Sapienza University of Rome

Ph.D. program in Computer Engineering

XXVII Cycle/

On Deterministic Counting in Anonymous Dynamic Networks

Giuseppe Antonio Di Luna

Sapienza University of Rome

Ph.D. program in Computer Engineering

XXVII Cycle/

Giuseppe Antonio Di Luna

On Deterministic Counting in Anonymous Dynamic Networks

Thesis Committee

Prof. Roberto Baldoni (Advisor)

Reviewers

Prof. Hugues Fauconnier
Prof. Roger Wattenhofer

Author's address:
Giuseppe Antonio Di Luna
Dipartimento di Ingegneria Informatica,
Automatica e Gestionale (DIAG)
Sapienza Università di Roma
Via Ariosto 25, 00185 Roma, Italy
e-mail: diluna@dis.uniroma1.it
www: http://www.dis.uniroma1.it/∼dottoratoii/
students/giuseppe-antonio-di-luna

Contents

Chapter 1

Introduction

Recently, highly dynamic distributed systems have attracted a lot of interest from the relevant research community [21,47], due to the fact that static distributed systems do not capture anymore the new kind of software applications that are emerging on the ICT field. Mobile devices, sensors networks and cloud computing definitely changed the way of looking at distributed systems where, for example, the underlying communication graph connecting distributed system's processes is constantly changing. Mobility of devices, intermittent connectivity due to lossy links create very challenging system models where what was trivially solvable in a static distributed systems, is far from being trivial in this new landscape.

A critical element in such future distributed systems is the *anonymity* of the devices; the uniqueness of the node IDs is not guaranteed due to operational limit (e.g., in highly dynamic networks maintaining unique IDs may be infeasible due to mobility and failure among nodes [65]) or to maintaining user's privacy (e.g., where users may not wish to disclose information about their behavior [28, 42]). We refer to this setting as *anonymous dynamic networks*. The impacts of anonymity on distributed computation has been largely investigated [18, 19, 24, 32, 33, 68, 71]. The difficulties are created by the inherent symmetries created in peculiar families of networks configuration. These symmetries do not allow nodes to locally distinguish the particular network in which they are running, this leads to the impossibility of solving many problems. A classical result in such setting is the impossibility of doing any non trivial computation on an anonymous synchronous edge-unlabeled ring of unknown size [3, 18].

Early studies deal with causes of dynamicity such as failures and changes in the topology that eventually stabilize, [2, 13, 30, 35]. However this low rate of topological changes is unsuitable for reasoning about truly dynamic networks. Recently there have been several studies about dynamic networks that constantly change [44, 63, 65]. In these studies the dynamicity is governed by a fictional omniscient entity *the adversary* that deploys the worst possible network configuration in order to challenge computations. Studying which kind of problems can be deterministically solved is a non trivial task and it is the focus of a wide number of recent papers: [43, 45, 60]. Ex-

ample of key problems in such environments are terminating broadcast and counting.

The importance of broadcast with explicit termination is clear, this problem is a basic building block for more complex primitives. Its solvability has been investigated in adversarial [44] and non adversarial dynamic networks [20]. Other recent works, [1, 31, 36], investigated lower bounds on the time needed to solve the related problem of k-token dissemination [44], where k tokens initially distributed among a subset of processes have to be disseminated and received by all processes.

The problem of counting, or estimating, the number of processes in a system is among the fundamental problems of distributed computing [12, 15, 41, 44, 58, 64]. Estimating the size of the network is indispensable for topological change detection or automatic network reconfiguration. In a network when IDs are not unique counting is the basic building block to answering predicates like *"Are there k nodes with initial input x?"*, that is in turn necessary to compute generic aggregated functions. Moreover if we consider a synchronous model where the perpetually changing graph is connected at each computational round (i.e., the 1-interval connected model proposed by [44]) we have that a terminating counting algorithm implies terminating broadcast [44]: Intuitively, as result of the connectivity assumption, we have that if a process floods in the network a certain message m then at each round the set of nodes that has not received m will decrease by at least one unit. In 1-interval connected network, counting has a further notable byproduct. Let us remark that such model has been widely accepted in the context of dynamic computation [1, 11, 31, 36–38, 44].

When we consider mobile computing a common way to abstract processes communication is to consider a local broadcast primitive, a process has to send the same message to all neighbors. Considering this communication capability we have that in anonymous interval connected graph the presence of the leader node is necessary in order to compute non trivial tasks, including counting and terminating broadcast. This is showed by [59] where is observed that otherwise the adversary could perpetually generate a ring graph. The presence of a leader node is an acceptable assumption in many realistic settings (e.g., the presence of a base station in a mobile network) and it is easier to ensure than an unique ID for each process. When processes communicate using broadcast assuming a leader is strictly weaker than assuming unique IDs [59].

Even with leader and connectivity the task of counting in adversarial networks is not trivial. The difficulties introduced by the anonymity are exacerbated by the presence of a continuously changing graph. In this model there is no assumption of fairness, as in contrast with population protocols [4,25] or dynamic graphs where links are created by using peers sampling mechanics [41, 64]. Therefore we cannot lightly assume: a certain break in symmetry (e.g., the highly dynamic assumption in [61]), a known recurrent set of edges, a certain expansion factor in the graph or a bound on the time needed to flood the entire network.

In the light of these difficulties it has been conjectured [61, 62] that even with a leader non trivial computations are impossible: *It is impossible to compute (even with*

a leader) the predicate $Na \geq 1$, that is "exists an a in the input", in general anonymous unknown dynamic networks with broadcast. The previous conjecture implies a conjecture on the impossibility of counting. Therefore there is an important open question about what can be computed in this environment.

In this manuscript we study the problem of exact counting. Specifically the main focus is on deterministic terminating algorithms. A terminating counting algorithm, in contrast with an algorithm that only converges to the correct count, is necessary in many realistic situations. As example we may think about a mission critical environment where information contained in each node matters, in this case an incorrect termination, e.g. counting one node less, could have nefarious consequences. We are not interested in algorithms where actions are based on the outcome of a coin toss, e.g. [44, 58, 64], since randomness could be used to break symmetries increasing the computational capabilities of nodes.

Ultimately, we are interested in the solvability of exact counting, and as consequence on what can be computed, in an anonymous network where a single distinguished leader is present, nodes communicate by broadcast and the underling connected communication graph changes at each round governed by an omniscient adversarial entity that picks links in such a way to challenge the execution of our task.

1.1 System Model

We consider a synchronous distributed system composed by a finite static set of processes V (also called *nodes*). Processes communicates through a communication network which is a *dynamic* graph.

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.*

We assume at each round r the network is stable and represented by a graph $G_r = (V, E(r))$ where V is the set of nodes and $E(r)$ is the set of bidirectional links at round r connecting processes in V.

The neighborhood of a node 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 with $P^r_{v',v}$, the set of all paths between v, v' on graph G_r. Graph G_r is connected iff $\forall v', v \in G$ we have $P^r_{v',v} \neq \emptyset$. 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 use the term family to indicate a specific set of dynamic graph that share certain properties. Let us formally introduce the family of interval connected graphs.

Definition 2. **[44]** 1-*Interval Connected: Let us consider a dynamic graph G. A graph G belongs to* 1-*Interval Connected Family , denoted $\mathcal{G}$(1-IC) , iff $\forall G_r \in G$ we have that G_r is connected.*

The main focus of this manuscript is on graphs in $\mathcal{G}$(1-IC). Therefore when is not explicitly specified we assume that the graphs are 1-interval connected.

We assume that each node $v \in V$ starts with a variable i_v assigned at the beginning of the computation, i.e. this variable is called initial state. Initial states are not unique, we could have two nodes v_1, v_2 such that $i_{v_1} = i_{v_2}$. We say that in a dynamic network there is a *leader* node v_l if $\forall v_1 \in V \setminus \{v_l\}$ we have $i_{v_1} \neq i_{v_l}$. A network is anonymous with a leader if $\forall v_1, v_2 \in V \setminus \{v_l\}$ we have $i_{v_1} = i_{v_2}$.

The Adversarial Entity We introduce a fictional entity called *adversary*. The adversary generates the sequence that will compose the dynamic graph. As in [44], we use the term *worst case* adversary to denote an omniscient adversary that is able to read nodes internal memory. Moreover we assume that the worst case adversary is able to control the source of randomness available to nodes, i.e. the outcome of local coin tosses could be arbitrarily biased. Therefore in this manuscript we consider only deterministic algorithms.

We use the term *random*-adversary to denote an adversary that generates the dynamic graph according to some probabilistic strategy. When is not explicitly specified we assume that the adversary is worst case.

Communication and Computation At round r a process v communicates with the set of processes $N(v, r)$ by sending and receiving messages. Every round is divided in two phases: (i) *send* where processes send all the messages for the current round, (ii) *receive* where processes receive all the messages sent at the beginning of the current round and where nodes process received messages and prepare those that will be sent in the next round. Each node has access to the current round number via a local variable that we usually denote by r. Messages are sent by an *anonymous broadcast primitive*: if node v sends a message m during the send phase of round r, message m will be received by all nodes in $N(v, r)$ during the receive phase of round r.

Now we introduce the concept of dynamic diameter, the definitions are adapted from [43]:

Definition 3. *Node v starts a flooding at round r by broadcasting a message* Flood *to its neighbors. If a node receives* Flood *it will broadcasts* Flood *to its neighbors for all successive rounds. We say that the flood has completed when all nodes have received message* Flood.

Definition 4. *A dynamic network has dynamic diameter D, if for each node $v \in V$ a flooding that starts from v at round r has completed at most at round $r + D$.*

Intuitively the dynamic diameter is the maximum time a node needs to transfer information to all nodes in the network. Networks in $\mathcal{G}(1\text{-IC})$ have a dynamic diameters that is at most $|V|$, see [44].

The Counting Problem Let us introduce our formalization of the counting problem. As shown in [59], the leader is necessary to solve the counting. Therefore our definition is leader based.

Definition 5. *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.*

$\mathcal{G}(\mathbf{PD})_h$ networks Let us introduce a characterization of dynamic networks where we restrict the adversary to ensure certain properties on paths among the leader v_l and other nodes. That properties lead to specific structures of the dynamic graph that is kept among rounds. We first characterize dynamic graphs according to the distances among a node v and the leader v_l.

Definition 6. *Persistent Distance over G: Let us 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 family of dynamic graphs based on the distance between the leader and the nodes of a graph.

Definition 7. *Persistent Distance family: A graph G belongs to Persistent Distance family, denoted $\mathcal{G}(PD)$, iff $\forall v \in G$, $\exists d \in \mathbb{N}^+ :: D(v, v_l) = d$*

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.

We also consider networks where for each node v there exists a path p_{v,v_l} that persist among rounds. In this case we have a resulting dynamic graph that is ∞-interval connected.

Definition 8. **[44]** *∞-Interval Connected: Let us consider a dynamic graph G. A graph G belongs to ∞-Interval Connected Family , denoted $\mathcal{G}(\infty\text{-}IC)$, if considered $G^\infty : \cap_{r=0}^{\infty} G_r$ we have that G^∞ is connected.*

The aforementioned networks will the be focus of Chapter 3

Local Degree Detector Oracles. We also focus on counting algorithms for networks that are unstructured, but where nodes have access to some local additional knowledge. Specifically v has information on its local degree at the beginning of round r, before exchanging messages. Let us recall that without the oracle in our model a node has no information about its degree before the receive phase. This is modeled by using the concept of local oracle, it will be formally introduced and investigated in Chapter 4.

1.2 Related Work

The problem of counting the network size has been widely investigated in different contexts. In the following, we will discuss the main results starting from the static non-anonymous networks. Then we orderly discuss related results in static anonymous network, in non-anonymous dynamic networks and finally in anonymous dynamic networks. When needed each section is divided in adversarial and non adversarial dynamicity.

Counting in Static Non-Anonymous Networks.

In non-anonymous undirected network the seminal work of Awerbuch [12] shown the connection between the counting problem and the construction of a spanning tree, their time and communication complexity are related by a constant factor. In this setting counting can be solved in $\mathcal{O}(D)$ rounds and $\mathcal{O}(\log |V|)$-size messages.

Recently, Khun and Oshman [46] have shown that in **directed** networks, if each node can send at most $B = \Omega(\log |V|)$-bits, the time needed to solve counting is function of the network size, specifically $\Omega(\frac{|V|}{B})$ rounds, even in networks where the diameter D is 2. The bound is obtained by a showing a reduction from the two-player game of deciding if the hamming distance of two strings falls inside a certain range to counting. The presence of directed edges is a fundamental point in their proofs. Therefore this result cannot be ported in undirected static (or dynamic) networks.

Counting in Static Anonymous Networks.

The question concerning which problems can be solved on top of an anonymous network has been pioneered by Angluin in [3]. Yamashita and Kameda [69] proposed several characterizations for problems that are solvable under certain topological constraints when considering known the size, or an upper bound, of the network. Further investigation led to the classification of computable functions [8, 18, 19, 69, 70]. The work presented in [17] has been the first one that removed the assumption of knowing the network size and provided characterizations of the relations that can be computed with arbitrary knowledge.

A computational tool used in many of the previous work is the *view* of a node v, this concept has been introduced in [70]. The view of v is an infinite tree rooted in v and constructed by exchanging topological information, i.e. the local incoming/outgoing configuration of edges, with neighbors. Each path from v to a node in the tree corresponds to a walk in the original graph. Edges of this tree are labelled or not according to the communication model used. The broadcast model considered in this manuscript is equivalent to the *broadcast-to-mailbox* [71] and obviously lead to an unlabeled view. Unfortunately views cannot be used in an highly dynamic environments where a node may change neighborhood at each round.

It is well known, [34], that in the broadcast model the presence of the leader does not allow to assign unique IDs to each node, i.e. solving the *naming problem* [71]. The possibility of having a terminating naming algorithm would immediately lead to a trivial possibility result for counting. In contrast with naming is easy to see that in anonymous network with broadcast the presence of a leader is enough to solve counting, we may easily obtain an algorithm based on the technique used in [71].

In [61] is shown a counting algorithm that employs $\mathcal{O}(D)$ rounds and $\mathcal{O}(\log |V|)$-size messages. We are not aware of any non-trivial, i.e. different from $\Omega(D)$, lower bound on the counting time for static anonymous network in models that could be related to the one used in this manuscript.

Counting in Dynamic Non-Anonymous Networks.

In this paragraph we discuss known results on the counting for dynamic networks where IDs, or local edge-labeling, are provided. We first discuss the works where the dynamicity is governed by a random process and then when is governed by a worst-case adversary.

Non-Adversarial Networks In the context of p2p systems there is a vast majority of works that investigate the problem of counting when dynamicity is governed by a random process. Usually in such systems the set of nodes is not static, as assumed in this manuscript, but it is dynamic. Nodes can continuously leave and join the system, this is modeled by the concept of *churn*. Moreover the distribution of links and churn is usually assumed to be governed by some random distribution, in contrast with a worst case entity. In this context have been developed many size estimation algorithms based on probabilistic approaches. In [57, 58, 67] the size is estimated by using random walks. This approach is unsuitable in our model for, mainly, two reasons: (1) in order to do a random walk we have to send a specific message to one neighbor, this implies some sort of local naming that is not available in the broadcast model, (2) we have to *randomly* select the next hop in the walk and this would implies some sort of randomness source that we would to avoid. This need of randomness is shared by the majority of techniques develop for p2p system. In [40] a locally averaging approach is used, and it is leveraged to obtain a convergent counting algorithm. This method does not use randomness. However in their model unique IDs are assumed and used to implement unicast communication and request-reply patterns. In our model the unicast communication is impossible to achieve, and the same holds for the request reply pattern, two neighbors at round r could be positioned arbitrarily in the graph at successive rounds. Moreover the algorithm only converges to the correct count, there is no explicit termination condition, and bounds on the estimation error and convergence time are known only for dynamic topologies where links are created uniformly random. That is not the case studied in our model. This lack of an explicit exact termination, common to algorithms for size estimation in p2p systems, make them unsuitable for our purpose.

The results in [15] formally show that when there is churn it is impossible to have the exact count of network participants at some time t (i.e. impossibility of snapshot validity semantics). This impossibility result obviously holds also for anonymous dynamic network. This is one of the motivation for our focus on a static set $|V|$. The impossibility proof presented in [15] leverage a worst case adversary that governs churn.

Apart from the aforementioned paper, We are aware of very few works that cope with a worst case adversary governing churn and all are investigating subjects that are unrelated to counting: In [48] authors propose algorithms to maintain a specific overlay structure. Very recently, [9, 10] propose algorithms to solve distributed storage and byzantine consensus. These problems are not directly connected to counting, system models assume IDs and solutions are heavily based on random walks and their mixing properties in sequences of expander graphs.

Adversarial Networks Distributed systems with worst-case adversary and a static set of processes were first studied in [65], in this work the worst case adversary is conceptually replaced by the equivalent highly dynamic graph. The communication model assumed can be seen as an *asynchronous* version of 1-interval connected model, the graph changes locally and it is not governed by a round mechanism. A node v can detect if the neighborhood changes, and if v sends a message immediately after the detection of this event then the message will be received by all neighbors. As in 1-interval connected network this model does not allow request reply pattern, the neighborhood of v may change immediately after v receives a message.

They studied implicitly terminating flooding and routing problems. A flooding algorithm is implicitly terminating if eventually nodes stop to send flooding related messages. For the flooding they shows various algorithms under one of this two different conditions: (C1) a solution that uses storage restricted to $\mathcal{O}(\log |V|)$, i.e. each nodes can store a constant numbers of IDs and (C2) a solution that is *uniform*, i.e. there is no initial knowledge on the network size. Moreover with an additional assumption on the ID space, i.e. the maximum ID has to be an upper bound on the network size, they show an algorithm that fulfills both conditions. However solutions that ensure (C2) heavily relay on the presence of IDs, or on a structure of ID space, to compute an upper bound on the network size. Therefore they cannot be used or adapted to work in our environment.

Then they investigate the problem of flooding when IDs are not present. In this case they conjecture that without IDs is impossible to design an implicitly terminating flooding algorithm that respects condition (C1) or (C2).

In [44] the 1-interval connected model is introduced an investigate. Authors provide terminating algorithms for counting and broadcast. A key property of this model, on which the algorithms proposed in [44] are based, is that at round r each node v has been influenced, i.e it has received directly or indirectly a message, by a set of nodes of size $X = max(|V|, r)$. Conversely if a node v starts to continuously flood a message m at round 0, and each process that receives m does the same, we have that after at most $|V|$ rounds each process in the network has received m. This can be intuitively seen if we imagine that v is the only colored node *red* at round 0. The coloring rule is that each node neighbor of a red node changes color to *red*, for the connectivity assumption in each G_r there is an edge between a red node and an uncolored one, therefore if there is an uncolored node at each round the set of colored node increases. By the end of round $|V| - 1$ all nodes will be colored in *red*.

On this observation is based a simple counting algorithm that requires $\mathcal{O}(|V|)$ rounds and $\mathcal{O}(|V|)$ bits per message. Each node v floods continuously the set S_v of known IDs, at the beginning a node knows only its ID. The algorithm terminates at node v when $r > |S_v|$. The first process that obtains the count can disseminate this value to other using a terminating broadcast. This idea is similar to the ListfFooding algorithm of [65]. The additional explicit termination is given by exploiting the property induced by the synchronous assumption.

Starting from this algorithm the authors propose a refined counting that requires $\mathcal{O}(|V|^2)$ rounds using $\mathcal{O}(\log |V|)$ bits per message. The algorithm is based on "k-

committee", that is a partition of nodes in sets of at most k members if $k < |V|$ and if $k \geq |V|$ there must exist a single committee that include all nodes, each committee is associated with an unique ID.

The idea is that nodes, knowing k, can locally verify if they are in a k-committee with $k \geq |V|$ by flooding the committee ID. For the aforementioned property after $k+1$ rounds if two, or more, committee exist then each node knows at least the ID of two committee.

Based on this the authors propose an algorithm that given a value k partition the network in k-committee, when a partitioning is obtained the nodes simply verify if there is an unique committee and if this is not the case they increment k. After at most $|V|$ step we have that in the network there is a single committee therefore nodes stop and output the count.

The algorithm to create the k-committee must guarantee that each committee does not have more than k members, otherwise the verification procedure will fail, leading to an incorrect count. This is done in their model by using IDs to invite note to join a committee, that is initially each node belongs to the committee with ID equal to its own. After a flooding phase, each committee leader, that is the node with ID equal to the committee ID, invites to join the node with lowest ID heard.

This cannot be implemented in our model, because of the symmetry induced by anonymity. We cannot guarantee, at any point during the computation, that we can send a message that could be associated to only one receiver. Therefore we cannot have a bound on the numbers of node that will join a certain committee.

Very recently, in [60] is examined a worst case dynamic model in which the graph is not connected at each round. They define the oit:*outgoing influence time* that is the number of rounds needed for a node v to influence another node w, and the iit:*ingoing influence time* that is the number of rounds needed to v to be influenced by another node w. They show relationship between this two metrics. Moreover they define a kind of network where the adversary is restricted to keep a *bounded cover time*. In this model each node knows a local bound on the number of rounds necessary to be connected, at least once, to all its possible neighbors. They show that, if IDs exist, it exists a terminating counting algorithm for this kind of networks. The model of bounded cover time bear some resemblance with the model of *recurrent edges* of [20] where each edges has to appear after at most δ rounds with δ known to nodes, in this model [20] proposes solutions and impossibility results for terminating broadcasts.

In the context of **directed** strongly connected adversarial network [1] shows a connection between the counting and the problem of two-party k-token dissemination, a variant of the k-token dissemination where the tokens have to be sent by a source process to a destination process. In their model, if the size of messages is limited to $\mathcal{O}(\log |V|)$, a lower bound for the two-party k-token dissemination is also a lower bound for counting. However the authors do not specify bounds for two-party k-token dissemination. Therefore the only non trivial, different from $\Omega(D)$, lower bound for counting in directed network is still the one of [46].

Other known bounds for adversarial network are related to the k-token dissemination: [44] proved that when each node may send only one token at each round any k

token dissemination algorithm based on token forwarding [1] terminates in $\Omega(|V|\log k)$ round. In [31], the authors improved the bound to $\Omega(\frac{|V|k}{\log|V|})$. Starting from these results, [36] provides bounds for dynamic networks where more than one token at time can be sent. When IDs are present, the k-token dissemination is connected to the counting problem, a solution for n-token dissemination solves also counting.

Considering networks with limited dynamicity, the T-interval connected model is defined in [44]. It is shown that in ∞-interval connected networks with IDs counting requires $\mathcal{O}(|V|)$ rounds and $\mathcal{O}(\log|V|)$-size messages.

Counting in Dynamic Anonymous Networks.

Non-Worst Case Dynamic Networks Counting algorithms in the field of anonymous non worst case dynamic networks have been investigated in the context of passive mobile model [25]. Where nodes interacts, i.e. they influence the state of each other, when they meet. The interaction between agents are usually governed by a random schedule that ensures some fairness condition. In [25] we have a convergent counting algorithm that has no terminating condition, the algorithm works under the assumption that eventually two agents interacts infinitely often, since agents are assumed to have space $\mathcal{O}(\log|V|)$ the agents exchange $\mathcal{O}(\log|V|)$-size messages. In the same interaction model but by using population protocol and a distinguished leader, namely base station, [16, 39] show algorithms to solve a self-stabilizing convergent counting and lower bounds on space complexity. These results are not related to our model, they assume pair-wise asymmetric interactions and a fair scheduling.

In [41], the authors propose a convergent gossip-based protocol to compute aggregated function in a dynamic network by exploiting an invariant, called *conservation of mass*, defined over the whole set of processes. The underline network graph considered in [41] is complete and connections between nodes are obtained through a sampling function, i.e. the network dynamics is governed by a fair random adversary that exchanges edges between nodes according to an uniform distribution. On the contrary, in our work we do not assume such fairness in the network dynamics but we rather consider a graph where edges are selected in such a way to choose at each step the worst configuration possible to challenge the protocol, moreover the algorithm of [41] is only convergent and terminating conditions, or method to evaluate the error on the estimate, are not proposed.

Always in the context of gossiping protocol the work of [64] uses a randomized algorithm to compute an approximated counting. The idea behind this algorithm will be adapted to reach an approximated counting in worst-case networks with an *oblivious* adversary in [44]. The authors remark that their algorithm does not suffer from the oscillating behavior of [41]. The algorithm is randomized, and the randomization is used to implicitly break the network symmetry. Moreover a loose upper bound on the network size is needed in order to determine the quantity of random-

[1] "Token-forwarding algorithms are not allowed to combine, split, or change tokens in any way " [36].

ness that nodes have to sample. As previously discussed we want to avoid the use of any random source since it drastically changes the computational power of the model.

Worst Case Dynamic Networks The problem of Counting in interval connected anonymous networks has been attacked for the first time in [44]. Where is proposed a probabilistic algorithm for approximated-counting in anonymous worst case adversarial network, the algorithm leverages the ideas proposed in [64]. However this algorithm suffer from the need of randomness, that we want to avoid, and from a non exact terminating condition. Therefore cannot be used to our purposes. Moreover, the algorithm is given for the *oblivious* adversary model, where the adversary has to create in advance the dynamic graph without knowing the random choice of processes.

The first work investigating the problem of counting in a model that is analogous to our is [61, 62]. The authors conjectured the impossibility of counting when the broadcast communication is available. In the light of this conjecture they show an algorithm to compute an upper bound on $|V|$ when an upper bound D on the maximum degree that nodes may assume in the graph is known. This is assumption is equivalent to the $\mathcal{O}^{FOE}$ oracle that we will use in Chapter 4. The idea behind this upper bound algorithm is that nodes may compute an upper bound $U(r) = \sum_{i=1}^{r}(D+1)^i$ on the number of nodes that will be influenced, i.e. nodes that have received directly or indirectly a message from the leader, by round r. Each node that has not been influenced starts a flooding of the current round number. For the property of 1-interval connected network if at round $r+1$ exist nodes that have not been influenced then the leader will receive a message from one of them by round $r + U(r) + 1$. The upper bound computed is worst case exponential with respect to the actual network size. It could be $\mathcal{O}(D^{|V|})$. An exponential upper bound is particularly disadvantageous when used to implement terminating broadcast, the time that each broadcast takes to terminate is exponential with respect to the actual time needed to flood the network, that is $\mathcal{O}(|V|)$. They introduce an *high dynamicity* assumption, that is a quite involved form of fairness. They assume that there exists a k after which nodes must have broken the symmetry. Under this assumption they show a terminating counting algorithm. Apart from the broadcast model the authors also consider a model where a node can send a different message to each neighbors, namely *each-to-each*. In this model they show that terminating naming is solvable: the leader is able to assign an unique set of names at each round. When a node has an unique ID it could act as a leader assigning names on its own. This basic idea is refined in a naming algorithm that takes $O(|V|^3)$ rounds and $\mathcal{O}(\log |V|)$ messages.

1.3 Contribution

We will show that counting in our model is possible, albeit by using an exponential algorithm, closing the conjecture of [61, 62]. Showing that the presence of leader and interval connectivity is enough to do non trivial computations, such as terminating broadcast or computing aggregated function. In the path towards this result we will investigate the problem of counting in more restricted dynamic networks, proving bounds and giving optimal or efficient algorithms. Leading to a better understanding

on the anonymity impact. Specifically the contributions are listed in the following:

Bounds on Counting in $\mathcal{G}(\mathbf{PD})_2$ In Chapter 2 we investigate the complexity of counting in the family $\mathcal{G}(\text{PD})_2$. In $\mathcal{G}(\text{PD})_2$, processes in V_2 are connected to the leader by an unknown number of paths that dynamically changes at each round. If nodes are anonymous, ambiguity is created among these multiple dynamic paths. In this Section we show that this ambiguity leads to a bound of $\Omega(\log |V|)$ rounds for counting. Starting from this analysis we show a tradeoff lower bound on the accuracy of any estimation algorithm on $\mathcal{G}(\text{PD})_2$. Then we show how these bounds on $\mathcal{G}(\text{PD})_2$ lead to non trivial bounds on dynamic networks. Showing that counting, for any network where $D > 3$, always requires a number of rounds that is function of $|V|$. Finally we introduce and investigate the Fault Detection Problem in $\mathcal{G}(\text{PD})_2$ networks, showing a lower bound for it.

Optimal and Efficient Algorithms in $\mathcal{G}(\mathbf{PD})_h$ and $\mathcal{G}(\infty\textbf{-IC})$ In Chapter 3 we show counting algorithms for structured networks. We start by showing optimal algorithm for networks in $\mathcal{G}(\text{PD})_h$, obtained by extending an optimal counting technique developer for networks in $\mathcal{G}(\text{PD})_2$. Then we show an algorithm $\mathcal{FD}$ solving the Fault Detection Problem with a small gap with respect to the lower bound shown in Chapter 2. Finally we leverage the algorithm $\mathcal{FD}$ to obtain an algorithm that counts in any $G \in \mathcal{G}(\infty\text{-IC})$ using at most $\mathcal{O}(|V|^5)$ rounds.

Counting using Local Degree Detector Oracles In Chapter 4 we present counting algorithms for unstructured networks that work under different Local Degree Detector Oracles. These algorithms use a set of techniques that is different from the one used in the previous Section. The chapter starts with the formal definition of the oracles. The techniques used are introduced by showing a counting algorithm that uses a fixed known upper bound on the degree of all nodes. The idea is then refined to obtain a counting algorithm $\mathcal{A}_{\mathcal{O}^{OE}}$ that uses a local overestimate on the number of neighbors that a node may have at some round. This counting algorithm has a round complexity that is worst case exponential. However we also design a variant $\mathcal{A}^*_{\mathcal{O}^{OE}}$ that performs well when the adversary is random. The performance are evaluated experimentally at the end of the Chapter. The important point to remark is that $\mathcal{A}^*_{\mathcal{O}^{OE}}$ always terminates giving a correct count. Moreover we present and evaluate a convergent counting algorithm $\mathcal{A}_{NoK}$, that works without needing any additional information.

Counting in $\mathcal{G}(\textbf{1-IC})$ In Chapter 5 we show a terminating algorithm that works on $\mathcal{G}(1\text{-IC})$ without using additional knowledge. The idea and the techniques behind the algorithm derive from Chapters 2-3.

The algorithm implies that:

- We can design a terminating broadcast algorithm.
- By counting the initial states of nodes, we can answer to any predicates like: *"Are there k nodes with initial state i_0 in G?"*. This in turn implies that we can exactly compute many aggregated functions (e.g, *max, avg, count, sum,...*).

The proposed algorithm is not practical, it needs an exponential number of rounds and its memory requirements grow exponentially with the round number.

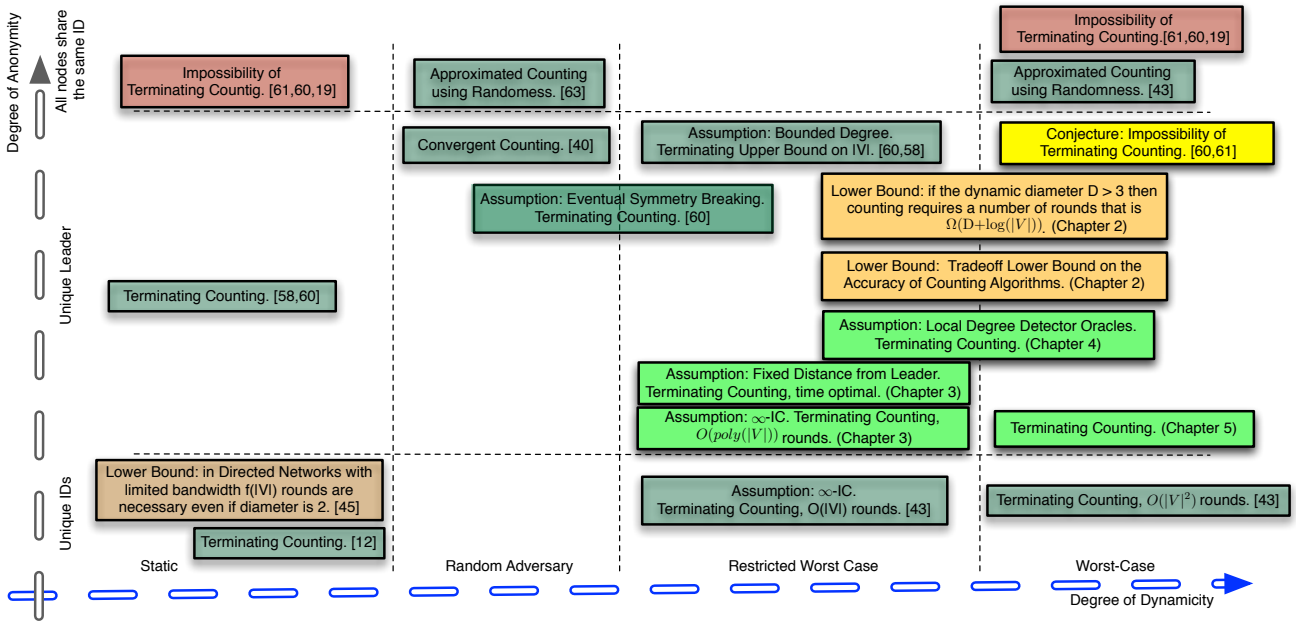


Figure 1.1: Contributions of this manuscript and relationships with related work. In the picture are considered only works where the communication primitive is broadcast

In Figure 1.1, there is a comparison of the contribution of this thesis with the respect to the relevant related work, in particular dynamic model where the communication is broadcast based. We can see that in this model the problem of terminating counting has been considered in only few works, and that no algorithm for terminating counting is known, even considering restricted environments as $\mathcal{G}(\infty\text{-IC})$ or $\mathcal{G}(\text{PD})_h$. The bounds presented in Chapter 2, are also new. At the best of our knowledge no trivial bounds, i.e. different from $\Omega(D)$, on counting are known in the case of unlimited bandwidth.

