show that the rows vector of $\mathbf{U}$ are linearly independent. If we consider the sets of 3 columns in the form $[x_0, \ldots, x_{r-1}, (\{1\}|\{2\}|\{1,2\})^1]$, only two rows have some elements different from zero: they are either $\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$ or $\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$ that are linearly independent.

This implies, for the rank-nullity theorem [22], that the size of the kernel is $|ker(\mathbf{M}_r)| = column(r) - row(r) = 3^{r+1} - 2\sum_{k=0}^{r} 3^k = 1$. □

Lemma 4. *Let us consider the matrix* $\mathbf{M}_r$ *of the family* $\mathcal{M}(DBL_2)$ *at round* r*. We have* $\mathbf{k}_r = \begin{bmatrix}\mathbf{k}_{r-1} & \mathbf{k}_{r-1} & -\mathbf{k}_{r-1}\end{bmatrix}^\intercal$ *with* $\mathbf{k}_{-1} = 1$.

Proof. The proof is done by induction:

- Base Case, *round* $= 0$. $\mathbf{k}_0 = \begin{bmatrix}1 & 1 & -1\end{bmatrix}^\intercal$ implies $\mathbf{0} = \mathbf{M}_0\mathbf{k}_0$.
- Inductive Case, *round* $= r$. We assume $\mathbf{k}_{r-1} = \begin{bmatrix}\mathbf{k}_{r-2} & \mathbf{k}_{r-2} & -\mathbf{k}_{r-2}\end{bmatrix}^\intercal$. We show a vector $\mathbf{k}$ such that its product for the rows of $\mathbf{M}_r$ corresponding to $c' = (j, l' : [x_0, \ldots, x_{r'-1}])$, with $r' < r$ and $j \in \{1, 2\}$, is 0. Then we show that the same holds for the remaining rows of $\mathbf{M}_r$ that corresponds to connection $c = (j, l : [x_0, \ldots, x_{r-1}])$, and finally we show that $\mathbf{k} = \mathbf{k}_r = \begin{bmatrix}\mathbf{k}_{r-1} & \mathbf{k}_{r-1} & -\mathbf{k}_{r-1}\end{bmatrix}^\intercal$. Let us consider the row-vector product $(\mathbf{M}_{r-1})_{c'}\mathbf{k}_{r-1}$ at round $r-1$, by definition of kernel we have:

$$0 = \sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} (\mathbf{k}_{r-1})_{|[l',\{j\},s]|} + \sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} (\mathbf{k}_{r-1})_{|[l',\{1,2\},s]|} \tag{8}$$

Let us build a vector $\mathbf{k} = \begin{bmatrix}(\mathbf{k}_{r-1})_1\mathbf{k}_0 & (\mathbf{k}_{r-1})_2\mathbf{k}_0 & \ldots & (\mathbf{k}_{r-1})_{3^r}\mathbf{k}_0\end{bmatrix}^\intercal$ and let us examine the row-vector product $(\mathbf{M}_r)_c\mathbf{k}$:

$$\sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} ((\mathbf{k})_{|[l',(j),s,\{1\}]|} + (\mathbf{k})_{|[l',(j),s,\{2\}]|} + (\mathbf{k})_{|[l',(j),s,\{1,2\}]|}) + \sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'}} (\mathbf{k})_{|[l',\{1,2\},s]|} \tag{9}$$

the first term of Eq. 9 can be expressed as follow

$$\sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} ((\mathbf{k})_{|[l',(j),s,\{1\}]|} + (\mathbf{k})_{|[l',(j),s,\{2\}]|} + (\mathbf{k})_{|[l',(j),s,\{1,2\}]|}) =$$
$$\sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} ((\mathbf{k}_{r-1})_{|[l',(j),s]|})((\mathbf{k}_0)_1 + (\mathbf{k}_0)_2 + (\mathbf{k}_0)_3) =$$
$$\sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} (\mathbf{k}_{r-1})_{|[l',(j),s]|}$$

The second term of Eq. 9 can be rewritten as the first term, then by applying Eq. 8, we have:

$$(\mathbf{M}_r)_{c'}\mathbf{k} = (\mathbf{M}_{r-1})_{c'}\mathbf{k}_{r-1} = 0$$

Now let us consider the row $c = (j, l : [x_0, \ldots, x_{r-1}])$ the row-vector product is $(\mathbf{M}_r)_c\mathbf{k}$:

$$(\mathbf{k})_{|[l,\{j\}]|} + (\mathbf{k})_{|[l,\{1,2\}]|} = (\mathbf{k}_{r-1})_{|[l]|}((\mathbf{k}_0)_j + (\mathbf{k}_0)_3) = 0$$

Thus we have $\mathbf{0} = \mathbf{M}_r\mathbf{k}$. Now we have to prove that $\mathbf{k} = \begin{bmatrix}\mathbf{k}_{r-1} & \mathbf{k}_{r-1} & -\mathbf{k}_{r-1}\end{bmatrix}^\intercal$.
For inductive hypothesis we have $\mathbf{k}_{r-1} = \begin{bmatrix}\mathbf{k}_{r-2} & \mathbf{k}_{r-2} & -\mathbf{k}_{r-2}\end{bmatrix}^\intercal$, moreover we have, for Lemma 3, that $\mathbf{k}_{r-1} = \begin{bmatrix}(\mathbf{k}_{r-2})_1\mathbf{k}_0 & (\mathbf{k}_{r-2})_2\mathbf{k}_0 & \ldots & (\mathbf{k}_{r-2})_{3^{r-1}}\mathbf{k}_0\end{bmatrix}^\intercal$. This means the first 3^r components of $\mathbf{k}'$ are $\mathbf{k}_{r-1}$, the same holds for the second 3^r components, and the last 3^r are $-\mathbf{k}_{r-1}$. This completes the proof.

□

Lemma 5. *Let us consider the matrix* $\mathbf{M}_r$ *of the family* $\mathcal{M}(DBL_2)$ *at round* r*.*
We have: $\begin{cases} min(\sum^+ \mathbf{k}_r, \sum^- \mathbf{k}_r) = \sum^- \mathbf{k}_r = \frac{1}{2}(3^r + 1) - 1 \\ \sum \mathbf{k}_r = 1 \end{cases}$

Proof. Thanks to lemma 4, we have $\sum^+ \mathbf{k}_r = 2\sum^+ \mathbf{k}_{r-1} + \sum^- \mathbf{k}_{r-1}$, $\sum^- \mathbf{k}_r = 2\sum^- \mathbf{k}_{r-1} + \sum^+ \mathbf{k}_{r-1}$ and $\sum^+ \mathbf{k}_0 = 2$, $\sum^- \mathbf{k}_0 = 1$. We first prove, by induction, that $\sum \mathbf{k}_r = (\sum^+ \mathbf{k}_r - \sum^- \mathbf{k}_r) = 1$.

- Base case *round=0*: $\sum^+ \mathbf{k}_0 - \sum^- \mathbf{k}_0 = 1$.
- Inductive Case *round=r:* $(\sum^+ \mathbf{k}_r - \sum^- \mathbf{k}_r) = (2\sum^+ \mathbf{k}_{r-1} + \sum^- \mathbf{k}_{r-1} - \sum^+ \mathbf{k}_{r-1} - 2\sum^- \mathbf{k}_{r-1}) = (\sum^+ \mathbf{k}_{r-1} - \sum^- \mathbf{k}_{r-1}) = 1$.

This result leads to the following recursive relation $\sum^+ \mathbf{k}_r = 3\sum^+ \mathbf{k}_{r-1} - 1$ for $\sum^+ \mathbf{k}_r$ with base condition $\sum^+ \mathbf{k}_0 = 2$. Solving the recursive relation we obtain $\sum^+ \mathbf{k}_r = \frac{1}{2}(3^{r+1}+1)$. This implies $min(\sum^+ \mathbf{k}_r, \sum^- \mathbf{k}_r) = \frac{1}{2}(3^{r+1}+1)-1$ where the last term takes into account the fact that the minimum is always the negative component and that $\sum \mathbf{k}_r = 1$.

□

From the previous lemma we have:

Lemma 6. *Let us consider $M, M' \in \mathcal{M}(DBL_2)$ such that their sizes are: $|W| = n$ and $|W'| = n+1$. Does not exist an algorithm $\mathcal{A}_l$ that at round $r \leq \lfloor log_3(2|n|+1) \rfloor$ is able to distinguish if it is running on multigraph M or M'.*

Proof. Let us suppose by contradiction that such algorithm $\mathcal{A}_l$ exists. For lemma 5 we have $\sum^- \mathbf{k}_r = \frac{1}{2}(3^{r+1}+1) - 1 \leq n$. Let us consider a configuration of non leader processes represented by vector $\mathbf{s}_r$ with $\sum \mathbf{s}_r = n$ and such that $(\mathbf{s}_r)_j \geq 1$ for each $j \,|\, (\mathbf{k}_r)_j < 0$ and $(\mathbf{s}_r)_j = 0$ otherwise. This implies there exists a dynamic multigraph $M : \{M_1, \ldots, M_r, \ldots\}$, of size n, obtained from $\mathbf{s}_r$ such that the leader state $S(v_l, r)$ at round r is represented by $\mathbf{m}_r = \mathbf{M}_r \mathbf{s}_r$. Thus $\mathcal{A}_l$ outputs n on the state $S(v_l, r)$.

Now let us consider $\mathbf{s'}_r = \mathbf{k}_r + \mathbf{s}_r$, by construction we have $\forall j | (\mathbf{s'}_r)_j > 0$ thus $\mathbf{s'}_r$ represents an instance of dynamic multigraph $M' : \{M'_1, \ldots, M'_r, \ldots\}$, let us denote $S'(v_l, r)$ the state of v_l in M'. Since we have $\sum \mathbf{s'}_r = \sum \mathbf{k}_r + \sum \mathbf{s}_r = n+1$, let us recall that from Lemma 5 we have $\sum \mathbf{k}_r = 1$, by hypothesis $\mathcal{A}_l$ outputs $n+1$ on the state $S'(v_l, r)$.

But by definition of kernel $\mathbf{M}_{r'} \mathbf{s}_r = \mathbf{M}_r \mathbf{s'}_r = \mathbf{m}_r$, thus $S(v_l, r) = S'(v_l, r)$ therefore $\mathcal{A}_l$ has to give the same output on the two different instances, that is a contradiction.

□

From the previous lemma and considering that $\mathcal{M}(\mathrm{DBL}_2) \subseteq \mathcal{M}(\mathrm{DBL}_k)$, we can state the following theorem whose proof is straightforward.

Theorem 2. *Any algorithm $\mathcal{A}$ cannot solve the counting on an instance $M \in \mathcal{M}(DBL_k)$ at round $r < \lfloor log_3(2|W|+1) \rfloor - 1$.*

From Theorem 2 and Lemma 2 the next theorem immediately follows.

Theorem 3. *Given an instance $G \in \mathcal{G}(PD)_2$ any counting algorithm $\mathcal{A}$ on G requires $\Omega(log|V|)$ rounds.*

From the previous Theorem we have the following corollary.

Corollary 1. *Given a dynamic network with dynamic diameter D, where D is constant w.r.t. $|V|$. We have that any counting algorithm $\mathcal{A}$ requires at least $D + \Omega(log|V|)$ rounds.*

Proof. We create a configuration where v_l is connected to two nodes v_1, v_2 by a static chain of $D-1$ nodes. Nodes v_1, v_2 are connected to the remaining $\mathcal{O}(|V|)$ nodes mimicking a $\mathcal{G}(\mathrm{PD})_2$ network. From this observation and Theorem 3 the next corollary follows. □

5.3 Smoothed Analysis of the Bound

In this section we will show that our bound is robust to a non-constant number of adversarial and uniform perturbations, in our model a perturbation is the addition or the removal of one edge. The smoothed analysis of bounds in the dynamic network has been first proposed in [12], where the case of uniform random perturbation has been investigated. We will follow their basic definitions and extend them to define *adversarial* perturbations.

Give two static graphs $G = (V, E), G' = (V, E')$ the *edit distance* between them is the minimum number of edge additions and removals that we need to transform G in G'. Given a static graph G and $k \in \{0, \ldots, \binom{n}{2}\}$, we define the set of static graphs: $editdist(G, k) = \{G' |$ if the *edit distance* between G, G' is less or equal to k $\}$. Now we are able to define the adversarial k-smooth of a $G \in \mathcal{G}(\mathrm{PD})_2$ adapting the definition from [12].

Definition 10. *Given $G = \{G_1, G_2, \ldots\} \in \mathcal{G}(PD)_2$ the dynamic graph $G' = \{G'_1, G_2, \ldots\} \in \mathcal{G}(PD)_2$ is the adversarial k-smooth of G if for each $G'_j \in G'$ we have $G'_j \in editdist(G_j, k)$.*

Intuitively we allow an external adversary to add or delete at most k edges at each round, with the restriction of generating graph always in $\mathcal{G}(\text{PD})_2$. We can weak this definition allowing the addition or deletion of up to k edges chosen uniformly random. In this case we obtain the k-smooth as defined in [12].

Definition 11. *Given $G = \{G_1, G_2, \ldots\} \in \mathcal{G}(PD)_2$ the dynamic graph $G' = \{G'_1, G_2, \ldots\} \in \mathcal{G}(PD)_2$ is the k-smooth of G if each $G'_j \in G'$ is uniformly sampled from $editdist(G_j, k)$.*

Essentially, when we k-smooth G we consider that, at each round r, up to k edges of the possible $\binom{v}{2}$ edges are selected uniformly random. Each of this k edges is added, if absent in G_r, or deleted, if present in G_r. We will show that our bound is resilient up to an adversarial $\mathcal{O}(\frac{\sqrt{|V|}}{\log(|V|)})$-smooth and up to an $\mathcal{O}(\sqrt{|V|})$-smooth.

Theorem 4. *Consider counting on an adversarial k-smoothed $G \in \mathcal{G}(PD)_2$ graph, with $k < \frac{\sqrt{|V|}}{\log(|V|)+2}$ and sufficiently large $|V|$. There is no counting algorithm that terminates before $\Omega(\log(|V|))$-th round.*

Proof. As we did in previous proof we first prove the same result for $M \in \mathcal{M}(\text{DBL}_2)$ and size $|V| + 1$. Let us consider round $r = c\log_3(|V|)$ where $c \in (0, 1)$ is a constant that we will specialise later. Let us consider the vector $\mathbf{u}_r$ of size 3^{r+1} defined as $(\mathbf{u}_r)_j = \frac{|V|}{3^r - \frac{1}{2}}$ if $(\mathbf{k}_r)_j == -1$ and $(\mathbf{u}_r)_j = 0$ otherwise. By immediate substitutions we have $(\mathbf{u}_r)_j > 2|V|^{1-c}$. This vector represents a dynamic multigraph $M : \{M_1, \ldots, M_r\}$. Let us consider a single round, $r' < r$, adversarial 1-smooth of M. Since the smooth operation cannot disconnect the multigraph, it corresponds to the addition or removal of some edges between the leader and the non leader node. This operation essentially changes the state $s = S(v, r')$ of one node v to s', obtaining the dynamic multigraph M' represented by a vector $\mathbf{u}'_r$. It is easy to see that $\mathbf{u}'_r = \mathbf{u}_r + \mathbf{x}$, where $\mathbf{x}$ is a vector with only two components different from zero and equal respectively to $-1, +1$. Where the -1 is the component corresponding to state s and $+1$ is the one corresponding to state s'. Generalising this reasoning we have that an adversarial k-smooth of M is a dynamic multigraph M' represented by a vector, with non negative components, $\mathbf{u}'_r = \mathbf{u}_r + \mathbf{x}$ where $|(\mathbf{x})_j| \leq k \cdot r \leq k \cdot c\log_3(|V|)$. For previous Lemma (see proof of Lemma 6) we have that as long as $\mathbf{u'}_r + \mathbf{k}_r$ has non negative components counting is impossible. It is easy to see that the previous statement holds if for each j such that $(\mathbf{k}_r)_j == -1$ we have $(\mathbf{u}_r)_j - |(\mathbf{x})_j| > 1$, the previous inequality is true if $(\mathbf{u}_r)_j - \lceil k \cdot c\log_3(|V|)\rceil > 0$. This holds if $k < \frac{2|V|^{1-c}}{c\cdot\log_3(|V|)+1}$ setting $c = \frac{1}{2}$ we have $k < \frac{4|V|^{\frac{1}{2}}}{\log_3(|V|)+2}$. Therefore despite the adversarial k-smoothing at round $r = \frac{1}{2}\log_3(|V|)$ counting is impossible.
Now we show that the same result is valid for dyn. graphs in $\mathcal{G}(\text{PD})_2$. The edges that the smoothing modifies between nodes in V_1, V_2 have been already considered in the analysis for $\mathcal{M}(\text{DBL}_2)$. It remains to analyze the effect of edges between nodes in V_2. At round r an edge between two nodes in V_2 allows them to exchange their state and to record in their history the round r, therefore this can be modelled in $\mathcal{M}(\text{DBL}_2)$ by adding to $\mathbf{u}_r$ a vector $\mathbf{x}$ that has a sum of negative component equal to -2 and 0 positive components, i.e. we consider as this two nodes are immediately counted by v_l at next round. This leads essentially to the same analysis of the previous case where we have a multiplicative factor of 2 for components of vector $\mathbf{x}$. The inequality for k is now $2k < \frac{4|V|^{\frac{1}{2}}}{\log_3(|V|)+2}$, that is satisfied under our assumptions. □

Theorem 5. *Consider counting on k-smoothed $G \in \mathcal{G}(PD)_2$ graph, with $k < \sqrt{|V|}$ and sufficiently large $|V|$. With high probability, that is with a probability greater or equal to $1 - \frac{1}{|V|}$, there is no counting algorithm that terminates before $\Omega(\log(|V|))$-th round.*

Proof. In this case we have that edges introduce by the smoothing are sampled uniformly random. Let us first prove our result for $M \in \mathcal{M}(\text{DBL}_2)$ and size $|V| + 1$. Let us consider round $r = \frac{1}{2}\log_3(|V|)$ and let us define vectors $\mathbf{u}_r$ as in the proof of Theorem 4. Let $\mathbf{u}'_r$ be the vector that represents the k-smooth of $\mathbf{u}_r$. Let $S : e_1, \ldots, e_{r\cdot k}$ be the set of edges involved in the smoothing. Each $e_i \in S$ added/removed corresponds to the addition of a vector $\mathbf{x}_{e_i}$, such vector has only two components different from zero and equal respectively to $-1, +1$. For simplicity let us assume that each $\mathbf{x}_{e_i}$ has only one component equal to -1 and all the others equal to 0, this component is chosen uniformly random from the 3^{r+1} possibility.

At round r we consider that $k \cdot r$ such vectors are added to $\mathbf{u}_r$ obtaining $\mathbf{U}$. Let us define as C the event: there exists an index j for which $(\mathbf{U})_j < 1$ and $j \in J$ with $J : \{j|(\mathbf{k}_r)_j == -1\}$, by Lemma 5 we have $|J| = \frac{|V|^{\frac{1}{2}}}{2} - \frac{1}{2}$. For previous Lemma (see proof of Lemma 6) we have that as long as $(\mathbf{u'}_r)_j > 1$ for j such that $(\mathbf{k}_r)_j == -1$ counting is impossibile, it is easy to see that the probability that such event does not hold is less or at most equal to the probability of event C.

In the following we will upper bound $\Pr[C]$. For the union bound we have $\Pr[C] \leq |V| \cdot \Pr[(\mathbf{U})_j < 1 \,|\, j \in J]$. Moreover it holds $\Pr[(\mathbf{U})_j < 1 \,|\, j \in J] \leq \Pr[|(\sum_{\forall e_i \in S} \mathbf{x}_{e_i})_j| > 2|V|^{\frac{1}{2}} \,|\, j \in J]$. Let us notice that the random variable $X_j = |(\sum_{\forall e_i \in S} \mathbf{x}_{e_i})_j|$ is the sum of $r \cdot k$ independent Bernoulli random variables. Therefore we have $\mathrm{E}[X_j] = \frac{k \log_3(|V|)}{2|J|}$. By applying the multiplicative form of the Chernoff Bound we have $\Pr[X_j > 5\mathrm{E}[X_j]] < (\frac{e^4}{5^5})^{\mathrm{E}[X_j]} < \frac{1}{|V|^2}$.

It remains to prove that $5\mathrm{E}[X_j]] \leq 2|V|^{\frac{1}{2}}$, by substitution we have $5\mathrm{E}[X_j]] \leq 5\frac{k \log_3(|V|)}{2|V|^{\frac{1}{2}}}$ and we have $5\frac{k \log_3(|V|)}{2|V|^{\frac{1}{2}}} \leq 2|V|^{\frac{1}{2}}$ if $k \leq \frac{4}{5} \frac{|V|}{\log_3(|V|)}$, that for sufficiently large $|V|$ is implied by $k < \sqrt{|V|}$.

From this we have $\Pr[\mathrm{not}(C)] \geq 1 - \frac{1}{|V|}$. Now we have to prove that the result holds for $\mathcal{G}(\mathrm{PD})_2$, this part of the proof follows the same line of Th. 4. This leads essentially to the same analysis of the previous case where we have a multiplicative factor of 2 for components of vector $\mathbf{x}$. The inequality for k is now $2k \leq \frac{4}{5} \frac{|V|}{\log_3(|V|)}$ that is satisfied under our assumptions. □

Discussion Our results shows that the $\Omega(\log n)$ bound for counting in $\mathcal{G}(\mathrm{PD})_2$ is not fragile. On the contrary it is rather robust and it resists to a substantial amount of adversarial and non-adversarial smoothing. Intuitively, these results imply that our bound represents a rather wide set of dynamic graphs.

5.4 Trade-Off Lower Bound on Accuracy Versus Time

The second results on dynamic networks with constant diameter is about the accuracy of counting algorithms terminating before $D + \Omega(\log |V|)$. We consider a leader based estimation algorithm $\mathcal{A}_e$ that takes as input an upper bound U on $|V|$ and at round r v_l outputs a guess on the network size and terminates.

Theorem 6. *Let $\mathcal{A}_e$ be any counting algorithm taking as input an upper bound U on the network size. If v_l outputs a guess on the network size $n_{\mathcal{G}}$ at a certain round r with $D < r < D + log_3(\frac{U}{4})$, then there always exists a dynamic network G of size $|V| \in [\frac{U}{2}, U]$ such that v_l outputs $n_{\mathcal{G}}$ on G at round r and $||V| - n_{\mathcal{G}}| \geq \frac{U}{4(3^r)}$.*

Proof. To prove the claim we need to introduce two intermediate results. We first prove in Lemma 7 that the claim of Theorem 6 holds for the family of multigraphs $\mathcal{M}(\mathrm{DBL}_2)$. Lemma 7 needs, in its turn, an intermediate result (Lemma 8) showing that a system of equations $\mathbf{u}_r = \mathbf{M}_r \mathbf{s}_r$ with a given $\mathbf{u}_r$ has a set of solutions sharing a specific structure. Secondly, we introduce a corollary of Lemma 2 extending the results obtained for $\mathcal{M}(\mathrm{DBL}_2)$ to the family of graphs $\mathcal{G}(\mathrm{PD})_2$.

Lemma 7. *Let $\mathcal{A}_e$ be any counting algorithm on $\mathcal{M}(DBL_2)$ taking as input an upper bound U on the network size. If v_l outputs a guess on the network size $n_{\mathcal{G}}$ at a certain round r with $D < r < D + log_3(\frac{U}{4})$, then there always exists a dynamic multigraph, $M \in \mathcal{M}(DBL_2)$ of size $|W| \in [\frac{U}{2}, U]$ such that v_l outputs $n_{\mathcal{G}}$ on M at round r and $||W| - n_{\mathcal{G}}| \geq \frac{U}{4(3^r)}$.*

Proof. The proof is constructive. We show that the leader could get in a state $S(v_l, r)$ such that, each multigraph of the family $\mathcal{M}(\mathrm{DBL}_2)$ that could generate $S(v_l, r)$ has a size that falls in the interval $[\frac{U}{2}, U]$. Let $SET \subseteq \mathcal{M}(\mathrm{DBL}_2)$ be the the set of such multigraphs. SET has the following properties: SET cardinality is $\frac{U}{2(3^r)}$ and each multigraph belonging to SET has a different size.

Therefore if $\mathcal{A}_e$ makes a choice and outputs a guess at round r, we have a multigraph $M \in SET$ such that the difference between the guess and the actual size of M is at least $\frac{U}{4(3^r)}$.

Without loss of generality, let us assume that $U = 2(3)^{\Delta}$ for some $\Delta \in \mathbb{N}^+$.

We first prove an intermediate results:

Lemma 8. *Let consider the system* $\mathbf{u}_r = \mathbf{M}_r\mathbf{q_r}$ *with* $\mathbf{u}_r = \begin{bmatrix}\frac{U}{2} & \frac{U}{2} & \frac{U}{2(3^1)} & \frac{U}{2(3^1)} & \frac{U}{2(3^1)} & \cdots & \frac{U}{2(3^r)}\end{bmatrix}^\intercal$ *where* $\mathbf{u}_r$ *is a vector of size* $\sum_{i=0}^{r} 2\cdot 3^i$ *whose components are defined as follows:*

- $(\mathbf{u}_r)_1 = (\mathbf{u}_r)_2 = \frac{U}{2}$*;*
- *each component* $(\mathbf{u}_r)_j$ *with index* j *in the interval* $[\sum_{i=0}^{k-1} 2\cdot 3^i + 1, \sum_{i=0}^{k} 2\cdot 3^i]$ *with* $k \in \mathbb{N}^+$ *has value equal to* $\frac{U}{2(3^k)}$ *(i.e.* $(\mathbf{u}_r)_3 = \frac{U}{6}$ *we have* $3 \in [[\sum_{i=0}^{k-1} 2\cdot 3^i + 1, \sum_{i=0}^{k} 2\cdot 3^i]$ *with* $k = 1$ *).*

The solution vector $\mathbf{q}_r$ *of size* 3^{r+1} *has the following structure:*

- *if* $(\mathbf{k}_r)_j = 1$ *then* $(\mathbf{q}_r)_j = 0$*;*
- *if* $(\mathbf{k}_r)_j = -1$ *then* $(\mathbf{q}_r)_j = \frac{U}{2\cdot 3^r}$*.*

Proof. The proof is by induction on r:

- **Base case** $r = 0$**:** By substitution we immediately obtain $\mathbf{u}_0 = \mathbf{M}_0\mathbf{q}_0$.
- **Inductive case** r: Our inductive hypothesis is: $\mathbf{u}_{r-1} = \mathbf{M}_{r-1}\mathbf{q}_{r-1}$. Let us recall, see proof of Lemma 3, that $\mathbf{M}_r = \begin{bmatrix}\mathbf{M}'_{r-1}\\ \mathbf{U}\end{bmatrix}$, where $\mathbf{M}'_{r-1}$ is the matrix obtained by $\mathbf{M}_{r-1}$ substituting each element $1/(0)$ of $\mathbf{M}_{r-1}$ with a row vector $\begin{bmatrix}1 & 1 & 1\end{bmatrix}/(\begin{bmatrix}0 & 0 & 0\end{bmatrix})$. $\mathbf{M}'_{r-1}$ has $\sum_{i=0}^{r-1} 2(3)^i$ rows.
  We first show that for each row of $\mathbf{U}$, we have $(\mathbf{U})_j\mathbf{q}_r = \frac{U}{2(3^r)}$. Let us recall that $\mathbf{U}$ has $2\cdot 3^r$ rows and that $(\mathbf{u}_r)_j = \frac{U}{2(3^r)}$ for each $j \in [2\cdot 3^{r-1}+1, 2\cdot 3^r]$. For each row $(\mathbf{U})_j$ considering the generic subvector $\mathbf{v}_i : \begin{bmatrix}((\mathbf{U})_j)_{3i-2} & ((\mathbf{U})_j)_{3i-1} & ((\mathbf{U})_j)_{3i}\end{bmatrix}$, with $i \in [1, 3^r]$, we have that $\mathbf{v}_i$ could be either $\begin{bmatrix}1 & 0 & 1\end{bmatrix}$ or $\begin{bmatrix}0 & 1 & 1\end{bmatrix}$ or $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$. Moreover there is only one $\mathbf{v}_i$, namely $\mathbf{v}'$, different from the zero vector. For the structure of $\mathbf{k}_r$, see proof of Lemma 4, we have that the groups of three components of $\mathbf{q}_r$ (i.e., the subvectors $\begin{bmatrix}(\mathbf{q}_r)_{3i-2} & (\mathbf{q}_r)_{3i-1} & (\mathbf{q}_r)_{3i}\end{bmatrix}$ with $i \in [1, 3^r]$) are either $\begin{bmatrix}\frac{U}{2(3^r)} & \frac{U}{2(3^r)} & 0\end{bmatrix}$ or $\begin{bmatrix}0 & 0 & \frac{U}{2(3^r)}\end{bmatrix}$. Therefore we have $\mathbf{v}'\cdot\begin{bmatrix}0 & 0 & \frac{U}{2(3^r)}\end{bmatrix}^\intercal = \mathbf{v}'\cdot\begin{bmatrix}\frac{U}{2(3^r)} & \frac{U}{2(3^r)} & 0\end{bmatrix}^\intercal = \frac{U}{2(3^r)}$. This implies $(\mathbf{U})_j\mathbf{q}_r = \frac{U}{2(3^r)}$.
  To complete the proof, we have to prove that $(\mathbf{M}'_{r-1})_j\mathbf{q}_r = (\mathbf{u}_r)_j$. From the structure of $\mathbf{u}_r$, this is equivalent to prove that $\mathbf{M}'_{r-1}\mathbf{q}_r = \mathbf{u}_{r-1}$. Let us build the vector $\mathbf{q}'$ of size 3^r obtained by $\mathbf{q}_r$ in such a way that $(\mathbf{q})'_i = (\mathbf{q}_r)_{3i-2} + (\mathbf{q}_r)_{3i-1} + (\mathbf{q}_r)_{3i}$. In the proof Lemma 4 we have shown that if $(\mathbf{k}_{r-1})_i = 1/(-1)$ then $(\mathbf{k}_r)_{3i-2} = 1/(-1), (\mathbf{k}_r)_{3i-1} = 1/(-1), (\mathbf{k}_r)_{3i} = -1/(1)$, from this follows:
  - if $(\mathbf{k}_{r-1})_j = 1$ then $(\mathbf{q})'_j = \frac{U}{2(3^r)}$;
  - if $(\mathbf{k}_{r-1})_j = -1$ then $(\mathbf{q})'_j = 2\frac{U}{2(3^r)}$

  Therefore $\mathbf{q}' - \frac{U}{2(3^r)}(\mathbf{k}_{r-1}) = \mathbf{q}_{r-1}$. Since $\mathbf{k}_{r-1}$ is a kernel vector, we have $\mathbf{M}_{r-1}\mathbf{q}_{r-1} = \mathbf{M}_{r-1}\mathbf{q}' = \mathbf{u}_{r-1}$. Considering the structure of $\mathbf{M}'_{r-1}$, we have $\mathbf{M}'_{r-1}\mathbf{q}_r = \mathbf{M}_{r-1}\mathbf{q}'$. This complete the proof.

□

(cont. Lemma 7) The construction works as follows:

- $\mathcal{A}_e$ *outputs the guess at round* $r = 0$. At round $r = 0$, the leader state is $\mathbf{u}_0 = \begin{bmatrix}\frac{U}{2} & \frac{U}{2}\end{bmatrix}^\intercal$. The set of possible solutions for $\mathbf{u}_0 = \mathbf{M}_0\mathbf{s}_0$ is given by $\mathbf{q}_0 + t_0\mathbf{k}_0$, where $t \in [0, \frac{U}{2}]$, and $\mathbf{q}_0 = \begin{bmatrix}0 & 0 & \frac{U}{2}\end{bmatrix}^\intercal$. The best strategy for $\mathcal{A}_e$ is to pick $n_{\mathcal{G}}$ such that $n_{\mathcal{G}} = \sum(\mathbf{q}_0 + x\mathbf{k}_0)$ with $x \in [0, \frac{U}{2}]$. Therefore there exists a multigraph with size $n = \sum(\mathbf{q}_0 + x_{\mathcal{A}}\mathbf{u}_0^0)$, thus we have $|n - n_{\mathcal{G}}| = |x - x_{\mathcal{A}}| \geq \frac{U}{4}$ which represents the error done issuing a guess at round 0.
- $\mathcal{A}_e$ *outputs the guess at round* $r = 1$. The leader state is $\mathbf{u}_1 = \begin{bmatrix}\frac{U}{2} & \frac{U}{2} & \frac{U}{6} & \frac{U}{6} & \frac{U}{6} & \frac{U}{6} & \frac{U}{6} & \frac{U}{6}\end{bmatrix}^\intercal$ the set of possible solutions of $\mathbf{u}_1 = \mathbf{M}_1\mathbf{s}_1$ is $\mathbf{q}_1 + t_1\mathbf{k}_1$ with $t_1 \in [0, \frac{U}{6}]$,
  and $\mathbf{q}_1 = \begin{bmatrix}0 & 0 & \frac{U}{6} & 0 & 0 & \frac{U}{6} & \frac{U}{6} & \frac{U}{6} & 0\end{bmatrix}^\intercal$. These solutions are actually the multigraphs in *SET*. The number of possible solutions (i.e., the cardinality of *SET*) is $\frac{U}{6} + 1$

Let $n_{\mathcal{G}}$ be the guess of $\mathcal{A}_e$, there is at least a multigraph in SET whose size is n such that $|n - n_{\mathcal{G}}| > \frac{U}{12}$ which represents the error done issuing a guess at round 1.