Part of the results contained in this manuscript have been published in the following papers [29, 50–53]. During the Ph.D., the author has also investigated: systems and algorithm security issues [5–7, 14] and collisionless pattern formation algorithms in the context of oblivious anonymous robots when they obstruct the view of each other [54–56].

Chapter 2

Bounds on $\mathcal{G}(\mathrm{PD})_2$ networks

The main result contained in this chapter is a lower bound $\Omega(log\,|V|)$ on counting time for networks in $\mathcal{G}(\mathrm{PD})_2$. This lower bound is proved in Section 2.1. An additional result that we obtain from our investigation $\mathcal{G}(\mathrm{PD})_2$ is a trade-off lower bound for the accuracy of size estimation algorithms: any size estimation algorithm that takes an input an upper bound $U \geq |V|$ and outputs a guess $n_{\mathcal{G}}$ on $|V|$ at round $r < \log_3(\frac{U}{4})$ makes an error $||V| - n_{\mathcal{G}}|$ that is at least $\frac{U}{4(3^r)}$.

These results show that, for any networks where $D > 3$, when anonymity and dynamicity are present the counting time is always function of the network size, this is explained in Section 2.2.

In Section 2.3, we define the Fault Detection Problem on $\mathcal{G}(\mathrm{PD})_2$. In this problem the leader has to distinguish between two runs one with processes that crash and another one without faults. We show that to solve this problem $\Omega(|V_1|\lfloor log(\lfloor\frac{|V_2|}{|V_1|}\rfloor+1)\rfloor)$ rounds are necessary.

At the best of our knowledge the only other known bound that shows a model where the time necessary for counting is not only function of D is in [46]. [46] considers directed static networks with IDs with limited bandwidth. The technique that we use to prove our result is completely unrelated to the one used in [46].

At the end of the Chapter we present some conclusions and we identify interesting open problems.

2.1 Lower bound for $\mathcal{G}(\mathrm{PD})_2$

In this section we consider the family $\mathcal{G}(\mathrm{PD})_2$ 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 than counting over $\mathcal{M}(\mathrm{DBL}_k)$ and by finally showing a lower bound on the number of rounds needed to count over $\mathcal{M}(\mathrm{DBL}_k)$. In this section we assume that at each round a node is able to read neighbors memory.

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, $f_r : E(r) \rightarrow \{v_l\} \times W$ a function that maps each edge to the endpoints nodes and $l_r : E(r) \rightarrow \{1, 2, \ldots, k\}$ a function labeling edges. M belongs to $\mathcal{M}(\text{DBL}_k)$ 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 $\forall r, \forall v \in W, E^v(r) = f^{-1}(v_l, v) :: 1 \leq |E^v(r)| \leq k$; Given $e', e'' \in E^v(r)$ we have $l_r(e') \neq l_r(e'')$. For simplicity we will refer as M_r the instance of M at round r. Figure 2.1 shows an example of a dynamic connected multigraph M at round r belonging to $\mathcal{M}(\text{DBL}_3)$. We consider that when a node $v \in \{v_l\} \cup W$ reads the memory of a node w at round r from edge e, it also obtains the label $l_r(e)$.

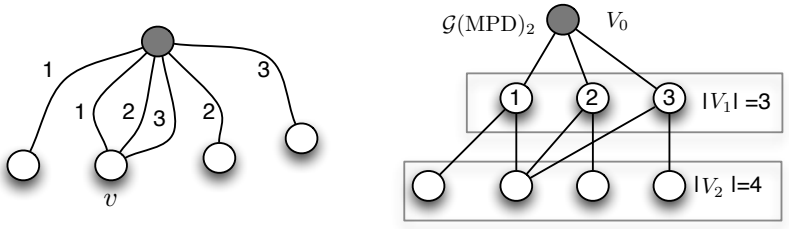


Figure 2.1: Trasformation, at round r, from $\mathcal{M}(DBL_3)$ multigraph to $\mathcal{G}(\text{PD})_2$.

Lemma 1. *Let 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 $G^{id} \in \mathcal{G}(\text{PD})_2$ 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 2.1 shows the transformation at round r between a dynamic graph in $\mathcal{M}(\text{DBL}_3)$ and one in $\mathcal{G}(\text{PD})_2$. 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. □

Let consider an instance M of the family $\mathcal{M}(\mathrm{DBL}_k)$, we introduce the following notions on M:

Definition 9. *(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 2.1, the edge label set of node v at round r is $\{1,2,3\}$.

Definition 10. *(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*[1]*.*

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 11. *(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 where $(j, S(v,i)) \in C(v_l, i)$ 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 each node sends to the leader its own state at each round.

The next lemma shows that a bound given for algorithms in which the decision of the leader is based on the state $S(v_l, r)$ is an asymptotic lower bound for any counting algorithm $\mathcal{A}$. Even algorithms in which the decision about the size is not taken by v_l.

Lemma 2. *Let us consider any message passing algorithm $\mathcal{A}$ that runs on top of dynamic multigraph $M \in \mathcal{M}(DBL_k)$. We have that exists an algorithm $\mathcal{A}$ that round r decides about a certain property of M, i.e. the size $|W|$, iff exists an algorithm $\mathcal{A}_s$ that decides on the same property using $S(v_l, r)$ or $S(v_l, r+1)$.*

Proof. sketch The $\leftarrow$ direction is trivial. We have to prove the $\rightarrow$. Let us consider the protocol $\mathcal{P}^*$ in which a process $v \in \{v_l\} \cup W$ at each round sends the initial content of its memory and the entire history of messages received at the previous round. We use $S^*(v,r)$ to define the memory content of a process v that runs $\mathcal{P}^*$ until round r. Any algorithm $\mathcal{A}$ will use a subset of the information contained in $S^*(v,r)$. Now we will show that from $S(v_l, r)$ we can reconstruct $S^*(v_l, r)$. In $\mathcal{P}^*$ at round r a non leader process v receives, from a set of labeled edges, the entire memory of v_l at round $r-1$; moreover v sends the copy of its memory at round $r-1$. The new meaningful information for v_l received from the node v and round $r-1$ its only the history of edges labels incident to v; and not the old copy of the leader memory $S^*(v_l, r-2), \ldots, S^*(v_l, 0)$ that v is sending back. Now let us suppose that at the end of round r, $\mathcal{A}$ allows the leader v_l to decides using the information in $S^*(v_l, r)$. We have that also $\mathcal{A}_l$ can decide at round r since the state $S(v_l, r)$ is equivalent to

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

$S^*(v_l, r)$. Now let us suppose that at the end of round r, $\mathcal{A}$ allows $v \neq v_l$ to decides using information in $S^*(v, r)$, that contains the $S*(v_l, r-1), \ldots, S^*(v_l, 0)$ received by the leader in the previous rounds plus and the information $L(v, r)$ associated to the edges incident to v at round r, and the old history of edge incident to v in the previous rounds. So $\mathcal{A}_l$ can simulates the decision of $\mathcal{A}$ on v, at leader v_l by using $S(v_l, r+1)$ at round $r+1$ since the history of edge labels of v until round r is in $S(v_l, r+1)$. □

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

In order to ease the presentation of the proof we present the underline intuition starting with a small example for the restricted family $\mathcal{M}(\mathrm{DBL}_2)$ and for only the first round, later we first generalize the example for runs of any number rounds thus proving the bound for the family $\mathcal{M}(\mathrm{DBL}_2)$, and then we show that the same bound holds for $\mathcal{M}(\mathrm{DBL}_k)$. The proof will use linear algebra concepts. Thus we will introduce some notation on vector and Matrices that will we use in this section.

Linear algebra notation Given a vector $\mathbf{a} \in \mathbb{N}^n$, 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 $\mathbf{M} \in \mathbb{N}^{n,m}$ we indicate 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 $\mathbf{a} \in \mathbb{N}^m$ 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.

The case of $k = 2$ and $r = 0$. In this case we have only two edge labels: $1, 2$. Let us consider the leader state on a generic multigraph $M \in \mathcal{M}(\mathrm{DBL}_2)$. We have that at the end of round 0 the leader has state $S(v_l, 0) : [\{(1, [\bot]), (2, [\bot])\}]$, 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:

$$\begin{cases} |(1, [\bot])| = |[\{1\}]| + |[\{1, 2\}]| \\ |(2, [\bot])| = |[\{2\}]| + |[\{1, 2\}]| \end{cases}_{r=0} \tag{2.1}$$

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 2.1 can be written in a matrix form as follows:

$$\mathbf{m}_0 = \mathbf{M}_0 \mathbf{s}_0 \tag{2.2}$$

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 $\mathcal{M}(\text{DBL}_2)$.

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 2.2 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 (2.1) 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 of the solutions represents a distinct graph belonging to $\mathcal{M}(\text{DBL}_2)$ 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 2.2, the system of equation at round 0 for the multigraph M is the following

$$\underset{r=0}{\begin{cases} 2 = |[\{1\}]| + |[\{1,2\}]| \\ 2 = |[\{2\}]| + |[\{1,2\}]| \end{cases}} \tag{2.3}$$

where $\mathbf{m}_0 : \begin{bmatrix}2 & 2\end{bmatrix}$. For such a 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 corresponds to two $M, M' \in \mathcal{M}(\text{DBL}_2)$ of different size that generates the same state $S(v_l, 0)$ as depicted in Figure 2.2. These two graphs are indistinguishable from the leader at round 0 thus the leader is not able to output a correct count.

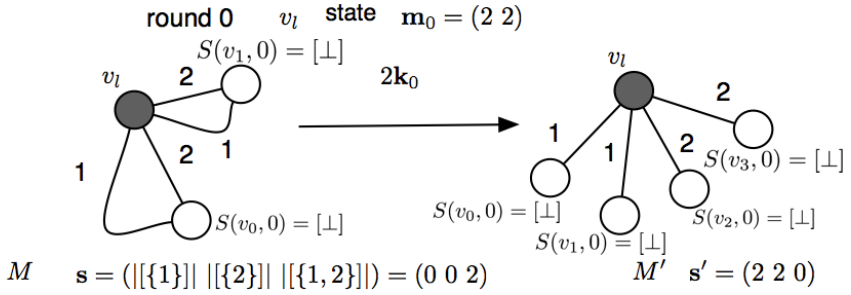


Figure 2.2: Two dynamic multigraphs $M, M' \in \mathcal{M}(\text{DBL}_2)$ of different size that are indistinguishable at round $r = 0$, the relationship among M, M' is given by the kernel vector $\mathbf{k}_0$

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 can have an unique solution, that corresponds to an unique size $|W|$. Essentially there is an intimate connection between the kernel space of a certain succession of matrices and the possibility counting in $\mathcal{M}(\text{DBL}_2)$.

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 ($2\cdot3^0+2\cdot3^1$) and 9 variables (i.e. 3^2 rows) and the associated matrix $\mathbf{M}_1$ are:

$$\begin{cases} |(1,[\perp])| = \sum_{\forall j\in\{\{1\},\{2\},\{1,2\}\}} |[\{1\},j]| + \sum_{\forall j\in\{\{1\},\{2\},\{1,2\}\}} |[\{1,2\},j]| \\ |(2,[\perp])| = \sum_{\forall j\in\{\{1\},\{2\},\{1,2\}\}} |[\{2\},j]| + \sum_{\forall j\in\{\{1\},\{2\},\{1,2\}\}} |[\{1,2\},j]| \\ |(1,[\{1\}])| = |[\{1\},\{1\}]| + |[\{1\},\{1,2\}]| \\ |(1,[\{2\}])| = |[\{2\},\{1\}]| + |[\{2\},\{1,2\}]| \\ |(1,[\{1,2\}])| = |[\{1,2\},\{1\}]| + |[\{1,2\},\{1,2\}]| \\ |(2,[\{1\}])| = |[\{1\},\{2\}]| + |[\{1\},\{1,2\}]| \\ |(2,[\{2\}])| = |[\{2\},\{2\}]| + |[\{2\},\{1,2\}]| \\ |(2,[\{1,2\}])| = |[\{1,2\},\{2\}]| + |[\{1,2\},\{1,2\}]| \end{cases}_{r=1} \tag{2.4}$$

$$\mathbf{M}_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.5}$$

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:

$$|(j,[x_0,\ldots,x_{r'}])| = \sum_{\forall s\in(\{1\}|\{2\}|\{1,2\})^{r-r'}} |[x_0,\ldots,x_{r'-1},\{j\},s]| + \\ + \sum_{\forall s\in(\{1\}|\{2\}|\{1,2\})^{r-r'}} [x_0,\ldots,x_{r'-1},\{1,2\},s]| \tag{2.6}$$

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 2.1) and the equation associated with $|(1, [\perp])|$ at round 1 (see Equation 2.4).

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 $n = 5$. Ut 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 2.3.

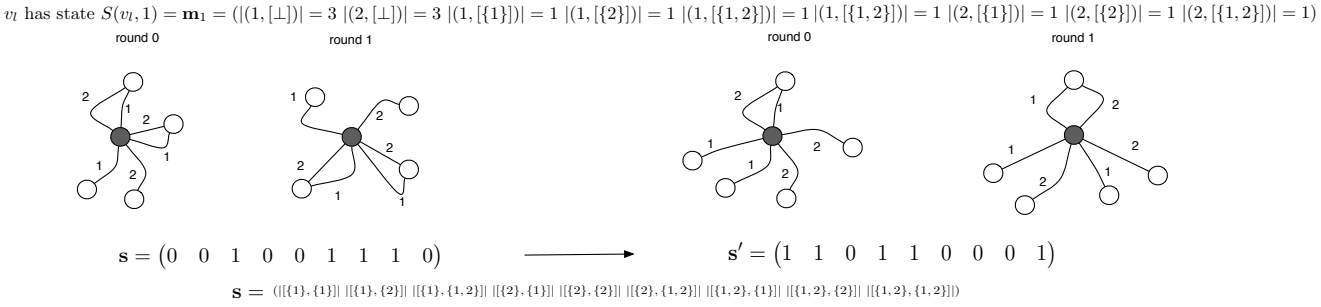


Figure 2.3: Two dynamic multigraph $M, M' \in \mathcal{M}(\mathrm{DBL}_2)$ of different size that are indistinguishable at round $r = 1$, they induce the same leader state $S(v_l, 1) = \mathbf{m}_1$, the relationship among the two is given by the kernel vector $\mathbf{k}_1$

Ordering Columns and rows in $\mathbf{M}_r$. For the sake of simplicity of 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: $|[\{1\}, \dots \{1\}]|$, the second column $|[\{1\}, \dots \{1\}, \{2\}]|$ and the last one $|[\{1, 2\}, \dots, \{1, 2\}]|$. Rows are ordered in the same way. This ordering has been used in Equation 2.2 and Equation 2.5. Fixed this ordering we can use a connection $(j, [x_0, \dots, x_{r'-1}])$ to indicates a row $\mathbf{v} = (\mathbf{M}_r)_{(j,[x_0,\dots,x_{r'-1}])}$ and a node state to indicate a single component of a vector, i.e. $(\mathbf{v})_{[x_0,\dots,x_{r-1}]}$. Moreover we have that the row vector $(\mathbf{M}_r)_{(j,[x_0,\dots,x_{r'-1}])}$ will have two trails of ones, with length $3^{r-r'}$, for all columns in the form

$$|[x_0, \dots, x_{r'-1}, \{j\}, s]|, |[x_0, \dots, x_{r'-1}, \{1, 2\}, s]|$$

with $s \in (\{1\}|\{2\}|\{1, 2\})^{r-r'}$, and zero for all the other columns (as reference see Eq. 2.5).

In the following lemmas we specifically characterize the structure of the kernel space of $\mathbf{M}_r$ in order to check if there is one solution to the system of equations.

Lemma 3. *Let 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 independents.
- 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 $jmod3 = 0$, 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 [49], 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 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 $r - 1$, that by definition of kernel we have:

$$0 = \sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} (\mathbf{k}_{r-1})_{|[l',\{j\},s]|} + \\ + \sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} (\mathbf{k}_{r-1})_{|[l',\{1,2\},s]|} \quad (2.7)$$

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

$$\sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'-1}} ((\mathbf{k})_{|[l',(j),s,\{1\}]|} + (\mathbf{k})_{|[l',(j),s,\{2\}]|} + (\mathbf{k})_{|[l',(j),s,\{1,2\}]|}) + \\ + \sum_{\forall s \in (\{1\}|\{2\}|\{1,2\})^{r-r'}} (\mathbf{k})_{|[l',\{1,2\},s]|} \quad (2.8)$$

the first term of Eq. 2.8 can be expressed as follow

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

Also the second term of Eq. 2.8 can be rewritten as the first term, then by applying Eq. 2.7, we have:

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

Now let us consider the row $c = (j, l : [x_0, \dots, x_{r-1}])$ 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 that $\mathbf{k}_{r-1} = \begin{bmatrix}\mathbf{k}_{r-2} & \mathbf{k}_{r-2} & -\mathbf{k}_{r-2}\end{bmatrix}^\intercal$, moreover we have, for Lemma 3, that
$\mathbf{k}_{r-1} = \begin{bmatrix}(\mathbf{k}_{r-2})_1\mathbf{k}_0 & (\mathbf{k}_{r-2})_2\mathbf{k}_0 & \dots & (\mathbf{k}_{r-2})_{3^{r-1}}\mathbf{k}_0\end{bmatrix}^\intercal$. This means that the first 3^r components of $\mathbf{k}'$ are $\mathbf{k}_{r-1}$, the same holds for the other 3^r components, and the last 3^r are $-\mathbf{k}_{r-1}$. This completes the proof.

□

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

Proof. Thanks to lemma 4, we have that $\sum^+ \mathbf{k}_r = 2\sum^+ \mathbf{k}_{r-1} + \sum^- \mathbf{k}_{r-1}$, $\sum^- \mathbf{k}_r = 2\sum^- \mathbf{k}_{r-1} + \sum^+ \mathbf{k}_{r-1}$ 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*: $\sum^+ \mathbf{k}_0 - \sum^- \mathbf{k}_0 = 1$.
- *Inductive step:* $(\sum^+ \mathbf{k}_r - \sum^- \mathbf{k}_r) = (2\sum^+ \mathbf{k}_{r-1} + \sum^- \mathbf{k}_{r-1} - \sum^+ \mathbf{k}_{r-1} - 2\sum^- \mathbf{k}_{r-1}) = (\sum^+ \mathbf{k}_{r-1} - \sum^- \mathbf{k}_{r-1}) = 1$.

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 $\sum^+ \mathbf{k}_0 = 2$. Resolving the recursive relation we obtains $\sum^+ \mathbf{k}_r = \frac{1}{2}(3^{r+1}+1)$. This implies that $min(\sum^+ \mathbf{k}_r, \sum^- \mathbf{k}_r) = \frac{1}{2}(3^{r+1}+1)-1$ where the last term takes into account the fact that the minimum is always the negative component and that $\sum \mathbf{k}_r = 1$.

□

Lemma 6. *Let us consider $M, M' \in \mathcal{M}(DBL_2)$ with sizes $|W| = n, |W'| = n+1$. 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.

□

Moreover from Lemma 3 we have:

Lemma 7. *There is an optimal algorithm for counting on $M \in \mathcal{M}(DBL_2)$ that decides at round r if $|W| = n$ with $r = \lfloor log_3(2|W|+1) \rfloor + 1$.*

Proof. The optimal algorithm, is given by leveraging the same ideas that we used in the previous proof, when $r = \lfloor log_3(2n+1) \rfloor + 1$ we have that $\sum^- \mathbf{k}_r > n$ thus cannot exist a solution $\mathbf{s}_r$ such that $(\mathbf{s}_{r'})_j \geq 1$ for each $j \,|\, (\mathbf{k}_{r'})_j < 0$ and $\sum \mathbf{s}_r = n$. This implies that given $\mathbf{m}_r$, thanks to the unicity of the kernel vector (see Lemma 3), there is only one solution with non negative components for the linear system $\mathbf{m}_r = \mathbf{M}_r \mathbf{s}_r$. This unique solution gives the size of the network. □

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

Proof. On $\mathcal{M}(\mathrm{DBL}_k)$ multigraphs there exists a set of multigraphs that always only two labels, creating the same setting generated by an instance on $\mathcal{M}(\mathrm{DBL}_2)$, this means that each algorithm has at least to solve the counting also on $\mathcal{M}(\mathrm{DBL}_2)$, from Lemma 6 we know that in order to distinguish a networks of size $|W|$ from $|W|+1$ we have to wait at least $\lfloor log_3(2|W|+1) \rfloor$ rounds. This complete the proof. □

From Theorem 1 and Lemma 1 the next theorem immediately follows.

Theorem 2. *Given an instance $G \in \mathcal{G}(PD)_2$ any counting algorithm $\mathcal{A}$ on G requires $\mathcal{O}(log(|V|))$ rounds.*

Discussion We have that the same kernel based strategy could be used for any $k > 2$, it is clear that if v_l creates a system of equations in a manner similar to the one used for $k = 2$ it obtains an optimal algorithm for counting. One problem that makes difficult to investigate $ker(\mathbf{M}_r)$ for $k > 2$ is that the kernel space is not anymore of fixed size. The size of the kernel starts to grow exponentially with respect to r. Even if we consider $k = 3$, it grows like $\frac{7^{r+1}+1}{2}$.

Moreover Theorem 1 does not say anything about the possibility to count in $\mathcal{G}(\mathrm{PD})_2$. For this reason in Section 3.1 we show a simple algorithm to count in $\mathcal{G}(\mathrm{PD})_2$ in $\mathcal{O}(\log_2(|V|))$ rounds.

Consequences for size estimation algorithms. Given the leader state $S(v_l, r)$ the kernel vector $\mathbf{k}_r$ of $\mathbf{M}_r$ characterizes all networks that could generate that state. This has been exploited in Theorem 1, to prove a lower bound on counting time, but it could also be used to give an upper bound on the precision of deterministic size estimation algorithms, as we show in the next theorem.

Lemma 8. *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 $r < log_3(\frac{U}{4})$, 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 $\frac{U}{4(3^r)}$.

Let us recall that $\mathcal{A}_e$ is deterministic thus the adversary, that has access to the internal variables of all nodes, knows exactly: (1) at which round $\mathcal{A}_e$ outputs its guess and (2) the value of the guess. Without lost of generality, let us assume that $U = 2(3)^{\Delta}$ for some $\Delta \in \mathbb{N}^+$.

We first prove an intermediate results:

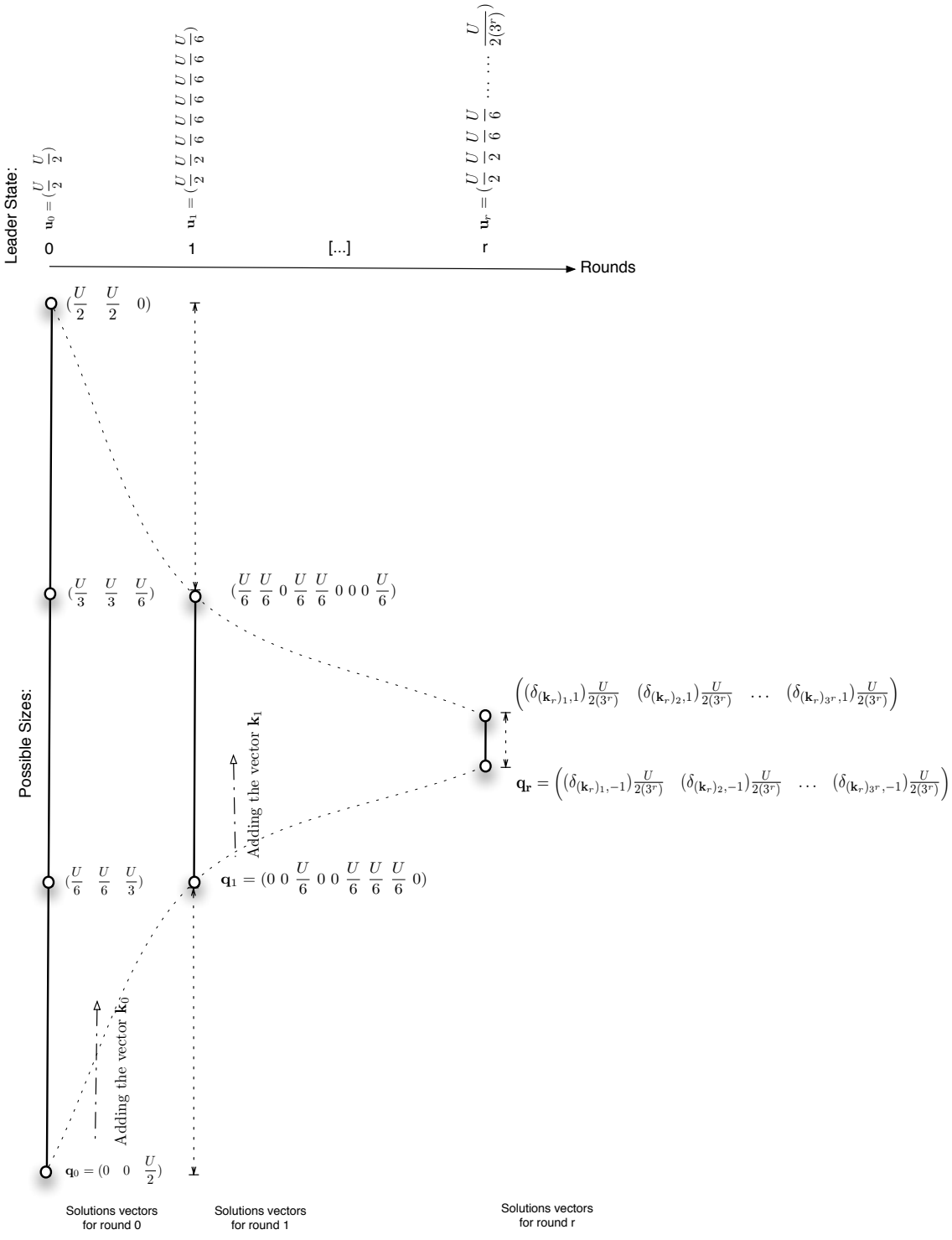


Figure 2.4: Structure of solutions used in Theorem 8. The symbol $\delta_{j,i}$ indicates the Kronecker's delta, its value is 1 if $i = j$ and 0 otherwise

Lemma 9. *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.

- $\mathcal{A}_e$ *outputs the guess at round* r. The leader state is the vector $\mathbf{u}_r = \begin{bmatrix}\frac{U}{2} & \frac{U}{2} & \frac{U}{2(3^1)} & \frac{U}{2(3^1)} & \frac{U}{2(3^1)} & \cdots & \frac{U}{2(3^r)}\end{bmatrix}^\intercal$ 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 9. 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.

□

2.2 Anonymity and Dynamic Diameter

The results found in the previous section have non trivial consequences on the connection of the counting with the dynamic diameter. Let us recall that when IDs are present, in case of unlimited node space and bandwidth counting requires exactly D rounds.

On contrary counting in anonymous dynamic networks has a complexity that is function of the network size even if $D = \mathcal{O}(1)$. Given a constant $D > 3$, we can 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 $\mathcal{G}(\text{PD})_2$ network. From this observation and Theorem 2 the next corollary follows.

Corollary 1. *Given a dynamic network with fixed known 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.*

It is easy to see that even if $D = \mathcal{O}(|V|)$ we can create configurations where we have still to pay a term of $\Omega(\log |V|)$ rounds, e.g. we create a static line of length $D = \frac{|V|}{2} - 1$ and we use the remaining $\frac{|V|}{2}$ nodes to create a $\mathcal{G}(\text{PD})_2$ network at the end of the line. However this term is asymptotically dominated by the diameter.

The second results is about the accuracy of counting algorithms. 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 3. *Let us consider a dynamic networks with dynamic diameter D, and $D > 3$. 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 $D < r < D + log_3(\frac{U-D}{4})$, 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 $||V| - n_\mathcal{G}| \geq \frac{U-D}{4(3^r)}$.*

Proof. We first prove an intermediate Lemma

Lemma 10. *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 10. Using this transformation we have for each multigraph $M \in \mathcal{M}(\text{DBL}_2)$, it exists a dynamic network $G^{id} \in \mathcal{G}(\text{PD})_2$, 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. □

The accuracy bound obtained in Lemma 8 for $\mathcal{M}(\text{DBL}_2)$, by Lemma 10, it also holds for $\mathcal{G}(\text{PD})_2$. From the bounds obtained on $\mathcal{G}(\text{PD})_2$, the claim of the theorem follows by using the same construction used in Corollary 1. □

The lower bounds have been obtained by using a dynamic labeled multigraph. An unlabelled multigraph models also an anonymous static network where broadcast messages may be duplicated, if a message from node v is delivered 3 times to its neighbor w at round r in G_r there will be at least 3 edges from v to w. Therefore our bounds on time and accuracy also holds for this kind of static anonymous networks.

Corollary 2. *Let us consider a synchronous static anonymous network where nodes communicates by using an anonymous broadcast primitive. Let us assume that there*

exists a leader node v_l, and that the maximum distance from v_l to a node v is h. If a message m sent by node v can be delivered multiple times to one of its neighbors then counting requires at least $h + \Omega(max_{\forall i} log|V_i|)$ rounds, where V_i is the set of nodes at distance h from v_l.

Proof. It is easy to see that counting in an undirected edge-unlabeled dynamic multigraph is easier than counting in a static anonymous network where messages may be duplicated. In order to model the duplication with an undirected multigraph we are assuming that each time a message m from v to w is duplicated also a message m' from w to v is duplicated. Counting in an edge-labeled multigraph is easier than counting in one that is edge-unlabeled. Let us consider a construction that generalize $\mathcal{M}(\text{DBL})_2$ in order to have nodes at distance h from v_l. That is v_l is connected, by a dynamic multigraph as in $\mathcal{M}(\text{DBL})_2$, to nodes in V_1. Then only one node in $v^1 \in V_1$ is connected to nodes in V_2 as in $\mathcal{M}(\text{DBL})_2$ and so on until set V_h. This construction models a static anonymous networks that is a tree where the leader is the root and where nodes at level h are all connected to the same node at level $h-1$. With the additional assumptions that duplications are reciprocal and that edges are labeled. It is easy see that v^j, in order to count nodes in V_{j+1}, has to wait $\Omega(max_{\forall i} \log |V_i|)$ rounds, and that counting at different levels do not interfere with each other. Therefore the leader outputs the count only after it has received the partial sum from each v_j. From this the claim follows.

□

2.3 Investigating Faults in $G(\text{PD})_2$ networks

In this section we will study the case of faulty processes. We say that v_i is faulty at round r if it leaves the system before the broadcast operation of round r. We model failures as a set of processes that permanently stop sending messages at the beginning of certain round r. We assume that the failures start from round $r = 1$: processes send at least one message before their departure.

Let us introduce a basic problem:

Problem 1. Fault Distinguisher Problem (FDP): *Given two run R_f, R such that: in the run R no processes fail; in the run R_f there is a failure at $r = 1$ and there could be other failures in successive rounds. An algorithm FD solves the Fault Distinguisher problem if at some round r it outputs a value and terminates. The value has to be Red on the run R_f, and Green on the run R.*

A simple broadcast algorithm solves this problem in a $\mathcal{G}(\text{PD})_2$ network with identifiers, moreover v_l is able to terminates at round $r = 3$ independently from the network size.

Lower Bound on Fault Detection for $\mathcal{G}(\text{PD})_2$ networks.

In this subsection we show a lower bound for FD problem. It is interesting to notice that the lower bound that we shows is greater than the time needed to count. It is clear that the time needed to detect a fault is at least the time that we need to count: let us imagine that all nodes have the same state at the end of round 0, then

one nodes fails, the leader has to count the remaining nodes in order to detect the failure and this needs $\Omega(\log(|V_2|))$ rounds, see optimal counting algorithm for $\mathcal{G}(\mathrm{PD})_2$ in next Chapter. With the next theorem we show that for $\mathcal{G}(\mathrm{PD})_2$ the adversary can exploit a "*chain of failures*" 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 $|V_1|$, this is in contrast with the simple counting studied in the previous sections.