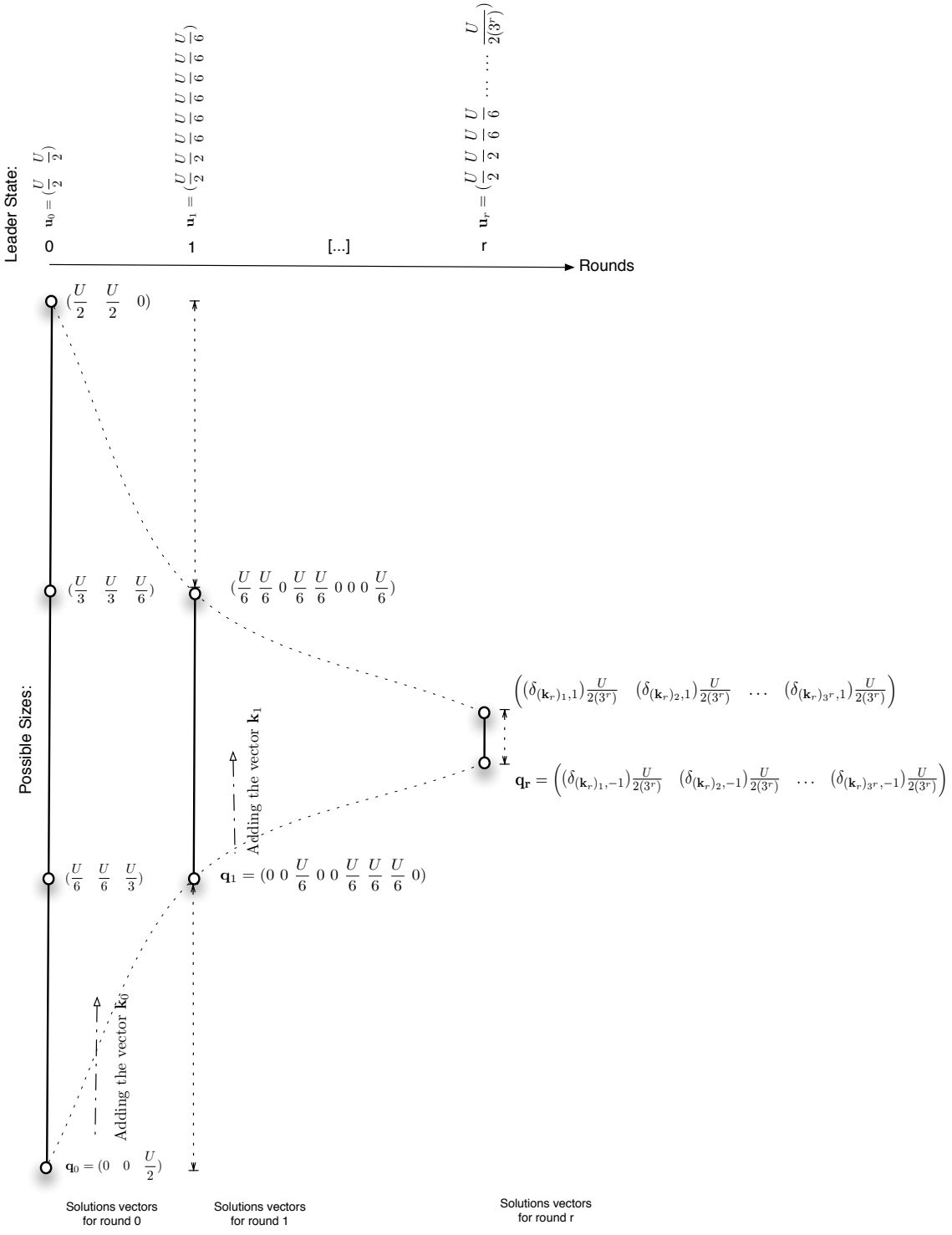

Figure 6: Set of possible solutions as round passes

- $\mathcal{A}_e$ *outputs the guess at round* r. The leader state is the vector

$\mathbf{u}_r = \left[\frac{U}{2} \quad \frac{U}{2} \quad \frac{U}{2(3^1)} \quad \frac{U}{2(3^1)} \quad \frac{U}{2(3^1)} \quad \cdots \quad \frac{U}{2(3^r)}\right]^\intercal$ of size $\sum_{i=0}^{r} 2 \cdot 3^i$. The element $(\mathbf{u}_r)_j$ with $j \in [\sum_{i=0}^{k} 2 \cdot 3^i, \sum_{i=0}^{k+1} 2 \cdot 3^i]$ and $k \in [0, \ldots, r]$ is equal to $\frac{U}{2(3^k)}$.
The set of possible solutions of $\mathbf{u}_r = \mathbf{M}_r \mathbf{s}_r$ is $\mathbf{q}_r + t_r \mathbf{k}_r$ with $t_r \in [0, \frac{U}{2(3^r)}]$ and $\mathbf{q}_r$ obtained by Lemma 8. For Lemma 5 we have $\sum \mathbf{q}_r = (\frac{1}{2}(3^r+1)-1)\frac{U}{2(3^r)}$ and $\sum(\mathbf{q}_r + \frac{U}{2(3^r)}\mathbf{k}_r) = (\frac{1}{2}(3^r+1))\frac{U}{2(3^r)}$, and that the difference of processes number of two adjacent solutions $\mathbf{q}_r + x\mathbf{k}_r$ and $\mathbf{q}_r + (x+1)\mathbf{k}_r$ is 1.

These solutions are actually the multigraphs in *SET*. The number of possible solutions (i.e., the cardinality of *SET*) is $\frac{U}{2(3^r)} + 1$.

Let $n_{\mathcal{G}}$ be the guess of $\mathcal{A}_e$, there is at least a multigraph in *SET* whose size is n such that $|n - n_{\mathcal{G}}| > \frac{U}{4(3^r)}$ which represents the error done issuing a guess at round r.

□

Corollary 2. *Let $S(v_l, r)$ be the state of v_l, in a dynamic multigraph $M \in \mathcal{M}(DBL_2)$. This state can be generated by a set SET of dynamic multigraphs with different sizes. There exists a state $S'(v'_l, r)$ for a leader node v'_l, in a dynamic network $G \in \mathcal{G}(PD)_2$ such that (1) this state can be generated by a set SET' of dynamic networks with different sizes and such that (2) $|SET'| \geq |SET|$.*

Proof. Let us consider the transformation used in the proof of Lemma 2. Using this transformation we have for each multigraph $M \in \mathcal{M}(\mathrm{DBL}_2)$, it exists a dynamic network $G^{id} \in \mathcal{G}(\mathrm{PD})_2$, with $|V_1| = 2$ and $|V_2| = |W|$, such that, at each round r, the state of $v_l \in M$ contains exactly the same information contained in the union of memories of nodes in $\{v'_l\} \cup V_1$. Leveraging this observation: We have that each for each multigraph $M \in SET$ that generates the state $S(v_l, r)$, there exists a $G^{id} \in SET'$ that generates $S'(v_l, r)$. Therefore the claim follows. □

(cont. Theorem 4) The accuracy bound obtained in Lemma 7 for $\mathcal{M}(\mathrm{DBL}_2)$, by Corollary 2, it also holds for $\mathcal{G}(\mathrm{PD})_2$. From the bounds obtained on $\mathcal{G}(\mathrm{PD})_2$, the claim of the theorem follows by using the same construction used in Corollary 1. □

Discussion This trade-off lower bounds complements our lower bound of $D + \Omega(\log |V|)$ rounds, by showing that any algorithms that runs in less then $D + o(\log |V|)$ rounds has to make an error that is function of the actual size, hence, it can be made arbitrarily big. This cannot be derived immediately from the lower bound on counting, since it shows the impossibility to distinguish a network of size n from a network of size $n + 1$, they have a constant difference of just one process.

5.5 Counting on $\mathcal{G}(\mathrm{PD})_2$ with halting processes

In this section we consider a graph in $\mathcal{G}(\mathrm{PD})_2$ where processes in V_2 may halt at some point. We say that v_i *halts at round* r if it has send messages for any round $r'' < r$, and it does not send messages for any $r' \geq r$. We assume that processes halt from round $r \geq 1$; that is they send at least one message before their departure. The results shown in this section point out the complexity of the counting problem in presence of failures, i.e. crash-stop processes. Nevertheless, the counting algorithm that we propose it is also a a tool used to solve counting in $\mathcal{G}(1\text{-IC})$, (see Sec. 6).

Let us introduce the **Valid Count Detection Problem**:

Definition 12. VCDP *Given two run R, R_{NC} such that: in the run R no process halts; in the run R_{NC} there are processes that halt. An algorithm solves the Valid Count Detection Problem if at some round r it outputs a value and terminates. The output could be either a special value* NOCOUNT *or a number $C = |V_2|$. When the algorithm is executed in R the output value has to be C, when it is executed in R_{NC} it could be either C or* NOCOUNT.

Lower Bound on VCDP problem One natural question is about lower bounds for the specific **VCDP** problem on $\mathcal{G}(\text{PD})_2$. One could ask if for this problem it holds the same lower bound that we have shown for counting. The answer is negative, we can show that the lower bound for this problem is exponentially worse with respect to the lower bound to solve the simple counting. In order to show this lower bound we introduce the intermediate problem of detecting if we are in a run R_{NC} where processes halt starting from round 1 from a run R where processes do not halt, that is the **Halt Detection** problem. An algorithm $\mathcal{A}_{\mathbf{VCDP}}$ that solves **VCDP** can be used to solve the **Halt Detection**. We just starts two instances of $\mathcal{A}_{\mathbf{VCDP}}$, the instance i_0 at round 0 and the instance i_1 at round 1. It is easy to verify that we are in the run R if and only if the output of the two instances is the same and it is different from NOCOUNT. On the contrary if the two instances returns different values or one of the two outputs NOCOUNT we are in R_{NC}.

Theorem 7. *Let us consider a graph $G \in \mathcal{G}(PD)_2$, where processes in V_2 may fail. If $|V_2| > |V_1|$ we have that does not exists any algorithm that solves* **Halt Detection** *in less then $|V_1|\lfloor\log(\lfloor\frac{|V_2|}{|V_1|}\rfloor + 1)\rfloor$ rounds, if $|V_2| \leq |V_1|$ the number of rounds is $|V_2|$.*

Proof. We assume that a failure happens at round $r = 1$, we denote as v_i^x that $v_i \in V_x$.

- $|V_2| \leq |V_1|$. Let us first consider the case $|V_2| = |V_1|$. We consider the G_0 at round 0 in which each process v_j^2 in V_2 is connected to only one process v_j^1 in V_1 and viceversa. At the end of round 0 thanks to the anonymous broadcast all processes in V_2 have the same state, the same holds for all processes in V_1. At the beginning of round 1, v_1^2 halts. The adversary takes process v_2^2 in V_2 and connects it to v_1^1 and v_2^1, for processes in v_1^1,v_2^1 this configuration is not distinguishable from the one of the previous round, and clearly is not distinguishable for the other processes. At round r, v_r^2 halts and v_{r+1}^2 is connected to $v_1^1, v_2^1, \ldots, v_{r+1}^1$. At round $r = |V_2| - 1$ only one process of V_2 is left $v_{|V_2|}^2$ and it is connected to $v_1^1, \ldots, v_{|V_2|}^1$. For processes in V_1, and thus for the leader, this run is not distinguishable from a dynamic graph $\{G_0, G_1 = G_0, \ldots, G_{|V_2|-1} = G_0\}$. This complete the proof for $|V_2| = |V_1|$, the case $|V_2| < |V_1|$ is analogous.

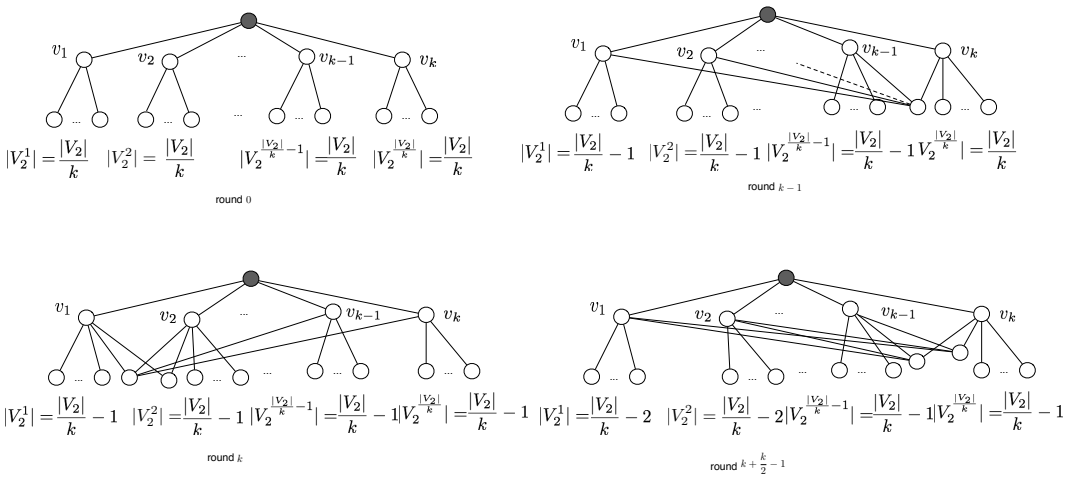

Figure 7: Halt Detection Lower bound: indistinguishability example for $|V_1| > |V_2|$.

- $|V_2| > |V_1|$. Now let us consider the case $|V_2| > |V_1|$, we consider the subset of instances in which $|V_1| = x!$ and $|V_2| = x|V_1|$ for some $x \in \mathbb{N}^+$.

  Nodes in V_2 are partitioned in x sets, let us define as V_2^j the j-th set and as $v_i^{j,2}$ the i-th process in the set V_2^j. In G_0 we have that each $v_i^{j,2}$ is connected only to v_j^1, see Figure 7. In G_1, $v_1^{1,2}$ halts and $v_1^{2,2}$ is connected to v_1^1, v_2^1. Thanks to the anonymous broadcast the memory content of $v_1^{2,2}$ is equal to the memory content of $v_1^{1,1}$ thus from the point of view of v_1^1, v_2^1 this graph is not distinguishable from $G_1 = G_0$. This strategy is iterated for the first $|V_1|$ rounds, at round $j \in [0, |V_1| - 1]$ the

adversary connects $v_1^{j,2}$ to $v_1^1, \ldots, v_j^1$, all processes in V_1 are kept in the same state, at each round a process in V_1 receives x identical messages by processes in V_2, this implies that the non halted processes in V_2 are kept in an identical state. As result we have at round $r = |V_2|$, $|V_2^j| = x - 1$ for all $j \in [1, |V_1|]$. Now at round $r = |V_2|$ the adversary uses two processes to mask the halted process of the previous rounds, $v_2^{1,2}$ is connected to $v_1^1, \ldots, v_{|V_1|}^1$ and $v_2^{2,2}$ to v_1^1, v_2^1, since at the beginning of round $r = |V_2|$ all processes in V_2 are in the same state each process in V_1 will receive x identical messages. At round $r = |V_2| + 1$ the adversary connects $v_2^{3,2}$ to all process in V_1 and $v_2^{4,2}$ to $v_1^1, v_2^1, v_3^1, v_4^1$.

Following this strategy given a round r such that the non halted processes in V_2 are $|V_2| - a \cdot |V_1| - (a+1) \cdot j$ we have that V_2^i with $i \in [1, (a+1)j]$ have $x - (a+1)$ non halted processes and the remaining sets have $x - a$ non halted processes, thus the adversary connects the processes $v_{a+1}^{(a+1)j+1,2}, \ldots, v_{a+1}^{(a+1)j+(a+1),2}$ to all processes in V_1 and the process $v^{(a+1)(j+1)+1}$ to $v_1^1, \ldots, v_{(a+1)(j+1)+1}^1$.

Now we have to prove that this behaviour can be iterated for $|V_1|\lfloor \log(\lfloor \frac{|V_2|}{|V_1|} \rfloor) \rfloor$ rounds. Let us focus on the number of active processes in the system, as long as non halted processes in V_2 are $[|V_2| - a \cdot |V_1|, |V_2| - (a+1) \cdot |V_1|]$ the adversary needs to use $(a+1)$ processes, at each round, to mask the halted processes. These processes will halt in the next round. This means that starting from $|V_2| - a \cdot |V_1|$ non halted processes we remain with $|V_2| - (a+1) \cdot |V_1|$ non halted processes after $\frac{|V_1|}{a+1}$ rounds. After $|V_1|$ rounds we have $|V_2| - |V_1|$ non halted processes, after $\frac{|V_1|}{2}$ rounds we have $|V_2| - 2|V_1|$ non halted processes, and so on until we have 0 non halted processes. The total number of rounds is $|V_1| \sum_{i=1}^{\frac{|V_2|}{|V_1|}} \frac{1}{i} \geq |V_1| \log(\frac{|V_2|}{|V_1|} + 1)$. For processes in V_1, and for v_l, this run is not distinguishable from a dynamic graph $\{G_0, G_1 = G_0, \ldots, G_{|V_1| \log(\frac{|V_2|}{|V_1|}+1)-1} = G_0\}$.

□

Discussion Theorem 7 and the previous considerations lead to a lower bound for the solution of **VCDP** problem on $\mathcal{G}(\mathrm{PD})_2$. It is interesting to notice that in this case the adversary can exploit a "*chain of halts*" leading to a number of rounds that in the worst case is exponentially far from $\mathcal{O}(\log(|V_2|))$, i.e. $\Omega(|V_2|)$. moreover the asymptotical number of rounds necessary to solve the problem depends from $|V_1|$. It is worth to highlight the difference with respect to $\mathcal{G}(\mathrm{PD})_2$ networks with IDs where the time needed to count is the same time needed to detect crash stop faults, therefore this bound points out another aspect impacted by anonymity.

5.5.1 VCD Algorithm to solve VCDP

When identifiers are present a simple broadcast algorithm solves **VCDP** in $\mathcal{G}(\mathrm{PD})_2$. In our model we solve it by using an extension OPT, denoted as OPT*. When processes halt OPT* has a peculiar *"overestimation"* property (see Lemma 10).

Algorithm OPT* The algorithm OPT* differs from OPT in:

- Its output is considered not valid if one of the following conditions holds: (i) the value n computed for some node of the tree is not in $\mathbb{N}^+$; (ii) if some of the Equations 1 are violated, i.e. $m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1}]} \gtrless \sum_{i=1}^{|V_1|} i \cdot n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},i]}$; (iii) if at round $r+2$ there exists a node in T with label $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ and at round $r+3$ does not exists a node with label $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r, *]$.
- Its counting rule is a restricted version of the OPT counting rule. Specifically: when in T there is a non-leaf node with label $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ such that *it has only one child* $[\perp, x_0, \ldots, x_{r-1}, x_r, j]$ the leader computes $n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]}$ using:

$$m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r]} = j \cdot n_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r,j]}$$

When the leader knows the values n for each of the children of a non leaf-node t, it sums the children values and sets the n_t (see the second equation of Eq. 1).

Algorithm OPT^* has the following properties:

Lemma 9. *Let R be a run produced by OPT^*. R terminates in $\mathcal{O}(|V_2|)$ rounds.*

Proof. The counting rule on leaf is applied only when the leaf has no siblings. Let x_0 be a node with at least two children x_1, x_2. We have by construction that the number of processes with history x is strictly greater that the number of processes with history x_1 and with history x_2. Since a node x may have more then one child if and only if the number of processes with history x is greater than one we have that after at most $\mathcal{O}(|V_2|)$ either each leaf has no siblings or the run R is not valid. In both cases OPT^* terminates. □

Lemma 10. *Let R be a run produced by OPT^* that starts at round 0 and $|V_2^f|$ be the number of non-halted processes in V_2 at the end of the execution of OPT^*. If at some round $r > 0$ processes in V_2 halt, then if the output C of OPT^* is valid we have $C > |V_2^f|$.*

Proof. For Lemma 9 OPT^* terminates we have to show that its output satisfies the lemma statement. Without loss of generality we consider only the case of a valid output of the algorithm. Let $l_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r]}$ be the number of processes with degree history $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ that halt at round $r+1$. Let us first consider the application of the counting rule on a leaf $[\perp, x_0, \ldots, x_{r-1}, x_r, j]$ of T with father $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$. For the counting rule of OPT^* we have that the leaf has no siblings. We have $m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r]} = L + j(n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]} - l_{[\perp,x_0,\ldots,x_{r-1},x_r,j]})$. Where $L = \sum_{\forall a:[\perp,x_0,\ldots,x_r,i]|l_a>0} i \cdot l_{[\perp,x_0,\ldots,x_r,i]} \geq 0$ is the number of edges from halted processes that have been counted in $m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1},x_r]}$. We have $(n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]} - l_{[\perp,x_0,\ldots,x_{r-1},x_r,j]}) > 0$ otherwise the outputs is not valid. Thus the leader computes a value for the node with label $[\perp, x_0, \ldots, x_{r-1}, x_r, j]$ that is $n^*_{[\perp,x_0,\ldots,x_{r-1},x_r,j]} = \frac{L}{j} + n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]} \geq n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]}$. If $L > 0$ the assigned value is clearly greater w.r.t to the actual number of non-halted processes with that degree history. The other case that we have to examine in our induction on T is when the counting rule sets the value of a non-leaf node $[\perp, x_0, \ldots, x_r]$. In this case we have that if $\exists\, l_{[\perp,x_0,\ldots,x_r,y]} > 0$ then $m_{[\perp,x_0,\ldots,x_r]} = L + \sum_1^{|V_1|} i(n_{[\perp,x_0,\ldots,x_r,i]} - l_{[\perp,x_0,\ldots,x_r,i]})$ thus if each $n_{[\perp,x_0,\ldots,x_r,i]}$ is not set to an overestimate of the number of processes with degree history $[\perp, x_0, \ldots, x_r, i]$ we would have $m_{[\perp,x_0,\ldots,x_r]} > \sum_1^{|V_1|} i(n_{[\perp,x_0,\ldots,x_r,i]} - l_{[\perp,x_0,\ldots,x_r,i]})$ leading to a non valid output. From these observations it is easy to see that each $v \in V_2^f$ is counted at least once in the value associated to some leaf node. If $\exists l_{[X]} > 0$ it follows, since the leader aggregates the value of nodes in T towards the root, that $n_{[\perp]} > |V_2^f|$.

□

Informally the previous Lemma says that, if there are halted processes, the output of OPT^* is always an overestimate on the number of non-halted processes. The following lemma states that if no process halts then the output is the number of processes.

Lemma 11. *Let R be a run produced by OPT^* that starts at round 0. If no process in V_2 halts during the run, then the output of OPT^* is valid and it is the correct count of processes in V_2.*

Proof. For Lemma 9 OPT^* terminates we have to show that its output satisfies the lemma statement. When no process halts the proof of correctness follows the same step of the correctness proof of OPT (Lemma 1): the base case is the same, the inductive case it is slightly different in the fact that the condition to set node value has to be $\nexists v_1 : [x_0, \ldots, x_{r+1}] \in C_{v_0} \setminus X_{v_0}$. It is straightforward that the modification on the counting rule only impacts on counting time, i.e. $\mathcal{O}(|V_2|)$ instead of $\mathcal{O}(\log |V_2|)$ (see proof of Th. 1 of OPT), and not on its correctness. □

Algorithm VCD The algorithm executes sequentially k runs of OPT^* starting from round 0, for some $k > |V_2|$. The leader compares the output of these runs: if they are all equal and valid, then VCD outputs the count obtained by the first run of OPT^*. Otherwise VCD outputs $\mathsf{NOCOUNT}$. The value k is computed by counting the edges connecting processes in V_1 with processes belonging to V_2 at round 0. This can be done trivially by v_l using messages from nodes in V_1. Each node in V_1 has to simply count neighbors in V_2, the sum of these partial counts is equal to $k-1$. Since OPT^* runs in $\mathcal{O}(|V_2|)$ rounds and $k = \mathcal{O}(|V_1||V_2|)$, we have VCD runs in $\mathcal{O}(|V|^3)$ rounds.

Theorem 8. *Algorithm* VCD *solves the* **VCDP** *problem.*

Proof. Let $\{i_0, i_1, \ldots, i_k\}$ be k runs of VCD. Let us first consider the execution of VCD on R. In R no process halts, thus we have for Lemma 11 that all outputs $\{c_0, c_1, \ldots, c_k\}$ will be equal. Therefore VCD terminates outputting $c_0 = |V_2|$. Let us consider the execution of VCD on R_{NC}. Let us define as $\{|V_2^{f_0}|, |V_2^{f_1}|, \ldots, |V_2^{f_k}|\}$ the number of non-halted processes in the system at the end of instance $1, 2, \ldots$. Now two cases may arise:

- $|V_2| = |V_2^{f_0}|$, in this case for Lemma 11 we have $c_0 = |V_2|$. Therefore if all $\{c_0, c_1, \ldots, c_k\}$ are equal and valid then VCD outputs the correct count otherwise the algorithm outputs NOCOUNT. In any case the output is correct.
- $|V_2| > |V_2^{f_0}|$ in this case by using Lemma 10 we have $c_0 > |V_2^{f_0}|$. Processes in V_2 are less than k, this implies that we have at least one instance i_j for which no process halts during its execution. For Lemma 11 the instance i_j outputs the value $c_j = |V_2^{f_{j-1}}| = |V_2^{f_j}| \neq c_0$. Thus the algorithm outputs NOCOUNT on R_{NC}. That is a correct output.

□

5.6 Optimal Counting in $\mathcal{G}$(PD): OPT_h

As in OPT, OPT_h begins with a *get_distance* phase over $\mathcal{G}(\text{PD})_h$ where each process obtains its distance from the leader. Using a simple flooding and convergecast algorithm this phase takes at most $2h + 1$ rounds. In the first h rounds (flooding step) each process computes its distance from v_l, in the $h + 1$ successive rounds (convergecast step) the leader computes the maximum distance h.

Non-Leader process behavior in OPT_h. The code of a non-leader process in OPT_h is reported in Figure 2, the function *count_distance_neighbors* returns the number of messages in MS generated by processes at distance $distance - 1$, the function *get_messages_from_distance* returns only messages generated by processes at distance $distance + 1$. If there is no such message the function returns $\perp$. As in OPT, a process v updates its degree history $v.H(r)$ by counting the number of processes in $N(v, r)$ whose distance is equal to $v.distance - 1$. Moreover v updates a multiset $v.M(r)$ that contains messages received by neighbors at distance $v.distance + 1$, if v has not received any of these messages, it adds $\perp$ to the multiset. In the sending phase, v broadcasts $< v.distance, v.M(r), v.H(r) >$ to its neighbors.

algorithm 2: OPT_h algorithm for $\mathcal{G}(\text{PD})_h$: algorithm run by a non-leader process

```
1  M(0) = [⊥];
2  H(0) = [⊥];
3  distance = −1;
4  Procedure SENDING_PHASE()
5      send(Message :< distance, M(r), H(r) >);
6      return;
7  Procedure RCV_PHASE(MultiSet MS)
8      H(r + 1) = H(r).append(count_distance_neighbors(MS, distance − 1)) ;
9      M(r + 1) = M(r).append(get_messages_from_distance(MS, distance + 1)) ;
10     return;
```

Leader process behavior in OPT_h. From an high level point of view the algorithm works as follow: the leader first computes the number of processes in V_1, then it executes OPT to count the processes in V_2, this count will be completed by round $(2h + 1) + (3 + log(|V_2|))$ (see Theorem 1). At this point, the leader simulates an execution of OPT counting processes in V_3 exploiting the information obtained by processes in V_2, the leader uses OPT to obtain the exact multiset of messages received by processes in V_2. This counting will be completed by round $(2h+1)+6+log(|V_2|)+log(|V_3|)$. Iterating this procedure till processes at distance h we obtain the final count in $(2h + 1) + 3h + \sum_{i=2}^{h} log_2(|V_i|)$ rounds.

Operationally, the purpose of the leader is to reconstruct the multiset MS_j of messages $< distance, M(r), H(r) >$ sent by processes in V_j at some round r, from MS_j we have $|V_j| = |MS_j|$.

At each round the leader receives MS_1. Starting from MS_1 content, OPT_h iteratively reconstructs the sets MS_j for $j > 1$. This is done in the loop 17-30 of Figure 3. The leader uses the variable i to store

algorithm 3: OPT_h algorithm for $\mathcal{G}(\mathrm{PD})_h$: algorithm run by the leader

```
1  distance_count[];
2  Procedure SENDING_PHASE()
3      send(< leader >);
4      return;
5  Procedure RCV_PHASE(MultiSet MS :< distance, M, H >)
6      i = 1;
7      distance_count[i] = |MS|;
8      i++;
9      while true do
10         if i > h then
11             count = Σ_{∀j|distance_count[j]≠⊥} distance_count[j] ;
12             output(count);
13         end
14         MS =BUILDLASTSET (MS);
15         if MS = ⊥ then
16             break ;
17         end
18         distance_count[i] = |MS|;
19         i++ ;
20     end
21     return;
22 Function BUILDLASTSET(MS)
23     MS_last = ⊥;
24     if OPT(MS, 0) ≠ ⊥ then
25         r_last = r'|OPT(MS, r') ≠ ⊥ ∧ ∄r'' > r'|OPT(MS, r'') ≠ ⊥;
26         MS_last = OPT(MS, r_last);
27     end
28     return MS_last;
```

the maximum distance at which processes of the network have been already counted by OPT_h (initially $i = 1$).

At the beginning of the loop, the leader checks if the count is over (i.e. it checks if $i > h$), in the affirmative the leader outputs the count. Otherwise v_l continues to execute the code in the loop (see Lines 10-12 of Figure 3). The leader now uses the information in the last reconstructed multiset, denoted MS_i, to obtain the multiset MS_{i+1}. Specifically v_l simulates an instance of the algorithm OPT for each element contained in MS_i, i.e., if MS_i contains only two different elements, namely $M(r)$ and $M'(r)$, then the leader uses two trees in order to count the exact number of processes that sent $M(r)$ and $M'(r)$. An example can be found in Figure 8.

The reconstruction is executed by the function BUILDLASTSET.

This function calls $\mathrm{OPT}(MS_i, r')$. The function takes two parameters, and it works on the set of messages MS' where the elements in MS_i received before round r' are removed. Therefore the call $\mathrm{OPT}(MS_i, r')$ returns either the multiset MS_{i+1} sent at round r' or $\perp$. The round r_{last} is the most recent round at which the multiset MS_{i+1} can be reconstructed (see Line 25). In the worst case $r_{last} \geq r - (log_2(|V_{i+1}|) + 3) + \sum_{j=2}^{i}(log_2(|V_j|) + 3)$. Thus the function BUILDLASTSET returns either MS_{i+1} sent at round r_{last} or $\perp$.

At line 15 of Figure 3 the leader obtains MS_{i+1} or $\perp$. If the value obtained is $\perp$, the leader exits from the loop and it waits for the next round; otherwise it computes the count of $|V_{i+1}|$, it updates the distance index and it starts the next iteration of the loop (lines 18-19, Figure 3).

Lemma 12. OPT_h *requires at most* $(2 \cdot h + 1) + 3 \cdot h + \sum_{1 \leq i \leq h} log_2(|V_i|)$ *rounds to output the count.*

Proof. We consider a generic run after $2h + 1$ rounds, so that each process has set $my\_distance \neq -1$. For easy of explanation we consider that the algorithm starts at round 0 and that all processes know their distance. Let us consider the processes in V_h, at round $r = 0$. They start to send their degree history to processes in V_{h-1}, in the worst case at round $r_h = \log_2(|V_h|) + 3$ the union of variables of processes in V_{h-1} allows to compute the multiset MS_h sent at round $r = 0$, see Th. 1 for OPT algorithm, thus at round $r_{h-1} = 6 + \log_2(|V_h|) + \log_2(|V_{h-1}|)$ the multiset MS_{h-1} generated at round r_h can be reconstructed by the union of variables of processes in V_{h-2}, let us recall that MS_{h-1} at round r_{h-1} contains the information to reconstruct MS_h at round r_h. By induction it is easy to show that at round $r_1 = \sum_{i=2}^{h}(\log_2(|V_i|) + 3)$ the multiset MS_1 contains all the information to reconstruct MS_2 at round $r_2 = \sum_{i=3}^{h}(\log_2(|V_i|) + 3)$ and so on. Thus the leader at round $r_1 + 1$ will execute the