Theorem 4. *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 Fault Distinguisher algorithm that outputs a results 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 indicate with v_i^x that $v_i \in V_x$.

Let us first consider the case $|V_2| = |V_1|$. We consider the G_0 at round 0 in which each node v_j^2 in V_2 is connected to only one node v_j^1 in V_1 and viceversa. It is easy to see that at the end of round 0 thanks to the anonymous broadcast all processes in V_2 have the same memory content, the same hold for all processes in V_1. Now at the beginning of round 1, v_1^2 leaves. The adversary takes process v_2^2 in V_2 and connects it to v_1^1 and v_2^1, from the point of view v_1^1,v_2^1 this configuration is not distinguishable to the one of the previous round, and clearly is not distinguishable for the other processes in the system. At round r, v_r^2 leaves 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 equivalent.

Now let us consider the case $|V_2| > |V_1|$, since we are proving a lower bound we restrict ourselves to 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. In G_1, $v_1^{1,2}$ leaves 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 nodes in V_1 are kept in the same state, at each round a node in V_1 receives x identical messages by nodes in V_2, this implies that also the non faulty nodes 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 nodes to mask the faults 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 faulty 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 faulty processes and the remaining sets have $x - a$ non faulty 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 behavior can be iterated for $|V_1|\lfloor\log(\lfloor\frac{|V_2|}{|V_1|}\rfloor)\rfloor$ rounds. Let us focus on the number of alive processes in the systems, as long as the non faulty the 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 failures, processes that will leave the system at the next round, this means that starting from $|V_2| - a \cdot |V_1|$ non faulty processes we remain with $|V_2| - (a+1) \cdot |V_1|$ non faulty processes after $\frac{|V_1|}{a+1}$ rounds. After $|V_1|$ rounds we have $|V_2| - |V_1|$ non faulty processes, after $\frac{|V_1|}{2}$ rounds we have $|V_2| - 2|V_1|$ non faulty processes, and so on until we have 0 non faulty 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 $\{G_0, G_1 = G_0, \ldots, G_{|V_1|\log(\frac{|V_2|}{|V_1|}+1)-1} = G_0\}$.

□

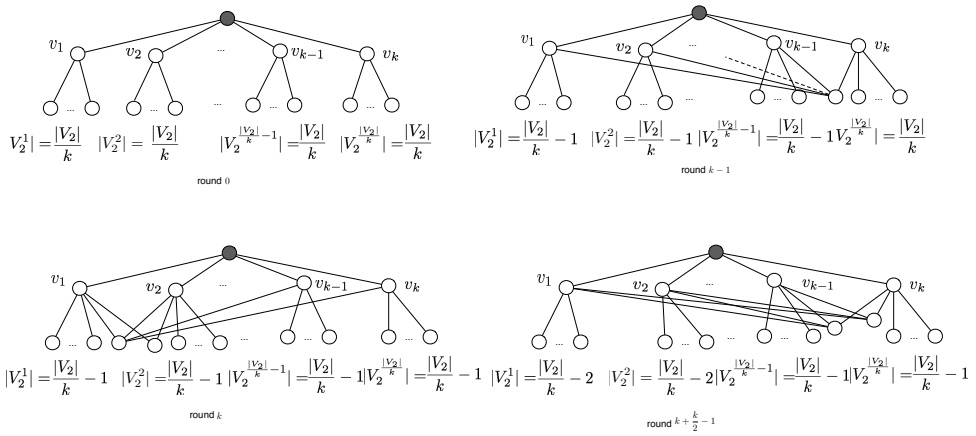


Figure 2.5: **FDP** Lower bound: indistinguishability example for $|V_1| > |V_2|$.

Robust Counting If we are in a system where processes in $V \setminus \{v_l\}$ may leave, it could be of interest defining the following notion of counting:

Definition 12. *Robust Terminating Counting: Given a dynamic network G with an initial set V of processes. Processes in $V \setminus \{v_l\}$ may fail and permanently stop to send messages. Let us define as V^r the set of non-faulty processes at round r. A distributed algorithm $\mathcal{A}$ solves robust terminating counting on G if it satisfies the following properties:*

- *Termination: It exists a round r at which the leader outputs a pair (X, r') and terminates.*
- *Correctness: We have $|V^{r'}| = X$.*

This notion is similar to the snapshot validity of [15]. It is easy to see that if we do not ensure some restriction on the pattern of failures any robust counting algorithm could only output the trivial $X = 1$. This is straightforward if we consider a run in which at round 1 all processes but the leader fail. Therefore in order to study non trivial instances of robust anonymous counting, let us define the robust dynamic diameter D^*.

Definition 13. *Robust Dynamic Diameter: Given a dynamic network G, we say that G has robust dynamic diameter D^* if at round r each process $v \in V^r$ can start a flooding of message m and by round $r + D^*$ m will be received by all non faulty process in V^{r+D^*}.*

A network in $\mathcal{G}(\mathrm{PD})_2$ where only processes in V_2 fails is network with robust diameter $D^* = 4$. It is easy to see that in a network with IDs and D^* known a simple flooding algorithm solves robust counting. The leader at round D^* outputs the number of IDs sent at round $r = 0$. We have the following corollary,

Corollary 3. *Given a dynamic network with robust dynamic diameter D^*. We have that if $D^* > 3$ any robust counting algorithm $\mathcal{A}$ requires $\Omega(|V^0|)$ rounds to give an output.*

Proof. The proof follows from Th.4, a robust algorithm has to solve **FDP** on $G \in \mathcal{G}(\mathrm{PD})_2$, if we have $|V_1| = |V_2|$ the claim follows. □

The previous corollary implicitly used a simple *"trick"*: we have exploited the anonymity and the unbounded degree of nodes to allow a node in V_2 to *"take place"* of up to $\mathcal{O}(|V^0|)$ failed nodes. Therefore it is natural to ask what could happen if we bound the degree of failing nodes to be $\mathcal{O}(1)$ with respect to $\mathcal{O}(|V^0|)$. We also restrict the number of nodes that could depart in a certain round, otherwise we would simply exploit $\mathcal{O}(|V^0|)$ nodes with degree $\mathcal{O}(1)$ to replace the past failures of $\mathcal{O}(|V^0|)$ nodes. We will show that even in networks with robust diameter and constant number of failures in the worst case the adversary could always force any robust counting algorithm to a trivial output that does not depend on the initial size of processes, i.e. $X \in \{1, 3\}$.

As we did for counting we consider the networks in $\mathcal{M}(\mathrm{DBL}_2)$. We prove the impossibility result in these networks as it could be easily ported to $\mathcal{G}(\mathrm{PD})_h$. Let us notice that for $\mathcal{M}(\mathrm{DBL}_2)$ we have $D^* = 1$ and that failing nodes could have at most degree $d = 2$. We have the following Theorem.

Theorem 5. *Let us consider a network in $\mathcal{M}(DBL_2)$, where at each round $f = \mathcal{O}(1)$ processes may fail, the number of failure is constant with respect to the initial network size. We have that any robust counting algorithm has to wait $\Omega(|W|)$ rounds and it can only output a trivial value $X = 1$.*

Proof. Let us consider two networks $M, M' \in \mathcal{M}(\mathrm{DBL}_2)$. M has $|W| = 2x$ processes and M' has $|W'| = 2x + 1$ processes with $x \in \mathbb{N}^+$. In Figure 2.6 there is a small example that we generalize in the following description.

- The network M has the following dynamicity. At round r we have $|W| - 2\lfloor \frac{r}{2} \rfloor$ non-faulty processes. if r is even, including $r = 0$, we have that all processes are

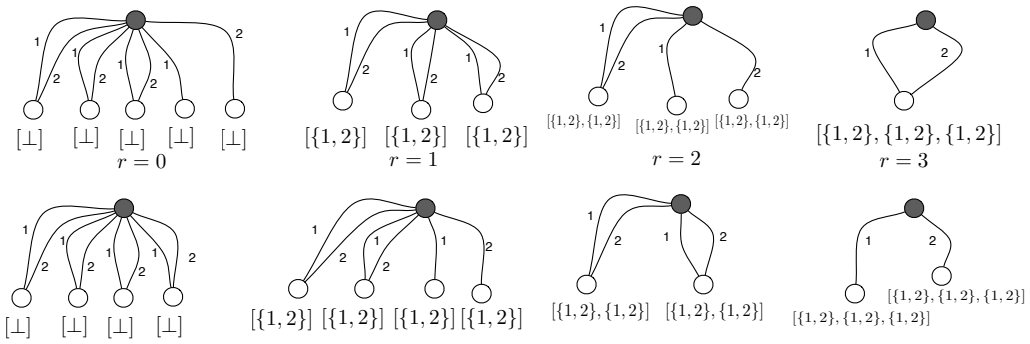


Figure 2.6: Impossibility of non-trivial robust counting with a constant failures rate on $\mathcal{M}(\mathrm{DBL}_2)$.

connected to the leader by two labels, if r is odd we have that $|W| - 2\lfloor \frac{r}{2} \rfloor - 2$ processes are connected with the leader by two labels and the remaining two will be connected to the leader respectively with label 1 and 2, the two processes will fail at the beginning of round $r + 1$.

- The network M' has the following dynamicity. At round r we have $|W'| - 2\lfloor \frac{r+1}{2} \rfloor$ non faulty processes. If r is even, including $r = 0$, we have that $|W'| - 2\lfloor \frac{r+1}{2} \rfloor - 2$ processes are connected with the leader by two labels and the remaining two will be connected to the leader respectively with label 1 and 2, this two processes will fail at the beginning of round $r + 1$. If r is odd we have that all processes are connected by the leader with two labels.

It is easy to verify that for each r we have $|W| - 2\lfloor \frac{r}{2} \rfloor \neq |W'| - 2\lfloor \frac{r+1}{2} \rfloor$, the absolute difference between the number of processes in the two networks is always 1. Eventually we will reach a round r_{end} in which one network has two processes and the other only one. At the beginning of round $r_{end} + 1$ we can force the failure of all non leader processes. We now show that at each round $S(v_l, r)$ on M is equal to $S'(v_l, r)$ on M'. This can be done inductively on r.

- round 0: At the end of round 0 on M the leader receives x states $[\perp]$ from label 1 and x states $[\perp]$ from label 2. The same holds on M'.

- round r: Our hypothesis is that the state of the leader is the same until round $r - 1$. Let us suppose w.l.o.g that r is even, in case r is odd we simply switch the role of two networks. On M we have that all the $|W| - 2\lfloor \frac{r}{2} \rfloor$ processes have state $[\{1, 2\}, \ldots, \{1, 2\}]$. All processes are connected to the leader by both edge labeled $\{1, 2\}$. Therefore the leader receives $|W| - 2\lfloor \frac{r}{2} \rfloor$ states $[\{1, 2\}, \ldots, \{1, 2\}]$, i.e. a list with exactly $r - 1$ elements equal to $\{1, 2\}$, from edge labeled with 1. It receives exactly the same from the edge with label 2. On M' we have $|W'| - 2\lfloor \frac{r+1}{2} \rfloor = |W| + 1$ non faulty processes, of the processes exactly $|W| - 1$ are connected to v_l with both labels, the remaining two processes are connected respectively with label 1 and 2. It is easy to see that v_l receives $|W|$ lists equal to $[\{1, 2\}, \ldots, \{1, 2\}]$ from label 1 and the same from label 2.

Thus if the leader outputs (X, r') and $r' \leq r_{end}$ we have that it would generate a contradiction on M or M'. Therefore the leader can only outputs (X, r') with $r' > r_{end}$ but in this case $X = 1$. □

From the previous proof we get the following corollary.

Corollary 4. *Given a dynamic network with robust dynamic diameter D^*. We have that if $D^* > 3$ any robust counting algorithm $\mathcal{A}$ can only output the trivial values $X \in \{1, 3\}$. This holds even by restricting the number of processes that may fail at each round to be constant with respect to the initial network size.*

Proof. For any integer $x \in \mathbb{N}^+$ we can create two networks $G, G' \in \mathcal{G}(\text{PD})_2$ of size $2x + 2, 2x + 3$. Until round r_{end} the networks follows the same dynamicity of the two networks $M, M' \in \mathcal{M}(\text{DBL}_2)$ of the proof of Th. 2.6. We use the two additional processes as dummy process in V_1 one associated with label 1 and the other 2. It is easy to see that these two networks have $D^* \leq 4$ and that the number of failure is at most 2 independently from x. At beginning of round $r_{end} + 2$ in G, G' both processes in V_1 fail. It follows from the proof of Th. 2.6 that the state of the leader will be exactly the same in the two networks. Therefore the leader can only output the trivial pair (X, r') with $X = 1$ if $r' \geq r_{end} + 2$ or $X = 3$ if $r' = r_{end} + 1$. □

2.4 Conclusive Remarks

We have shown that counting in anonymous dynamic networks could be harder than information dissemination. This difference is actually not present in dynamic networks with IDs and anonymous static networks, counting algorithms for anonymous static networks also works on static networks with IDs. In all these networks there are algorithms showing that the cost of counting is at most of the same order of the cost of information dissemination [44, 59]. When D is constant with respect to V, we proved that counting in anonymous dynamic network requires to pay at least $\Omega(\log |V|)$ rounds.

Let us finally remark that the proposed bound does not hold only for counting but also for other aggregation problem in which the leader has to count the number of nodes in a certain state. Another interesting result is the trade-off lower bound on the accuracy of estimation counting algorithms.

Moreover we have shown that on synchronous static network message duplication leads to a counting time that is function of the network size independently from the network diameter. This is not true for static networks where messages cannot be duplicated. This connection will be considered more in depth in Chapter 3 where we show possibility results for counting in $\mathcal{G}(\text{PD})_h$ networks that can be applied to static anonymous networks as well.

Finally we have proved a lower bound for the **FDP** problem on $G(\text{PD})_2$. The importance of this problem for counting algorithms will be clear in Chapter 3 and 5. Where an algorithm to solve **FDP** will be used as a fundamental building block for a polynomial counting algorithm on $G(\infty\text{-IC})$ and for a terminating counting algorithm on $G(1\text{-IC})$. Let us remark that we mainly interested in networks where $|V|$ is static, but also in this case the solution of **FDP** will be useful to our purposes.

FDP on $G(\text{PD})_2$ networks with IDs is solvable in a number of rounds that is at most 3, but in the anonymous case the time needed to solve the problem is function of the network size, and this was intuitive since counting is the only way to detect if someone is missing when processes are anonymous. However we have shown that in the worst case, $|V_1|, |V_2| = \mathcal{O}(|V|)$, we need $\Omega(|V|)$ rounds that is exponential with respect to the time needed to count, $\Omega(\log |V|)$, and linear with respect to the number of processes.

Open Problems From the results presented in this chapter we can identify the following open problems, ordered by their relevance:

- **Bound on $G(1\text{-IC})$:** An important open problem is to find out if on anonymous $G(1\text{-IC})$ the lower bound for counting is $\Omega(|V|)$, as in $G(1\text{-IC})$ with IDs. We are investigating a generalization of our kernel based technique to dynamic graphs when there is mobility among nodes at various distances from the leader.
- **Bound on the memory needed by nodes to count in $G(\mathbf{PD})_2$**: Let us consider a network in the family of multigraphs $\mathcal{M}(\text{DBL}_k)$ where $k = \mathcal{O}(|V|)$. Our feeling is that in such setting messages and space of $\Omega(\log^2 |V|)$ bits are necessary. This would be in contrast with dynamic networks with IDs or static anonymous networks where messages of $\mathcal{O}(\log |V|)$ bits are enough, see [44] and [59]. Probably both algorithms can be adapted to use $\mathcal{O}(\log|V|)$ space. Our feeling is based on the following facts:
  - Each node has to keep an history. Collectively these histories must involve at least $\Omega(\log |V|)$ rounds; otherwise we can create an ambiguity, i.e. the system of equations leader side has at least two solutions.
  - Let us assume that at each round a node has to memorize in his history information about its degree. This cannot be done by using less than $\mathcal{O}(\log |V|)$ bits without creating ambiguity.

  However to formally prove the bound we have both to show: (1) that nodes cannot agree on some strategy that allows them to "forget" about some rounds in such a way to store locally less than $\mathcal{O}(\log |V|)$ rounds, (2) the degree at each round is a necessary information.

Chapter 3

Counting on $\mathcal{G}(\mathrm{PD})_h$ and $\mathcal{G}(\infty-\mathrm{IC})$

In this chapter we present terminating counting algorithms for networks where the dynamicity introduced by the adversary is limited, keeping intact an inherent structure in the dynamic graph. We first present an optimal counting algorithm, namely OPT, for the family $\mathcal{G}(\mathrm{PD})_2$, this is done in Section 3.1.

In Section 3.2 this algorithm is extended to an optimal algorithm, OPT_h, for networks in $\mathcal{G}(\mathrm{PD})_h$. At the end of Section 3.2 we discuss the implications of this algorithm with respect to static anonymous networks, showing that OPT_h is an optimal algorithm for static networks where messages are duplicated and the duplication is detectable sender side.

In Section 3.3 we present an efficient algorithm, namely $\mathcal{FD}$, that uses the idea behind OPT to detect faulty processes in $\mathcal{G}(\mathrm{PD})_2$, solving the FDP problem introduced in Section 2.3. This algorithm has a gap with respect to the lower bound that is at most $\mathcal{O}(log(|V_2|)^2)$.

We conclude the chapter with Section 3.4 where we show a polynomial counting algorithm, $\mathcal{O}(|V|^5)$, for networks in $\mathcal{G}(\infty-\mathrm{IC})$. Let us remark that at the best of our knowledge this is the first deterministic and polynomial terminating algorithm for anonymous $\mathcal{G}(\infty-\mathrm{IC})$ networks.

3.1 OPT algorithm for $\mathcal{G}(\mathrm{PD})_2$

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

No-leader node behavior Starting from round 1, a node broadcasts its distance from the leader (i.e., 1 or 2) and each node v in V_2 builds its *degree history* $v.H(r)$

with $r \geq 0$ where $v.H(1)$ represents the number of neighbors of v belonging to V_1 at round 1 and $v.H(0) = \perp$. 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 node in V_2 broadcasts $v.H(r)$. These histories are collected by each node $v' \in V_1$ and sent to the leader at the beginning of round $r+1$.

Let us define as $Num(H'(r)) : |\{v \in V_2 | v.H(r) == H'(r)\}|$, i.e. the number of nodes in V_2 that at round r have degree history $H(r)$.

Leader behavior Starting from the beginning of round $r \geq 2$ the leader receives degree histories from each node in V_1. The leader merges histories in a multiset denoted $v_l.M(r)$. Let us remark that $v_l.M(r)$ can contain a 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 has 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 $n_{[\perp]} =?$. It is important to remark that m values are directly computed at each round r 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 that is a child of the node with label $[\perp, x_0, \ldots, x_{r-2}]$ otherwise it does nothing, see lines 7-12 Figure 3.2. It is straightforward to see that the following equations holds:

$$\begin{cases} m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1}]} = \sum_{i=1}^{|V_1|} i \cdot Num([\perp, x_0, \ldots, x_{r-2}, x_{r-1}, i]) \\ Num([\perp, x_0, \ldots, x_{r-2}, x_{r-1}]) = \sum_{i=1}^{|V_1|} Num([\perp, x_0, \ldots, x_{r-2}, x_{r-1}, i]) \end{cases} \tag{3.1}$$

Where $i \cdot Num([\perp, x_0, \ldots, x_{r-2}, x_{r-1}, i])$ 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 $[\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r]$ and 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 $n_{[\perp,x_0,\ldots,x_{r-1},x_r,j]} = Num([\perp, x_0, \ldots, x_{r-1}, x_r, j])$ using $m_{[\perp,x_0,\ldots,x_{r-2},x_{r-1}],x_r} = \sum_{i=1}^{|V_1|} i \cdot Num([\perp, x_0, \ldots, x_{r-2}, x_{r-1}, x_r, i])$ See lines 19-22 Figure 3.2

Due to the fact that the number of processes is finite, there will be a non-leaf node in T that will have only one child (a leaf). Then 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 $n_{[\perp]}$.

When the leader knows the values n for each of the children of a non leaf-node t, it sum the children values and set the n value of the t, see the second equation of Eq. 3.1, see line 24 Figure 3.2. The count terminates when the leader knows $n_{[\perp]}$.

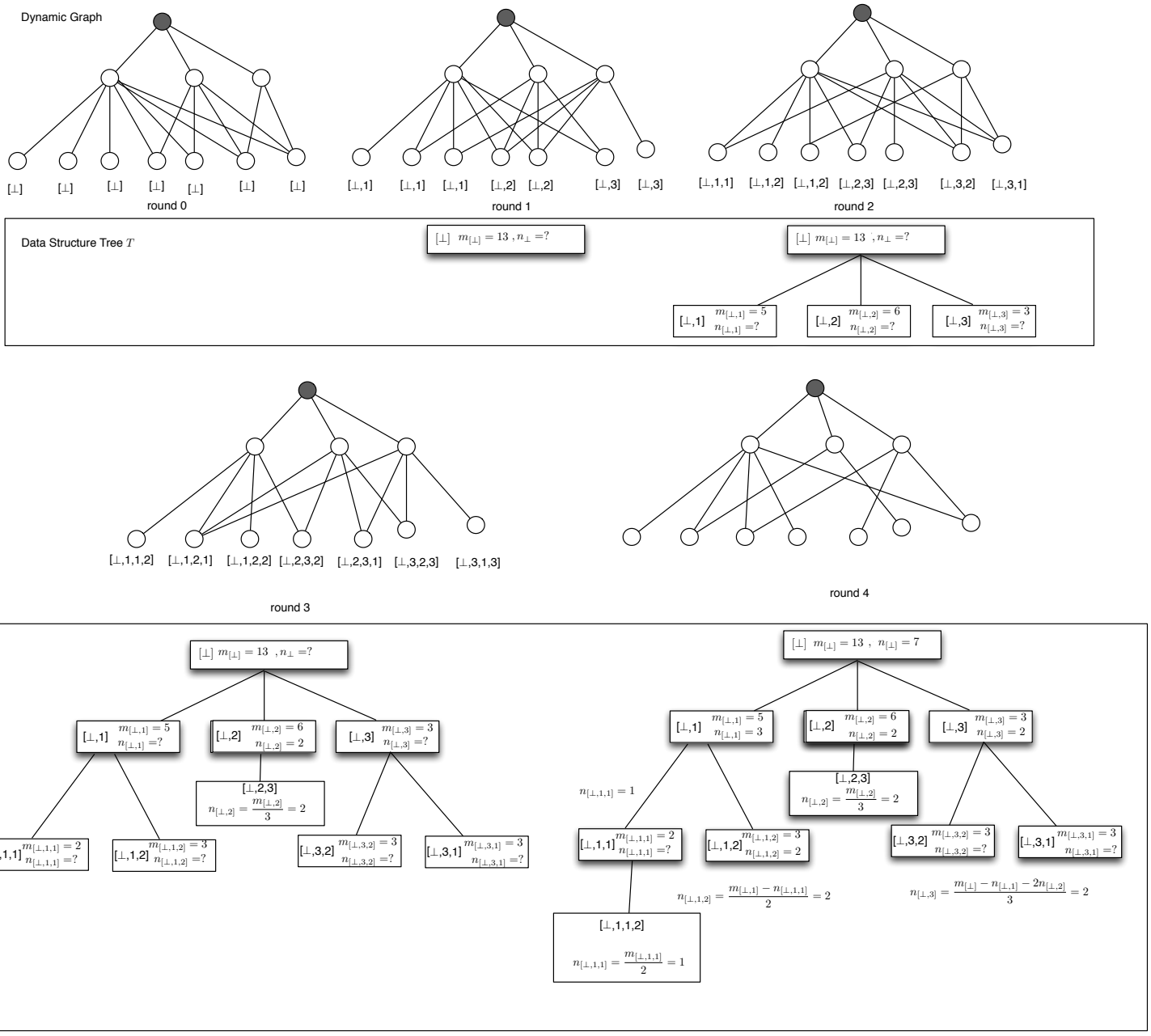


Figure 3.1: Example run for Algorithm for $\mathcal{G}(\text{PD})_2$

```
1: T = ⊥
2:
3: function BUILDT(MultiSet M_r)
4:     if r == 2 then
5:         Assert(∃[⊥] ∈ M_1)
6:         T.setRoot([⊥] :< m_[⊥], ⊥ >)
7:     for all [x_0, ..., x_{r-2}, x_{r-1}] ∈ M_r do
8:         v_l creates a node [x_0, ..., x_{r-2}, x_r − 1] :< m_[x_0,...,x_{r-1}], n_|[x_0,...,x_{r-1}]| : ⊥ >
9:         t : T.FINDNODE([x_0, ..., x_{r-2}])
10:        if n_t ≠?||t = null then
11:            continue
12:        T.ADDCHILD([x_0, ..., x_{r-2}, x_r − 1])
13:    COMPUTE(T)
14:    if T.root.n_[⊥] ≠ ⊥ then
15:        output(T.root.n_[⊥])
16:
17: function COMPUTE(Tree T)
18:    for all t ∈ T starting from the level of the leaves until the root do
19:        C : T.findChildren(t)
20:        X ⊆ C such that [x_0, ..., x_k] ∈ X iff n_[x_0,...,x_k] ≠ ⊥
21:        if ∃!c : [y_0, ..., y_k] ∈ C \ X then
22:            n_c : (m_t − Σ_{∀[x_0,...,x_k]∈X} (x_k · (n_[x_0,...,x_k]))) / y_k
23:        if X = C then
24:            n_t : Σ_{∀[x_0,...,x_k]∈X} n_[x_0,...,x_k]
```

Figure 3.2: Creation of T - Leader Code , this algorithm is used by both OPT and OPT_h

Correctness proof

Lemma 11. *Let 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 $n_{[A,x']|} \geq 1$ since there must be at least one sending process for each of this two different degree-history. Since we have $Num([A]) = \sum_{j=1}^{k} Num([A, j])$ then $Num([A, x]) < Num([A]) - 1$. Iterating this reasoning we have that when the height of the subtree rooted in $[A]$ is greater than $Num([A]) - 1$, then each leaf has no sibling: when there is a single processes sending a certain degree history H in the next round there will be only one degree history with H as suffix. Thus after at most $Num([A])$ rounds we may apply the counting rule for any leaf of the subtree rooted in $[A]$.

Now we prove by induction that: for each node $v \in T$ if $n_v \neq ?$, then $n_v = Num(v)$.

- Base case, leaf without siblings: Let $v_1 : [x_0, \ldots, x_{r+1}]$ a leaf without sibling and $v_0 : [x_0, \ldots, x_r]$, v_l will sets $n_{v_0} = n_{v_1} = \frac{m_{v_0}}{x_{r+1}}$ it is straight from Eq 3.1 that $n_{v_0} = n_{v_1}$ which is equal to $Num([x_0, \ldots, x_r])$;

- Inductive case: Let us consider $v_0 : [x_0, \ldots, x_r]$ and the set of its children C_{v_0}, with $|C_{v_0}| > 1$, defined $X_{v_0} : \{x \in C_{v_0} | n_x \neq ?\}$ if $\exists! v_1 : [x_0, \ldots, x_{r+1}] \in C_{v_0} \setminus X_{v_0}$ the leader sets

$$n_{v_1} : \frac{m_{v_0} - \sum_{\forall [x_0,\ldots,x_k] \in X}((x_k) \cdot ([x_0, \ldots, x_k].n_{[x_0,\ldots,x_k]}))}{y_k}$$

  and by inductive hypothesis we have $\forall x \in X_{v_1}$, n_x is equal to $Num(x)$. Due to Eq. 3.1, we also have 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 leaf of $[\perp]$ have no siblings, thus the counting rule will be applied recursively until the value $n_{[\perp]}$ is set to $|V_2|$, at round 1 all processes in V_2 sends $\perp$. □

Theorem 6. *Let G be a dynamic graph of size $|V|$ belonging to $\mathcal{G}(PD)_2$. There exists an asymptotically optimal counting algorithm for G that terminates in $\lceil log_2(|V|) \rceil + 3$ rounds.*

Proof. Let consider the algorithm OPT. The latter counts the node in V_2, since the number of nodes in V_1 is immediately known by v_l at round 0, thus let us suppose that we are in the worst case that is $|V_2| = O(n)$. Let us consider the tree T, given a node $[A]$ the maximum height of the subtree rooted in $[A]$ is a function $h_{max}(Num([A]))$. We have that h_{max} is non decreasing, that is $h_{max}(Num([A])-1) \leq h_{max}(Num([A]))$: let us consider the worst scheduling that the adversary may use with $Num([A]) - 1$ nodes in order to obtain the maximum height, it easy to see that the same scheduling can be created with $Num([A])$ nodes, the adversary will simply force two nodes to follow the behavior of a single node in the hold schedule. Let us restrict to the case when $[A]$ has only two children: $[A, x], [A, x']$, for the counting rule $h_{max}(Num([A])) = min(h_{max}(Num([A, x])), h_{max}(Num([A, x']))) + 1$ that for the second equation of Eq. 3.1 becomes, considering $\delta \in [0, \frac{n_{[A]}}{2}]$, $h_{max}(n_{[A]}) = 1 + min(h_{max}(\frac{n_{[A]}}{2} - \delta), h_{max}(\frac{n_{[A]}}{2} + \delta)) \leq 1 + min(h_{max}(\frac{n_{[A]}}{2}), h_{max}(\frac{n_{[A]}}{2}))$ thus is clear that the optimal height can be reached by having $Num([A, x]) = Num([A, x']) = \frac{Num([A])}{2}$. It is easy to see that the case when $[A]$ has more than two children, gives a maximum height that cannot be greater than the case examined. Iterating this reasoning we have that the worst case is that T is a balanced tree with degree at most 2 for each non leaf node and with exactly n leaf. The height of this tree is $\lceil log_2(n) \rceil$. Each level of T corresponds to one round of OPT, this complete the proof. Due to theorem 2, OPT is asymptotically optimal. □

Discussion An interesting point is that if OPT terminates at early stages we have that many nodes in V_2 may share the same state, i.e. if the algorithm terminates at round 1 all nodes have the same state. But the more rounds OPT runs, the more nodes have to assume a different history, ending up with a different state for each node. This trade-off between the speed of the counting and the different states that processes may assume is up to our adversary. This property will be investigated more in depth at the end of the next section. Let us remark that this algorithm is not robust.

3.2 OPT_h: Extending OPT to count in $\mathcal{G}(\mathbf{PD})_h$

As in OPT, OPT_h begins with a *get_distance* phase over $\mathcal{G}(\text{PD})_h$ where each node 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 node computes its distance from v_l, in the $h+1$ successive rounds (convergecast step) the leader computes the maximum distance h.

```
1: M(0) = [⊥]
2: H(0) = [⊥]
3: distance = d                          ▷ The distance is obtained in the get_distance phase
4:
5: procedure SENDING_PHASE
6:     send(Message :< distance, M(r), H(r) >)
7:
8: procedure RCV_PHASE(MultiSet MS)
9:     H(r+1) = H(r).append(count_distance_neighbors(MS, distance − 1))
10:    M(r+1) = M(r).append(get_messages_from_distance(MS, distance + 1))
```

Figure 3.3: OPT_h algorithm for $\mathcal{G}(\text{PD})_h$: algorithm run by a non-leader node

Non-Leader node behavior in OPT_h. The code of a non-leader node in OPT_h is reported in Figure 3.3, the function *count_distance_neighbors* returns the number of messages in MS generated by nodes at distance $distance-1$, the function *get_messages_from_distance* returns only messages generated by nodes at distance $distance+1$. If there is no such message the function returns $\perp$. As in OPT, a node v updates its degree history $v.H(r)$ by counting the number of nodes 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.