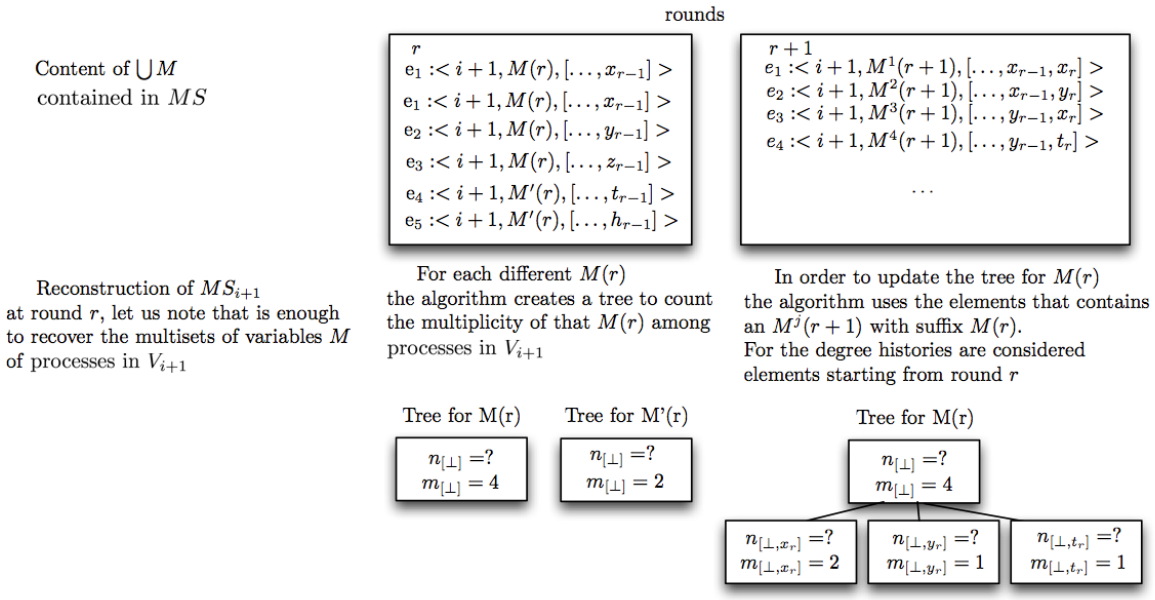

Figure 8: Reconstruction of M variables sent by the set of processes V_{i+1}

reconstruction loop, Lines 23-28 of Figure 3, on the multiset MS_1, and it will obtain the multiset MS_2 sent by processes in V_2 at round $r' \geq r_2$, thus using MS_2 it will reconstruct the multiset MS_3 sent at rounds $r' \geq r_3 = \sum_{i=4}^{h}(\log_2(|V_i|) + 3)$. The leader iterates the computation till it obtains the multiset MS_h, then the leader terminates the reconstruction and the count. □

Complexity discussion From the bound that we will show in the next section we can easily obtain that a lower bound on counting time for $\mathcal{G}(\text{PD})_h$ is $h + max_{\forall V_i}(\log_3(2|V_i| + 1))$. This lower bound holds for a configuration where at each round processes at distance x could be connected to only two processes at distance $x - 1$. The complexity of OPT_h is upper bounded by $5h + 1 + \sum_{i=2}^{h} \log_2(|V_i|)$. Now let us discuss two cases:

- Case 1: If h is constant w.r.t $|V|$, we have $5h+1+\sum_{i=2}^{h} \log_2(|V_i|) \leq 5h+1+h \cdot max_{\forall V_i}(\log_3(2|V_i|+1))$ that is the same order of the lower bound.
- Case 2: If h is not constant w.r.t. $|V|$. We have that $\sum_{i=2}^{h} |V_i| \leq |V|$ and that $\Pi_{i=2}^{h}|V_i| \leq \frac{|V|}{x}^x$, since the product of numbers with a given sum is maximized when all numbers are equals [4]. The maximum of $\frac{|V|}{x}^x$ is obtained when $x = \frac{|V|}{e}$ and thus $\log_2(\Pi_{i=2}^{h}|V_i|) \leq \frac{|V|}{e}\log_2(e)$. Since also the lower bound is worst case $\mathcal{O}(|V|)$ we have our algorithm is asymptotically optimal.

6 Counting in $\mathcal{G}$(1-IC)

In this section we first introduce EXT counting algorithm for $\mathcal{G}$(1-IC) networks then we prove its correctness.

6.1 High level view of $\mathcal{G}$(1-IC) counting algorithm

Let us introduce the underlying structure we use to build EXT. We consider networks in $\mathcal{G}$(1-IC). In such networks, at each round, processes can change their distance from the leader in $[1, V-1]$. When a

[4] This is a well known result obtained by using the relationship between geometric and arithmetic mean.

process changes distance we say that the process *"moved"* from distance x to distance y. Let us introduce the notion of temporal subgraph G' of G:

Definition 13. *(Temporal Subgraph) Given a dynamic graph G, a dynamic graph G' is a temporal subgraph of G ($G' \subseteq G$) if and only if $G' : [G_{i_1}, G_{i_2}, \ldots]$ is an ordered subsequence of $G : [G_0, G_1, G_2, \ldots]$.*

We can show that in each $G \in \mathcal{G}(1\text{-IC})$ there exists a temporal subgraph G' that belongs to $\mathcal{G}(\text{PD})_h$:

Lemma 13. *Let us consider a dynamic graph $G : [G_0, G_1, G_2, \ldots] \in \mathcal{G}(1\text{-}IC)$. There exists $h \in \mathbb{N}^+$ and $\exists G' \subseteq G$ such that G' is composed by an infinite sequences of graphs and $G' \in \mathcal{G}(PD)_h$.*

Proof. The proof is by contradiction. Given a $G : [G_0, G_1, \ldots] \in \mathcal{G}(1\text{-IC})$ let us assume that each distinct graph $G_j \in G$ appears a bounded number of times, let us say $m_{G_j} \in \mathbb{N}^+$. Now let us consider the set X of all possible graphs of $|V|$ processes, clearly we have that this set is finite. Now let us consider the round $x = \sum_{\forall G_j \in X} m_{G_j} + 1$ and the sub-sequence $S : [G_0, G_1 \ldots G_x]$ of G, let us consider the set of distinct graphs X_s of S, we have $|S| \leq \sum_{\forall G_j \in X_s} m_{G_j} \leq \sum_{\forall G_j \in X} m_{G_j}$ but $|S| = x + 1$ which is a contradiction. Thus it exists at least one graph $G_j \in G$ that appears in G an infinite number of times.

Let us consider the subsequence $G' = [G_{r_0}, G_{r_1}, G_{r_2}, \ldots]$ of G such that each $G_{r_i} = G_j$. It is clear that $G' \in \mathcal{G}(\text{PD})_h$ for some $h \leq Diameter(G_j)$ and that G' is infinite. □

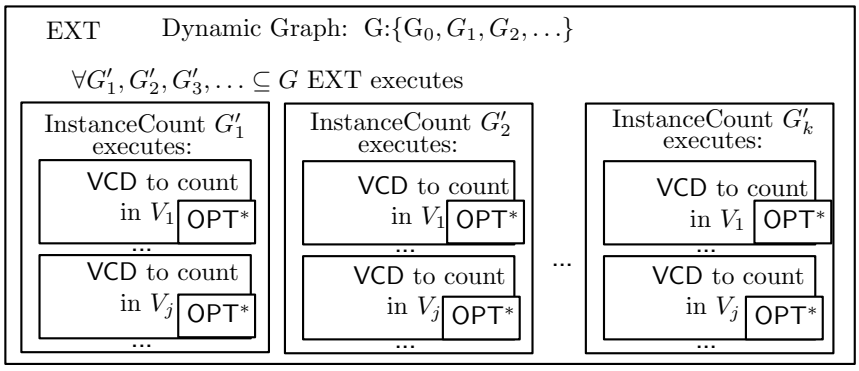

Figure 9: Counting algorithm EXT and the relationship among its subalgorithms.

Now, let us define a counting algorithm InstanceCount. Such algorithm works on $G \in \mathcal{G}(1\text{-IC})$ and it has two properties: (P1) it terminates giving the correct count on instance $G' \in \mathcal{G}(\text{PD})_h$; (P2) it does not give an incorrect count on $G' \notin \mathcal{G}(\text{PD})_h$. Thus, if $G' \notin \mathcal{G}(\text{PD})_h$ it can terminate giving either a correct count of the network or a special invalid value, namely INVCNT. The strategy of EXT is to run a different instance of InstanceCount on each temporal subgraph of G. Due to properties (P1) and (P2), EXT terminates correctly when an instance of InstanceCount outputs a valid count value. For the property (P1) and for Lemma 13, one instance of InstanceCount outputs a valid count value. Consequently, EXT is a correct terminating counting algorithm.

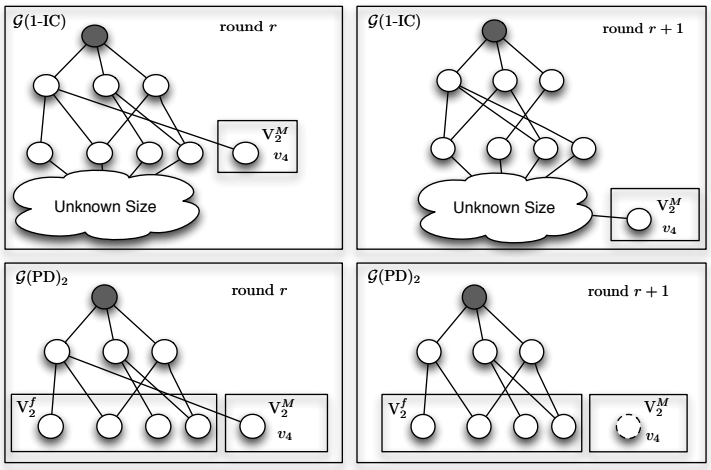

Figure 10: In $\mathcal{G}$(1-IC) a subset V_2^M of processes in V_2 may move changing the distance from the leader invalidating thus the correct count of processes in V_2. Network size is unknown therefore messages from process v_4 need an unknown number of rounds to reach the leader. We are interested in an algorithm that detects this distance change using information from processes in $V_2 \setminus V_2^M$. This is equivalent to solve the problem on a network in $\mathcal{G}(\text{PD})_2$ where the subset V_2^M stops sending messages after a certain round. In the example process v_4 halts at round $r+1$.

InstanceCount counts as if the network is in $\mathcal{G}(\text{PD})_h$. Therefore, the leader first counts processes in V_1, then processes in V_2 and so on. This is done until v_l counts processes of a set V_h such that no set V_{h+1} exists. The tricky part is to detect if the counting algorithm is operating on a network in $\mathcal{G}(\text{PD})_h$. In the affirmative, the count done with such strategy will be correct. The procedure that counts each set V_j is algorithm VCD. VCD allows to detect if the count obtained for V_j is correct, returning the count value, or if it is not possible to count V_j because some process moved during the counting, returning NOCOUNT. See Figure 10.

6.2 InstanceCount

This algorithm assumes that the communication graph belongs to $\mathcal{G}(\text{PD})_h$ then if InstanceCount notices that some process changed the distance from the leader along rounds, it invalids the count.

Non-leader process behavior (Figure 4) Each non leader process v has three variables: $v.distance$ indicating its distance from the leader and two lists $v.M$ and $v.H$. v assigns a value to $v.distance$ as follows: if, at round r, v has $v.distance = -1$ and it is neighbor of a process with $distance = r \neq -1$, v sets its distance to $r+1$ (Line 8). Initially, the leader is the only process with $distance = 0$. As in OPT, v updates its degree history $v.H(r)$ by counting the number of processes in $N(v, r)$ whose distance is equal to $v.distance - 1$. Moreover v updates a multiset $v.M(r)$ that contains messages received by neighbors at distance $v.distance + 1$; if v has not received any of these messages, it adds $\perp$ to the multiset. In the sending phase, v broadcasts $< v.distance, v.M(r), v.H(r) >$ to its neighbors. This is done by using functions $count\_distance\_neighbors$ and $get\_messages\_from\_distance$.

A process that has $distance = r$ adds the messages from processes with $distance = -1$ to M list, let us recall that these processes with $distance = -1$ will set $distance = r + 1$ at round r. Finally at Line 18 a process adds an INVCNT message to M if it detects that at least one its neighbor changed its distance from the leader which implies that the communication graph is not in $\mathcal{G}(\text{PD})_h$ (see condition at Line 17). In the following when we refer to the set V_h, we consider processes setting their distance from the leader to h.

Leader process behavior (Figure 5) The leader v_l first computes the number of processes in V_1, this is simply done by counting the messages received from these processes. After that, v_l executes VCD

algorithm 4: InstanceCount for $\mathcal{G}(1-\text{IC})$: pseudocode for Non-Leader process

```
1  M(−1) = [];
2  H(−1) = [⊥];
3  distance = −1;
4  Procedure SENDING_PHASE()
5      send(Message :< distance, M(r), H(r) >);
6      return;
7  Procedure RCV_PHASE(MultiSet MS)
8      if distance == −1 ∧ ∃m ∈ MS | m.distance ≠ −1 ∧ m.distance == r then
9          distance = m.distance + 1;
10     end
11     if r == distance then
12         forall the m ∈ MS | m.distance == −1 do
13             m.distance = distance + 1;
14         end
15     end
16     if distance ≠ −1 then
17         if r > distance ∧ ∃m ∈ MS | m.distance ∉ {distance − 1, distance, distance + 1} then
18             M(r + 1) = M(r).append(INVCNT);
19         end
20         H(r + 1) = H(r).append(count_distance_neighbors(MS, distance − 1)) ;
21         M(r + 1) = M(r).append(get_messages_from_distance(MS, distance + 1));
22     end
23     return;
```

to count processes in V_2. This is done (i) by receiving the multi-set of messages MS from processes in V_1 (these processes are immediate neighbors of v_l) and (ii) by calling at Line 18 the function BUILDLASTSET. This function takes the multi set MS and starts an instance of VCD to construct the multi-set MS_{last} of messages sent by processes in V_2. We define as VCD(MS, r) the local leader side simulation of a run of VCD that starts at round r using the content of messages in MS. The function returns one out of three possible values: (i) $\perp$ if the messages in MS are not enough to terminate the execution of VCD; (ii) NOCOUNT if VCD detects an halt ; (iii) A multi-set MS_{last} of messages sent by processes belonging to V_2 at round r.

This multi-set leads to the actual count of processes in V_2 (see Line 25). This procedure is iterated: each time the leader obtains the multi-set MS sent by processes in V_{h-1}, v_l calls BUILDLASTSET to reconstruct the most recent multi-set sent by processes in V_h.

The leader returns INVCNT if either (i) there is a INVCNT message in some MS (see Lines 31) or (ii) if one of the instances of VCD terminates returning NOCOUNT. If an halt is detected then a process $v \in V_j$ at some round had a distance from v_l different than j. Additionally, at Line 8 the leader checks if processes in V_1, from which it receives messages, are stable; if this set changes the current instance is considered INVCNT.

The leader outputs the count when it counts a set V_h such that no process in V_h has a neighbor in V_{h+1}, see Line 14.

Correctness Proof

Lemma 14. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(PD)_h$. We have that v_l will never output* INVCNT *in R.*

Proof. The leader returns INVCNT at Line 20. The line is executed either (a) if an halt is detected by VCD, i.e. a process $v \in V_h$ at some round $r > h-1$ is not neighbor of processes in V_{h-1}, or (b) if some set MS contains a INVCNT element. The latter happens if a non leader-process v' executes Line 18-Figure 4, that is v' has a neighbor with distance value that is not in $\{v'.distance - 1, v'.distance, v'.distance + 1\}$. By definition of $\mathcal{G}(\text{PD})_h$ condition (a) cannot happen on G, see Th 8 , the same holds for condition (b). Both conditions would implies that a process v is at distance h in a graph $G_j \in G$ and at distance $h' \neq h$ in $G_i \in G$ with $i > j$. Therefore the claim follows. □

Lemma 15. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$. If $V_h \neq \emptyset$ in R, either (1) the leader obtains the count V_h or (2) the leader outputs* INVCNT.

Proof. The processes in V_h set their distance at round $r = h - 1$, see Line 8 of Figure 4. At the same round an instance of VCD between processes in V_{h-1}, V_h is started, see Line 12 of Figure 4 where messages

algorithm 5: InstanceCount for $\mathcal{G}(1-\text{IC})$: pseudocode for Leader process 0

```
1  distance_count[] = ⊥;
2  distance = 0;
3  Procedure SENDING_PHASE()
4      send(< distance, ⊥, ⊥ >);
5      return;
6  Procedure RCV_PHASE(MultiSet MS :< distance, M, H >)
7      i = 1;
8      if (distance_count[i] ≠ ⊥ ∧ distance_count[i] ≠ |MS|) ∨ (∃m ∈ MS|m.distance > 1) then
9          output(INVCNT);
10     end
11     distance_count[i] = |MS|;
12     i + +;
13     while true do
14         if MS ≠ ∅ ∧ (∀m ∈ MS : m.M = [⊥, ..., ⊥] ∧ size(m.M) > 1) then
15             count = Σ_{∀j|distance_count[j]≠⊥} distance_count[j] ;
16             output(count) ;
17         end
18         MS =BUILDLASTSET (MS) ;
19         if ∃INVCNT ∈ MS then
20             output(INVCNT) ;
21         end
22         if MS = ⊥ then
23             break ;
24         end
25         distance_count[i] = |MS| ;
26         i + +;
27     end
28     return;
29 Function BUILDLASTSET(MS)
30     MS_last = ⊥;
31     if MS.containsSymbol(INVCNT) then
32         return {INVCNT} ;
33     end
34     for r =MinRound(MS); r <MaxRound(MS); r + + do
35         if VCD(MS, r) ==NOCOUNT then
36             return {INVCNT} ;
37         end
38         if VCD(MS, r) ≠ ⊥ then
39             if MS_last ≠ ⊥ ∧ |MS_last| ≠ | VCD(MS, r)| then
40                 return {INVCNT} ;
41             end
42             MS_last = VCD(MS, r);
43         else
44             break ;
45         end
46     end
47     return MS_last ;
```

from Nodes in V_h are taken into account by Nodes in V_{h-1} starting from round $r = h - 1$.

Now we discuss how VCD could be used to reconstruct the exact multi set of memory of processes in V_2. This is equivalent to consider a case where each process v in V_2 have a certain initialisation input value i_v, and where we want to compute the multiset of these values when no process in V_2 halts. Essentially, the value i_v is attached to each algorithm message. The leader starts a different instance of VCD for each of these values. Each VCD instance outputs the multiplicity of a certain value, this is due to Th. 8.

Let us assume that processes in $V_0, \ldots, V_{h-1}$ do not change distance we have for Th. 8 that if no process in V_h moves the set of processes in V_{h-1} will eventually obtain the multiset MS_h . Now the multiset of messages MS_h sent at round $h-1$ by process in V_h will be propagated from processes in V_h to v_l during rounds $[h-1, \ldots, r_{count_h}]$. At each different frontier between distances this is done by using other instances of VCD: between processes in V_{h-1}, V_{h-2}, processes in V_{h-2}, V_{h-3} and so on.

Now let us consider condition (a) that is instances of VCD between processes in $V_0, \ldots, V_h$ that are propagating towards the leader MS_h never detect an invalid count and no processes in $V_0, \ldots, V_h$ has an INVCNT element in its multiset of messages MS. If condition (a) holds we have that the leader will obtains the correct count of processes in V_h by reconstructing the multiset of messages MS_h sent by process in V_h at round $h-1$. This is ensured by Th. 8 of algorithm VCD and by a simple induction on the count for each set V_i.

Otherwise if condition (a) does not hold the leader will output INVCNT. This is done either because of Line 31 or because some instance of VCD terminated with NOCOUNT before the leader is able to reconstruct MS_h.

□

Lemma 16. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$. If v_l outputs a value distinct from* INVCNT *in R, then that value is $|V|$.*

Proof. The Leader terminates at Line 14, that is the leader has reconstructed a multiset MS_h from processes in V_h such that for each $M \in MS_h$: M contains at least two elements and M contains only $\perp$ value. The condition on the size implies that MS_h has been sent at round $r' \geq h$. For Lemma 15 we have that if this happen the leader has correctly counted processes in $V_0, \ldots, V_h$, we have to show that when Line 14 is triggered we have $V \setminus V_0 \setminus \ldots \setminus V_h = \emptyset$.

Let us assume Line 14 is executed and that it exists $v \in V \setminus V_0 \setminus \ldots \setminus V_h$. We must have, for connectivity assumption, that such v at round r' is neighbor of some process in $V_0, \ldots, V_h$. If it is neighbor of a process $v_1 \in V_j$ then v_1 will put INVCNT in $v_1.M(r')$. In order to reconstruct MS_h the leader will also reconstruct the multiset of messages MS_j sent at round r'; two things may happen:

- (1) That the leader receives the INVCNT message thus the Line 31 will be triggered, then the leader outputs INVCNT and the Line 14 will not be executed;
- (2) That v_1, or some other process that is sending the INVCNT message to v_l, is moved away. This triggers the detection in VCD during the reconstruction of MS_h therefore Line 14 is not executed. If v is neighbor of processes in V_h and $r' > h$ we have the same behavior of the previous case.

The only possibility left is v neighbor of processes in V_h and $r' = h$. In this case at least one process v' in V_h will execute Line 12 setting $v'.M(h) = [\perp, \neg\perp]$ therefore Line 14 cannot be executed.

□

Lemma 17. *Let R be a run of* InstanceCount *on a dynamic graph $G \in \mathcal{G}(PD)_h$. We have that v_l terminate and it outputs $|V|$ in R.*

Proof. For Lemma 14 v_l will never outputs INVCNT. This means that, see Lemma 15, the leader eventually obtains the count for each set $V_1, \ldots, V_h$. Since the set V_{h+1} is empty we have that v_l executes the terminating condition when it obtains MS_h, see Line 14. For Lemma 16 the leader output is correct. □

6.3 EXT Counting Algorithm

EXT executes an instance of InstanceCount for each temporal subgraph of G. Let us define as $\mathcal{P}_G$ the set of subgraphs of G such that (i) processes execute for each $G' \in \mathcal{P}_G$ a different instance $I_{G'}$ of InstanceCount

and (ii) such instances do not interfere with each other. Let us remark that the system is synchronous and the current round number r is known by all processes. Therefore each $I_{G'}$ is uniquely identified by a binary string that has value 1 in position j if $G_{r_j} \in G'$ and 0 otherwise. The uniqueness guarantees that instances can run in parallel. At each new round r the number of instances is doubled, half of the new instances will consider the messages exchanged within round r and the remaining ones will not consider these messages. As example at the end round 0 we have two instances I_1, I_0. In instance I_1 the counting is started and processes have received the message exchanged in G_0. In instance I_0 the counting has not been started, the messages exchanged in round 0 are ignored. At round 1 we have four instances $I_{11}, I_{10}, I_{01}, I_{00}$: I_{11} is an instance of counting in which messages exchanged in G_0, G_1 are considered; in I_{10} are considered only messages exchanged in G_1 and ignored messages exchanged in G_0; in I_{01} are considered only messages exchanged in G_0 and ignored messages exchanged in G_1; in I_{00} the counting has not been started. The pseudocode to implement the this procedure is trivial, thus it is omitted.

Theorem 9. *Let R be a run of* EXT *on a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$. Eventually, v_l terminates and it outputs the correct count in R.*

Proof. For Lemma 13 there exists $G' \subseteq G$ with $G' \in \mathcal{G}(\text{PD})_h$. Therefore for Lemma 17 the leader has to terminate correctly on $I_{G'}$. Moreover also for Lemma 16 if another instance $I_{G''}$ with $G'' \subseteq G$ outputs a value, then this value is also correct. From these considerations the claim follows.

□

From the previous Theorem and from the impossibility of non trivial computation without a leader presented in [24, 25] we have:

Theorem 10. *Let us consider an anonymous unknown 1-interval connected networks with broadcast. A distinguished leader process is necessary and sufficient to do non trivial computations.*

Besides counting and existence predicates other non-trivial problems are solvable using simple variation of EXT. Let us assume that each process has an initial input value. If this initial input is attached in the messages of EXT the leader can compute the exact multiset of these values. Thanks to this multiset, the leader may compute aggregation functions as average,min,max.

EXT Complexity EXT has an exponential complexity: Let us consider our $G : [G_0, G_1, \ldots] \in \mathcal{G}(1\text{−IC})$ and $G' \subseteq G$ with $G' : [G'_{i_0}, G_{i_1}, \ldots] \in \mathcal{G}(\text{PD})$. Let compute an upper bound on $max_j(|i_j - i_{j+1}|)$ with $G_{i_j}, G_{i_{j+1}} \in G'$. If we consider that distances of each node from v_l are in $[1, |V|-1]$, then it is easy to see that the number of possible combinations of distances over the set of nodes is upper bounded by $|V|^{|V|}$, therefore by definition of $\mathcal{G}(\text{PD})$ we have $max_j(|i_j - i_{j+1}|) \leq |V|^{|V|}$. Now what we have to bound is the number of instances of G' needed by EXT to terminate, but this can be easily computed by considering when counting terminates with InstanceCount on a graph $\mathcal{G}(\text{PD})$. At each level i we count in at most $\mathcal{O}(|V_{i-1}|^2|V_i|)$ rounds, therefore it is easy to show that the total cost is $\mathcal{O}(|V|^3)$. So EXT terminates in at most $\mathcal{O}(|V|^{|V|+3})$ rounds.

6.4 Equivalence between Counting and Average Consensus

In this section we show that average consensus and a generalisation of terminating counting are equivalent in $\mathcal{G}(1\text{-IC})$ networks. Let us first define the average consensus:

Definition 14. Sum-Preserving Average Consensus: *Let us consider a set of nodes V where each $v_i \in V$ has an initial value $x(0)_i$. An Average Consensus algorithm is Sum-Preserving if the following properties hold:*

- *(1) At each round $r > 0$ each node $v_i \in V$ has a consensus variable $x(r)_i$.*
- *(2) At each round $r > 0$ it holds $\sum_{v_i \in V} x(r)_i = \sum_{v_i \in V} x(0)_i$.*
- *(3) For any $\epsilon > 0$ there exists a round r_ϵ such that $\forall v_i \in V$ it holds $|\frac{\sum_{v_i \in V} x(0)_i}{|V|} - x(r_\epsilon)_i| < \epsilon$.*

We notice that any local averaging algorithm with a doubly stochastic matrix of weights satisfies the specification of Definition 14.

We define a problem that generalise Terminating Counting:

Definition 15. Multi-set Counting: *Let us consider a set of nodes V where each $v_i \in V$ has an initial value $x(0)_i$. An algorithm satisfies Multi-set Counting if at round r the algorithm terminates and v_l outputs for each $x(0)_i$ the number of processes that initially had that value.*

Multi-set counting is useful to compute commutative and associative functions. In this paper we focus on solving Terminating Counting, however all the algorithms that we proposed can be adapted to solve Multi-set Counting. It is trivial to see that the following observation holds

Observation 1. *Any algorithm solving Multi-set Counting can be trivially adapted to solve Terminating Counting.*

One way of the reduction is straighforward, having a terminating Multi-set Counting processes can compute $\frac{\sum_{v_i \in V} x(0)_i}{|V|}$ (Average Consensus). Let us analyze the other direction of the transformation, having a Average Consensus Algorithm, we first show how to build a terminating counting algorithm, then we adapt it to obtain Multi-set Counting.

Lemma 18. *Given a graph $G \in \mathcal{G}(1\text{-}IC)$, a leader node v_l, and a Sum-Preserving Average Consensus Algorithm, we can build a terminating Counting algorithm.*

Proof. Let us first show the reduction and then prove its correctness. Each node $v_i \in V \setminus \{v_l\}$ starts the avg. consensus algorithm with input 0. The leader starts with input 1. The guess of the leader for round r is $N_r = \lceil (x(r)_l)^{-1} \rfloor$ that is the nearest integer to $x(r)_l$. At each round r each process v_i broadcasts $m_r :< r, (x(r)_i) >$. For each messages $m_x :< x, l >$ received only the one with minimal and maximal value l for a certain x is propagated.

The leader starts a *Verification Procedure* for the guess N_r: it waits and collects in the set M_r the m_r messages that it receives until round $r + 2N_r + 1$. The leader guess is true only if $\forall < r, x > \in M_r$, we have $N_r = \lceil (x)^{-1} \rfloor$.

We first show that the *Verification Procedure* always terminates correctly on interval connected networks, the proof is by contradiction.

- Let us suppose that $N_r < |V|$ and that the *Verification Procedure* is true. This implies, thanks to interval connectivity, that there exists at least $x = min(|V|, 2(N_r))$ nodes with value $\frac{1}{N_r - \frac{1}{2}} > x(r) > \frac{1}{N_r + \frac{1}{2}}$. Thus the sum over these nodes is at least $S \geq x \frac{1}{N_r + \frac{1}{2}}$. If $x = 2(N_r)$ we have that $S > 1$ that is an absurd for the condition on consensus variable. Otherwise if $x = |V|$ we have $S \geq \frac{|V|}{N_r + \frac{1}{2}}$ and it must hold $S = 1$ therefore we have $N_r + \frac{1}{2} \geq |V|$. This is a contradiction since N_r and $|V|$ are integers, therefore we have $|V| - N_r \geq 1$
- Let us suppose that $N_r > |V|$ and that *Verification Procedure* is true. This implies, thanks to interval connectivity, that for all nodes $|V|$ we have $\frac{1}{N_r - \frac{1}{2}} > x(r) > \frac{1}{N_r + \frac{1}{2}}$. Thus the sum over these nodes is at most $S \leq |V| \frac{1}{N_r - \frac{1}{2}}$. Since it must hold $S = 1$ we have $|V| \geq N_r - \frac{1}{2}$. This is a contradiction since N_r and $|V|$ are integers, therefore we have $N_r - |V| \geq 1$

It remains to prove that it exists a round r in which the *Verification Procedure* terminates. Let us consider $\epsilon = \frac{1}{2|V|}$ for the convergence condition at round r_ϵ for each $x(r_\epsilon)_i$ we have that $|\frac{1}{|V|} - x(r_\epsilon)_i| < \frac{1}{2|V|}$. It is easy to show that since $x(r) \leq 1$, for the condition on consensus variable, we have that the previous inequality implies $\frac{1}{|V| - \frac{1}{2}} > x(r_\epsilon) > \frac{1}{|V| + \frac{1}{2}}$. Therefore each node has the same value N_{r_ϵ}. This in turns implies that the *Verification Procedure* for the leader terminates correctly for the guess N_{r_ϵ} at round r_ϵ. □

Theorem 11. *Given a graph $G \in \mathcal{G}(1\text{-}IC)$, a leader node v_l, and a Sum-Preserving Average Consensus Algorithm. We can build a terminating Multiset Counting algorithm.*

Proof. From Lemma 18, we can build a terminating counting algorithm obtaining the value $|V|$. To obtain the number of processes with initial value $x(0)$, that we indicate with $|x(0)|$ we use a reduction similar to Lemma 18. Each process with initial value $x(0)$ starts a run of consensus with initial value 1, other

processes with value 0. The leader computes a guess G_r as $x(r)_l \cdot |V|$. The m_r messages and its handling by processes is analogous to Lemma 18. For each guess G_r the leader verifies it by waiting until round $r + |V| + 1$ collecting the messages m_r. A guess is verified if $G_r = \lfloor min_x(M_r) \rfloor |V| = \lceil max_x(M_r) \rceil |V|$. Let us prove by contradiction that if a guess is verified then $G_r = |x(0)|$. Assume that $|x(0)| > G$, we have $S = |x(0)| \geq max_x(M_r)|V| > G_r < |x(0)|$ that is a contradiction. For $|x(0)| < G$ the proof is analogous. □

Convergent counting In the following we will show how build a Convergent Counting Algorithm on $\mathcal{G}$(1-IC) that converges in at most $O(|V|^4 \log(2|V|))$ rounds. It is well known that a local averaging algorithm with fixed weight $q = \frac{1}{max_{v,r}|N(v,r)|}$ leads to a sequence of doubly stochastic matrices, for each possible sequence of interval connected graphs. From [29] it is possible to show that such algorithm verifies the convergence for a fixed ϵ in $\mathcal{O}(|V|^3 \log(\frac{1}{\epsilon}))$ rounds. The only difficulty is to show how to cope with the indecision about q since its value is not known a-priori. But this can be done trivially, each node updates its value q if it sees that its maximum degree its greater than it, and the value is used to restart previous instances of the local averaging algorithm, the maximum value for q it is also disseminated. Since the value of q changes at most $|V|$ times and for our transformation, see proof of Lemma 18, we have $\epsilon = \frac{1}{2|V|}$ our claim on round complexity follows. The aforementioned strategy does not implement a Sum-Preserving Average Consensus Algorithm, therefore our transformation for terminating counting does not apply.

Discussion At the best of our knowledge in $\mathcal{G}$(1-IC) when there is no bound on the maximum degree or when there is no knowledge of node degree prior to the send phase, there is no known strategy that implements a Sum-Preserving Average Consensus Algorithm. For this reason we cannot use our transformation to obtain terminating counting from a Sum-Preserving Average Consensus Algorithm.

7 Conclusion

	$\mathcal{G}(\text{PD})_h$		$\mathcal{G}(1-\text{IC})$		$\mathcal{G}(\infty-\text{IC})$
	$h=2$	$h=\mathcal{O}(\lvert V\rvert)$	Bounded Degree	Unbounded Degree	
Terminating Count.	$\mathcal{O}(\log \lvert V\rvert)$ (Sec. 5)	$\mathcal{O}(\lvert V\rvert)$ (Sec. 5.6)	$\mathcal{O}(\lvert V\rvert^3 \log \lvert V\rvert)$ Red. to Avg. Consensus (Sec. 6.4)	$\mathcal{O}(\lvert V\rvert^{\lvert V\rvert+3})$ (Sec. 6.3)	$\mathcal{O}(\lvert V\rvert^5)$ (Appendix)
Convergent Count.	$\mathcal{O}(\log \lvert V\rvert)$	$\mathcal{O}(\lvert V\rvert)$	$\mathcal{O}(\lvert V\rvert^3 \log \lvert V\rvert)$ [9] [29]	$\mathcal{O}(\lvert V\rvert^4 \log \lvert V\rvert)$ Red. to Avg. Consensus (Sec. 6.4)	
Lower Bound	$\Omega(\log \lvert V\rvert)$ (Sec. 5.2)	$\Omega(\lvert V\rvert)$ Trivial Bound			

Figure 11: Lower Bounds and Upper Bounds for Counting Algorithms on different anonymous dynamic networks

Terminating counting is a basic block for implementing more complex abstractions on any computation model. Therefore, bounds on such problem have an impact on a wide set of more complex tasks that rely on counting. The study we presented represents a wide picture of characterising results of counting over different topologies of anonymous dynamic networks. This is done by presenting original results and by establishing reductions to other well known problems in control theory, namely Average Consensus. Table 11 summarises this picture, it shows upper bounds for terminating counting and convergent counting over $\mathcal{G}$(PD), $\mathcal{G}$(1-IC) and $\mathcal{G}(\infty$-IC). The table also shows a tight bounds for $\mathcal{G}(\text{PD})_2$ and $\mathcal{G}$(PD). As far as $\mathcal{G}$(1-IC) is concerned, we provide an exponential upper bound for terminating counting, namely EXT Algorithm. Interestingly, the reduction to Average Consensus, shows a polynomial convergent counting algorithm for this case. Therefore, it is a relevant open question to quantify if the price of terminating is actually exponential.