Leader node behavior in OPT_h. From an high level point of view the algorithm works as follow: the leader first computes the number of nodes in V_1, then it executes OPT to count the nodes in V_2, this count will be completed by round $(2h+1)+(3+log(|V_2|))$ (see Theorem 3.2). At this point, the leader simulates an execution of OPT counting nodes 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 nodes 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 nodes 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 16-26 of

```
1: distance_count[]
2: procedure SENDING_PHASE
3:     send(< leader >)
4:
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 = ∑_{∀j|distance_count[j]≠⊥} distance_count[j]
12:            output(count)
13:        MS =BUILDLASTNEXTDISTANCESET (MS)
14:        if MS = ⊥ then
15:            break
16:        distance_count[i] = |MS|
17:        i++
18:
19: function BUILDLASTNEXTDISTANCESET(MS)
20:     MS_last = ⊥
21:     if TREE(RECENT(MS, 0)) ≠ ⊥ then
22:         r_last = r'|TREE(RECENT(MS, r')) ≠ ⊥ ∧ ∄r'' > r'|TREE(RECENT(MS, r'')) ≠ ⊥
23:         MS_last = TREE(RECENT(MS, r_last))
24:     return MS_last
```

Figure 3.4: OPT_h algorithm for $\mathcal{G}(\mathrm{PD})_h$: algorithm run by the leader

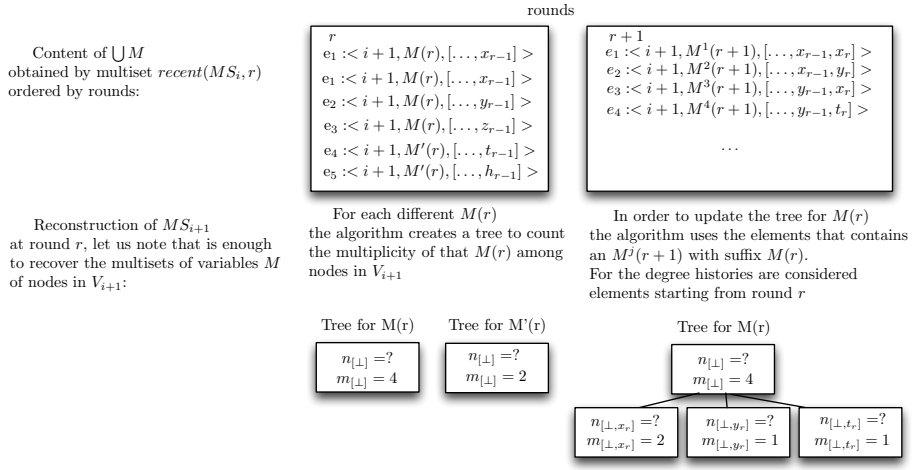


Figure 3.5: Reconstruction of M variables sent by the set of nodes V_{i+1}

Figure 3.4. The leader uses the variable i to store the maximum distance at which nodes 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 3.4). The leader now uses the information in the last reconstructed multiset, denoted MS_i, to obtain the multiset M_{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 3.5.

The reconstruction is executed by the function BUILDLASTNEXTDISTANCESET.

This function calls TREE(RECENT(MS_i, r')). The function RECENT(MS_i, r') returns the set of messages MS' where the elements in MS_i received before round r' are removed. Therefore the call TREE(RECENT(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 22). 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 BUILDLASTNEXTDISTANCESET returns either MS_{i+1} sent at round r_{last} or $\perp$.

At line 14 of Figure 3.4 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 16-17, Figure 3.4).

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 a precise count.*

Proof. We consider a generic run after $2h + 1$ rounds, so that each node has set *my_distance* $\neq -1$. For easy of explanation we consider that the algorithm starts at round 0 and that all nodes know their distance. Let us consider the nodes in V_h, at round $r = 0$. They start to send their degree history to nodes in V_{h-1}, in the worst case at round $r_h = \log_2(|V_h|) + 3$ the union of variables of nodes in V_{h-1} allows to compute the multiset MS_h sent at round $r = 0$, see Th. 6 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 nodes 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 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 reconstruction loop, Lines 19-24 of Figure 3.4, on the multiset MS_1, and it will obtain the multiset MS_2 sent by nodes 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 leader terminates the reconstruction and the count. □

Complexity analysis From Section 2.1, 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 nodes at distance x could be connected to only two nodes at distance $x - 1$. The complexity of OPT_h is $5h + 1 + \sum_{i=2}^{h} \log_2(|V_i|)$.

If h has to be 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. Let consider the case when 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 [1].

We have from simple calculus that the maximum $\frac{|V|}{x}^x$ 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.

Negligible impact of the adversary in case of continuos counting The proposed algorithm has a nice property that could be useful in many realistic setting, we illustrate this with an example. Let us suppose to have a network of sensors deployed around a base station (BS). Nodes are partitioned in levels. At each level is assigned an unique ID, that is shared among all nodes in that level, this sharing could be necessary because of cost or time deployment constraint. Nodes are concentrically disposed around the BS in such a way that nodes at level l can always communicate with nodes at level $l - 1$, but due to constraint of the environment the communication among levels is not stable therefore a node at level l could communicated at each different round with a different set of nodes at level $l - 1$. The measurements done by sensors change periodically and these have to be continuously obtained by the BS. By continuously starting new instances of the algorithm OPT_h we have an asymptotically optimal solution. Moreover we have that, considering the delay in counting, the term that is function of the network size will be payed only once in the whole computation. The impact of the worst case adversary in the long run will be negligible. The reason behind this is simple, in order to delay the OPT algorithm the adversary has to create new histories, these histories if memorized by nodes could be used to speed up successive computations. When the number of histories for nodes at level i is $\mathcal{O}(|V_i|)$ we have that the instance of OPT, that counts node on level i, terminates in $\mathcal{O}(1)$ rounds. So let us consider the cost that we have to pay in the whole computation, and not the single counting instance, for level i. The adversary could force us to pay at most $\mathcal{O}(|V_i|))$ rounds, each round it delays the count there is at least a new history in V_i. Therefore on all instances of counting we can pay at most a delay of $\mathcal{O}(|V|)$ rounds.

Consequences for static anonymous networks

Let us consider synchronous static anonymous networks where the broadcast primitive may deliver to neighbors duplicated messages, with this duplication governed by a worst case adversary but bounded by a value unknown to nodes. If duplication is detectable receiver side, duplicated messages can be ignored and any algorithm for the setting where messages cannot be duplicated will work, for a counting algorithm that without duplications counts in $\mathcal{O}(D)$ rounds see [59]. Is thus interesting to examine three complementary settings:

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

- (S1) Duplication cannot be detected.
- (S2) Duplication can be detected sender side: the sender obtains the number of duplicated messages delivered to each set of neighbors with the same state. More precisely: we define as $s(v_j)$ the state of node v_j, i.e. the memory content of v_j. We consider node v and its neighbors $N(v) : \{v_0, \ldots, v_l\}$, with states $\{s_0, \ldots, s_k\}$ with $k \leq l$. When v broadcasts m to $N(v)$ at the end of the broadcast it receives a set $Dup : \{< m_0, s_0 >, \ldots, < m_k, s_k >\}$ where m_i is the number of messages delivered to nodes with state s_i.
- (S3) Duplication can be detected sender side: the sender obtains the number of duplicated messages delivered. We consider node v and its neighbors $N(v) : \{v_0, \ldots, v_l\}$, with states $\{s_0, \ldots, s_k\}$ with $k \leq l$. When v broadcasts m to $N(v)$ at the end of the broadcast it receives an integer I_{dup} that corresponds to the number of messages delivered to its neighbors.

Let us recall that, even in static anonymous network, the leader is necessary to count [59], thus we assume the existence of v_l.

In setting (S1) counting cannot be solved:

Lemma 13. *Let us consider an anonymous networks with broadcast communication where a message m sent by a node v could be received a bounded number of times by one of its neighbors w. In this setting counting is not possible.*

Proof. Let us first consider the left network of Figure 3.6 and a run $\mathcal{R}$ where at each round r a message from v is always delivered two times to v_l and messages from v_l are never duplicated. At each round r of run $\mathcal{R}$ the state of v is the same of the state of nodes a, b in the right network, when we consider a run $\mathcal{R}'$ where messages are never duplicated. Thus v_l will receive the same set of messages in the two runs $\mathcal{R}, \mathcal{R}'$ persisting in a state that is reachable in a network of size 1 and a network of size 2. This indistinguishability can be easily extended for any network size.

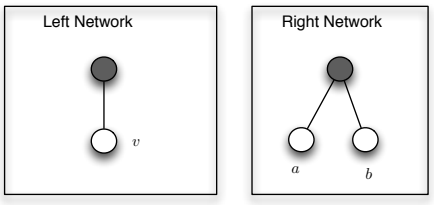


Figure 3.6: Two static networks that are indistinguishable if duplications are not detectable

□

In the previous proof we considered that the same message is delivered multiple times. Usually the message duplication is introduced to model an underling broadcast algorithm that sends a message multiple times in order to overcome message loss. A realistic setting is the following: node v knows that if it sends a message x times at round r at least one message is delivered to its neighbors. In this case v could number each broadcast by using a local sequential counter.

Thus v broadcasts a set $B : \{< m, 0 >, < m, 1 >, \ldots, < m, x >\}$ to its neighbors $N : \{v_0, \ldots, v_l\}$. The adversary for each $v_j \in N$ picks a subset $S_{v_j} \subseteq M$ with $|S_{v_j}| \geq 1$ and delivers S_{v_j} to v_j. Unfortunately even in this stronger setting, where two duplicated message are always different, counting is still impossible, the proof is essentially the same of Lemma 13, in the left network at each round the leader receives $\{< m, 0 >, < m, 1 >\}$ from v, in the right network the leader receives $\{< m, 0 >\}$ from a and $\{< m, 1 >\}$ from b.

Let us consider setting (S2), we have that a lower bound for counting is $h + max_{\forall V_i}(log_3(2|V_i| + 1))$ see Chapter 2. In setting (S2) the algorithm $\mathcal{G}(\text{PD})_h$ still works, this is due to the fact that the algorithm only uses information about the *outdegree/indegree* of nodes among the various distances from the leader. Information that can be obtained by the set Dup and by counting the multiplicity of received messages.

Therefore we have an optimal algorithm for static networks with communication modeled as (S2). Let us remark that the majority of works on computation for generic static anonymous networks do not consider a broadcast primitive that may duplicate messages thus introducing a dynamicity of communication network among rounds, see [18, 24, 32, 34, 68], and recently [22, 23]. To the best of our knowledge we are not aware of any optimal counting algorithms designed for this setting.

It is interesting to consider a setting (S3) where the sending node obtains only the cardinality of duplicated messages without knowing how many messages have reached a certain set of neighbors.

In (S3) OPT_h does not work. Let us consider a star graph with v_l at the centre and where nodes in V_1 could be connected among each other. In this kind of graph we cannot use the techniques used in Chapter 2 to obtain a better lower bound. The reason is that a nodes cannot know in advance how many messages has been delivered to neighbors in V_1 or to v_l. Therefore the matrix M_r seen by the leader does not have a peculiar structure that allows to characterize the kernel space. Counting is possible in (S3), this will be clear by seeing the techniques used in Chapter 4. We can adapt the algorithm $\mathcal{A}_{\mathcal{O}^{FOE}}$ presented in Section 4.1.

The idea is that the leader can compute the diameter D, this is done by a trivial flooding and convergecast algorithm. When D is known the leader knows that at round r it has received all messages sent at round $r' < r - D$. The purpose of nodes is to build an eventual $\mathcal{O}^{FOE}$, see definition in Section 1.1, that gives to node a bound on the number of duplications. This is done by disseminating the maximum value I_{dup} seen by each node. The leader starts with a guess $max_{I_{dup}} = 1$, each time this guess changes it restarts the counting. Each instance of the algorithm is associated with an increasing epoch number assigned by v_l. Eventually, since duplications are bounded, the value $max_{I_{dup}}$ will never change. Therefore there will be a final instance of the counting where all nodes have access to a correct $\mathcal{O}^{FOE}$. For the termination condition the leader terminates D rounds after the usual termination of $\mathcal{A}_{\mathcal{O}^{FOE}}$, in this way it knows that the current instance of $\mathcal{A}_{\mathcal{O}^{FOE}}$ has been executed with a correct $\mathcal{O}^{FOE}$. Unfortunately $\mathcal{A}_{\mathcal{O}^{FOE}}$ is a time exponential algorithm. As final note, the algorithm presented in Chapter 5 cannot be adapted to work in (S3).

3.3 An FD algorithm for $\mathcal{G}(\mathbf{PD})_2$

In this section we show an algorithm to solve the **FDP** problem introduced in the previous Chapter. It is easy to see that a fault in V_1 is immediately detected by the leader, therefore we assume that faults are only localized in the set V_2. We consider a variation of the counting algorithm for $\mathcal{G}(\mathrm{PD})_2$, the only different part is the structure of the tree that the leader constructs using the received messages. In this case the leader builds a tree $T_{\mathcal{FD}}$, a node in $v \in T_{\mathcal{FD}}$ is associated to the usual label and two variables $< m_v, w_v >$. When the leader receives a new degree history it creates a node in $T_{\mathcal{FD}}$ using the same rule of T. Moreover If there exist a leaf $v : [x_0, \ldots, x_{r-2}] \in T_{\mathcal{FD}}$ and at round $r-1$ the leader does not receive any message that contains a degree history $[x_0, \ldots, x_{r-2}, x_{r-1}]$ then the leader adds to v a dummy child $v' : [x_0, \ldots, x_{r-2}]$ with $< m_{v'} = -1, w_{v'} = 0.5 >$.

The leader executes a set of actions on the tree, according to the rule explained in the following. The first two rules are used by the leader to mark a node v of $T_{\mathcal{FD}}$ as verified, that is $w_v \neq ?$. When v is verified the leader will not add any child node to v, thus the relative degree histories will be ignored. The last rule is used to check the consistency of the tree and to verify if there has been a fault.

- **Verification Rule 1**: Let us consider a node $v_{\Delta+1} : [x_0, \ldots, x_{k+\Delta+1}] \in T_{\mathcal{FD}}$ with $w_v = ?$ and such that $\exists v_1 : [x_0, \ldots, x_k] \in T_{\mathcal{FD}}$ ancestor of v: if (1) denoted with $p_{v_1, v_{\Delta+1}}$ the path between $v_1, v_{\Delta+1}$ we have $\forall v \in p_{v_1, v_{\Delta+1}}$ that $w_v = ?$ and that defined as C_v the set of children of v and defined the set $X_v : \{x | x \in C_v \wedge w_x \neq ?\}$, $|C_v| - |X_v| = 1$ and (2) we have $|p_{v_1, v_{\Delta+1}}| > 2|V_1| \cdot log(2\lceil \frac{m_{v_1}}{|V_1|} \rceil + 1) + 1$ then defined $v_\Delta : [x_0, \ldots, x_{k+\Delta}] \in p_{v_1, v_{\Delta+1}}$, the father of $v_{\Delta+1}$, the leader computes $t : \frac{m_{v_\Delta} - \sum_{\forall [x_0, \ldots, b] \in X_{v_\Delta}} b \cdot w_{[x0, \ldots, b]}}{x_{k+\Delta+1}}$ and sets $w_{v_\Delta} : t + \sum_{\forall [x_0, \ldots, b] \in X_{v_\Delta}} b \cdot w_{[x0, \ldots, b]}$.
- **Verification Rule 2**: Let us consider a node $v : [x_0, \ldots, x_k] \in T_{\mathcal{FD}}$ with $w_v = ?$ and such that $|C_v| > 0$ and $\forall v' \in C_v \ w_{v'} \neq ?$, then the leader sets $w_v = \sum_{\forall v'} w_{v'}$.
- **Fault Detection Rule 1**: If $\exists v \in T_{\mathcal{FD}}$ with $w_v \neq ?$ such that $w_v \notin \mathbb{N}$ or $m_v \neq \sum_{\forall v' : [x_0, \ldots, x_k] \in C_v} x_k w_{v'}$ or $w_v \neq \sum_{\forall v' : [x_0, \ldots, x_k] \in C_v} w_{v'}$ then v_l detects a failure.

The algorithm terminates when $w_{[\perp]} \neq ?$ or when the leader detects a failure. In the next lemmata we show that the $\mathcal{FD}$ algorithm is correct, and that its cost in rounds is $\mathcal{O}(|V_1| log^2(|V_2|))$.

Lemma 14. *If the $\mathcal{FD}$ algorithm terminates, i.e. $w_{[\perp]} \neq ?$, and the leader does not detect any failure then the number of failures at round $r = 1$ is 0 and $w_{[\perp]} = Num([\perp])$.*

Proof. The proof is done by induction:

- **Base Case:** In the base case we consider a node $v_1 : [x_0, \ldots, x_{r-1}]$ such that the subtree $T^{v_1}_{\mathcal{FD}}$ rooted in v_1 that has only one leaf, and where the Verification Rule 1 has been applied for the father, $v_\Delta : [x_0, \ldots, x_{r+\Delta}]$, of the leaf; this implies that Verification Rule 2 has been applied for all nodes in $T^{v_1}_{\mathcal{FD}}$ but

the leaf, setting all the variables w_*. We show that if the Fault Detection Rule 2 is not triggered on $T^{v_1}_{\mathcal{FD}}$ then there has been no failure at round $r+1$ among processes with degree history $[x_0,\dots,x_{r-1}]$ and moreover we have $Num([x_0,\dots,x_{r-1}]) = w_{v_1}$. This base case has to exist since processes are finite and we have $Num([x_0,\dots,x_k]) \geq \sum_{i=1}^{|V_1|} Num([x_0,\dots,x_k,i])$, thus the tree will eventually contains a subtree without branches, where the Verification Rule 1 will be applied. In the following we only refers to nodes in $T^{v_1}_{\mathcal{FD}}$ that are between v_1 and v_Δ, and the respective processes in the dynamic graph, since the degree history until round $r-1$ is in common we use $[x_{r-1},\dots,x_{r+j+1}]$ as label for v_j.

Let us first notice that all w_* has to be set to the same value w otherwise the FD rule will be triggered.

The proof of the base case is by contradiction. Let us assume, that there as been a single failure among processes with degree history $[x_{r-1}]$ and that $\forall (v_j, v_{j+1}) \in p_{v_1,v_\Delta}\ m_{v_j} = x_{r+j} w_{v_{j+1}}$ otherwise the Fault Detection rule will be trigger, but this condition means that $w_{v_{j+1}} = \frac{m_{v_j}}{x_{r+j}}$ and since the Verification Rule 2 has set $w_{v_j} = w_{v_{j+1}}$ we must have $w_{v_j} = \frac{m_{v_j}}{x_{r+j}}$. Thus we have $w_{v_1} = \frac{m_{v_1}}{x_r}$, that is an upper bound on $Num([x_{r-1}])$, since we have $Num([x_{r-1},x_r]) = Num([x_{r-1}]) - 1$ and $m_{v_1} = Num([x_{r-1},x_r])x_r + f_1$ where $f_1 \geq 1$ is the degree of the faulty process, let us notice that $\frac{f_1}{x_r} \in \mathbb{N}$ otherwise $w_{v_1} \notin \mathbb{N}$, thus f_1 has to be a multiple of x_r. Now suppose that there are no failures among processes with degree history $[x_{r-1},x_r]$ thus $Num([x_{r-1},x_r]) = Num([x_{r-1},x_r,x_{r+1}])$, and we must have $w_{v_2} = \frac{m_{v_2}}{x_{r+1}} = w_{v_1}$ thus we have, since $m_{v_2} = Num([x_{r-1},x_r,x_{r+1}])x_{r+1}$, $w_2 = w_1 \rightarrow Num([x_{r-1},x_r,x_{r+1}]) = Num([x_{r-1},x_r]) + \frac{f_1}{x_r}$ that is an absurd since the no failure condition implies $Num([x_{r-1},x_r]) = Num([x_{r-1},x_r,x_{r+1}])$. Thus we must have at least one failure in $Num([x_{r-1},x_r])$, i.e. $Num([x_{r-1},x_r,x_{r+1}]) \leq Num([x_{r-1},x_r]) - 1$, let f_2 be the degree of the failed process. The condition on w_{v_2} becomes $Num([x_{r-1},x_r,x_{r+1}]) + \frac{f_2}{x_{r+1}} = Num([x_{r-1},x_r]) + \frac{f_1}{x_r}$, thus $-1 + \frac{f_2}{x_{r+1}} \geq \frac{f_1}{x_r}$ that is $f_2 \geq x_{r+1}(1 + \frac{f_1}{x_r})$ since $\frac{f_1}{x_r} \in \mathbb{N}$ we have $f_2 \geq x_{r+1} + 1 \geq 2$, thus the faulty process has to have degree at least 2. Now let us examine the condition $w_{v_3} = w_{v_2}$, it easy to see that we may reach the same contradiction of before if we assume that no node with history $Num([x_{r-1},x_r,x_{r+1}])$ may fail, thus we must have $Num([x_{r-1},x_r,x_{r+1},x_{r+2}]) \leq Num([x_{r-1},x_r,x_{r+1}]) - 1$, and we will have $Num([x_{r-1},x_r,x_{r+1},x_{r+2}]) + \frac{f_3}{x_{r+2}} = Num([x_{r-1},x_r,x_{r+1}]) + \frac{f_2}{x_{r+1}}$ thus $f_3 \geq x_{r+2}(1 + \frac{f_2}{x_{r+1}}) \geq x_{r+2}3$ since $\frac{f_3}{x_{r+2}} \in \mathbb{N}$ we must have $f_3 \geq 3$. Iterating this we have that the cumulative degree of faulty nodes at round $r + \delta_f$ has to be greater or equal to δ_f, but the degree of a single node is bounded by $|V_1|$, thus we have that the minimum number of faulty nodes at round $r + \delta_f$ has to be $\lceil \frac{\delta_f}{|V_1|} \rceil$, since $|[x_{r-1}]|$ is bounded by n_{v_1} we have that defined $\delta_{f_{max}}$ such $r + \delta_{f_{max}}$ is the maximum round at which $m_{v_{\delta_{f_{max}}}} = x_{\delta_{f_{max}}+1} w_{\delta_{f_{max}}+1}$ and must hold $\delta_{f_{max}} \leq T = \sum_{i=1}^{\frac{m_{v_1}}{|V_1|}} \frac{|V_1|}{i}$, since after $r + T$ we have no more active processes with $[x_{r-1}]$ in their history. This means the leader will add a dummy child, but this will trigger the Fault Detection Rule, thus we must have $\delta_{f_{max}} > \Delta$. By

construction we have $\Delta = 2|V_1| \cdot log(2\lceil \frac{m_{v_1}}{|V_1|} \rceil + 1) + 1 > T \geq \delta_{f_{max}}$, that is a contradiction. The initial assumption of a single failure is not restrictive since f_1 could be seen as the sum of the degrees of the set of failed processes at round $r+1$, thus the previous contradiction complete the proof. Since there is no failure we have $Num([x_{r-1}]) = Num([x_{r-1}, x_r])$ that implies $w_{v_1} = Num([x_{r-1}])$, otherwise we can reach a contradiction.

- **Inductive Step:** The inductive step is divided in two cases, our inductive hypothesis is that for each node $v : [l]$ with $w_v \neq ?$ and such that v is examined in conditions of Ver. Rule 1 or Ver. Rule 2, we have that $w_v = Num([l])$ and that there is no fault among processes with degree history $[l, x]$.

  - **Case 1** In this case we examine the application of Ver. Rule 1 on a subtree $T^{v_1}_{\mathcal{FD}}$ with two or more leaves, and the successive application of the Ver. Rule 2.
    In this case we consider a node $v_1 : [x_0, \ldots, x_{r-1}]$ such that in the subtree $T^{v_1}_{\mathcal{FD}}$ rooted in v_1 there exists a node v' and where the Verification Rule 1 has been applied for the father, $v_\Delta : [x_0, \ldots, x_{r+\Delta}]$, of v'; this implies that Verification Rule 2 has been applied for all nodes in $T^{v_1}_{\mathcal{FD}}$ but the leaf. For inductive hypothesis we assume that given $v \in p_{v_1,v_\Delta}$, $\forall v' : [h] \in X_v$ we have $w_{v'} = Num([h])$. We show that if the Fault Detection Rule 2 is not triggered on $T^{v_1}_{\mathcal{FD}}$ then there has been no failure at round $r+1$ among processes with degree history $[x_0, \ldots, x_{r-1}]$ and moreover we have $w_{v_1} = Num([x_0, \ldots, x_{r-1}])$. Let us assume, that there as been a single failure among processes with degree history $[x_{r-1}]$ and that $\forall (v_j, v_{j+1}) \in p_{v_1,v_\Delta}$ $m_{v_j} = \sum_{\forall v' : [x_0,\ldots,x_k] \in C_v} x_k w_{v'}$ otherwise the Fault Detection rule will be triggered. Thus we have

$$m_{v_1} = \sum_{x \in \{1,\ldots,|V_1|\} \setminus \{x_r\}} xNum([x_{r-1}, x]) + x_r Num([x_{r-1}, x_r]) + f_1 =$$

$$= \sum_{x \in \{1,\ldots,|V_1|\} \setminus \{x_r\}} x w_{[x_{r-1},x]} + x_r w_{[x_{r-1},x_r]}$$

    , since for inductive hypothesis we have $w_{[x_{r-1},x]} = Num([x_{r-1}, x])$ this implies that we must have $w_{[x_{r-1},x_r]} = Num([x_{r-1}, x_r]) + \frac{f_1}{x_r}$ and that $w_{[x_{r-1},x_r]} \in \mathbb{N}^+$, now let us assume that there are no faults among processes with degree history $[x_{r-1}, x_r]$, thus $Num([x_{r-1}, x_r]) = \sum_{\forall x} Num([x_{r-1}, x_r, x]) + Num([x_{r-1}, x_r, x_{r+1}])$ for inductive hypothesis and since we have $w_{[x_{r-1},x_r]} = \sum_{\forall x} w_{[x_{r-1},x_r,x]}$ otherwise the FD rule would have been triggered, we must have $Num([x_{r-1}, x_r]) = w_{[x_{r-1},x_r]}$ that leads to a contradiction $Num([x_{r-1}, x_r]) = w_{[x_{r-1},x_r]} = Num([x_{r-1}, x_r]) + \frac{f_1}{x_r}$. Thus we must have at least one fault in the set $[x_{r-1}, x_r]$, let us indicate with f_2 the degree of the faulty processes. We have $m_{v_2} = \sum_{x \in \{1,\ldots,|V_1|\} \setminus \{x_r\}} xNum([x_{r-1}, x_r, x]) + x_{r+1} Num([x_{r-1}, x_r, x_{r+1}]) + f_2$ as for the previous case this implies $w_{v_{[x_{r-1},x_r,x_{r+1}]}} = Num([x_{r-1}, x_r, x_{r+1}]) + \frac{f_2}{x_{r+1}}$, we have $w_{v_{[x_{r-1},x_r]}} = \sum_{x \in \{1,\ldots,|V_1|\} \setminus \{x_r\}} w_{|[x_{r-1},x_r,x]} +$

$w_{v_{[x_{r-1},x_r,x_{r+1}]}}$, for i.h.

$w_{v_{[x_{r-1},x_r,x_{r+1}]}} = Num([x_{r-1},x_r]) - \sum_{x\in\{1,\dots,|V_1|\}\setminus\{x_r\}} Num([x_{r-1},x_r,x]) + \frac{f_1}{x_r}$ since $Num([x_{r-1},x_r]) - \sum_{x\in\{1,\dots,|V_1|\}\setminus\{x_r\}} Num([x_{r-1},x_r,x]) \geq Num([x_{r-1},x_r,x_{r+1}]) + 1$ we have $f_2 \geq x_{r+1}(1+\frac{f_1}{x_r})$, thus $f_2 \geq f_1 + 1 \geq 2$. Iterating this as for the base case we have that $f_\delta \geq \delta$, thus we can reach the same contradiction of the base case using the fact that $\Delta > 2|V_1| \cdot log(2\lceil \frac{m_{v_1}}{|V_1|} \rceil + 1) + 1$. This implies that we cannot have failure at round $r+1$, thus $[x_0,\dots,x_{r-1}] = \sum_{\forall [x_0,\dots,x_{r-1},x]} Num([x_0,\dots,x_{r-1},x])$ this and the inductive hypothesis implies $w_{[x_0,\dots,x_{r-1}]} = Num([x_0,\dots,x_{r-1}])$.

– **Case 2** In this case we examine the application of Ver. Rule 2 for nodes that were not included in the checking condition of Ver. Rule 1. Thus let us consider a node $v : [x_0,\dots,x_{r-1}]$ for which w_v is set using Ver. Rule 2. For Inductive hyp. we have that $\forall v' : [x_0,\dots x_r] \in X_v$ $w_{v'} = Num([x_0,\dots x_r])$ and for the condition of Ver. Rule 2 $C_v = X_v$. Let us suppose that $Num([x_0,\dots,x_{r-1}]) \neq \sum_{\forall v':[x_0,\dots x_r]\in X_v} Num([x_0,\dots,x_r])$ but since F.D. Rule is not triggered we have $m_v = \sum_{\forall v':[x_0,\dots x_r]\in X_v} x_r Num([x_0,\dots,x_r]) + f_1 = \sum_{\forall v':[x_0,\dots x_r]\in X_v} x_r w_{v'}$ thus $f_1 = 0$ but this means that $Num([x_0,\dots,x_{r-1}]) = \sum_{\forall v':[x_0,\dots x_r]\in X_v} Num([x_0,\dots,x_r])$.

From the inductive proof we have that if $w_{[\perp]} \neq ?$ we must have $w_{[\perp]} = Num([\perp])$ and that there is no failure at round 1. □

Lemma 15. *If there are no failures then $\mathcal{FD}$ terminates without detecting a failure and $w_{[\perp]} = Num([\perp])$.*

Proof. The tree used by $\mathcal{FD}$ is an extension of the tree used by algorithm OPT to count in $\mathcal{G}(\mathrm{PD})_2$, the verification rule 1, 2 are an extension of the counting rule explained in Section 3.1 where to assign a value we wait to have a path from an ancestor to some node of a certain length. Therefore when no failures are present, the value w_* assigned to one node is exactly the same of n_* in $\mathcal{G}(\mathrm{PD})_2$, the correctness of OPT is proved in Lemma 11. The detection rule cannot be triggered since in case of no failures the equations 3.1 of Section 3.1 hold. □

Lemma 16. *The FD algorithm terminates in a number of rounds that is order of $\mathcal{O}(|V_1|log(|V_2|)^2)$.*

Proof. Let us notice that for each $v \in T_{\mathcal{FD}}$ we have $m_v \leq |V_1|Num(v)$, this happens if all nodes with degree history v are connected to all nodes in V_1. Let us indicate with $h_{|V_1|}(|V_2|)$ the maximum height of a tree that can be built by a run of $\mathcal{FD}$ on a $G \in \mathcal{G}(\mathrm{PD})_2$ with $|V| = |V_1| + |V_2|$ nodes. It is easy to see that $h_{|V_1|}(x)$ is finite for each value x: (1) we have $h(0) = 1$, (2) if the adversary creates a branch in the tree we have that each branch has height less or equal to $h(x-1)$, if the adversary creates no branch then the algorithm terminates in at most $2|V_1| \cdot log(2\lceil \frac{|V_1|x}{|V_1|} \rceil + 1) + 1$ rounds. Moreover we have that $h_{|V_1|}(x) \leq h_{|V_1|}(x+1)$ for the same argument used in Th. 6.

Let us define the function $g(x) = 2|V_1| \cdot log(2\lceil \frac{x}{|V_1|} \rceil + 1) + 1$, and the function $secondMax$ that given a multiset S of integer returns the second maximum element, i.e $secondMax(3, 2, 2, 1) = 2$.

Let us consider a tree that have at least height $g(m_v)$ where v is the root of the tree, a generic configuration that starts from $|V_2|$ nodes, see the Figure 3.7 left, reach an height that is always less or equal to a configuration with the structure C, see the Figure 3.7 right Let us consider the generic configuration, w.l.o.g let us restrict ourselves to the case when each node may have only two children, and a path p of length $g(m_v)$, with where b_j is the degree history of the node at distance j from the leader in the right branch. We have that the height of the tree is less or equal to $g(|V_1|) + max(\overline{max}(h_{|V_1|}(Num(b_1), Num(b_2), \ldots, Num(b_{g(m_v)}))))$, this derives directly from the rules used in the algorithm, if the value w_j is set for all b_j but one the rule 2 is applied on the tree, setting the value w_{v_0}. But it is easy to see that $g(|V_1|) + secondMax(h_{|V_1|}(Num(b_1), Num(b_2), \ldots, Num(b_{g(m_v)}))) \leq g(|V_1|) + h((\frac{\sum_{j=1}^{g(m_1)} Num(b_j) + Num(v_{g(m_v)})}{2}))$, otherwise we should have that the argument of the second maximum value in $(Num(b_1), Num(b_2), \ldots, Num(b_{g(m_v)}))$ is greater than $\frac{|V_2|}{2} = \frac{\sum_{j=1}^{g(m_1)} Num(b_j) + Num(v_{g(m_v)})}{2}$ that leads to a contradiction since this will implies then the sum of the arguments of the maximum and the second maximum elements is greater than $|V_2|$. But $g(m_v) + h_{v_1}(\frac{|V_1|}{2})$ is exactly the height of configuration C. Reiterating this argument on the two children of the two subtree of configuration C we have that $h_{|V_1|}(|V_2|) \leq log(|V_2|)g(m_v)$, since $m_v \leq |V_1||V_2|$ we have that the upper bound is $\mathcal{O}(|V_1|log(|V_2|)^2)$.

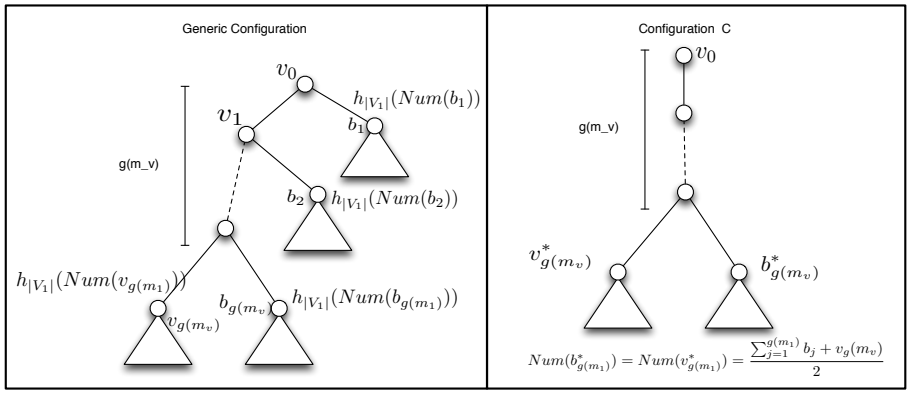


Figure 3.7: Maximum height generable by the adversary

□

From Lemmas 14,15 and 16 the next Theorem follows

Theorem 7. *On $\mathcal{G}(PD)_2$ is possible to solve* **FDP** *problem in $\mathcal{O}(|V_1|log(|V_2|)^2)$ rounds.*

There is gap between the time needed by the proposed algorithm and the lower bound of Th. 4, the ratio between the algorithm and an hypothetical optimal algorithm is at most $\mathcal{O}(\frac{log(|V_2|)^2}{log(\frac{|V_2|}{|V_1|})})$, an open question is if there exists an algorithm that matches the lower bound. The $\mathcal{FD}$ algorithm could be used as a fault detection algorithm, we continuously start a new instances of $\mathcal{FD}$ at each round, and if at round r there is a fault then the instance that starts at round r will terminate detecting the failure. But the proposed algorithm does not ensure that all the instances starting before round r terminate without detecting a fault.

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

In this section we consider networks in $\mathcal{G}(\infty-\text{IC})$. For the assumption of ∞-Interval Connectivity there is a path $p^{\infty}_{v_l,v}$ from v_l to any node v on a stable spanning tree, thus each message that is flooded by the leader will reach v in at most $|p^{\infty}_{v_l,v}|$ rounds. We define as $Len(v) = |p^{\infty}_{v_l,v}|$, that is the distance of each node from the leader on the stable path.

The algorithm works in epochs, the main idea is that each node v, with 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 $G(\text{PD})_h$, thus each non leader node v at length l has variables M, H used to compute the history degree and to collects 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 $\mathcal{FD}$ 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.

Essentially, we continuously run instances of $\mathcal{FD}$ between nodes at length l and nodes at length $l-1$, if a node v at length l at round r is disconnected from nodes at length $l-1$ we have that the instance of $\mathcal{FD}$ started at round $r-1$ will eventually detect this event, since it is analogous to a crash failure studied in the problem **FDP**. Let us remark that if also nodes in $l-1$ are moved then the instance of $\mathcal{FD}$ between nodes in $l-2, l-1$ will detects this, and so on. Let us remark that if nodes with $l=1$ are moved at a different length the leader will immediately detect this since these nodes are their direct neighbors.

Let us introduce in details the strategies used by our algorithm. The algorithm for non leader node is reported in Figure 3.8, the algorithm for the leader node in Figure 3.10.

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 send 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 checks, see line 13:

- (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 this three checks are verified then v sets its epoch and length according to the content of the message, see Lines 14-20. The check (c2) ensures that the length of v, variable $v.length$, between epochs is non decreasing, details in Lemma 17; the checks (c1,c3) are done to limit the propagation of the epoch on dynamic paths that are compatible with paths on MST^{∞}.

Specifically the check (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 check (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 check (c4) on the messages that it receives, see Line 22. The check 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 increase 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 25-26. If the check (c4) is triggered then node v was not able to route its degree history towards the leader 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 check (c4).

In the formal proofs we show that the combination of (c1,c3,c4) ensures two properties on nodes pair $(u, v) \in MST^{\infty}$ that are: (p1) if they have the same epoch value then $|u.length - v.length| \leq 1$ and (p2) if one nodes, let us say u, switch 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 18. 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 20. This and the non decreasing property of the length value, see the check (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 check (c4) cannot be triggered anymore.

```
1: length = −1
2: epoch = −1,
3: repoch = −1
4: changeepoch = false
5: M(−1) = []
6: H(−1) = [⊥]
7:
8: procedure SEND_PHASE(r)
9:     send(< length, repoch, epoch, M(r − 1), H(r − 1), changeepoch >)
10:
11: procedure RCV_PHASE(MultiSet MS : {< len, rep, ep, Ms, Hs, change >})
12:     for all m ∈ MS ordered by increasing ep do
13:         if (epoch = ep − 1) ∧ ((r − rep) + 1 ≥ length) ∧ (len == r − rep) then      ▷ Checks (c1)(c2)(c3)
14:             length = (r − rep) + 1
15:             repoch = rep
16:             epoch = ep
17:             changeepoch = false
18:             M(r − 1) = []
19:             H(r − 1) = [⊥]
20:             break
21:     if ¬changeepoch ∧ epoch > −1 then
22:         if (∄m ∈ MS|len == length − 1 ∧ ep ≥ epoch) then                          ▷ Check (c4)
23:             changeepoch = true
24:             length = length + 1
25:         MS′ : {m ∈ MS|(len == length + 1 ∧ ep == epoch ∧ change == false)}
26:         forall (m ∈ MS|(change == true∧r == length+repoch∧len ≤ length∧ep = epoch−1)) do MS′∪
    {[⊥]}
27:         M(r) = M(r − 1).append(if(MS′ ≠ ∅) {{MS′}} else { ⊥})
28:         H(r) = H(r − 1).append(count(m ∈ MS|(len == length − 1 ∧ ep == epoch)))
29:     else
30:         M(r) = M(r − 1), H(r) = H(r − 1)
```

Figure 3.8: Counting Algorithm for $\mathcal{G}(\infty$-IC): code for Non-Leader Node

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 check (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 FD algorithm, see Section 2.3. Let us recall, see Lemma 15, that FD 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 *"fails"*, that is it stops sending messages to nodes in V_{h-1} because it is moved to a different length (let us recall that in our model the sets of nodes V is static). When FD it is executed at round r such that at round $r+1$ one, or more node, in V_h fails then it detects a failure see Lemma 14.

Thus the leader at a certain epoch uses FD 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 20, in which we informally 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 23 in which we show that if the leader updates its count for a certain V_h, see Line 14, then this count is correct, and Lemma 24 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 executes (c4); v stops routing information to the leader preventing the correct counting. The leader will detect this by using FD, and this detection takes at most $\mathcal{O}(|V|^3)$ see Lemma 22. After the detection it will issue a new epoch and reset the partial count, see Lines 21-23.

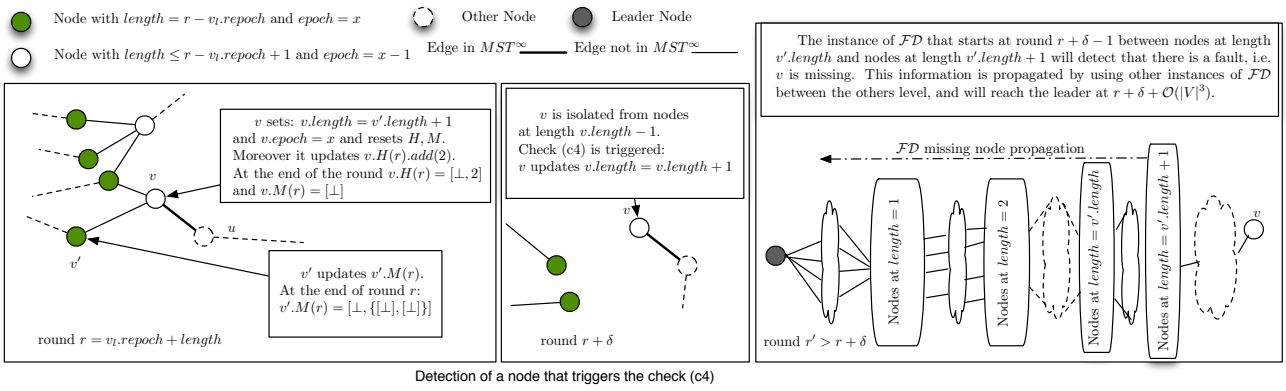


Figure 3.9: Detection of a node that changes length

To show the termination, we leverage the bound on the length value that a node may assume, see Lemma 19. Then we show that this leads to at most $|V|^2$ epochs, see Lemma 27. 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 21-22. This leads to a total worst case cost of $\mathcal{O}(|V|^5)$ rounds.

In Figure 3.9 there is a pictorial representation of the algorithm key points.

```
1: epoch = 0
2: repoch = 0
3: length_count[]
---
4: procedure SEND_PHASE(r)
5:     send(< 0, repoch, epoch, ⊥, ⊥ >)
---
6:
7: procedure RCV_PHASE(MultiSet MS :< len, ep, M, H >)
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:    index_length = 1
14:    length_count[index_length] = |MS|
15:    index_length + +
16:    while true do
17:        MS = buildLastNextLengthSet (MS)
18:        if MS == ⊥ then
19:            break
20:        if MS == failure then
21:            repoch = r + 1
22:            epoch + +
23:            reset(length_count)
24:            break
25:        length_count[index_length] = |MS|
26:        index_length + +
27:
28: function buildLastNextDistanceSet(MS)
29:    MS_last = ⊥
30:    if FDTree(Recent(MS, 0)) ≠ ⊥ then
31:        for r = (MinRound(MS); r < MaxRound(MS); r + + do
32:            if FDTree(Recent(MS, r)) == failure then
33:                return failure
34:            if FDTree(Recent(MS, r)) ≠ ⊥ then
35:                MS_last = FDTree(Recent(MS, r_last))
36:            else
37:                break
38:    return MS_last
```

Figure 3.10: Counting Algorithm for $\mathcal{G}(\infty$-IC): code for Leader Node

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

Lemma 17. *For each node v the value $v.length$ is not decreasing.*

Proof. $v.length$ is set at line 14 where the check in the if Line 22 avoid decreasing, and it is increased at line 24. □

Lemma 18. *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 14 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 13 by setting its length to $v_{b\oplus 1}.length = r+2$, (2) that $v_{b\oplus 1}$ does not execute line 13; 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 24: 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 24 during epoch *ep* cannot force the nodes to have $|v_0.length - v_1.length| > 1$.

□

Lemma 19. *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 14 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 24 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 16 by setting $v.epecoh = ep$ for Lemma 18 we have that u will execute line 16 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 18 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 20. *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^a_r, I^b_r s.t. $\forall v \in I^a_r$ we have $v.epoch = ep$ and $|I^a_r| > 0$ and $\forall v \in I^b_r$ we have $v.epoch < ep$. We have that (1) $|I^a_{r+1}| > |I^a_r|$ or (2) $|I^b_r| = 0$.*

Proof. Let us suppose, by contradiction, that at round r we have $|I^b_r| > 0$ and that at round $r+1$ we have $|I^a_{r+1}| = |I^a_r|$. 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 $v \in I^b_r$ 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 $v_1 \in I^a_r$ this and Lemma 18 implies that at round $r' \leq rep + v_1.length + 1$ the node v has to be in $I^a_{r'}$, thus we have that $v \notin I^b_{r+1}$ this implies $|I^a_{r+1}| = |I^a_r|$. So we must have $v_1 \in I^b_r$, 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 $v_2 \in I^b_r$ otherwise we can reach the same contradiction, but this implies that iterating the chain we reach a $v_k \in I^b_r$ with $Len(v_k) = 0$ that is a contradiction.

□

From the previous lemma the next observation immediately follows

Corollary 5. *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 $|I^b_r| \neq 0$ then exist $v \in I^b_r$ s.t. $v \in I^a_{r+1}$.*

Lemma 21. *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 20 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 22, thus we are equivalent to the $G(\text{PD})_h$ where messages are routed from nodes at length l to nodes at length $l - 1$. The leader v_l uses $\mathcal{FD}$ to counts node at each different length, the correctness of this procedure derive from Lemma 15. If we see line 27 - Figure 3.8 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 $\mathcal{O}(\sum_{i=1}^{x-1} |V_i| log^2(|V_{i+1}|)) \leq \mathcal{O}(|V|^3)$. Let us define as l_{last} the maximum length

assigned, we have that at most at round $rep+\mathcal{O}(l_{last}+1+\sum_{i=1}^{l_{last}-1}|V_i|log^2(|V_{i+1}|)) \leq rep+\mathcal{O}(|V|^3)$ 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 27- Figure 3.8). The terminating condition will be triggered when the leader reconstruct MS sent by nodes in V_{last} at round $rep+l_{last}+1$. □

Lemma 22. *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 24. And let us assume that $\exists v_f|v_f.length = l+1$ that at some round $r_f > rep+l$ executes line 24. We have that if v_l does not terminates then it will issue a new epoch at a round $r_n < r_f+\mathcal{O}(|V|^3)$.*

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 26 adding these messages to M. At round r_f+1 v_f executes line 24 this means that v_f is not neighbor of any node with length l (see checks at Lines 21-22), 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 check at line 25. At most at round $rep+l+1+\mathcal{O}(\sum_{i=1}^{l-1}|V_i|log^2(|V_{i+1}|))$ the leader reconstructs the set MS of messages sent by nodes with length l at round $l+1$. Starting a simulation of the $\mathcal{FD}$ 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 $r > rep+l$, i.e. equivalent to a failure discussed in Section 2.3, this implies that these simulation of $\mathcal{FD}$ see Th. 7 will terminates with a failure at most at round $r_{end} = rep+l+|V_l|log^2(|V_{l+1}|)$, since there is a propagation time from nodes in V_l to v_l of at most $\mathcal{O}(\sum_{i=1}^{l-1}|V_i|log^2(|V_{i+1}|))$ we have that at round $r = rep+l+\mathcal{O}(\sum_{i=1}^{l-1}|V_i|log^2(|V_{i+1}|)+|V_l|log^2(|V_{l+1}|))$ the leader detects a failure of the $\mathcal{FD}$, see line 32. We have $r = rep+l+\mathcal{O}(\sum_{i=1}^{l-1}|V_i|log^2(|V_{i+1}|)+|V_l|log^2(|V_{l+1}|)) < rep+l+\mathcal{O}(|V|^3)$. If $ref > rep+l+1$ we have an analogous proof since at lines 31, the leader executes a for loop in which it sequentially starts simulation s_j of $\mathcal{FD}$ such that each s_j started at round $rep+l+j$ with $j \geq 0$, for Th. 7 we have that the simulation $s_{r_f-(rep+l)-1}$ will detect the failure. □

Let us recall that when a node sets a new epoch its erase the content of M, H (Lines 18,19-Figure 3.8), the same is done by the leader (Line 23- Figure 3.10), 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 25,28-Figure 3.8 and Line 8-Figure 3.10). This is equivalent to restart the counting process at each new epoch.

Lemma 23. *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 25 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 25 then a simulation of the $\mathcal{FD}$ between nodes in V_{l-1}, V_l terminated without detecting a failure. 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 26 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. Now for lemma 22 if one nodes in V_l changes its length, or if the same is done by some node in $V_{i<l}$, we have that the leader will detects this before obtaining the counting. Otherwise if the leader does not detects this then $\mathcal{FD}$ terminated without detecting failure. Thus we have that all the instances executed in the for loop (Lines 31- Figure 3.10) s_j of $\mathcal{FD}$ such that each s_j started at round $rep + l - 1 + j$ with $j \geq 0$ terminate in a correct way. The various instances are obtained by shifting the starting round of 1, then all have to terminate detecting the same count; otherwise we have a contradiction in which the count s_j is different from s_{j+1} without detecting a failure, that is not possible for Lemma 15. Also for Lemma 15 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 that for lemma 22 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 24. *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 check 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 check that each M has to contains at least two elements and it contains only $\perp$ value. For Corollary 5 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 26 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 18 and line 27. For lemma 23 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 25. *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 23 it has correctly counted all nodes in $V_0, \ldots, V_{last}$ moreover for Lemma 24 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 26. *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 24.*

Proof. A new epoch is issued by the leader at line 22 - Figure 3.10. This means that the $\mathcal{FD}$ algorithm terminated detecting a failure, see lines 32,33,22- Figure 3.10. A

failure is detected by $\mathcal{FD}$ only if, see Lemma 15, 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 24- Figure 3.8. □

Lemma 27. *We have that the maximum number of epochs in any run is at most* $\mathcal{O}(|V|^2)$

Proof. From Lemma 26 we have that an epoch is issued only if there is node that changes its *length*. For Lemmas 19-17 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 8. *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 25. For Lemma 27 we have that the maximum number of epochs is $\mathcal{O}(|V|^2)$. For Lemmas 21,22 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. □

3.5 Conclusive Remarks

In this section we have presented an optimal counting algorithm for $\mathcal{G}(\text{PD})_h$, that is also an optimal algorithm for static networks with message duplication. Then we have presented an efficient algorithm for **FDP**. Finally we have shown a polynomial counting algorithm for $\mathcal{G}(\infty\text{-IC})$ networks.

The techniques designed in this section will be used, in Chapter 5, to build a terminating algorithm for $\mathcal{G}(1\text{-IC})$.

Open Problems From the results presented in this chapter we can identify the following open problems, ordered by their relevance:

- It is unknown if counting in $\mathcal{G}(\infty\text{-IC})$ requires more than $\Omega(|V|)$ rounds. Therefore an open question is to quantify the gap, if it exists, between the algorithm proposed in this chapter and the optimal counting time for $\mathcal{G}(\infty\text{-IC})$.
- Considering static anonymous network with message duplication. It is unknown if it is possible to count in setting (S3) with a number of rounds that is better than $\mathcal{O}(Exp(|V|))$.

Chapter 4

Counting and LDD Oracles

In this chapter we consider the problem of counting when addition knowledge about the degree of a node in G is given. We assume that each process endows a local oracle that, at the beginning of each round r, reveals to the inquiring node, an estimation on the actual number of neighbors in the current round. We will call such oracles *Local Degree Detector* oracles (LDD).

Each node is equipped with its own LDD oracle and, at the beginning of each round r, an event NUMBER_OF_NEIGHBORS(lvs) is generated by the oracle that reports the expected number of neighbors. Depending on the accuracy of the revealed information, it is possible to define several LDD oracles as follow:

- **Perfect LDD** $\mathcal{O}^P$: At the beginning of any round r the oracle $\mathcal{O}^P$ outputs the exact number of neighbors of the inquiring process v. i.e.,
$$\forall v \in V, \forall r \in \mathbb{N}^+ :: \mathcal{O}^P(v,r) = |N(v,r)|$$
- **LDD with Over Estimation** $\mathcal{O}^{OE}$: At the beginning of any round r the oracle $\mathcal{O}^{OE}$ outputs an upper bound on number of neighbors of the inquiring process v. i.e.,
$$\forall v \in V, \forall r \in \mathbb{N}^+ :: \mathcal{O}^{OE}(v,r) \geq |N(v,r)|$$
- **LDD with Fixed Over Estimation**: $\mathcal{O}^{FOE}$: At the beginning of the computation the oracle $\mathcal{O}^{FOE}$ outputs an upper bound d_{max} on the number of neighbors for any process v and any round r. i.e.,
$\exists d_{max} \in \mathbb{N}^+ : \forall v \in V, \forall r \in \mathbb{N}^+ ::$
$$\mathcal{O}^{FOE}(v,r) = d_{max} \text{ with } d_{max} \geq |N(v,r)|$$
- **NoK** : The local oracle does not give any additional knowledge.

Let us notice that $\mathcal{O}^P$ can be easily implemented by using the information collected at the MAC layer or it could be implemented in systems where there is enough stability to have a request-reply pattern between processes (that in our model means having no topology changes for two consecutive rounds).

It is easy to see that $\mathcal{O}^{FOE}$ satisfies the definition of $\mathcal{O}^{OE}$ while the viceversa is not true. Thus $\mathcal{O}^{OE}$ is strictly weaker than $\mathcal{O}^{FOE}$. At the same time, $\mathcal{O}^{OE}$ is strictly weaker than $\mathcal{O}^P$.

The algorithms contained in this Chapter are based on a technique that mimics an *energy-transfer*. Informally it works as follows: each node v_i is assigned with a fixed unitary energy charge, stored in the variable e_{v_i}, and during each round it discharges itself by disseminating energy around to its neighbors i.e., e_{v_i} decreases of a value $k \leq e_{v_i}/2$, then this quantity k is equally split among the neighbors of v_i and this value is added to v_i's neighbors variable. The leader acts as a sink collecting energy (i.e., energy is not transferred by the leader to neighbors). Our technique enforces, at each round, a global invariant on the sum of energy among networks' nodes (i.e., $\sum_{v_i} e_{v_i} = |V|$), that resorts to the fact that energy is not created or destroyed in the anonymous dynamic distributed system (energy conservation property). Considering the behavior of the nodes, the energy is eventually transferred to the leader and stored there. The crucial point is that the energy is concentrated in the leader, and this ensures a monotonically increasing behavior of variable e_{v_l} that will converge towards $|V|$. This behavior is fundamental for the correctness of the $\mathcal{A}_{\mathcal{O}^{OE}}$ algorithm presented in the successive Section.

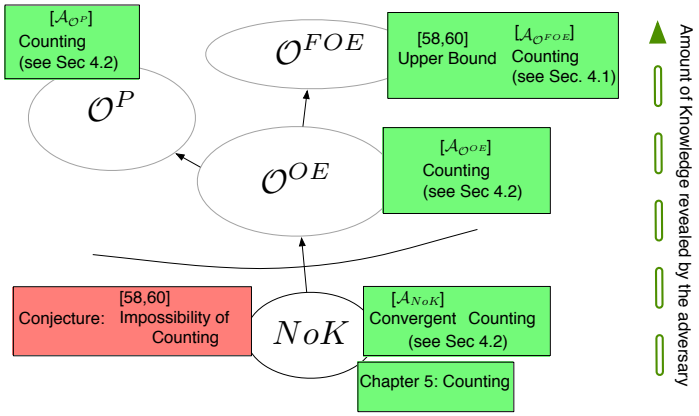


Figure 4.1: Dependency relation between oracles and results on counting in anonymous networks under worst-case adversary

The technique shares some similarities with the averaging gossiping used in [40,41], where nodes continuously exchange fractions of input variables in order to reach an average of the inputs. Both previous works, that are coping with a random adversary, suggest the same way to have a convergent counting algorithm: one node starts with input 1 the others with input 0, the average of the inputs will converge to $\frac{1}{|V|}$. We are not aware of terminating condition or convergence study for averaging gossiping under worst case adversary. The convergence in both works is proved only for an uniform gossiping scheme, i.e at each send phase a node samples uniformly a set of nodes of known size for interaction, therefore it is not clear if they converge also in case of worst case adversary where the interaction scheme is far different from uniform gossiping.

Our approach concentrates the sum of inputs in the leader, this allows to solve the following problem: There are $x < |V| - 1$ nodes in $V \setminus \{v_l\}$ with initial input 1 and

all the other nodes have input 0; the leader knows that the number of nodes is x but it does not know $|V|$; v_l has to terminates when it receives, directly or indirectly, a message from these nodes. With our technique when the energy $e_{v_l} > x - 1 + \epsilon$, with $\epsilon > 0$, the leader knows that it has received information from all x nodes; despite its lack of knowledge about the network size $|V|$. This simple problem is fundamental for terminating counting and its solution is a building block for algorithm $\mathcal{A}_{\mathcal{O}^{OE}}$.

The next sections are organized as follows: In Section 4.1 is presented a terminating counting algorithm, $\mathcal{A}_{\mathcal{O}^{FOE}}$, that uses $\mathcal{O}^{FOE}$, this algorithm is used to introduce the aforementioned technique and to prove its convergence. In Section 4.2 is presented a terminating counting algorithm, $\mathcal{A}_{\mathcal{O}^{OE}}$, that uses $\mathcal{O}^{OE}$. In Section 4.3 we present a **convergent**, i.e. non terminating, algorithm that uses no additional knowledge, $\mathcal{A}_{\mathcal{O}^{NoK}}$. In the same section we also present a termination heuristics designed to have an approximated counting in environments where the adversary is not worst case. At the end of the Chapter, in Section 4.4, we present an evaluation of our algorithms under different random adversaries

4.1 Counting Algorithm using $\mathcal{O}^{FOE}$

This section presents a distributed algorithm, denoted as $\mathcal{A}_{\mathcal{O}^{FOE}}$, that assumes (i) the existence of a leader node starting from a different initial state and (ii) all other nodes execute identical programs. In addition, $\mathcal{A}_{\mathcal{O}^{FOE}}$ assumes that the bound on nodes degree d_{max} (which limits the powerfulness of the adversary) is known by all processes.

$\mathcal{A}_{\mathcal{O}^{FOE}}$ is composed by a sequence of iterations $i = 0 \dots \ell$ run by the leader and each iteration takes several rounds. Each iteration i starts considering two parameters: (i) the upper bound on nodes degree d_{max} and (ii) an upper bound K_i on the network size; then, the algorithm computes a guess for the network size. If the guess matches the current considered bound K_i, then the algorithm terminates and K_i corresponds to the size of the network $|V|$. Otherwise iteration $i + 1$ is started taking d_{max} and $K_{i+1} = K_i - 1$ as inputs.

Let us notice that, starting from the upper bound d_{max} on nodes degree, an upper bound K_0, to be used in the first algorithm iteration, can be easily computed (e.g. by using the algorithm shown in [59]). Thus, the hard part of $\mathcal{A}_{\mathcal{O}^{FOE}}$ is to compute the guess on the network size at the end of each iteration. To this aim, $\mathcal{A}_{\mathcal{O}^{FOE}}$ employs the idea of *energy-transfer* and exploits the invariant on the energy globally stored in network: the sum of the energy stored by each node at the end of each round remains constant during the algorithm execution. Note that the energy we are considering is not the real energy that nodes may have but it is rather an abstraction used to explain the details of the algorithm.

At the beginning of each iteration i (with $i \geq 0$), every node is assigned with a fixed energy charge, and during each round r it discharges itself by disseminating the energy to its neighbors. The leader acts as a sink collecting energy (i.e., it does not transfer energy to its neighbors).

Considering the behavior of the nodes, the energy is eventually transferred to the leader. The leader measures the level of energy received to verify if the energy level

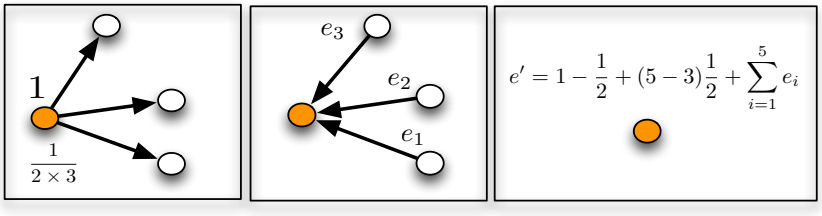


Figure 4.2: Send, Receive and Update of energy for a non leader node

matches the bound K_i considered in the current iteration i. More specifically, the leader starts iteration i of $\mathcal{A}_{\mathcal{O}FOE}$ algorithm by assuming that the network has size K_i and computes the first round r_final_i where, in the worst case, it should have received strictly more than $K_i - 1$ energy by all the others (see Lemmas 29,30). If at round r_final_i, the leader has received such amount of energy then it outputs $|V| = K_i$. Otherwise the algorithm starts iteration $i+1$ with $K_{i+1} = K_i - 1$.

Algorithm $\mathcal{A}_{\mathcal{O}FOE}$ – *Terminating Counting Algorithm resilient to a Dynamic Graph Adversary with Bounded Degree.*

(Step 1) The algorithm starts taking as input d_{max} and computes an upper bound K on the network size by running an estimation algorithm as the one shown in [59];

(Step 2) The leader and the anonymous nodes start the step 2 at round r_start;

leader : The leader v_l maintains the following local variables: $e_{v_l} \leftarrow 1$ representing the initial energy charge, $rcv_{v_l} \leftarrow \emptyset$ is a set variable where v_l stores all the messages received during each round, $i \leftarrow 0$ is the iteration number, $K_i \leftarrow K$ is the upper bound on the network size at iteration i, $r\_init_i \leftarrow r\_start$ is the starting round of iteration i, r_final_i is the termination round of iteration i when the leader will be able to either validate the guess (i.e., $|V| = K_i$) or not (i.e., $|V| < K_i$) - see Lemmas 29, 30.

start iteration i at round r_init$_i$: take d_{max} and K_i as inputs

The leader computes the minimum $R \in \mathbb{N}^+$ such that R satisfies $K_i - (1 - (\frac{B-1}{B})^R)K_i < 1$, where $B = (2d_{max})^K$ and sets $r\_final_i \leftarrow r\_init_i + RK_i$.

for each round r:

- **Send phase of round** r**:** at the beginning of each round, v_l broadcasts a ENERGY_RELEASE(0) message releasing no energy.
- **Receive and Computation phases of round** r**:** ENERGY_RELEASE(e') messages are stored in the rcv_{v_l} variable. At the end of the round, when all the messages have been received, v_l updates, its local energy charge as follows:

  $$e_{v_l} \leftarrow e_{v_l} + received$$

  where *received* is the sum of all the charges received by neighbors (i.e., $\sum_{e' \in rcv_{v_l}} e'$).
- **When** $(r = r\_final_i)$ **% Terminating Condition %**
  if $(K_i - e_{v_l} < 1)$
  then v_l terminates and broadcasts for K_i rounds STOP(K_i) message to other nodes and then it outputs $count \leftarrow K_i$;
  else v_l sends a message RE-START($r\_final_i + K_i$) to other nodes declaring that it will start a new iteration. This message is broadcast for K_i rounds and then v_l starts iteration $i+1$ considering the new bound $K_{i+1} = K_i - 1$ and $r\_init_{i+1} = (r\_final_i + |K_i|)$.

anonymous node: Each non leader node v_i maintains the following local variables: $e_{v_i} \leftarrow 1$ representing the initial energy charge of v_i, $rcv_{v_i} \leftarrow \emptyset$ is a set variable where v_i stores all the messages received during each round.

for each round r:

- **Send phase of round** r**:** at the beginning of each round, v_i broadcasts a ENERGY_RELEASE($\frac{e_{v_i}}{2d_{max}}$) message, releasing at most half of its energy to its neighbors (at most d_{max}).
- **Receive and Computation phases of round** r**:**
  switch

  case ($\exists$ m =STOP($count$) $\in rcv_{v_i}$)
  v_i broadcasts STOP($count$) for $count$ rounds, then v_i stops to execute the algorithm and outputs the value $count$.

  case ($\exists$ m =RE-START(r') $\in rcv_{v_i}$)
  v_i broadcasts RE-START(r') until the current round is r' and then v_i initializes all local variables.

  default **%$\mathbf{rcv_{v_i}}$ contains only** ENERGY_RELEASE(e') **messages %**
  v_i updates its local energy charge as follows:

$$e_{v_i} \leftarrow e_{v_i} - released + recharged + received =$$

  where the energy *released* is given by the the quantity of energy sent to its current neighbors (i.e. $(d_{max} \times \frac{e_{v_i}}{2d_{max}})$), the energy *recharged* is the amount of energy not effectively released due a possible neighborhood over estimation (that, in the current round, is smaller than d_{max}) and computed by considering the difference between the estimated number of neighbors d_{max} and the effective ones $|rcv_i|$ (i.e. $(d_{max} - |rcv_{v_i}|) \times \frac{e_{v_i}}{2d_{max}}$). Thus,

$$e_{v_i} \leftarrow e_{v_i} - \big(d_{max} \times \frac{e_{v_i}}{2d_{max}}\big) + (d_{max} - |rcv_{v_i}|) \times \frac{e_{v_i}}{2d_{max}} + \sum_{e' \in rcv_{v_i}} e'$$

Figure 4.2 shows the execution of a generic round r at node v_i with initial energy 1 where $d_{max} = 5$ and the number of v_i neighbors is 3.

Correctness Proofs.

In the following, we will prove that protocol $\mathcal{A}_{\mathcal{O}^{FOE}}$ is correct. We first prove in Lemma 28 the existence of the following invariant: the global energy stored in the system is preserved (i.e. the sum of all the energy variables e_{v_i} at each $v_i \in V$ remain constant). Note that, this is a fundamental property to ensure the correctness of the guess computed and returned by the leader. Then, in Lemma 29, we derive a lower bound on the energy that the leader will receive during the computation. Finally, in Lemma 30, we prove the terminating condition that the leader has to verify in order to check if the bound matches the network size. From these results Theorem 9 naturally follows. In the following, we use the notation $e^r_{v_i}$ to indicates the quantity of energy stored at round r in $v_i \in V$.