The lower bound on $\mathcal{G}(\text{PD})_2$ proves that in networks where the diameter D is fixed and constant with respect to $|V|$ we need $D + \Omega(\log |V|)$ rounds to count the number of nodes. Interestingly this result points out an inherent cost of the combined effect of anonymity and dynamicity. Counting on dynamic networks with IDs requires indeed $\Theta(D)$ rounds [19], the same holds for static anonymous networks [25].

Moreover, this marks a clear difference on anonymous dynamic graphs between the cost of disseminating information, that can be done in $O(D)$ rounds, and the cost of aggregating information, that requires always a number of rounds function of the network size.

Acknoweldge

We are in debt with Ioannis Chatzigiannakis and Silvia Bonomi with whom started the investigation of anonymous dynamic networks.

References

[1] S. Abshoff, M. Benter, M. Malatyali, and F. Meyer auf der Heide. On two-party communication through dynamic networks. In *OPODIS 2013*, pages 11–22. Springer, 2013.

[2] D. Angluin. Local and global properties in networks of processors (extended abstract). In *STOC '80*, pages 82–93. ACM, 1980.

[3] R. Baldoni, S. Bonomi, A. Kermarrec, and M. Raynal. Implementing a register in a dynamic distributed system. In *IEEE International Conference on Distributed Computing Systems*, pages 639–647, 2009.

[4] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. *J. Comput. Syst. Sci.*, 73(3):245–264, May 2007.

[5] Joffroy Beauquier, Janna Burman, Simon Clavière, and Devan Sohier. Space-optimal counting in population protocols. In *(to appear) DISC '15*, 2015.

[6] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In *PODC '99*, pages 181–188. ACM, 1999.

[7] P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3):21–66, 2002.

[8] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating machines that use restricted space. *Theor. Comput. Sci.*, 412(46):6469–6483, October 2011.

[9] Bernard Chazelle. The total s-energy of a multiagent system. *SIAM J. Control and Optimization*, 49(4):1680–1706, 2011.

[10] G. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Conscious and unconscious counting on anonymous dynamic networks. In *ICDCN '14*, pages 257–271. Springer, 2014.

[11] G. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Counting in anonymous dynamic networks under worst case adversary. In *ICDCS '14*, pages 338–347. IEEE, 2014.

[12] Michael Dinitz, Jeremy Fineman, Seth Gilbert, and Calvin Newport. Smoothed analysis of dynamic networks. In Yoram Moses, editor, *Distributed Computing*, volume 9363 of *Lecture Notes in Computer Science*, pages 513–527. Springer Berlin Heidelberg, 2015.

[13] C. Dutta, G. Pandurangan, R. Rajaraman, Z. Sun, and E. Viola. On the complexity of information spreading in dynamic networks. In *SODA '13*, pages 717–736. SIAM, 2013.

[14] Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. Anonymous networks: Randomization = 2-hop coloring. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 96–105, New York, NY, USA, 2014. ACM.

[15] B. Haeupler and F. Kuhn. Lower bounds on information dissemination in dynamic networks. In *DISC '12*, pages 166–180. Springer, 2012.

[16] T. Izumi, K. Kinpara, T. Izumi, and K. Wada. Space-efficient self-stabilizing counting population protocols on mobile sensor networks. *Theoretical Computer Science*, 552(0):99 – 108, 2014.

[17] M. Jelasity, A. Montresor, and Ö. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005.

[18] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS '03*, pages 482–491. IEEE, 2003.

[19] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *STOC '10*, pages 513–522. ACM, 2010.

[20] F. Kuhn and R. Oshman. The complexity of data aggregation in directed networks. In *DISC' 11*, pages 416–431, 2011.

[21] F. Kuhn and R. Oshman. Dynamic networks: Models and algorithms. *SIGACT News*, 42(1):82–96, March 2011.

[22] S. Lang. *Linear algebra*. Addison-Wesley Series in Mathematics. Addison-Wesley Publishing Company, 1966.

[23] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: Random walk methods. In *PODC '06*, pages 123–132. ACM, 2006.

[24] O. Michail, I. Chatzigiannakis, and P. Spirakis. Brief announcement: Naming and counting in anonymous unknown dynamic networks. In *DISC '12*, pages 437–438. Springer, 2012.

[25] O. Michail, I. Chatzigiannakis, and P. Spirakis. Naming and counting in anonymous unknown dynamic networks. In *SSS '13*, pages 281–295. Springer, 2013.

[26] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. In *OPODIS '12*, pages 269–283, 2012.

[27] A. Milani and M. A. Mosteiro. A Faster Counting Protocol for Anonymous Dynamic Networks. *ArXiv e-prints*, September 2015.

[28] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *PODC' 06*, pages 113–122. ACM, 2006.

[29] A. Nedic, A. Olshevsky, A. Ozdaglar, and J.N. Tsitsiklis. On distributed averaging algorithms and quantization effects. *Automatic Control, IEEE Transactions on*, 54(11):2506–2517, Nov 2009.

[30] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *DIALM-POMC' 05*, pages 104–110, 2005.

[31] A. Olshevsky and J.N. Tsitsiklis. A lower bound for distributed averaging algorithms on the line graph. *Automatic Control, IEEE Transactions on*, 56(11):2694–2698, Nov 2011.

[32] Alex Olshevsky and John N. Tsitsiklis. Convergence speed in distributed consensus and averaging. *SIAM J. Control Optim.*, 48(1):33–55, February 2009.

[33] Alex Olshevsky and John N. Tsitsiklis. Convergence speed in distributed consensus and averaging. *SIAM Review*, 53(4):747–772, 2011.

[34] B. Ribeiro and D. Towsley. Estimating and sampling graphs with multidimensional random walks. In *IMC '10*, pages 390–403, New York, NY, USA, 2010. ACM.

[35] John N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation*. PhD thesis, Department of EECS, MIT, November 1984.

[36] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. Distributed average consensus with least-mean-square deviation. *J. Parallel Distrib. Comput.*, 67(1):33–46, January 2007.

[37] M. Yamashita and T. Kameda. Computing on an anonymous network. In *PODC '88*, pages 117–130. ACM, 1988.

[38] M. Yamashita and T. Kameda. Computing on anonymous networks: Part 1-characterizing the solvable cases. *IEEE Trans. on Parallel and Distributed Systems*, 7(1):69–89, 1996.

[39] Y. Yuan, G.-B. Stan, M. Barahona, L. Shi, and J. Goncalves. Decentralised minimal-time consensus. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 4282–4289, Dec 2011.

Appendix

Polynomial Counting in $\mathcal{G}(\infty$-IC)

In this section we consider networks in $\mathcal{G}(\infty-\text{IC})$ which is an interesting subset of $\mathcal{G}(1-\text{IC})$. A dynamic graph G belongs to $\mathcal{G}(\infty\text{-IC})$ if there is a path $p^{\infty}_{v_l,v}$ from v_l to any node v on a stable spanning tree. Under this condition, each message flooded by the leader will reach v in at most $|p^{\infty}_{v_l,v}|$ rounds. We want to show that cunting is polynomial in $\mathcal{G}(\infty-\text{IC})$ networks.

Let us define as $Len(v) = |p^{\infty}_{v_l,v}|$ the distance of v from the leader on the stable path. The algorithm works in epochs, the main idea is that each node v, inside a certain epoch, computes an approximation $v.length$ of $Len(v)$, and communicates only with nodes that have length value equal to $v.length - 1$ and the same epoch value. The counting messages are analogous to messages used in $\mathcal{G}(\text{PD})_h$, thus each non leader node v at length l has variables M, H used to compute the history degree and to collect messages from nodes at length $l+1$. The counting messages are routed to v_l by paths of ordered decreasing length.

When the leader detects that some node at a certain length l at some round was not able to send messages to nodes at length $l-1$, it creates a new epoch. This detection is done by executing the VCD algorithm, between nodes at lengths $l-1, l$. In the new epoch nodes compute a new approximation of the value length and reset the data structures used in the old epoch, starting a new counting.

Let us introduce in details the strategies used by our algorithm. The algorithm for non leader node is reported in Figure 6, the algorithm for the leader node in Figure 7.

Non-Leader Node behavior Each non leader node v starts with the following variables:

- $v.epoch = -1$ its current epoch number;
- $v.repoch = -1$ the starting round of the current epoch;
- $changeepoch = false$ a flag that indicate if node v wants to change epoch;
- $v.length = -1$ its approximation of $Len(v)$ in the current epoch;
- lists M, H used to store the counting messages as in the algorithm for $\mathcal{G}(\text{PD})_h$.

During the send phase v broadcasts the content of its variables, with messages $< length, repoch, epoch, M(r-1), H(r-1), changeepoch >$.

When the leader issues the first epoch, it sends a message that contains the epoch number, stored in the variable $v_l.epoch == 0$, and the starting round of the epoch, variable $v_l.repoch == 0$. This message is flooded in the network. When v receives the message, at round r, it runs three predicates, see line 12:

- (c1) its epoch number has to be equal to $v_l.epoch - 1$;
- (c2) its length has to be less or equal than $r - v_l.repoch$;
- (c3) the length value, len, of the node that sends the message to v has to be equal to $r - v_l.repoch$.

If these three predicates are verified, then v sets its epoch and length according to the content of the message, see Lines 13-19. The predicate (c2) ensures that the length of v, namely variable $v.length$, between epochs is non decreasing, details in Lemma 19; predicates (c1,c3) are done to limit the propagation of the epoch on dynamic paths that are compatible with paths on MST^{∞}.

Specifically the predicate (c1) ensures that node may only accept epochs in a strict monotonically increasing order, thus if node v has $v.epoch = -1$ and at some round it is connected by an edge $e \notin MST^{\infty}$ to a node v' that is at epoch 1 it will discard the message waiting for epoch 0. Intuitively since v has a neighbor u on $p^{\infty}_{v_l,v}$ we have that v has to eventually receive a message with $epoch == 0$ from u, or it receives the message from some other node by means of an edge $e' \notin MST^{\infty}$. The predicate (c3) is inspired by the same principles but inside the same epoch, let us assume that v is at $Len(v) = 10$ and at round $r' = 2$ is connected by an edge $e \notin MST^{\infty}$ to a node v' at $Len(v') = 1$, i.e. a neighbor of v_l in MST^{∞}, then v will discard the message received by v' because v' cannot be its neighbor on $p^{\infty}_{v_l,v}$ otherwise v should have been the neighbor of v' at round $r' = 1$.

After the acceptance of an epoch ep by v, and thus after the acceptance by v of a length value $v.length$, the node v will do a continuos predicate (c4) on the messages that it receives, see Line 23.

The predicate is triggered if at some round does not exist a neighbor of v that has length value equal to $v.length - 1$, if this happens node v sets the variable $v.changeepoch = true$ and increases its length to $v.length + 1$. From now on nodes at epoch ep will not add messages from v to their M, H sets, see Lines 30-32. If the predicate (c4) is triggered then node v was not able to route its degree history over a path of ordered decreasing length. This could happen (1) if node v sets its length by receiving a message from a node v' through an edge $e' \notin MST^{\infty}$ and at some round this edge disappears or (2) if the node u s.t $(v, u) \in MST^{\infty}$ with $u.length = v.length - 1$ triggered its predicate (c4).

In the formal proofs we show that the combination of (c1,c3,c4) ensures two properties on nodes pair $(u, v) \in MST^{\infty}$: (p1) if they have the same epoch value then $|u.length - v.length| \leq 1$ and (p2) if one nodes, let us say u, switches to a new epoch ep at round r before the other, let us say v, then or v switches to ep at the same round or at the next round, Lemma 20. Based on these properties if $(u, v) \in p^{\infty}_{v_l,v}$ we also prove inductively that $v.length \leq Len(v)$, see Lemma 22. This and the non decreasing property of the length value, see the predicate (c2), guarantees that v may change length at most $Len(v)$ times, intuitively if $v.length = Len(v)$ than also the length of its neighbor u has $u.length = Len(u) = Len(v)-1$ and since v is always neighbor of u the predicate (c4) cannot be triggered anymore.

algorithm 6: Counting Algorithm for $\mathcal{G}(\infty$-IC): code for Non-Leader Node

```
1  length = −1;
2  epoch = −1;
3  repoch = −1;
4  changeepoh = false;
5  M(−1) = [];
6  H(−1) = [⊥];
7  Procedure SENDING_PHASE()
8      send(< length, repoch, epoch, M(r − 1), H(r − 1), changeepoch >) ;
9      return;
10 Procedure RCV_PHASE(MultiSet MS : {< len, rep, ep, Ms, Hs, change >})
11     forall the m ∈ MS ordered by increasing ep do
12         if (epoch = ep − 1) ∧ ((r − rep) + 1 ≥ length) ∧ (len == r − rep) then
               /* Predicates (c1)(c2)(c3)                                          */
13             length = (r − rep) + 1;
14             repoch = rep;
15             epoch = ep ;
16             changeepoch = false;
17             M(r − 1) = [] ;
18             H(r − 1) = [⊥];
19             break ;
20         end
21     end
22     if ¬changeepoch ∧ epoch > −1 then
23         if (∄m ∈ MS | (len == length − 1 ∧ ep ≥ epoch)) then
               /* Predicate (c4)                                                   */
24             changeepoch = true;
25             length = length + 1;
26         end
27         if (∃m ∈ MS | (change == true ∧ ep == epoch)) then
28             changeepoch = true ;
29         end
30         MS′ : {m ∈ MS|(len == length + 1 ∧ ep == epoch ∧ change == false)} ;
31         forall the (m ∈ MS|(change == true ∧ r == length + repoch ∧ len ≤ length ∧ ep = epoch − 1) do
32             MS′ ∪ {[⊥]}
33         end
34         M(r) = M(r − 1).append(if(MS′ ≠ ∅) {{MS′}} else { ⊥}) ;
35         H(r) = H(r − 1).append(count(m ∈ MS|(len == length − 1 ∧ ep == epoch)));
36     else
37         M(r) = M(r − 1), H(r) = H(r − 1);
38     end
```

Leader Node behavior The leader starts with the following variables: $epoch = 0$ the epoch counter; $repoch = 0$ the round at which the current epoch is started; $length\_count[]$ an array to store the count of nodes at different lengths. The duties of the leader are to issue new epochs and to count.

During the send phase v_l broadcasts the information about the current epoch, $< 0, repoch, epoch, \perp, \perp >$.

In order to create a new epoch the leader detects if a node triggered predicate (c4) by counting the nodes at each length, for simplicity let us call with V_h nodes with $length = h$ at the current epoch, using the VCD algorithm, see Section 5.5.1. Let us recall that VCD outputs the count of V_h when it is executed between two sets V_h, V_{h-1} such that no node in V_h stops sending messages to nodes in V_{h-1} because it is moved to a different length. This call is done by procedure BUILDLASTSET that is the same of Figure 5, therefore is omitted in Figure 7

Thus the leader at a certain epoch uses VCD to count iteratively and continuously nodes in $V_1, V_2, \ldots$. If the length value of nodes is enough stable, i.e. no nodes trigger (c4), then the leader eventually will count a set V_{last} such that no nodes in V_{last} had a neighbor at length $last + 1$, when this is done it terminates. The key point to prove the correctness of this termination procedure are: Lemma 22, in which we show that at each round a new node has to switch to the new epoch, if this does not happen then all nodes are in the current epoch. This ensures that if $V_{last+1} = \emptyset$ then all $V_{j>last+1}$ are empty. Lemma 25 in which we show that if the leader updates its count for a certain V_h, see Line 15, then this count is correct, and Lemma 26 in which we show that if Line 11 is executed then there is no node in V_{last+1}.

If the network is not stable, a node v verifies (c4); v stops routing information to the leader preventing the correct counting. The leader will detect this by using $\mathcal{VCD}$, and this detection takes at most $\mathcal{O}(|V|)$ rounds. See Lemma 24. After the detection it will issue a new epoch and reset the partial count, see Lines 23-25.