Lemma 28. *(Invariant - Energy Conservation) At the end of each round, the sum of the energy over all the nodes of the anonymous dynamic network is an invariant and it is equal to the total energy present in the system at the beginning of the computation.*

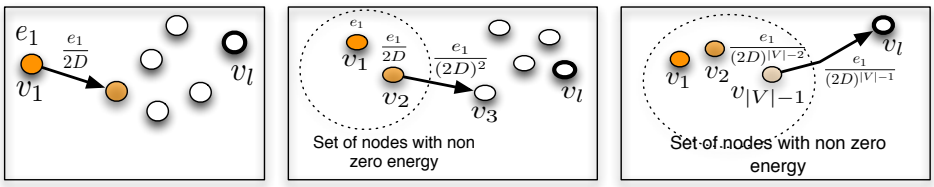


Figure 4.3: Lower bound on the energy received by the leader from a single node

Proof. Let us notice that energy is created only during the initialization phase and then it is only transferred. Thus, in the following, we will prove that energy is never lost. For ease of presentation, in the prove we refer only to the energy $e^r_{v_i}$ stored at non-leader nodes (i.e. $v_{\in}V \setminus \{v_l\}$) as v_l does not send energy. At the beginning of each round r, every v_i sends $\frac{e^r_{v_i}}{2d_{max}}$ energy to the set of its neighbors (i.e. the released energy is $d_{max} \times \frac{e^r_{v_i}}{2d_{max}}$). However, v_i does not know the exact number of current neighbors $N(v_i, r)$ but it just know an upper bound on them. Thus, while sending the energy, due to a possible over estimation of its neighborhood, v_i may transfer more energy that the one will never received by other processes (loss of energy of $(d_{max} - |N(v_i, r)|) \times \frac{e_{v_i}}{2d_{max}}$). However, v_i will know the exact number of neighbors by counting the number of received messages in the current round ($|N(v_i, r)| = |rcv_{v_i}|$) and it will compensate the extra energy released.

Thus, v_i will have a remaining energy of $\frac{e^r_{v_i}}{2} + e^r_v \frac{d_{max} - |N(v_i,r)|}{2d_{max}}$. The energy initially possessed by v_i was $e^r_{v_i}$, at the end of r the sum of energy over nodes in $N(v_i, r) \cup \{v_i\}$ is $e^r_{v_i}$. Iterating this reasoning over all nodes we have that the conservation of energy is not violated.

□

Lemma 29. *Let $G \in \mathcal{G}(1\text{-}IC)$, let K be an upper bound on the network size and let $R \in \mathbb{N}^+$. Given the algorithm $\mathcal{A}_{\mathcal{OFOE}}$, at the end of round RK, the energy stored at the leader is at least $e^{RK}_{v_l} \geq (1 - (\frac{B-1}{B})^R)|V|$, where $B = (2d_{max})^K$.*

Proof. Let us first compute the energy that a single node v_i provides to the leader v_l when it is the only one releasing energy (i.e. v_i has an energy charge of $e_{v_i} = 1$ while all the other non-leader nodes v_j has no energy and $e_{v_i} = 0$).

At round 0, v_i transfers $e = \frac{1}{2d_{max}}$ to each of its neighbors. Due to 1-interval connectivity, v_i has at least one neighbor v_j that will receive such energy charge and will update its residual energy to $e^0_{v_j} \geq \frac{1}{2d_{max}}$. As a consequence, an additional node (i.e. v_j) will have a quantity of energy greater than 0, no matter of the adversary move.

At round 1, due to the adversary action, v_i and v_j may change their neighbors and a node v_k, with no energy, may become neighbor of v_j. Due to the rules of the algorithm, v_j will transfer to v_k a quantity of energy at least $e_j \geq \frac{e}{(2d_{max})} \geq \frac{1}{(2d_{max})^2}$. Let us notice that the such quantity is a lower bound on the energy that an empty node may receive after 2 rounds.

Iterating the reasoning, it is possible to define the lower bound on the energy that the leader v_l received from v_i after $|V|-1$ rounds in case v_i is the only one releasing energy, i.e. $e_{v_l}^{|V|-1} \geq \frac{1}{(2d_{max})^{|V|}}$. Considering the individual contribution of each node in V, we obtain $e_{v_l}^{|V|-1} \geq \frac{|V|}{(2d_{max})^{|V|}}$. However, node contributions interfere and the real quantity of energy that each of them transfers to the leader at each round is greater than the computed one; thus the formula above represents a lower bound on the energy transferred from all nodes to the leader after $|V|-1$ rounds.

Due to Lemma 28, $e_{v_l}^{|V|-1} + \sum_{v_i \in V \setminus \{v_l\}} e_{v_i}^{|V|-1} = |V|$, this implies that after $|V|-1$ rounds, the energy not stored at the leader is, at most, $|V| - e_{v_l}^{|V|-1}$.

Using the same argument, after $2(|V|-1)$ rounds the lower bound on the energy that the leader stores is $e_{v_l}^{2(|V|-1)} \geq \frac{|V|-e_{v_l}^{|V|-1}}{(2d_{max})^{|V|}} + e_{v_l}^{|V|-1}$. We can use the same argument even if the distribution of energy at round $r = |V|$ is different from the uniform initial distribution, the basic idea is to consider the contribution that each node has to send to the leader independently, then summing up these contribution we will end up with the same bound.

After $R(|V|-1)$ rounds, the energy at the leader is at least $e_{v_l}^{R(|V|-1)} \geq \frac{|V|-e_{v_l}^{(R-1)(|V|-1)}}{(2d_{max})^{|V|}} + e_{v_l}^{(R-1)(|V|-1)}$ that is a recurrence equation with boundary condition $e_{v_l}^{0} = 0$.
Solving the equation we obtain: $e_{v_l}^{R(|V|-1)} \geq |V| \times (1 - (\frac{b-1}{b})^R)$ where $b = (2d_{max})^{|V|}$, that is a lower bound on the energy of the leader after $R \times (|V|-1)$ rounds. Let us recall that $K \geq |V|$, thus we can compute a lower bound for the energy in the leader after RK rounds on a network of size $|V|$, with $B = (2d_{max})^K$ since $\frac{|a|}{b} \geq \frac{|a|}{B}$ the following inequality holds:

$$e_{v_l}^{RK} \geq |V| \times \left(1 - \left(\frac{B-1}{B}\right)^R\right)$$

□

Lemma 30. *Let $G \in \mathcal{G}$(1-IC), let K_i be an upper bound on the network size. Given the algorithm $\mathcal{A}_{\mathcal{O}^{FOE}}$, at the end of each iteration i, the leader can either accept or reject the hypothesis $|V| = K_i$.*

Proof. Let us note that the algorithm outputs the exact count when the upper bound K_i is equal to the real network size $|V|$. Initially, K_i is set by running the algorithm in [59] that has been proved to provide an upper bound on the network size (i.e. $K_0 \geq |V|$). The leader can compute the difference Δ_r as follows

$$\Delta_r = K_i - e_{v_l}^r \geq K_i - \left(1 - \left(\frac{B-1}{B}\right)^R\right) K_i$$

that is the difference, at round $r = RK_i$, between the total energy stored in a network of size $|V| = K_i$ and the lower bound on the minimum amount of energy that the leader has to receive after r rounds if $|V| = K_i$ (as computed in Lemma 29).

According to the algorithm, the leader computes the minimum $R' \in \mathbb{N}^+$ such that $r\_final_i = R'K_i$ satisfies $\Delta_{r\_final_i} < 1$. At round r_final_i two cases are possible:

1. The energy accumulated in the leader $e_{v_l}^{r\text{-}final_i}$ is less or equal than $K_i - 1$: in this case, we have necessary that $|V| < K_i$. By construction, in fact, R' is selected in such a way that the quantity $\Delta_{r\_final_i} < 1$; thus, if the real network size is K_i, the leader must collect $e_{v_l}^{r\text{-}final_i} > K_i - 1$. On the contrary, the only possibility is that $|V| < K_i$ since in this case the quantity of energy in V will not allow the leader to reach a value $e_{v_l}^{r\text{-}final_i} = K_i - 1 + \epsilon$ with $\epsilon > 0$.

2. The energy accumulated in the leader $e_{v_l}^{r\text{-}final_i} = K_i - 1 + \epsilon$ (with $\epsilon > 0$): this implies that there are at least K_i nodes in V. Considering that K_i is an upper bound we have that necessary $|V| = K_i$.

□

Theorem 9. *Let $G \in \mathcal{G}$(1-IC) and let K_0 be an upper bound on the network size. The algorithm $\mathcal{A}_{\mathcal{O}^{FOE}}$ is a terminating algorithm.*

Proof. During the Step 1, computes an upper bound K_0 on the size of the network (i.e. $K_0 \geq |V|$) and then it moves to Step 2 starting a sequence of iteration $I = \{i_0, ..., i_{final}\}$, each one taking as input an upper bound K_{i_x}. According to Lemma 30, at the end of an iteration i_x, the leader checks the terminating condition, evaluates the hypothesis $K_{i_x} = |V|$ and in case of rejection, it decreases by one the upper bound considered in the following iteration. Considering that (i) K_0 is an upper bound on the real network size and (ii) every iteration i_{x+1} starts with a new upper bound $K_{i_{x+1}} = K_{i_x} - 1$ it follows that, in a finite number of iterations, $\mathcal{A}_{\mathcal{O}^{FOE}}$ will evaluate the real size of the network accepting the hypothesis $K_i = |V|$.

The **termination** is also ensured by Lemma 30, considering that when the leader accepts the hypothesis $K_i = |V|$ starts to disseminate a STOP message letting the algorithm terminate at each node.

□

Complexity and Discussion

Theorem 10. *Let $G \in \mathcal{G}$(1-IC) and let K_0 be an upper bound on the network size. The algorithm $\mathcal{A}_{\mathcal{O}^{FOE}}$ outputs the correct size of the network after at most $O(e^{K_0^2} K_0^2 (K_0 - 1))$ rounds.*

Proof. The claim follows from Lemma 29 and Lemma 30 by considering that each iteration i of the algorithm lasts $\Delta_R(K_i) \times K_i$, where $\Delta_R(K_i)$ is the number of $K_i - 1$ rounds that the algorithm has to execute in order to check the hypothesis $|V| = K_i$ (cfr. Lemma 30). Moreover, we have that the maximum number of iterations is K_0 (i.e., the worst case obtained starting from evaluating the hypothesis on the size $|V| = K_0$ when the real size is $|V| = 1$). $\Delta_R(K_i)$ can be computed by solving $\Delta_{r\_final_i} < 1$ inequality obtaining:

$$\Delta_R(K_i) \geq \left\lceil \frac{\log(\frac{1}{K_i})}{\log(\frac{B-1}{B})} \right\rceil + 1$$

where $B = (2d_{max})^{K_i}$.

Let us notice that, in the worst case, $d_{max} = K_i$, so we have $\Delta_R(K_i) = \Theta\left(\frac{\log\left(\frac{1}{K_i}\right)}{\log\left(1 - {K_i}^{-K_i}\right)}\right)$.

Since the limit $\lim_{K_i \to +\infty} \left(\frac{\Delta_R(K_i)}{e^{K_i^2}} \right) = 0$ we have that the number of rounds for each iteration is $O\left(e^{K_i{}^2}(K_i - 1)\right)$ this seamlessly leads to the theorem.

□

The complexity aforementioned is a direct consequence of the energy release mechanism where each non leader node transfers half of its energy at each round. Let us suppose to modify the algorithm in such a way that each node sends the whole energy to its neighbors. In this case the problem became equivalent to unique token circulation, the trivial example is a chain $\{v_1, v_2, ..., v_n\}$, that is transformed at odd rounds in $v_2, v_1, ..., v_n$ and in even rounds as $v_1, v_2, ..., v_n$, this dynamic adversary will prevent the initial energy present in v_1 to reach other nodes apart v_2. On the contrary halving the energy allow us to obtain the bound on energy dissemination (See Lemma 29). Let us remark that changing the quantity of energy sent by nodes (i.e. sending a fraction different from $\frac{1}{2}$) does not change the complexity from being exponential since the termination round is only influenced by a multiplicative factor.

4.2 Counting Algorithm using $\mathcal{O}^{OE}$

In this section we present a terminating counting algorithm, namely $\mathcal{A}_{\mathcal{O}^P}$, working in an anonymous network governed by a worst-case dynamic adversary and where processes are equipped with an $\mathcal{O}^{OE}$ LDD oracle. For ease of presentation, we introduce and prove the algorithm using oracle $\mathcal{O}^P$ and then we will explain the changes needed in the algorithm in order to let it work with the weaker oracle $\mathcal{O}^{OE}$. At the end of the section we discuss a variation of $\mathcal{A}_{\mathcal{O}^P}$, namely $\mathcal{A}^*_{\mathcal{O}^P}$, that is designed to terminate faster on graphs generated by random adversaries. Experimental results are presented at the end of the evaluation Section 4.4. The pseudo-code for the algorithm is shown in Figures 4.4,4.5.

Abstract Description Informally, the algorithm follows the approach of "coloring and counting": at each round, a new subset of processes is colored with a new color and then the algorithm counts the number of colored processes (for each color) until no new color is used. More in detail, at round r_0, the leader v_l starts with color c_0 while each other node has no color, i.e. its color is $\perp$. At any round r_i a new color c_i is used and the mapping between round number and color is unique. As a consequence, considering that round numbers are totally ordered, it follows that also colors (and set of nodes colored with a specific color) are totally ordered according to the round number (i.e. a node colored in round 0 will have a color that preceeds the color of a node colored in round 1 in the color order). In addition, exploiting the unique mapping between round numbers and colors, we can use the operators $<, \leq, =$ between colors as they can be represented as natural numbers, i.e. there exists an isomorphism between colors and $\mathbb{N}^+$.
Each non-colored node (i.e. a node with color $\perp$) will be colored in round r_i with color c_i if and only if it has at least one neighbor with a color different from $\perp$ and it does not change its color anymore. This is detected by letting processes exchange their colors at the beginning of each round. The multi-set defined by colors received by neighbors in a certain round is called *color view*.

At the beginning of each round r
(01) $color\_view_{v_i} \leftarrow \emptyset$;
(02) $n\_size_{v_i} \leftarrow$ NUMBER_OF_NEIGHBORS(lvs);

Send Phase of round r: each process v_i sends two different type of messages:
(03) it broadcasts a MY_COLOR$(color_{v_i})$ message to let its neighbors know if it is colored and its color;
(04) **for each** $< e, color\_view, col, r' > \in energy\_set_{v_i}$
(05) v_i broadcasts a ENERGY$(\frac{e}{2 * n_size_{v_i}}, color_view, col, r')$ message;
(06) v_i updates $energy\_set_{v_i} \leftarrow energy\_set_{v_i} \setminus \{< e, color\_view, col, r' >\})$;
(07) v_i updates $energy\_set_{v_i} \leftarrow energy\_set_{v_i} \cup \{< \frac{e}{2}, color\_view, col, r' >\}$;
(08) **endFor**

Receive Phase of round r
(09) **for each** ENERGY$(e', color_view, col, r')$ message in rcv_i **do**
(10) **if** $(\nexists \ < e'', color\_view, col, r' > \in energy\_set_{v_i})$ (i.e., with the same color view, same color and same round)
(11) **then** $energy\_set_{v_i} \leftarrow energy\_set_{v_i} \cup < e', color\_view, col, r' >$;
(12) **else** $energy\_set_{v_i} \leftarrow energy\_set_{v_i} \setminus \{< e'', color\_view, col, r' >\}$;
(13) $energy\_set_{v_i} \leftarrow energy\_set_{v_i} \cup \{< e' + e'', color\_view, col, r' >\}$.
(14) **endIf**
(15) **endFor**
(16) **For each** MY_COLOR(cl) message in rcv_i **do**
(17) $color\_view_{v_i} \leftarrow color\_view_{v_i} \cup \{cl\}$;
(18) **endFor**

Computation Phase of round r
(19) **if** $((\exists c_j \neq \perp \in color\_view_{v_i}) \wedge (color_{v_i} = \perp))$ **then** $color_{v_i} \leftarrow c_r$.
(20) **if** $(color_{v_i} \neq \perp)$ **then** $energy\_set_{v_i} \leftarrow energy\_set_{v_i} \cup \ < 1, color\_view_{v_i}, color_{v_i}, r >$.

Figure 4.4: The $\mathcal{A}_{\mathcal{O}^P}$ Algorithm. Non Leader Protocol

At the beginning of each round r
(01) $color\_view_{v_l} \leftarrow \emptyset$;

Send Phase of round r:
(02) The leader broadcasts a MY_COLOR($color_{v_l}$) message.

Receive Phase of round r: The leader handles the received messages as an anonymous node.

Computation Phase of round r
(03) v_l creates a tuple $< 1, color\_view_{v_l}, color_{v_l}, r >$ and adds this tuple in its $energy\_set_{v_l}$.
(04) **if** $(count = 0)$ **then** $count \leftarrow |color\_view_{v_l}| + 1$;
(05) **let** $S = \{< e, cv, col, r' >\in energy\_set_{v_l} \mid col \leq lcc \wedge r' = lcc + 1\}$
be the sub-set of energy charges created at round $r' = lcc + 1$
and coming from nodes colored at some round $r' \leq lcc$.
This energy is transferred to v_l by the nodes in $C_1 \cup C_2 \cup ... \cup C_{lcc}$.
(06) **let** $F = \{< e, cv, col, r' >\in energy\_set_{v_l} \mid col = lcc + 1 \wedge r' = lcc + 1\}$ be the sub-set of energy charges created at round $r' = lcc + 1$ and coming from nodes colored at round $r' = lcc + 1$.
This energy is transferred to v_l by the nodes C_{lcc+1}.
(07) **if** $(count - (\sum_{\forall < e, color\_view, col, r' > \in S} e) < 1)$
(08) **then if** $(\nexists < e', color\_view, col, r' >\in S \mid \exists \perp \in color\_view)$
(09) **then trigger** terminated_count($count$);
% Let us define $n^{\perp}_{color\_view}$ as the number of $\perp$ in color_view.
(10) **else** $upper\_bound \leftarrow \sum_{\forall < e', color\_view, col, r' > \in S} \lceil e' \rceil \times n^{\perp}_{color\_view}$;
% Let us define $n^{*}_{color\_view}$ as the number of processes with color less or equal than $*$ in the color view $color\_view$.
(11) $shrink \leftarrow \sum_{\forall < e', color\_view, col, r' > \in F} \lceil e' \rceil \times (n^{lcc}_{color\_view} - 1)$;
% Let us notice that $n^{lcc}_{color\_view}$ is the number of edges from process in
% $C_1 \cup C_2 \cup ... \cup C_{lcc}$ to a process with color view $color\_view$ in C_{lid+1}.
% So *shrink* is a *"surplus"* of processes that have tried to color processes in C_{lcc+1}.

(12) **if** $((upper\_bound - shrink) - \sum_{\forall < e', color\_view, col, r' > \in F} e' < 1)$
(13) **then** $lcc \leftarrow lcc + 1$;
(14) $count \leftarrow count + \lceil \sum_{\forall < e', color\_view, col, r' > \in F} e' \rceil$;
(15) **endIf**
(16) **endIf**
(17) **endIf**

Figure 4.5: The $\mathcal{A}_{\mathcal{O}^P}$ Algorithm. Leader Protocol

In the following, we indicate with C_i the set of nodes with color c_i.
At the beginning of the algorithm, the leader is able to count only itself. At the end of round r_0, the leader will be also able to count its direct neighbors. In the following rounds, the leader will count processes with color c_1, then processes with color c_2 and so on until everyone is colored and counted.
Let us note that counting processes in C_0 is straightforward as they are neighbors of the leader v_l at round r_0 and the leader itself. The tricky part is counting processes colored in the following rounds, i.e. counting $|C_j|$ for all $j > 0$. This is done, at the leader side, by collecting the color views from processes in $C_0 \cup C_1 \cup ... \cup C_{j-1}$. Through these color views it is possible to compute an upper bound on the actual size of a set C_j; this upper bound is then refined to the actual $|C_j|$ by collecting the views of processes in C_j at round r_j.
As an example, let us consider Figure 4.6(a) where we show on the left the network at the end of round r_0 and on the right the network an the end of round r_1. Processes v_1, v_2, v_3 have been colored by the leader during r_0 with color c_0 and they have exactly the same color view both at round r_0 and r_1. To properly reconstruct the frontier between nodes with color c_0 and nodes with color $\perp$, that will assume color c_1 at the end of round r_1 (processes with color blue in the right Figure), the leader must be able to count the correct multiplicity of each color view defined at round r_1.
Local Variables at node v_i. Each process v_i maintains the following local variables:

1. $color_{v_i}$: is a local variable storing the color of node v_i, that is the color c_j associated to the round at which v_j has been colored. Such variable is initialized to a default value $\perp$ except for the leader that initializes it to c_0.
2. $n\_size_{v_i}$: is an integer variable (initialized to 1) storing the current number of neighbors of node v_i.
3. $color\_view_{v_i}$: is a multi-set variable (emptied at the beginning of each round) where v_i collects the set of colors of its neighbors.
4. $energy\_set_{v_i}$: is a set (initially empty) where processes store tuples in the form $< e, clv, col, r >$ used to collects energy charges sent during different rounds. In particular, each tuple contains: (i) the quantity of energy e transferred by the sender, (ii) the color view clv of the sender and (iii) the sender color at the specified round r.
5. rcv_{v_i}: is a set variable (emptied at the beginning of each round), where v_i stores messages received during the current round r.

In addition, the leader v_l also maintains the following local variables:

1. *count*: is an integer variable (initialized to 0) used to store the partial count.
2. *lcc*: is an integer variable (initialized to 0) which stores the color of the last counted processes, that are the processes in C_{lcc}.

Detailed Algorithm Description.

At round r_0, the leader is the only process having a color (i.e. it is the only process having $color_{v_l} \neq \perp$). During the send phase of r_0, each process (included the leader) broadcasts a MY_COLOR message to all its neighbors (line 03 in Figure 4.4 and line 02 in Figure 4.5). Note that, during round r_0, MY_COLOR messages are the only ones sent

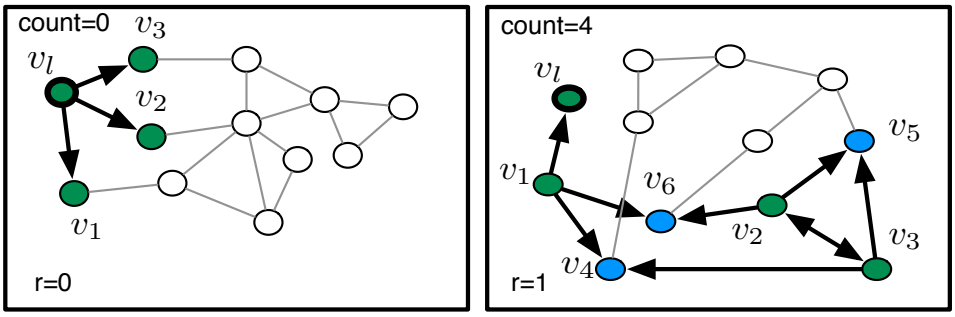
around as there are no colored nodes except the leader and the $energy\_set_{v_i}$ variable is initially empty, meaning that no colored energy is created. Every MY_COLOR message sent in r_0 by the leader v_l will trigger the coloring of the leader neighbors. In fact, when such processes will execute the computation phase at the end of r_0, they will find in their $color\_view_{v_i}$ variable the color c_0 sent by the leader (line 17, Figure 4.4) enabling the coloring with color c_0 (line 19, Figure 4.4). As an example, consider the scenario depicted in Figure 4.6(a) where node v_1 receives a MY_COLOR(c_0) message from the leader and a MY_COLOR($\perp$) message from v_i; this implies $color\_view_{v_1} = \{c_0, \perp\}$ (where color $c_0 =$ green).
Let us remark that the leader and its neighbors at round r_0 belong to the set C_0 and the cardinality of C_0 is known immediately by the leader. Thus, in the computation phase of round r_0, the leader can update the variable *count* to $1 + |color\_view_{v_l}|$ (line 04, Figure 4.5).
At round r_1, processes in C_0 send a MY_COLOR(c_0) message inducing the coloring of their non-colored neighbors that will belong to set C_1. Let us note that such processes will belong to the color views of processes in C_0 as they will send, during the send phase of r_1, a MY_COLOR($\perp$) message. Thus, in order to count the size of C_1, the leader has to collect color views at round r_1 of any process in C_0 as such color views will contain at least one $\perp$ value for each process colored during round r_1. Coming back to the example of Figure 4.6(a), the set $C_1 = \{v_4, v_5, v_6\}$ and the color views of v_1, v_2 and v_3 are respectively $color\_view_{v_1} = \{c_0, \perp, \perp\}$, $color\_view_{v_2} = \{c_0, \perp, \perp\}$ and $color\_view_{v_3} = \{c_0, \perp, \perp\}$. Let us note that making the union of all the color views collected at round r_1 and counting the number of $\perp$ provides an upper bound on the real number of processes that have been colored with color c_1 ($c_1 =$ blue in the example) as each non-colored processes may be neighbor of more than one colored process. In addition, note that there may exist different processes with the same color view that should be distinguished by the leader. Thus, to count how many processes have the same color view, each process generates an energy charge associated with (i) its color, (ii) the current round and (iii) its color view. From now on, each colored process v_i at any round r_j will generate one energy charge associated with its color c_k, the round r_j and the view $color\_view_{v_i}$ at that round. This energy charge will be stored in the $energy\_set_{v_i}$ and disseminated starting from the following round r_{j+1} (line 20, Figure 4.4). Let us consider the energy spreading in a generic round r_i. At the beginning of r_i, each process v_i updates its $n\_size_{v_i}$ variable with the value provided by the oracle and then the send phase starts (line 02, Figure 4.4). During the send phase, non-leader nodes start distributing colored energy stored in their *energy_set* variable to neighbors by sending a ENERGY message (line 09, Figure 4.4). More in details, for each tuple $< e, color\_view_{v_i}, color_{v_i}, r' >$ belonging to $energy\_set_{v_i}$, v_i broadcasts a ENERGY($\frac{e}{2*n\_size_{v_i}}, color\_view_{v_i}, color_{v_i}, r'$) message and updates the set halving the energy for each tuple. Let us remark that the $\mathcal{O}^P$ oracle provides the exact number of neighbors to each process v_i. Thus, v_i will send, to each of its $n\_size_{v_i}$ neighbors, a charge $\frac{e}{2*n\_size_{v_i}}$ and the sums of all the energy distributed will be exactly half of the energy initially stored in each tuple of the energy set. Receiving a ENERGY($\frac{e}{2*n\_size_{v_i}}, color\_view_{v_i}, color_{v_i}, r'$) message, any node v_j updates its $energy\_set_{v_j}$ as follows: (i) if there not exists a tuple in $energy\_set_{v_j}$ with $color\_view_{v_i}, color_{v_i}, r'$, v_j simply adds the received tuple to the energy set (line

11. Figure 4.4), (ii) otherwise it updates the tuple already stored in $energy\_set_{v_j}$ by adding the energy received in ENERGY($\frac{e}{2*n\_size_{v_i}}, color\_view_{v_i}, color_{v_i}, r'$) (lines 12-13. Figure 4.4).

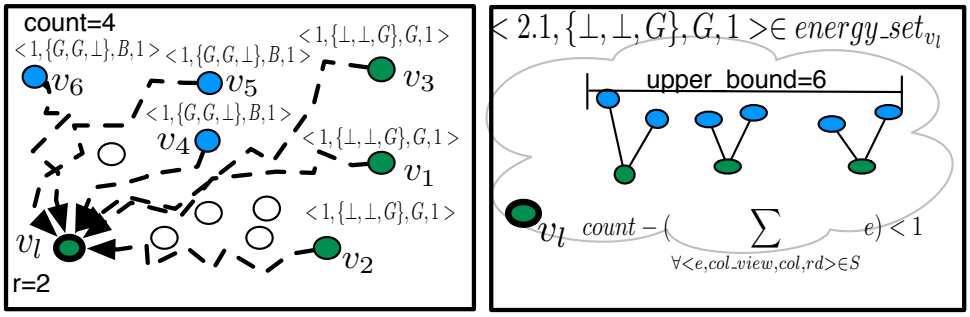
Let us now consider what happens at the leader side. The leader v_l is an energy sink and, receiving ENERGY messages from its neighbors, it behaves as a generic node storing the tuple in its $energy\_set_{v_l}$ variable. In addition, at the end of each round, it checks if it has collected enough information to count a new subset of colored processes or to terminate the count. In the computation phase of each round r_i, the leader computes two sets S and F defined as follows:

- $S = \{< e, cv, col, r' > \in energy\_set_{v_l} \mid col \leq lcc \wedge r' = lcc + 1\}$ is the sub-set of energy charges created at round $r' = lcc + 1$ and coming from nodes colored at some round $r \leq lcc$. Such set represents all energy charges collected by v_l from nodes in $C_1 \cup C_2 \cup ... \cup C_{lcc}$ (line 05, Figure 4.5).
- $F = \{< e, cv, col, r' > \in energy\_set_{v_l} \mid col = lcc + 1 \wedge r' = lcc + 1\}$ is the sub-set of energy charges created at round $r' = lcc + 1$ and coming from nodes colored at round $r' = lcc + 1$. Such set represents all energy charges collected by v_l from nodes in C_{lcc+1} (line 06, Figure 4.5).

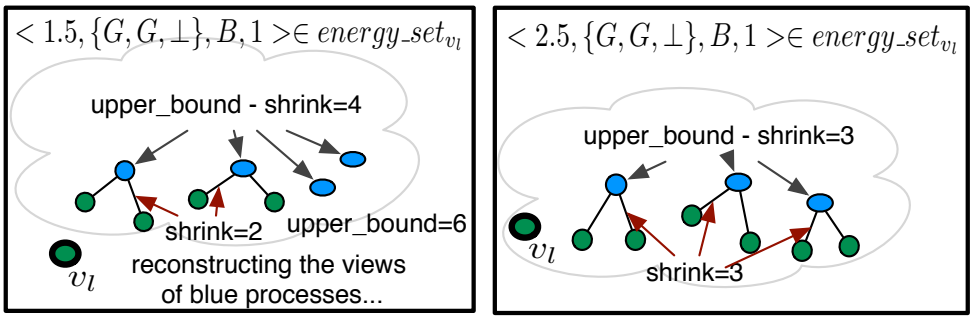
Once the leader computed such sets, it checks the condition in line 07, Figure 4.5 to verify that it waited long enough to receive enough energy charges from all the nodes that have been colored before round r_{lcc} to correctly count the number of nodes with the same color view. When this happens, the condition $count - (\sum_{\forall < e, color\_view, col, r' > \in S} e) < 1$ is satisfied. Then, the leader checks color views just collected to verify if there still exist nodes with color $\perp$ and if all the color views contains only colored nodes the counting terminates (line 09, Figure 4.5). Otherwise the leader counts the number of nodes with color $\perp$ in the color views collected (line 10, Figure 4.5). Such number is an *upper_bound* on nodes in C_{lcc+1}. So, v_l sets $upper\_bound \leftarrow \sum_{\forall < e', color\_view, col, r' > \in S} \lceil e' \rceil \times n^{\perp}_{lv}$, the rounded energy $\lceil e' \rceil$ is the actual number of processes in $C_{\leq lcc}$ with color *col* and view *color_view* in round r_{lcc+1}, where $C_{\leq lcc}$ is the union of all the sets C_i with $i \leq lcc$.
The bound is then used to check when the leader has collected enough energy from processes in C_{lcc+1} to let him able to obtain the correct count of these processes.
This is done by considering the set of tuple contained in F. By means of these color views, in fact, v_l detects if at round r_{lcc+1} there exists a node $v_j \in C_{lcc+1}$ that has been colored by multiples nodes in $C_{\leq lcc}$. If this happened, in the color view of v_j we will find more than one process with color less or equal to c_{lcc}. In this case v_l will lower the upper bound on $|C_{lcc+1}|$ until it matches the actual size. For each tuple $< e', color\_view, c_{lcc+1}, r_{lcc+1} >$ in F the leader counts the number of color c_x with $x < lcc + 1$, namely $n^{lcc}_{color\_view}$. If this number is greater than one, we have that a process in C_{lcc+1} with color view *color_view* at round r_{lcc+1} have been counted $n^{lcc}_{color\_view} - 1$ times more in the *upper_bound*. The leader computes the quantity *shrink* that is the surplus of times nodes in C_{lcc+1} have been counted in *upper_bound*. As last step the leader verifies if it has received all the color views from processes in C_{lcc+1} with the correct multiplicities (condition in line 12, Figure 4.5). When this condition is satisfied the leader updates *count* with the size of C_{lcc+1} (line 13, Figure 4.5) and updates *lcc* to $lcc = lcc + 1$ to move the count to the next set



(a) Coloring of nodes starting from round r_0 to round r_1



(b) Energy-Transfer from nodes to leader in order to construct the views of processes in C_0at round r_1 inside the leader v_l



(c) Energy-Transfer from nodes to leader in order to construct the views of processes in C_1 at round r_1 inside the leader v_l, when these views are correctly recovered the leader is able to count processes in C_1

Figure 4.6: Example of the algorithm execution.

of colored processes, if any (line 14, Figure 4.5). This procedure is iterated until all processes are colored and counted.
Let us consider again the example in Figure 4.6. In Figures 4.6(b) and 4.6(c) the leader, receiving energy from processes in C_0, C_1, obtains $|C_1|$. In details in Figure 4.6(b) left, processes in C_0 and C_1 start to send energy to the leader by means of energy-transfer technique. Each process attaches to the unitary charge of energy its color view at round r_1. It is possible to see that there are many processes with the same color view (i.e. v_1, v_2, v_3 and all processes in C_1, i.e. v_4, v_5, v_6). The leader will identify this multiplicity by collecting energy. The leader waits until energy collected from processes in C_0 is greater than $|C_0| - 1$. This can be done since the leader knows the number of processes in C_0. When this happens, (Figure 4.6(b) right), the condition $count - (\sum_{\forall <e,color\_view,col,r'> \in S} e) < 1)$ is satisfied. Then the leader computes an upper bound on C_1 by counting the number of $\perp$ in the views of processes in C_0. In this example, we have a total of six $\perp$ ($\{\perp, \perp, v_l\}, 2 \times \{\perp, \perp, G\}$), so the leader sets $upper\_bound = 6$. Now the leader has to collect energy from at most six blue nodes. During the collection of energy from nodes in C_1, it dynamically adjusts the upper bound until it matches the real size of C_1. In Figure 4.6(c) left, the leader collected enough energy to recognize two sources for view $\{G, G, \perp\}$; this means that two processes in C_1 have been counted twice in the upper bound because each of them is present in the color view of two processes in C_0. So each of these views brings 1 unit of shrink represented by the edges that are pointed by the red arrows in the figure. So the leader computes the new upper bound $upper\_bound - shrink = 4$ and it waits to collect energy from 4 blue processes. In Figure 4.6(c) right, the leader collected enough energy to detect that there are 3 processes in C_1 with view $\{G, G, \perp\}$, so it updates the shrink to 3 and updates the upper bound to $upper\_bound - shrink = 3$, but this value is equal to the number of processes in C_1 from which the leader has received energy. Thus, v_l detects this condition and updates the count to $count = count + 3$ and lcc to $lcc = 1$. We can see that the leader has the count of $C_0 \cup C_1$.