To show the termination, we leverage the bound on the length value that a node may assume, see Lemma 21. Then we show that this leads to at most $|V|^2$ epochs, see Lemma 29. As final part we show that after at most $\mathcal{O}(|V|^3)$ rounds the leader counts or it issues a new epoch, see Lemmas 23-24. This leads to a total worst case cost of $\mathcal{O}(|V|^5)$ rounds.

algorithm 7: Counting Algorithm for $\mathcal{G}(\infty$-IC): code for Leader Node

```
1  epoch = 0;
2  repoch = 0;
3  length_count[];
4  Procedure SENDING_PHASE()
5      send(< 0, repoch, epoch, ⊥, ⊥ >) ;
6      return;
7  Procedure RCV_PHASE(MultiSet MS : {< len, rep, ep, Ms, Hs, change >})
8      MS' : {m ∈ MS | m.ep = epoch};
9      MS = MS';
10     if MS ≠ ∅ ∧ (∀m ∈ MS : m.M = [⊥, ..., ⊥] ∧ size(m.M) > 1) then
11         count = Σ_{∀j | length_count[j] ≠ ⊥} length_count[j] ;
12         output(count) ;
13     end
14     index_length = 1;
15     length_count[index_length] = |MS|;
16     index_length + +;
17     while true do
18         MS = BUILDLASTSET (MS);
19         if MS == ⊥ then
20             break;
21         end
22         if ∃INVCNT ∈ MS ∨ ∃m ∈ MS | m.change == true then
23             repoch = r + 1;
24             epoch + +;
25             reset(length_count);
26             break;
27         end
28         length_count[index_length] = |MS| ;
29         index_length + +;
30     end
```

Correctness proof Let us begin by stating lemmas on the *length* variable of non-leader nodes

Lemma 19. *For each node v the value $v.length$ is not decreasing.*

Proof. $v.length$ is set at line 13 where the predicate in the if Line 23 avoid decreasing, and it is increased at line 25. □

Lemma 20. *Let us consider two nodes $v_0, v_1 \in V$ such that $(v_0, v_1) \in MST^\infty$ we have that:*

- *(p1): If $(v_0.len \neq -1 \wedge v_1.len \neq -1) \wedge (v_0.epoch == v_1.epoch)$ then $|v_0.length - v_1.length| \leq 1$*
- *(p2): Considered an epoch ep if v_0 is the first among the two to set $v.epoch = ep$ at round r then also v_1 sets $v_1.epoch = ep$ at round $r' \in \{r, r+1\}$*

Proof. The proof is by induction on the epoch:

- **base case epoch=0:** Let us consider the case when node v_b is the first among the two that round r executes for the first time line 13, thus setting $v_b.epoch = 0$ and $v_b.length = r + 1$, then we have that at round $r + 1$ node $v_{b\oplus 1}$ receives the message from v_b. Now two things may happen: (1) that $v_{b\oplus 1}$ executes line 12 by setting its length to $v_{b\oplus 1}.length = r + 2$, (2) that $v_{b\oplus 1}$ does not execute line 12; the possibilities are: (a) that $v_{b\oplus 1}.length > v_b.length$ but this is not possible since $v_{b\oplus 1}.length \leq r+1$, (b) that $v_{b\oplus 1}.epoch == v_b.epoch$ but this implies that $v_{b\oplus 1}.length = v_b.length$ thus the lemma holds, (c) that $v_{b\oplus 1}.epoch > v_b.epoch$ but this is not possible since $v_{b\oplus 1}.epoch \leq 0$. Let us consider the case when in epoch 0 one of the two executes line 25: it is easy to see that it can be executed first by the one among the two that have the small length value, thus the lemma still holds since both have the same length value, and at next round it could be executed by the other one, thus the lemma still holds since the difference is at most 1. Otherwise it can be executed at the same round r by both, but this implies that both have the same *length* value at round r and $r + 1$, thus in any case the lemma holds.
- **inductive case epoch=ep:** Let us suppose that epoch *ep* has been issued at round *rep*. If v_0 sets its epoch to $v_0.epoch = ep$ and its length to $v_0.length = r - rep$ at round r then at round $r - 1$ we have $v_0.epoch = ep - 1$. For ind. hyp. we have that $v_1.epoch = ep - 1$ at round $r - 1$ or r. Thus at most at round $r+1$, when it receives the messages from v_0, v_1 will set its epoch to *ep* and its length to a value $l \leq r + 1 - rep$. It is easy to see that by a reasoning analogous to the base case that execution of line 25 during epoch *ep* cannot force the nodes to have $|v_0.length - v_1.length| > 1$.

□

Lemma 21. *Let us consider a node $u \in V \setminus \{v_l\}$ with $Len(u) = k$ and a new epoch ep issued by the leader at round rep. We have that at round $r' \leq rep + k - 1$, $u.epoch = ep$ and $u.length \in [0, k]$.*

Proof. The proof is by induction on *Len*.

- **Len(u)=1**: In this case we have that $\exists (u, v) \in MST^\infty$ and $v = v_l$ it is easy to see that if v_l issues a new epoch *ep* we must have $ep > u.epoch$ thus, at round $rep + 1$ the only reason for u to not execute line 13 is that $u.length > v.length + 1$ but $u.length$ is set at round 0 to 1 let us recall that u will never execute line 25 since it is always neighbor of the leader. Thus $\forall u | Len(u) = 1$ we have that $u.length \in [0, Len(u)]$.
- **Len(u)=k**: We have that exists $(u, v) \in MST^\infty$ and that $LEN(v) = Len(u) - 1 = k - 1$. Thus let us consider the possible cases:
  (1) v **changes epoch before** u: At round r the node v executes line 15 by setting $v.epecoh = ep$ for Lemma 20 we have that u will execute line 15 at round $r + 1$, thus we have for inductive hypothesis that $v.length = u.length + 1 \leq Len(u) + 1 = k$.
  (2) u **changes epoch before** v: Let us assume that u changes epoch at round r, By using the same consideration of the previous case we have that v will change epoch at round $r + 1$ setting its length to $v.length + 1$. But we have that exists $(v, w) \in MST$ with $Len(w) = Len(v) - 1$ by using inductive hypothesis we have that w change epoch at round $r' - rep < Len(w)$, thus for Lemma 20 we have that $r + 1 \leq r' + 1$ this implies $u.length \leq Len(w) \leq k - 2$.

  (3) u, v **change epoch at the same round r**: by inductive hypothesis since $u.length = r - ep \leq Len(u)$ we have that $v.length \leq k - 1$.

□

Lemma 22. *Let us consider the beginning of a new epoch ep, at round rep, and the interval of rounds $r \in [rep, T]$, where T is the first round at which some nodes v_f sets changeepoch = true. Let us consider two partition of nodes I_r^a, I_r^b s.t. $\forall v \in I_r^a$ we have v.epoch = ep and $|I_r^a| > 0$ and $\forall v \in I_r^b$ we have v.epoch < ep. We have that (1) $|I_{r+1}^a| > |I_r^a|$ or (2) $|I_r^b| = 0$.*

Proof. Let us suppose, by contradiction, that at round r we have $|I_r^b| > 0$ and that at round $r+1$ we have $|I_{r+1}^a| = |I_r^a|$. For hypothesis we have that at each $r \in [rep, T]$ $\nexists w \in V$ with $w.changeepoch = true$ and $w.epoch = ep$. Let us consider a node $v \in I_r^b$ and let us recall that $\exists v_1 \in N(v)$ with $(v, v_1) \in MST^\infty$ and $Len(v) = Len(v_1) + 1$, let us suppose that $v_1 \in I_r^a$ this and Lemma 20 implies that at round $r' \leq rep + v_1.length + 1$ the node v has to be in $I_{r'}^a$, thus we have that $v \notin I_{r+1}^b$ this implies $|I_{r+1}^a| = |I_r^a|$. So we must have $v_1 \in I_r^b$, let us consider the node $v_2 \in N(v_1)$ s.t. $(v_1, v_2) \in MST^\infty$ and $Len(v_1) = Len(v_2) + 1$ we must have $v_2 \in I_r^b$ otherwise we can reach the same contradiction, but this implies that iterating the chain we reach a $v_k \in I_r^b$ with $Len(v_k) = 0$ that is a contradiction.

□

From the previous lemma the next observation immediately follows

Corollary 3. *Let us consider the beginning of a new epoch ep at round rep, at each round $r \in [ep, T]$ we must have that if $|I_r^b| \neq 0$ then exist $v \in I_r^b$ s.t. $v \in I_{r+1}^a$.*

Lemma 23. *Let us consider the beginning of a new epoch ep at round rep and such that $\forall r > rep$ no nodes v_f sets changeepoch = true. We have that after at most $rep + \mathcal{O}(|V|^3)$ the leader terminates.*

Proof. We have for Lemma 22 that after at most $|V|$ rounds all nodes have epoch *ep*. If no node sets *changeepoch* = *true* then we have that each node in V_k will have a neighbors in $V_k - 1$, see Line 23, thus we are equivalent to the $\mathcal{G}(\text{PD})_h$ where messages are routed from nodes at length l to nodes at length $l-1$. The leader v_l uses VCD to counts node at each different length. If we see line 34 - Figure 6 at round h nodes in V_h start to send their degree history, initially equal to $[\perp]$, to nodes in V_{h-1} that will puts that histories in their variable M. Thus to count up to length x the leader employs at most $\mathcal{O}(\sum_{i=1}^{x-1} |V_i|^2 |V_{i+1}|)) \leq \mathcal{O}(|V|^3)$. Let us define as l_{last} the maximum length assigned, we have that at most at round $rep + \mathcal{O}(|V|^3)$ the leader has to terminate: if l_{last} is the maximum length then we have that no nodes can send a message m such that a node $v \in V_{last}$ will add m to $v.M$ (see Line 34- Figure 6). The terminating condition will be triggered when the leader reconstruct MS sent by nodes in V_{last} at round $rep + l_{last} + 1$. □

Lemma 24. *Let us consider an epoch ep, issued at round rep and the interval of rounds $[rep+l+1, +\infty)$. Let us assume that $\forall v | v.length \in [0, l]$ we have that $\nexists v$ that executes line 25. And let us assume that $\exists v_f | v_f.length = l + 1$ that at some round $r_f > rep + l$ executes line 25. We have that if v_l does not terminate then it will issue a new epoch at a round $r_n = r_f + \mathcal{O}(|V|)$.*

Proof. Let us assume w.l.o.g that $r_f = rep + l + 1$, we have that at round $rep + l$ node v_f sends a message to a subset S of nodes with $length = l$, since we have that v_l sets its length to $l+1$ at round $rep + l$ we have that the nodes in S will execute line 32 adding these messages to M. At round $r_f + 1$ v_f executes line 25 this means that v_f is not neighbor of any node with length l (see predicates at Lines 22-23), let us remark that from round r_f and until $v_f.epoch = ep$ we have that messages coming from v_f will be not added to any set M of other nodes, see predicate at line 30. Moreover, for line 27 the positive value for variable $v_f.changeepoch$ will be propagated from v_f to other nodes with same epoch, this implies, thanks to interval connectivity, that after at most $\mathcal{O}(|V|)$ rounds a message with epoch *ep* and *changeepoch* = *true* will reach the leader triggering line 23.

At most at round $rep + l + 1 + \mathcal{O}(\sum_{i=1}^{l-1} |V_i|^2 |V_{i+1}|)$ the leader reconstructs the set MS of messages sent by nodes with length l at round $l+1$. Starting a simulation of the VCD algorithm between nodes with length l and $l+1$ from round $r = l$. But there is a node v_f that was present at round $r = rep + l$ and that was not any more present at round $r > rep + l$ this implies that these simulation of VCD will terminates with a failure at most at round at round $r = rep + l + \mathcal{O}(\sum_{i=1}^{l-1} |V_i|^2 |V_{i+1}| + |V_l|^2 |V_{l+1}|)$, and this trigger the execution of line 23. If $ref > rep + l + 1$ we have an analogous proof. Therefore in at most $min(\mathcal{O}(|V|), r)$ the leader issues a new epoch. □

Let us recall that when a node sets a new epoch its erase the content of M, H (Lines 17,18-Figure 6), the same is done by the leader (Line 25- Figure 7), moreover the messages using for counting in old

epochs do not influence the content of the sets M, H and the messages processed by the leader (Lines 30,35-Figure 6 and Line 8-Figure 7). This is equivalent to restart the counting process at each new epoch.

Lemma 25. *Let us consider an epoch ep, issued at round rep and the interval of rounds* $[rep+l+1, +\infty)$. *If we have* $v_l.epoch = ep$ *and the leader sets at line 28 that* $length\_count[l] = C$ *we have that the number of nodes that at epoch ep had length* $= l$ *is equal to* C.

Proof. If the leader executes line 28 then a simulation of the VCD between nodes in V_{l-1}, V_l terminated without detecting an halt. Let us recall that nodes in V_l sets their length to l at round $rep + l - 1$ and that at the same round in line 32 nodes in V_{l-1} detect the messages from their neighbors in V_l, by adding the default value $[\perp]$ for each neighbor in V_l. All the instances executed in the for loop (Lines 34,39 of Figure 5) s_j of VCD such that each s_j started at round $rep+l-1+j$ with $j \geq 0$ terminates in a correct way. The various instances are obtained by shifting the starting round of 1 (for at line 34), then all have to terminate detecting the same count; Also for VCD properties we have that this count is equal to the number of nodes V_l that where neighbors of nodes in V_{l-1} at round $rep + l - 1$. Let us remark that if a node, $v \in V_{i<l}$, on the path of the messages from V_l to v_l changes length when it is tunneling the messages from V_l in its set M we have for the aforementioned reasons that the leader will detects this when it compute the set of messages MS sent by nodes in V_i, this set is necessary to compute the set MS sent by nodes in V_l. Thus, the leader will detect a failure before computing an erroneous counting on V_l due to the delay of messages. □

Lemma 26. *Let us consider an epoch ep, issued at round rep and the interval of rounds* $[rep+l+1, +\infty)$. *If we have* $v_l.epoch = ep$ *and such that the leader terminates executing line 12 when* $index_{length} = l_{last}+1$ *we have that* $\nexists v$ *with* $v.epoch = ep$ *and* $v.length \geq l_{last} + 1$.

Proof. If the leader terminates then, see predicate at line 10, it has reconstructed the set MS of messages sent by nodes $V_{l_{last}}$ at round $rep+l_{last}+1$, this is enforced by the predicate that each M has to contains at least two elements and it contains only $\perp$ value. For Corollary 3 we have that if there is some node with $v.length \geq l_{last} + 1$ then there must be at least on neighbor of a node in $V_{l_{last}}$ that sets its length to $v.length = l_{last} + 1$. This implies that exists a node in $v' \in V_{l_{last}}$ that at round l_{last} add to its list $v'.M(l_{last})$ an element different from $\perp$ as second element of the list, see line 32 where a non empty set of lists is added to $M(l_{last})$. Let us recall that at round $l_{last} - 1$ nodes in $V_{l_{last}}$ reset the variable $M(l_{last} - 1)$ to $[\perp]$ line 17 and line 34. For lemma 25 when the leader reconstruct the set MS sent by nodes in $V_{l_{last}}$ it obtains the messages sent by all process thus it has to receive the messages from v', but this means that the terminating condition will not be triggered since $v'.M(l_{last}) = [\perp, \neg\perp, \ldots]$. □

Lemma 27. *Let us consider an epoch ep, issued at round rep and the interval of rounds* $[rep+l+1, +\infty)$. *If the leader* v_l, *with* $v_l.epoch = ep$ *terminates then it outputs the correct count.*

Proof. Let us suppose that $l_{last} + 1$ is the value of $index_{length}$ when the leader terminates. For Lemma 25 it has correctly counted all nodes in $V_0, \ldots, V_{last}$ moreover for Lemma 26 we have that there is no node with $v.length \geq l_{last} + 1$. This implies that the leader has counted all nodes in the network. □

Lemma 28. *If the leader issues an epoch* $ep > 0$ *then some node* v_f *at a certain round* r *with* $v_f.ep = ep - 1$ *executed line 25.*

Proof. A new epoch is issued by the leader at line 24 - Figure 7. This means that the VCD algorithm terminated detecting an halting. An halt is detected by VCD only if a node v_f with $v.length = h$ has sent messages to nodes with $length = h - 1$ at rounds $r < r'$ and then stopped sending messages at rounds greater than r', that is v_f executed line 25- Figure 6. □

Lemma 29. *We have that the maximum number of epochs in any run is at most* $\mathcal{O}(|V|^2)$

Proof. From Lemma 28 we have that an epoch is issued only if there is node that changes its *length*. For Lemmas 21-19 we have that the maximum number of length changes is bounded by V^2, that is each node, at most, changes length $|V|$ times from 0 to $|V|$. □

Theorem 12. *In* ∞*-interval connected network it is possible to count in* $\mathcal{O}(|V|^5)$ *rounds.*

Proof. If the leader terminates the count is correct, see Lemma 27. For Lemma 29, we have that the maximum number of epochs is $\mathcal{O}(|V|^2)$. For Lemmas 23,24 we have that in the worst case the count terminates or that a new epoch is issued after at most $\mathcal{O}(|V|^3)$. Thus in at most $\mathcal{O}(|V|^5)$ rounds we reach the last epoch where the counting terminates. □