Correctness

For ease of presentation, let us define E^r as the sum of energy charges generated at round r (i.e. all the energy charges created while executing line 20, Figure 4.4 at round r). In the next two Lemmas we show that: (i) the energy transfer mechanism does not duplicate or lose energy while moving energy from a node $v \in V \setminus \{v_l\}$ to v_l; this allows us to associate information related to color views with the energy that will be transmitted to v_l without duplication and (ii) the quantity of energy inside the leader set $energy\_set_{v_l}$ is a monotonically increasing function in the number of rounds.

Lemma 31. Conservation of Energy: *In each execution of $\mathcal{A}_{\mathcal{O}^P}$, $\forall r' > r$, E^r is an invariant.*

Proof. The proof is essentially the same of Lemma 28, thus it is omitted. □

Lemma 32. *In each execution of $\mathcal{A}_{\mathcal{O}^P}$, $\forall r'$ where r' is a round such that $r' > r + |V| - 1$. The energy in the form $< *, *, *, r >$ (that is the energy generated at round r) in $energy\_set_{v_l}$ of the leader is an increasing quantity that converges to E^r.*

Proof. The proof derives directly from Lemma 29, thus it is omitted. □

In order to prove that $\mathcal{A}_{\mathcal{O}^P}$ is correct, we show that, starting from round r_0, the leader is able to (i) compute an upper bound on C_1 in a finite time (Lemmas 33,34) and (ii) compute the actual cardinality of $|C_1|$ in a finite time. So, in the next lemmas, we show that the leader is able to count the number of processes in C_1 and if $C_1 = \emptyset$ to terminate outputting the actual count.

Lemma 33. *Let us consider a run of Algorithm $\mathcal{A}_{\mathcal{O}^P}$. The condition at line 07, Figure 4.5 will be satisfied at least once after a finite number of rounds.*

Proof. At round r_0, the leader v_l colors a set of nodes $C_0 = N(v_l, 0) \cup \{v_l\}$ (line 02, Figure 4.5 and lines 17 and 19, Figure 4.4). Please note that the leader knows the size of C_0 and will set the variable *count* to $|C_0| = |N(v_l, 0)| + 1$ (line 04, Figure 4.5). Moreover, each process v_j in C_0 will put in its $energy\_set_{v_j}$ a charge $< 1, color\_view_{v_j}, c_0, r_1 >$ (line 20, Figure 4.4). Each process in $V \setminus \{v_l\}$ broadcasts at each round r, for each energy charge in *energy_set*, half of the stored energy (lines 04-05, Figure 4.4). This implies that, for 1-interval connectivity and due to Lemma 32, energy charges with attributes $color = c_0$ and round r_1 will start flowing from each $v_j \in C_0 \setminus \{v_l\}$ and eventually they will reach the leader. The energy accumulated by the leader with attributes $color = c_0$ and round r_1, in the worst-case, follows a monotonic increasing behavior (see Lemmas 31-32). So there exists a round r after which the leader receives $|C_0| - 1 + \epsilon$ energy with $\epsilon > 0$ and the claim follows. □

Lemma 34. *Let us consider a run of Algorithm $\mathcal{A}_{\mathcal{O}^P}$. When the condition at line 07, Figure 4.5 is satisfied and $lcc = 0$, then $upper\_bound \geq |C_1|$.*

Proof. When the condition at line 07, Figure 4.5 is satisfied, and $lcc = 0$, we have that the leader received a quantity $|C_0| - 1 + \epsilon$ of energy (with $\epsilon > 0$) associated to tuples with $color = c_0$ and round r_1. This means that v_l has received at least a fraction ϵ' of energy from each $v_j \in C_0$. So v_l has collected all color views $color\_view_{v_j}$ generated at round r_1 from nodes in C_0. Let $C_0^{color\_view'_v(r_1)} \subseteq C_0$ be the subset of nodes in C_0 with color view equal to $color\_view'_v$ at round r_1. In order to prove the existence of the upper bound, we first need to prove that for each color view $color\_view'$ of processes in C_0, the energy generated and transferred to the leaser is equal to the number of processes having the same color view $color\_view'$, i.e. we have to prove that $m_{color\_view'} = \sum_{\forall < e', color\_view'_v, c_0, 1> \in S} \lceil e' \rceil$ is equal to $|C_0^{color\_view'_v(r_1)}|$.

Let us suppose by contradiction that $m_{color\_view'} \neq |C_0^{color\_view'_v(r_1)}|$. Obviously it is not possible that $m_{color\_view'} > |C_0^{color\_view'_v(r_1)}|$ as this would means that the leader received more energy than the energy present in the system but this is not possible due to Lemma 28. So it is only possible to have $m_{color\_view'} < |C_0^{color\_view'_v(r_1)}|$. Let us define $e_{v_i}^{v_l}$ as the energy with color c_0 transferred from v_i to v_l and round r_1. Then the previous condition, reported in a system of inequalities is:

$$\begin{cases} \sum_{\forall v_i \in C_0} e_{v_i}^{v_l} \geq |C_0| - 1 + \epsilon & (4.1) \\ \sum_{\forall v_i \in C_0^{color\_view'_v(r_1)}} e_{v_i}^{v_l} \leq |C_0^{color\_view'_v(r_1)}| - 1 & (4.2) \\ \forall v_i \in C_0 : 0 \leq e_{v_i}^{v_l} \leq 1 & (4.3) \\ \epsilon > 0 & (4.4) \end{cases}$$

where inequalities (1) and (4) are essentially a rewriting of the condition in line 07, Figure 4.5. Inequality (3) holds by construction of the energy-transfer mechanism since each process in C_0 during the receive phase of round r_1 creates only one energy charge with attribute round r_1 (cfr. Lemma 28). In addition, by hypothesis, equation (2) holds (that is a transposition of $m_{color\_view'} < |C_0^{color\_view'_v(r_1)}|$).

From inequalities (1) and (2) we have

$$\sum_{\forall v_i \in C_0 \setminus C_0^{color\_view'_v(r_1)}} e_{v_i}^{v_l} \geq |C_0| - |C_0^{color\_view'_v(r_1)}| + \epsilon$$

that violates the energy conservation Lemma, since this would implies that the leader would collect, from the set $C_0 \setminus C_0^{color\_view'_v(r_1)}$, strictly more energy (since $\epsilon > 0$) than the energy present in this set of nodes, this is an absurd. So, when the condition in line 07, Figure 4.5 is satisfied, we necessary have that $m_{color\_view'} = |C_0^{color\_view'_v(r_1)}|$.

The leader sets, executing line 10 in Figure 4.5, the upper bound on C_1 to

$$upper\_bound \leftarrow \sum_{\forall <e', color\_view, col, r'> \in S} \lceil e' \rceil \times n^{\perp}_{color\_view}$$

that means

$$upper\_bound \leftarrow \sum_{\forall <e', color\_view, col, r'> \in S} m_{color\_view} \times n^{\perp}_{color\_view}$$

that is equivalent to

$$upper\_bound \leftarrow \sum_{\forall <e', color\_view, col, r'> \in S} |C_0^{color\_view_v(r_1)}| \times n^{\perp}_{color\_view}$$

.

Let us note that the right term of the last inequality is essentially the number of edges at round r_1 from nodes in C_0 to non-colored nodes, that is obviously an upper bound on C_1. □

Lemma 35. *Let us consider a run of Algorithm $\mathcal{A}_{\mathcal{O}^P}$. If the condition at line 08, Figure 4.5 is satisfied and $lcc = 0$ then $C_1 = \emptyset$ and $count = |V|$.*

Proof. In the proof of Lemma 34 we shown that the computed upper bound is actually an upper bound on the number of edges from nodes in C_0 to non-colored nodes. Due to 1-interval connectivity, the network is always connected. Thus, if there not exists any $\perp$ in any color views it means that there not exists any non-colored node. Thus, if this number is equal to 0 we have $V \setminus C_0 = \emptyset$ and the claim follows. □

The next lemma shows that we can count the processes in C_1.

Lemma 36. *Let us consider a run of Algorithm $\mathcal{A}_{\mathcal{O}^P}$. If the condition at line 12, Figure 4.5 is satisfied and $lcc = 0$ then $|C_1| = \lceil \sum_{\forall <e',lv,col,r'> \in F} e' \rceil$. Moreover if $|C_1| > 0$ then the condition at line 12, Figure 4.5 will be eventually satisfied in a finite number of rounds.*

Proof. In order to evaluate the condition at line 12, Figure 4.5 , the leader computes the *shrink*, i.e. it counts the number of edges from nodes in C_1 to nodes in C_0 at round r_1. This counting is done similarly to what the leader does with color views of processes in C_0 (see Lemma 34). So, if a color view $color\_view_{v_j}$, with $v_j \in C_1$, contains a certain number n_{c_0} of color c_0 then v_j has been counted in *upper_bound* $n_{c_0} - 1$ times more than necessary. So, when the leader checks the condition at line 12, Figure 4.5 it adjusts the count of *upper_bound* subtracting the *shrink*. Let us remark that in this way $upper\_bound - shrink$ is still an upper bound on $|C_1|$. This follows by considering that

$$shrink \leftarrow \sum_{\forall <e',color\_view,col,r'> \in F} \lceil e' \rceil \times (n^{lcc}_{color\_view} - 1)$$

is equals, for each received *color_view*, to the number of edges between processes in C_1 and C_0 minus one, and $\lceil e' \rceil$ converges to the number of processes in C_1 at round r_1 with view *color_view*. So $upper\_bound - shrink$ is still an upper bound since it will contains at least one counted edge for each process in C_1.
When the condition at line 12, Figure 4.5 is satisfied we have that the difference between $upper\_bound - shrink$ and the energy collected from nodes in $|C_1|$ is smaller than one. Since $upper\_bound - shrink$ and $\lceil \sum_{\forall <e',lv,col,r'> \in F} e' \rceil$ are respectively an upper/lower bound on $|C_1|$ we have that $|C_1| = \lceil \sum_{\forall <e',lv,col,r'> \in F} e' \rceil$.
Let us note that, due to Lemma 28 $\lceil \sum_{\forall <e',lv,col,r'> \in F} e' \rceil$ is a lower bound on $|C_1|$ since the energy accumulated in the leader from the set C_1 at round r_1 cannot be more than $|C_1|$.
Due to Lemma 32 the leader eventually collects from nodes in $|C_1|$ a quantity of energy such that $|C_1| - \sum_{\forall <e',lv,col,r'> \in F} e' < 1$ and this will happen in a finite number of rounds. When this happens □

Theorem 11 (Convergence)**.** *$\mathcal{A}_{\mathcal{O}^P}$ converges to a correct count of the network size.*

Proof. For 1-interval connectivity we have that after at most $|V| - 1$ rounds all processes will be colored, this processes are partitioned in sets $C_0, C_1, C_2, ..., C_k$ with $k \leq |V| - 1$. We show by induction that the leader correctly counts each $|C_j|$ with $0 \leq j \leq k$.
Basic Step: It follows from Lemmas 31-36.
Inductive Step: Let us assume that the algorithm correctly counted the subset of processes in $C_0 \cup C_1 \cup \cdots \cup C_i$ and let us show that it counts also C_{i+1} (if any). From a logical point of view, when the leader updates *lcc* to i, it is equivalent to create a set of colored nodes $C'_{\leq i} = C_0 \cup C_1 \cup \cdots \cup C_i$. It easy to see that each one of the previous lemma still applies for $C'_{\leq i}$. Thus, the leader will end up with the correct count of $|C_{i+1}|$. When $lcc = k$ (i.e., the leader has counted all process), we have that it does not change the value of *count* before terminating, since all nodes have been

colored, so condition **if**($\nexists < e', color\_view, col, r' > \in S \mid \exists \perp \in color\_view$) cannot be false this implies that the updates of count cannot be triggered. □

Theorem 12 (Termination)**.** *There exists a round r in which the leader knows that its guess corresponds to the correct count.*

Proof. Let us recall from the previous proof that when $count = |V|$ we have that the set of processes is partitioned in sets $C_0, C_1, C_2, ..., C_k$ with $k \leq |V| - 1$ and that $lcc = k$. Since $lcc = k$ the leader collects all the views from processes in $C_0 \cup C_1 \cup ... \cup C_k$ in the set $S = \{< e, color\_view, col, r' > \in energy\_set_{v_l} \mid col \leq lcc \wedge r' = lcc + 1\}$. For Lemma 28, 32 this will happen and when it happens the condition at line 07, Figure 4.5 has to be satisfied. Moreover, since all processes have been colored we have that it cannot exist a process with $\perp$ in its color view. This implies that condition of line 08 is satisfied so the algorithm correctly terminates in a round r for which $count = |V|$, thus the algorithm terminates.

□

The counting algorithm $\mathcal{A}_{\mathcal{O}^{OE}}$

By inspection of the correctness proof of the previous section, we can notice that the knowledge encapsulated within $\mathcal{O}^P$ is used only in the proof of Lemma 28. This Lemma is necessary since it ensures that the quantity of energy in the system is always equal to the number of sources, this allow us to prove other Lemmas, in which the leader has to converge to the number of sources that are pushing a certain token t. It is then possible to modify the algorithm shown in Figures 4.4,4.5 ensuring that Lemma 31 still holds even though processes can access an oracle $\mathcal{O}^{OE}$. This can be done as we did in the Algorithm of Section 4.1.

From $\mathcal{A}_{\mathcal{O}^P}$ to the practical algorithm $\mathcal{A}^*_{\mathcal{O}^P}$

$\mathcal{A}_{\mathcal{O}^P}$ is able to provide an exact count in a finite time. However, it may take a large amount of time since the leader may not be able to count two nodes with the same color until it collects more than one unit of energy from them. Let us assume for example that there are y nodes that at round r have the same color and the same multi set of neighbors. The leader has to collect at least $y - 1 + \epsilon$ energy to count the correct multiplicity y. However, in practice such two nodes may be identified if we consider the history of their local views, i.e., the union of all the multi-sets they saw from round r_0 until the current round.

Based on this intuition, we defined a new algorithm, namely $\mathcal{A}^*_{\mathcal{O}^P}$, that takes advantage of symmetry breaking introduced by different local histories. Essentially, each non-leader process with color $c_i \neq \perp$ computes, at each round r, a round id $rid_r = CriptoHash(rid_{r-1} + color\_view(r-1))$; in this way, two nodes v, u that at round r' have a different multi set of neighbors will have, with high probability the hashing function is collision resistant, for each round $r > r'$ a different rid_r.

Such rid_r will be attached together with the other information to the energy created at round r. The rid_r influences only the collection of energy generated from round r. Let us notice that in the unlikely event of a collision of the hashing function the correctness of the algorithm is not impaired.

With this modification the algorithm uses the concept of energy to count when the symmetry has not been broken by the dynamic topology, i.e., two nodes that at each rounds have the same neighborhood. Otherwise it can count fast, let us suppose that at round r all nodes with $c_{id} \neq \perp$ have a different rid_r, this implies that the leader terminates the energy collection in at most V rounds. Considering the previous example if $\frac{y}{2}$ nodes have different rid_r the leader have to wait till it collects $\frac{y}{2} - 1 + \epsilon$ of energy that is in general faster.

In the Evalutation Section, at the end of the Chapter, we have evaluated both $\mathcal{A}_{\mathcal{O}^P}$ and $\mathcal{A}^*_{\mathcal{O}^P}$. From our tests we have that $\mathcal{A}^*_{\mathcal{O}^P}$ on graphs generated by a random adversary terminates roughly in a number of rounds that is less than twice the size of the network.

In this Section we study the problem of counting when the communication model is labeled in-edge. That is: when v receives at a certain round r a message m from nodes w it also receives an ID $in_{v,w}$, this ID is fixed and do not changes among rounds, moreover the ids are uniques. Node v will see ID $in_{v,w}$ if and only if the message is from w. Let us recall that the labels can be seen only at receive sides and are locally uniques, this means that could be another node $v' \neq v$ that associates to messages coming from w the same label $in_{v,w}$.

This implies that, in contrast to out-edge labeling [], the presence of the leader does not imply the possibility of solving the naming problem.

However, under this communication model is possible to solve the counting problem without additional assumption.

Building an $\mathcal{O}^{OE}$ when there is In-edge labeling

```
1: U^v = ∅, pending_send = ⊥
2: procedure TRY_TO_SEND(m)
3:     if pending_send = ⊥ then
4:         pending_send = m
5:
6: procedure SENDING_PHASE
7:     if pending_send ≠ ⊥ then
8:         send(< pending_send, |U^v| >)
9:
10: procedure RCV_PHASE(Set M : {< payload, n, id >})
11:     ID = ∪_{∀m∈M} {m.id}
12:     if pending_send ≠ ⊥ ∧ ID ∩ U^v ≠ ∅ then
13:         delivered(< m, |U^v|, |ID ∩ U^v| >)
14:         pending_send = ⊥
15:     for all < payload, n, id > ∈ M do
16:         if id ∈ U^v then
17:             delivery(< payload, n, id >)
18:     U^v = ID
```

Figure 4.7: Eventual Delivery with Upper Bound Algorithm for the In-Edge labeling Model

Let us assume that the communication model is labeled in-edge: when v receives at a certain round r a message m from nodes w it also receives an ID $in_{v,w}$, this ID

is fixed and it does not change among rounds. Moreover the ids are locally uniques. Node v will see ID $in_{v,w}$ if and only if the message is from w. Let us recall that the labels can be seen only at receive sides and are locally uniques. Therefore it could exist another node $v' \neq v$ that associates to messages coming from w the same label $in_{v,w}$.

From this observation is easy to see that in contrast to out-edge labeling, the each-to-each model of [59], the presence of the leader does not imply the possibility of solving the naming problem.

The algorithm The idea is to implement an algorithm, **Eventual Delivery with Upper Bound**, that eventually allows each node to send messages to a set of neighbors for which an upper bound is known. Let us define as ID_i^v the sets of ID attached to messages received by node v at round i. Let us define $U_i^v = \cup_{j=0}^{j=i} ID_j^v$. In order to send message m at round r, the node v sends $< m, |U_{r-1}^v| >$, the receiving neighbor w will receive the message $< m, |U_{r-1}^v|, in_{w,v} >$.

Now w checks if it has heard from v before, that is $in_{w,v} \in U_{r-1}^w$, in this case w deliver m to the upper layer associated with $|U_{r-1}^v|$ that is the upper bound on the number of nodes that may deliver m at round r.

At the end of the receive phase of round r, v checks if $|ID_r^v \cap U_{r-1}^v| \neq 0$ in this case, v knows that m has been delivered by exactly $|ID_r^v \cap U_{r-1}^v|$ nodes with an upper bound information associated that is $|U_{r-1}^v|$.

The detailed pseudocode of the algorithm can be found in Figure 4.7

Analysis: It is straightforward to see that by construction the algorithm ensures that U_{r-1}^v is an upper bound on the nodes that may deliver a message from v at round $r-1$, since a node delivers a message m from v if and only if it has already received a message from v in a previous round, and thanks to the local in edge labeling no ambiguity may arise.

Given a node v that try to send message m at round r', we have that the delivery of m could be delayed: m is not delivered if at round r we have that $|ID_r^v \cap U_{r-1}^v| = 0$. Therefore, in order to block a message m the adversary has to give v a new set of neighbors at each round. However there are only $|V|$ nodes thus the message m has to be delivered in the worst case by round $r' + |V| + 1$. Therefore v will send a messages m to all other nodes in at most $|V|^2$ rounds.

Under this communication procedure we can adapt the proof of Lemma 29 to show that it holds also in this case, taking into a count a multiplicative factor of $|V|$. Therefore it is possible to implement $\mathcal{A}_{\mathcal{O}^{OE}}$ in this model.

4.3 Convergent Counting Algorithm using NoK

In this section we propose an algorithm, namely $\mathcal{A}_{NoK}$, that is able to eventually count the number of processes in the dynamic anonymous network without having any assumption on the network. $\mathcal{A}_{NoK}$ is actually a modification of the algorithm presented in Section 4.1 where nodes have to cope with the uncertainty about the number of neighbors (and thus, with the uncertainty about the energy they have to release in each round).

$\mathcal{A}_{NoK}$ works in the following way: each non-leader node v_i starts, at round r_0, with

energy quantity $e_{v_i} \leftarrow 1$ and it transfers half of its current energy to the neighbors. However, v_i has no knowledge about the network and thus it cannot know the exact number of neighbors at round r before receiving messages, but it can only guess such number. Thus, v_i supposes to have D_{max} neighbors and it broadcasts a quantity of energy $\frac{1}{2D_{max}}$ (as if there are really D_{max} neighbors). Then v_i starts to collect messages transmitted by its neighbors at the beginning of the round and it stores such messages in a local variable rcv_{v_i}. At the end of the round, v_i updates its energy, as in the previous algorithm, to preserve the quantity of energy over all the network.

Notice that, if the real number of neighbors at round r is lower than the estimation (i.e., $|rcv_{v_i}| \leq D_{max}$) then the global energy conserved among all the processes is still constant (this is due to the compensation done by v_i at the end of the round, based on the effective number of received messages). On the contrary, if the number of neighbors is greater than the estimation (i.e., $|rcv_{v_i}| > D_{max}$) then, there is the release of a *local surplus* of energy. As an example, consider the case where v_i has energy e_{v_i} the estimation of neighbors is $D_{max} = 2$ and the real number of neighbors is $|rcv_{v_i}| = 8$. When v_i sends $\frac{e_{v_i}}{4}$ to each neighbors, the total amount of energy transferred is twice the energy stored by v_i (i.e., the energy transferred is $8 \times \frac{e_{v_i}}{4} = 2e_{v_i}$ while node v_i had only e_{v_i} residual energy). However, since v_i adjusts its local residual energy considering the number of received messages $|rcv_{v_i}|$, it follows that its residual energy will become negative and globally the energy is still preserved. Considering the previous example at the end of the round v_i stores a residual energy $-e_{v_i}$.
Unfortunately, the local surplus of positive/negative energy could create, in the leader, a temporary value of energy e_{v_l} that is greater than $|V|$ or negative. Moreover, the adversary could change, at each round, the degree of nodes in order to avoid the convergence of the leader. To overcome these issues each processes stores locally the highest number of neighbors it has ever seen and it uses the double of such number as D_{max}. In this way, the surplus of local negative/positive energy that the adversary can create is upper bounded by a function $f(|V|)$: each node v_i can increase D_{max} at most $log(|V|)$ times, from 1 to $|V|$. This implies that the worst case adversary cannot create an infinite surplus of local energy. Thus, the adversary could delay the convergence of the count only a finite number of rounds.

Algorithm $\mathcal{A}_{NoK}$ – Convergent Counting without Additional Knowledge

leader : The leader v_l maintains the following local variables: (i) $e_{v_l} \leftarrow 1$ representing the initial energy charge and (ii) $rcv_{v_l} \leftarrow \emptyset$ is a set variable where v_l stores all the messages received during each round. (iii) $count_{v_l} \leftarrow 0$ this variable is used to keep trace of the latest estimated count.
for each round r:

- **Send phase of round** r**:** at the beginning of each round, v_l broadcasts a ENERGY_RELEASE(0) message releasing no energy.
  v_l broadcasts COUNT($\lceil e_{v_l} \rceil, r$).

- **Receive and Computation phases of round** r**:** ENERGY_RELEASE(e') messages are stored in the rcv_{v_l} variable. At the end of the round, when all the messages have been received, v_l updates, its local energy charge as follows:

$$e_{v_l} \leftarrow e_{v_l} + \sum_{e' \in rcv_{v_l}} e'$$

  v_l sets $count_{v_l} \approx e_{v_l}$

anonymous node: Each non leader node v_i maintains the following local variables: (i) $e_{v_i} \leftarrow 1$ representing the initial energy charge of v_i, (ii) $rcv_{v_i} \leftarrow \emptyset$ is a set variable where v_i stores all the messages received during each round and (iii) $D_{max} \leftarrow 1$ is an integer variable storing the maximum number of neighbor v_i has ever had (initially set to 1 as it has no knowledge about the network). (iv) $count_{v_i} \leftarrow 0$ this variable is used to keep trace of the latest count estimate seen by the node (v) $r\_count_{v_i} \leftarrow 0$ is the round number associated with the latest count accepted.
for each round r:

- **Send phase of round** r**:** at the beginning of each round, v_i broadcasts a ENERGY_RELEASE($\frac{e_{v_i}}{2D_{max}}$) message, releasing at most half of its energy to its neighbors.
- **Receive and Computation phases of round** r**:**

  for each message $m \in rcv_{v_i}$ such that m is an ENERGY_RELEASE message, v_i updates its local energy charge as in the previous algorithm:

$$e_{v_i} \leftarrow e_{v_i} - (D_{max} \times \frac{e_{v_i}}{2D_{max}}) + ((D_{max} - |rcv_{v_i}|) \times \frac{e_{v_i}}{2D_{max}} + \sum_{e' \in rcv_{v_i}} e')$$

  In addition, if $|rcv_{v_i}| > D_{max}$, v_i also updates the maximum number of neighbors it has ever saw by setting $D_{max} \leftarrow 2|rcv_{v_i}|$.
  for each message $m \in rcv_{v_i}$ such that m is an COUNT message, if the round number in m is greater than $r\_count_{v_i}$ updates $count_{v_i}$ and $r\_count_{v_i}$ with the content of m.

Correctness Proofs. In the following, we will prove that protocol $\mathcal{A}_{NoK}$ converges to the exact count in a finite number of rounds. We first prove that the quantitive of negative energy that the dynamic adversary is able to create is bounded by a function of the network size $|V|$ (Lemma 37). From this result, we prove that the leader will obtain the correct count in a finite but unknown number of rounds (Lemma 39), so we will prove that $\mathcal{A}_{NoK}$ is convergent. Let us notice that since the energy transfer mechanism is the same of $\mathcal{A}_D$ the global invariant on energy still holds, so Lemma 28 keeps holding for $\mathcal{A}_{NoK}$ too. Moreover, it is clear that for each round r the leader will receive energy from its neighbors. So, even if it is not possible to have a bound like the one obtained in Lemma 29, it is straightforward to see that the absolute value of the sum of energy that the leader receives is a monotonically increasing function.

Lemma 37. *Let $G \in \mathcal{G}$(1-IC). During any execution of $\mathcal{A}_{NoK}$ on G, the amount of negative energy that can be generated is finite. Moreover, during any round r a single node $v_i \in V$ with energy $e^r_{v_i}$ can create at most $(|V| - 2)e^r_{v_i}$ negative energy.*

Proof. Let us focus on the negative energy that can be generated during the execution of the algorithm by a generic node $v_i \in V$. Let $E_n : \{r_{i,1}, r_{i,2}, ..., r_{i,t}\}$ be the set of rounds (not necessarily consecutive) in which v_i creates negative energy; we will show that the number t is finite. For the generic round $r_{i,j} \in E_n$ we must have that $|rcv_{v_i}| > 2D_{max}$; this follows by imposing $\frac{e}{2} + (|rcv_{v_i}| - D_{max})\frac{e}{2} < 0$. This means that we have at least $|rcv_i| \geq 2D_{max} + 1$, and at the end of round $r_{i,j}$ the number D_{max} will be doubled. Since $|rcv_i| \leq |V|$ and $D_{max} \geq 1$, we have that the condition $|rcv_i| > 2D_{max}$ could happen at most $log(|V|)$ times. So, $t \leq O(log(|V|))$. Moreover, the negative energy created by a node v_i with energy e during a single round is at most $(|rcv_{v_i}| - 2D_{max})e$ that is maximized when $|rcv_{v_i}| = |V|$ and $D_{max} = 1$. □

Lemma 38. *Let $G \in \mathcal{G}(1\text{-}IC)$. During any execution of $\mathcal{A}_{NoK}$ on G, for any $\epsilon \in \mathbb{R}^+$ there exists a round $r_{|V,\epsilon|}$ after which the amount of negative energy that could be transferred to the leader in the following rounds is less than ϵ.*

Proof. Due to the invariant on energy (Lemma 28), we have that the negative energy that each node could create is related to the energy that the nodes already possess (since the sum of negative and positive energy has to be equal zero). In addition, for Lemma 28 and for the monotonically increasing energy received by the leader, we have that, if no negative energy is created, the absolute value of energy in $V \setminus \{v_l\}$ is a monotonically decreasing function of r. The maximum amount of negative energy that can be created starting from round r is bounded (see lemma 37) and from the previous consideration it is a monotonic function f of $\sum_{\forall v \in V \setminus \{v_l\}} |e_v^r|$. From Lemma 29 we have that $\forall \epsilon_1 \in \mathbb{R}^+ \; \exists r_{\epsilon_1} \in \mathbb{N}^+ | \; \sum_{\forall v \in V \setminus \{v_l\}} |e_v^{r_{\epsilon_1}}| \leq \epsilon_1$, since f is monotonic exists ϵ_1 such that $f(\epsilon_1) \leq \epsilon$. □

Lemma 39. *Let $G \in \mathcal{G}(1\text{-}IC)$. Let us consider an execution of $\mathcal{A}_{NoK}$ on G. Then, $\forall v_i \; \lim_{r\to\infty} count_{v_i} = |V|$. So $\mathcal{A}_{NoK}$ is a convergent counting algorithm.*

Proof. Lemma 38 shows that there exists a round $r_{|V,\epsilon|}$ after which the maximal amount of negative energy that can be created in the network is bounded by a quantity ϵ that can be made arbitrarily small. For energy conservation Lemma, this means that the total amount of negative and positive energy in $V \setminus \{v_l\}$ is bounded at each round by a monotonically decreasing function of the round number. So $\lim_{r\to\infty}(|V| - e_{v_l}^r) = 0$, this means that there exists a round r such that $\forall r' \geq r$ holds $|V| - e_{v_l}^{r'} < 1$. So the variable $count_{v_l}$ will be equals to $|V| \; \forall r' \geq r$. □

The previous Lemma has an interesting implication: there exists a round r such that $\forall r' \geq r$ holds $|V| - e_{v_l}^{r'} < 1$. However, this condition is not detectable by any node in the network. Convergent algorithms can be used in any practical context where the safety of the application does not depend by a correct count and where liveness and/or fairness need just an eventually correct count (e.g. fair load balancing).

Termination Heuristic: $\mathcal{A}^*_{NoK}$

In this section, we will present the heuristic added to the basic $\mathcal{A}_{NoK}$ to obtain the new algorithm $\mathcal{A}^*_{NoK}$ working in an anonymous network with No Knowledge assumption and having a termination condition. The heuristic is used by the leader to decide at which time the current count can be considered as the final one. The heuristic is based on the assumption that the dynamicity of the graph is governed by a random process (i.e., a graph where links change according to a uniform probability distribution) and it considers the notion of *flow observed by the leader*.

At each round r, the leader v_l will receive a fraction of energy from all its neighbors. So the flow of energy to the leader at round r can be expressed as:

$$\Phi^r(v_l) = \sum_{\forall v \in N(v_l, r)} \frac{e_v(r)}{2d_v^{max}(r)}$$

where $e_v(r)$ is the energy of v at round r and $d_v^{max}(r)$ is the maximum number of neighbors that node v has so far. After a sufficient number of rounds, the estimation

of the flow observed by the leader is

$$\Phi^r(v_l) = \sum_{\forall v \in N(v_l,r)} \frac{e_v(r)}{2d_v^{max}(r)} \simeq \frac{|N(v_l,r)|}{2d_{avg}^{max(r)}} \overline{e_v(r)}$$

where $2d_{avg}^{max}(r)$ is the average of the maximum degrees seen by nodes in G at round r and $\overline{e_v(r)}$ is the average of the energy kept by all non-leader nodes at round r. Let us remark that, in the absence of the leader, the energy is always balanced among nodes in the network and let us recall that the leader is the only node absorbing energy. As a consequence, nodes being neighbors of the leader could have less energy than others as they transferred part of their energy to the leader without receiving nothing from it. Due to the assumption about the probabilistic nature of the edges creation process and considering the functioning of $\mathcal{A}_{NoK}$, those non-leader nodes will tend to have a similar quantity of energy as they will balance energy surplus. Thus, the leader can estimate $\overline{e_v(r)} \simeq \frac{|V|-e_{v_l}(r-1)}{|V|}$.

Due to the assumption about the probabilistic nature of the edges creation process, the leader will see almost the same maximum number of neighbors as the other nodes. Thus, $2d_{v_l}^{max}(r) \simeq 2d_{avg}^{max}(r)$. Thus, substituting we have

$$\Phi^r(v_l) \simeq \frac{|N(v_l,r)|}{2d_{v_l}^{max}(r)} \frac{|V| - e_{v_l}(r-1)}{|V|}$$

from which we obtain

$$|\bar{V}(r)| \simeq \frac{\rho(r) e_{v_l}(r-1)}{\rho(r) - \Phi^r(v_l)}$$

where $|\bar{V}(r)|$ is estimation of the number of processes in the network done by the leader at round r and $\rho(r) = \frac{|N(v_l,r)|}{2d_{v_l}^{max}(r)}$.

Let $k = \lceil e_{v_l}(r) \rceil$ be the number representing the count done by the leader at round r, and let $\Delta(r) = |\bar{V}(r)| - e_{v_l}(r)$ be difference between the network size estimated with the energy flow and the energy currently stored at the leader. We can finally define a termination condition as follows: as long as $\lceil e_{v_l}(r) \rceil$ remains stable, the leader computes the average $\overline{\Delta}$ of $\Delta(r)$ over the last k rounds and if after k consecutive rounds the quantity $\lceil e_{v_l}(r+k) + \overline{h} \rceil$ is equal to k and $\lceil e_{v_l}(r+k) \rceil = k$ the counting procedure terminates and the leader outputs k.

4.4 Experimental Evaluation

Simulator. In order to run our experiments, we developed a JAVA simulator using the Jung library [66] to keep track of the graph data structure. Each process v is seen as a node in the graph and it exposes an interface composed of two methods: the first one allowing to send a message for round r and the second one allowing to deliver messages for the round r. Moreover, each node has associated a queue q_v storing the messages that it has to receive. The simulation is done trough a set of threads; a thread T_j takes a node from a list l_m containing all of nodes to be examined in this round, removes it from the list and invokes the method send message. T_j also takes the message m generated by v, and adds it to the queues of $N(v,r)$. When l_m is

empty, a different set of threads is activated to deliver messages. T_j takes a node v from a list l_d and manage the delivery of all messages in q_v that v received during the current round. When all the messages in the queues are delivered to all the processes, the round terminates and the topology can be modified according to the dynamicity model considered and a new round can start.

Random Dynamic Graph Adversaries. In order to model the dynamicity of the network graph, we consider the following four adversary models:

1. **G(n, p) graph [26]**: at the beginning of each round r the set of edges is emptied and then for any pair of processes $u, v \in V$, the edge uv is created according to a given probability p. Let us recall that in the $G(n, p)$ graph model, there exist a connectivity threshold t, depending on the number of nodes n, such that if probability p is above the threshold, then $G(n, p)$ is connected with very high probability.

2. **Edge-Markovian (EM) graph [27]** : at each round r, edges are modified according to the following rules:

   (a) For each edge $uv \in E(r-1)$, uv is removed from $E(r)$ with a probability p_d (i.e., *death probability*).

   (b) For each edge $uv \notin E(r-1)$, uv is created and inserted in $E(r)$ with a probability p_b (i.e., *birth probability*).

   Clearly, connectivity of the graph at each round depends on p_d and p_b.

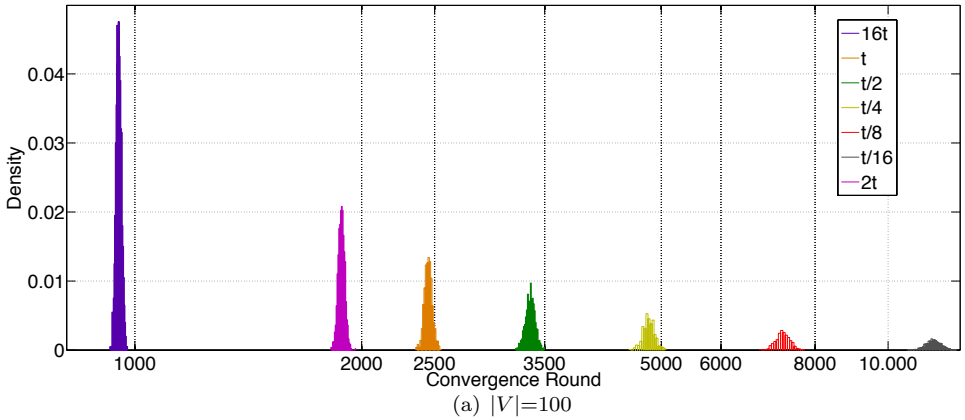
3. **Duty-cycle based graph**: at round r_0 the dynamic graph has a fixed, connected, topology. Each node follows a duty cycling phase during which, if at a given round r_i the node is awake it can receive and send messages according the topology of r_0 to any neighboring node that is also awake. While when at round r_j it is in sleep mode, all adjacent edges are removed from the graph. The presence of the duty cycle essentially brings some dynamicity in the graph since not all edges will be set at each round. This model constructs evolving graphs that reflect realistic deployments of resource constraint devices. Remark that this model does not guarantee that the graph will be connected at each round.

Metrics and parameters. We investigate three key performance metrics:

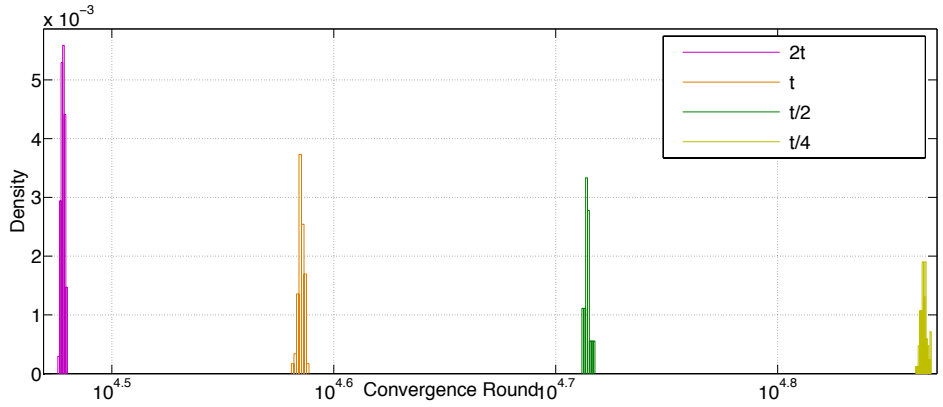
- **Convergence Time Distribution:** the convergence time is defined by the first round at which the algorithm outputs the correct value. In the following, we studied the probability distribution of the convergence time to show the average latency of the algorithms before reaching a correct count.

- **Flow Based Gain Δ**: such metrics represents the difference measured by the leader between the size estimated through the flow and the the size estimated trough the energy stored inside the leader (i.e., $\Delta(r) = |\bar{V}(r)| - e_{v_l}(r)$).

- **Error frequency** ρ: we measured the percentage of uncorrect termination obtained while adopting the heuristics-based termination condition defined in Section 4.3.



(a) $|V|$=100



(b) $|V|$=1000

Figure 4.8: $\mathcal{A}_{NoK}$ Convergence time distribution for $G_{(n,p)}$ graphs with different edge creation probabilities p.

The above metrics have been evaluated by varying the following parameters:

- **Dynamicity model:** we considered different types of random dynamic graph adversaries to evaluate the factors impacting every metrics.
- **Edges creation probability p:** such probability governs the graph dynamicity according to the specific model considered ($G_{(n,p)}$ or Edge-Markovian).

We have evaluated the performance of the algorithms under different metrics in networks comprised of $\{10, 100, 1000\}$ nodes. When not explicitly stated, tests are the results of 1000 independent runs.

Evaluation of $\mathcal{A}_{NoK}$

We implemented and tested $\mathcal{A}_{NoK}$ on both $G_{(n,p)}$, Edge-Markovian and Duty-cycle-based graphs. Let us first consider the case of $G_{(n,p)}$ graphs and let us recall that

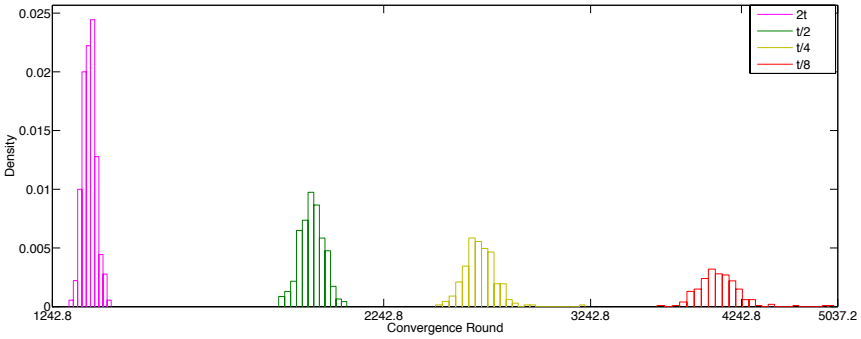


Figure 4.9: $\mathcal{A}_{NoK}$ Convergence time distribution for edge-Markovian graphs with different edge creation probabilities p_b and $|V| = 100$.

the connectivity threshold t is defined according to the number of nodes in the graph (i.e., $t = \frac{ln(|V|)}{|V|}$). We evaluate our algorithm for several probability p. In particular, for any probability greater than $2t$, we consider only connected graph instances, i.e., at each step, we check the connectivity and in case of disconnected graph we sample a new random graph. For probabilities smaller than $2t$ we allow disconnected graph instances.

Figure 4.8 shows $\mathcal{A}_{NoK}$ convergence time distribution when the algorithm runs on $G_{(n,p)}$ graphs. As expected the convergence time becomes worse when we consider disconnected instances. However, it is worth notice that the algorithm is able to converge to the correct count even in presence of disconnected instances. Moreover, the increment of convergence time is inversely proportional to p and there is an increment of the distribution variance due to the presence of disconnected instances.

When considering Edge-Markovian graphs, we set the probability of creating an edge as in the $G_{(n,p)}$ graphs and we fixed the probability of deleting an edge to 0.25 (i.e., $p_d = 0.25$ and $p_b = f(t)$).

Figure 4.9 shows $\mathcal{A}_{NoK}$ convergence time distribution; as we can see, it is comparable to $G_{(n,p)}$ graph one. In addition, the persistence of edges across rounds (due to $p_d \leq 1$) mitigates the low values of edge creation probability. As a consequence, the convergence is faster than the pure $G_{(n,p)}$.

Evaluation of $\mathcal{A}^*_{NoK}$

In the following, we evaluate the $\mathcal{A}^*_{NoK}$ algorithm on both $G_{(n,p)}$ and Edge-Markovian graphs. Figure 4.10 shows several measures related to the heuristic correctness. In particular, in addition to the error frequency ρ, we measured also the average error and maximum error done, by the heuristic, in terms of number of nodes missed with respect to the real number of nodes in the graph. We omit from the Figure some probabilities since they always terminate correctly ($p \geq \frac{t}{2}$ in case of $G_{(n,p)}$ graphs and $p_b \geq \frac{t}{4}$ for the Edge-Markovian). In case of disconnected topologies, i.e., $p \leq \frac{t}{4}$ for the $G_{(n,p)}$ or $p_b \leq \frac{t}{8}$ for the Edge-Markovian, we have that the percentage of counting instances terminating correctly is smaller that 100% and it becomes proportionally worse with the decrease of p. Moreover, it is possible to see a bimodal behavior of the

heuristic when it fails: two cases are frequent in the experiments (i) the heuristic forces the termination in the first rounds of the counting process with the consequence of having the leader outputting a count much smaller than the real number of processes and (ii) the heuristic fails when the energy accumulated by the leader is close to the current network size. In all our experiments we have not found a case in which the heuristics forces the termination in a case different from this two. Moreover in the table we indicate the *Convergence Detection Time*, that is the number of rounds after the first convergence that the heuristics employs to correctly terminate the count. It is possible to see that in the majority of experiments, even on disconnected instances the heuristic converges in a time that is equal to the size of the network.

Model	$G_{(n,p)}$						Edge-Markovian $p_d = 0.25$		
p	$\frac{t}{4}$			$\frac{t}{8}$	$\frac{t}{16}$	$\frac{t}{32}$	$\frac{t}{8}$	$\frac{t}{16}$	$\frac{t}{32}$
$\|V\|$	10	100	1000	100	100	100	100	100	100
ρ	22%	3%	2%	19%	25%	84%	30%	68%	76%
Average Error	2,02	8,96	1	9	44,5	41,4	1	3,12	11,8
Max Error in Nodes	8	96	1	99	99	99	1	99	99
σ of Error	2,1166	27,4	0	27,4	48,3	48,8	1	14,23	29,73
Convergence Detection Time Average	10,2	100	1000	100	100	100	100	100	100
Convergence Detection Time Max	40	100	1000	100	100	100	100	100	100
Convergence Detection Time Min	10	100	1000	100	100	100	100	100	100

Figure 4.10: Evaluation of the Termination Correctness ρ. The results are the outcome of 500 experiments

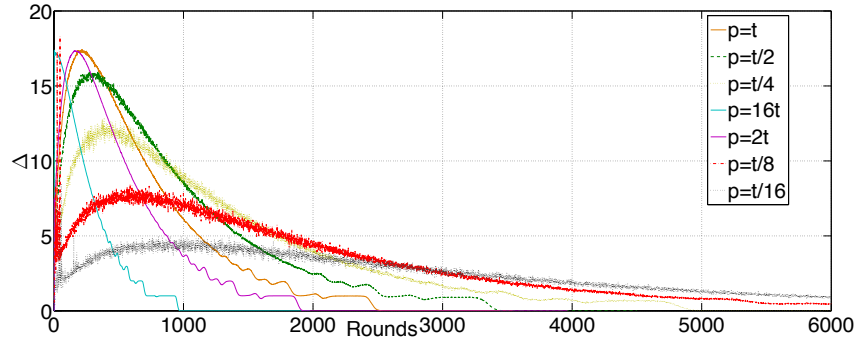
Comparison between $\mathcal{A}_{NoK}$ and $\mathcal{A}^*_{NoK}$

The flow could be used to estimate the size of $|V|$ obtaining a faster count. Figure 4.11 shows the evolution of Δ, i.e., difference measured by the leader between the size estimated through the flow and the the size estimated trough the energy stored inside the leader, both from a temporal perspective 4.11(a) and from the energy perspective 4.11(b).

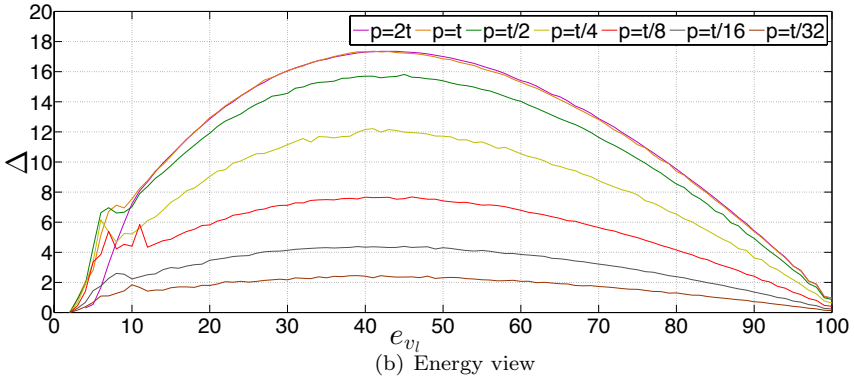
The value Δ reaches the maximum when the energy at the leader is approximately half of the network size; in this case, when the network is connected (i.e., $p \geq t$), the use of the heuristic allows the leader to predict, correctly, the presence of at least others 17 nodes.
So, on connected instances our approach could be useful to answer faster to predicates likes $|V| \geq t$. In addiction, the flow-based estimation continues to perform well on non-connected instances only until a certain threshold, then the gain obtained with the flow drops to one or two nodes more than the ones estimated by the energy.
Moreover the figures show why the termination heuristics works bad on instances where $p \leq \frac{t}{4}$, we can see that Δ falls behind the threshold of 1, both when the energy in the leader is low, and when the energy in the leader is approaching the value $|V|$ this could lead to two possible misbehavior, terminating after few rounds from the start, so with a value that could be sensibly distant from the value of $|V|$ or it could terminate near $|V|$, when Δ falls again behind 1.

Figure 4.11(a) shows the behavior of Δ along time. In particular,

- when the network is connected (i.e., $p \geq t$), the counting done by the leader fast approaches half of the network size (i.e., the maximum value for Δ). The energy-based count approaches the actual size with an exponential time; this



(a) Round view



(b) Energy view

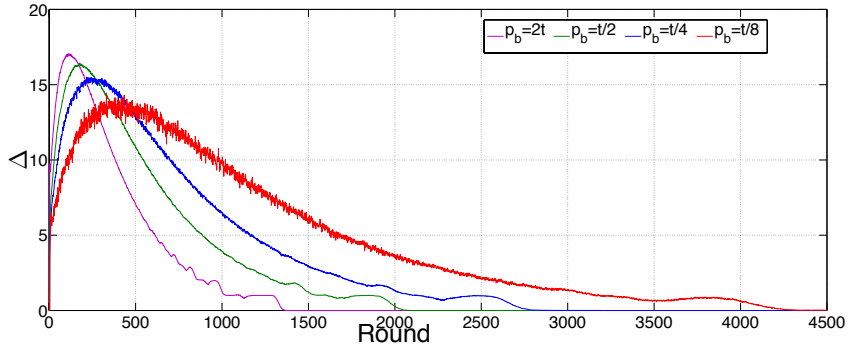
Figure 4.11: Difference between the size estimated with the flow ($\mathcal{A}^*_{NoK}$) and the size estimated by looking to the energy stored at the leader ($\mathcal{A}_{NoK}$) in a $G_{n,p}$ network of $|V|$=100.

is visible from the exponential decay of Δ. This behavior is present also when $p < t$, even tough there is a slower decay of Δ that obviously reflects a slower approach to the actual size.

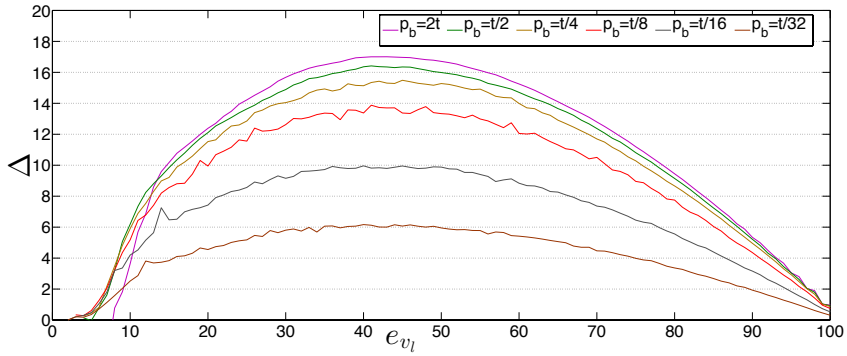
- for values of $p \leq t$ the curves show a high variance. This is due to the presence of disconnected topologies that introduce a variance in the convergence time for which the magnitude is proportional to the inverse of p. This high variance in convergence is due to the high variance of the flow that the leader will see during the execution.

The same behavior can be observed in Edge-Markovian graphs (cfr. Figure 4.12). The presence of more edges in the edge-markovian graph affects positively the Δ measures since it is less prone to the value of p. It is possible to notice a slightly low maximum value for the edge-markovian process, 17 against 17.3 of the $G_{(n,p)}$ graph.

We run also tests with larger graphs ($|V| = 1000$) but we omit them here since curves exhibit the same behavior of those shown in Figures 4.11 and 4.12, notably in this case the maximum delta is about 170 nodes.



(a) Round View



(b) Energy View

Figure 4.12: Difference between the size estimated with the flow ($\mathcal{A}^*_{NoK}$) and the size estimated by looking to the energy stored at the leader ($\mathcal{A}_{NoK}$) for Edge Markovian network with $p_d = 0.25$ of $|V|$=100.

Duty Cycle

In order to test the adaptiveness of our heuristic, we run $\mathcal{A}^*_{NoK}$ on regular topologies: rings and chains. Over those topologies, we simulate a duty-cyle of 80%. Each node independently sleeps for 20% of the time and during this period links of sleeping nodes are deleted. Considering a ring topology with $|V| = 100$, the average convergence time is around 26986 rounds for 100 experiments, for the chain the convergence time is on average 70000 rounds . We also tested random $G_{(n,p)}$ topologies where $p = 2t$, in this case the average over 200 experiments shows a convergence time of 1059 rounds. The most noticeable phenomenon is that on graphs with duty-cyle both the termination heuristic and the size estimation perform really bad: on rings and chains the termination heuristics always fails and on random graphs it fails on the 23% of the instances.

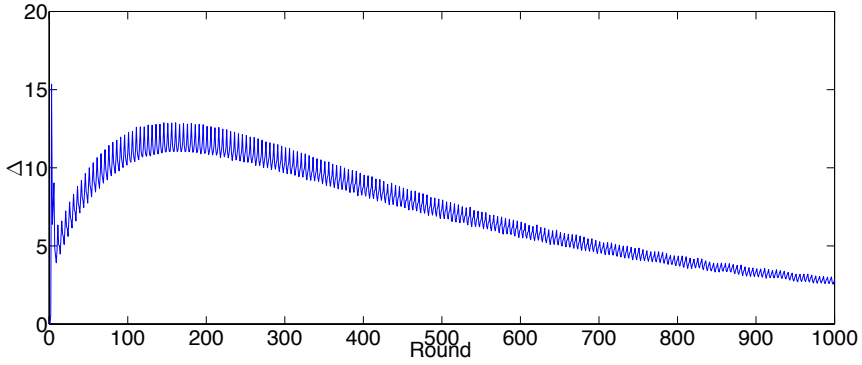


Figure 4.13: Difference between the size estimated with the flow ($\mathcal{A}^*_{NoK}$) and the size estimated by looking to the energy stored at the leader ($\mathcal{A}_{NoK}$) of 200 runs for duty cycle and random graph with $|V|$=100.

Evaluation of $\mathcal{A}^*_{\mathcal{O}^P}$

Model	$G_{(n,p)}$ $p = 2t$			Edge-Markovian $p_b = 2t$		
$\|V\|$	10	100	1000	10	100	1000
Average Termination	7,6	107,6	1690,6	9,9	113,4	1554,2
Max Termination	17	187	2117	19	222	2684
Min Termination	5	96	1175	5	99	807

Figure 4.14: Termination performance of $\mathcal{A}^*_{\mathcal{O}^P}$ on 1-interval connected instances.

We evaluate the termination time of $\mathcal{A}^*_{\mathcal{O}^P}$ over $G_{(n,p)}$, edge-markovian and Random Connected graphs. Let us recall that this algorithm only works on instances that are 1-interval connected. The basic $\mathcal{A}_{\mathcal{O}^P}$ algorithm employs, on average, 15 rounds for $|V| = 10$, 393 for $|V| = 100$ and 7753 for $|V| = 1000$, in Table 2 we can see the performance for the symmetry breaking version (i.e., $\mathcal{A}^*_{\mathcal{O}^P}$). As expected the symmetry breaking extension allows the algorithm to terminate faster, the termination time is close to the size of the network. Moreover we can see that the additional knowledge offered

by the $\mathcal{O}^P$ allows the counting algorithm to count faster then the $\mathcal{A}_{NoK}$ or the $\mathcal{A}^*_{NoK}$ one.

4.5 Conclusive Remarks

In this Chapter we have shown that when a structure to convey counting information towards the leader is missing, we can resort on a convergent processes that could lead to terminating counting. The main results of this chapter is the algorithm $\mathcal{A}_{\mathcal{O}^P}$. A perfect local oracle is implementable in many context: when the MAC layer could gives up information about the number of neighbors or when the network is enough stable to allow a local request-reply pattern. Another thing to notice is that $\mathcal{A}_{\mathcal{O}^P}$ can also be easily adapted to work when the network is directed and strongly connected at each round. Provided that it is known in advance the out-degree of nodes. This is due to the fact that the energy transfer mechanism does not care about the state of receiving nodes. The technique used to know the cardinality of the new colored node could be also adapted to work when coloring nodes do not know if all nodes at the receiving endpoint are colored or not. It is possible to solve this indecision receiver side.

Open Problems From the results presented in this Chapter we can identify the following open problem:

- The presence of $\mathcal{O}^{OE}$ is necessary for Lemma 37, on which are based all the terminating algorithms presented in this section. An open problem is to find out if it is possible to implement a monotonically convergent process similar to the one that we showed in this section without using $\mathcal{O}^{OE}$. This would seamlessly lead to a variation of $\mathcal{A}_{\mathcal{O}^{OE}}$ that works in other settings.
- In contrast with Chapter 3, the algorithm $\mathcal{A}_{\mathcal{O}^P}$ suffers from an upper bound of exponential complexity due to our exponential upper bound on the convergence of the energy-transfer mechanism. This mechanism is used to compensate the lack of structure, i.e. at each round the position of the leader and of the previous neighbors changes unpredictably. Therefore an open question is if it is possible to find a sub-exponential upper bound on the convergence. In case of $\mathcal{O}^{OE}$ the bound is given by the quality of the local overestimation. A polynomial upper bound for the energy transfer, if it exists, will show that when a local oracle is present the counting time is polynomial despite the anonymity.

Chapter 5

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

In this Section we show a terminating algorithm that solves the counting on the general family $\mathcal{G}$(1-IC), this algorithm shows the equivalence, from the point of view of the computability of predicates and aggregated functions, of $\mathcal{G}$(1-IC) with IDs and $\mathcal{G}$(1-IC) with anonymous nodes and a distinguished leader.

Definition 14. *(Subgraph) Given a dynamic graph G, a dynamic graph G' is a subgraph of G (i.e, $G' \subseteq G$) if and only if $G' : [G_{i_1}, G_{i_2}, \ldots]$ is an ordered subsequence of $G : [G_0, G_1, G_2, \ldots]$.*

Let introduce a lemma on $\mathcal{G}$(1-IC).

Lemma 40. *Let consider a dynamic graph $G : [G_0, G_1, G_2, \ldots] \in \mathcal{G}$(1-IC). We have that exists $h \in \mathbb{N}^+$ and $\exists G' \subseteq G$ such that G' is infinite 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-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 $|V|$ nodes, 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 that $|S| \leq \sum_{\forall G_j \in X_s} m_{G_j} \leq \sum_{\forall G_j \in X} m_{G_j}$ but $|S| = x + 1$ that is a contradiction. Thus 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. □

Lemma 40 tell us that to have a terminating counting algorithm on $\mathcal{G}$(1-IC) is sufficient to design a counting algorithm that always terminates correctly on instance $G' \in \mathcal{G}(\text{PD})_h$, and that do not give a wrong result on $G' \notin \mathcal{G}(\text{PD})_h$. Since for any $G \in \mathcal{G}$(1-IC) there exists $G' \subseteq G, G' \in \mathcal{G}(\text{PD})_h$ we have that eventually a counting algorithm designed in such way terminates correctly.

5.1 The Algorithm

Both leader and non leader nodes check all possible subgraphs of G, let us define as $\mathcal{P}_G$ such set. For each $G' \in \mathcal{P}_G$ they start a different instance $I_{G'}$ of the counting algorithm. Let us remark that the system is synchronous and the current round number r is known to all processes. Therefore each instance $I_{G'}$ is uniquely identified, for example by a binary string that has value 1 in position j if $G_{r_j} \in G'$ and 0 otherwise. The uniqueness guarantees that different instances cannot interfere with each other. At each new round r the number of instances is duplicated, half of the new instances will consider the messages exchanged in round r and the other half 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 nodes 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 necessary to handle instances is trivial therefore will be omitted.

```
1:  M(−1) = []
2:  H(−1) = [⊥]
3:  distance = −1
4:
5:  procedure SENDING_PHASE
6:      send(Message :< distance, M(r), H(r) >)
7:
8:  procedure RCV_PHASE(MultiSet MS)
9:      if distance == −1 ∧ ∃m ∈ MS | m.distance ≠ −1 ∧ m.distance == r then
10:         distance=m.distance+1
11:     if r == distance then
12:         for all m ∈ MS | m.distance == −1 do
13:             m.distance=distance+1
14:     if r > distance ∧ ∃m ∈ MS | m.distance ∉ {distance − 1, distance, distance + 1} then
15:         M(r + 1) = M(r).append(BAD)
16:     if distance ≠ −1 then
17:         H(r + 1) = H(r).append(count_distance_neighbors(MS, distance − 1))
18:         M(r + 1) = M(r).append(get_messages_from_distance(MS, distance + 1))
```

Figure 5.1: Counting algoritm for $\mathcal{G}(1-\text{IC})$: pseudocode for Non-Leader Node

In each instance $I_{G'}$, nodes run a slightly modified version of the OPT_h algorithm for $\mathcal{G}(\text{PD})_h$ (see Section 3.2 Chapter 3).

For simplicity in the explanation of the algorithm we refer to instance $I_{G'}$: $[G_{i_1}, G_{i_2}, \ldots]$ and we use a round numbering consistent with instance $I_{G'}$, that is if we say round 2 we refer to the round of i_2 of G. Thus when we use the term V_h we refer to the set of nodes that sets that in the instance $I_{G'}$ have set their distance to h at round $r = h - 1$ of G', that is the round i_{h-1} of G. The modifications are the

following:

- If v_l detects that the instance $I_{G'}$ is such that $G' \notin \mathcal{G}(\text{PD})_h$ for any $h \in \mathbb{N}^+$ then $I_{G'}$ is considered BAD. When an instance $I_{G'}$ is BAD it will be not considered for counting. Moreover all instances $I_{G''}$ such that G' is a prefix of G'' will be considered BAD.
- The *get_distance* phase is not executed as in OPT_h. Counting and distance determination are done in parallel, as in the algorithm of Section 3.4.
- The counting of nodes at distance V_h is done by using the $\mathcal{FD}$ algorithm explained in Section 3.3. This allows the leader, if it does not terminate the count, to detect:
  - if a node v, that had set its distance to h at round $r = h$, at some round $r' > r$ moves at some distance $h' \neq h$. When this is detected the current instance $I_{G'}$ is considered BAD.
  - If a node v that has no distance set, is neighbor of some node w at distance h at some round $r > h + 1$. When this is detected the current instance $I_{G'}$ is considered BAD.
- The condition of termination cannot be the same of OPT_h. The leader terminates the count when the current instance is not considered BAD and it counts a set V_h such that no node in V_h, has some neighbor in V_{h+1}. This same strategy is used in the algorithm of Section 3.4.

The pseudocode for instance $I_{G'}$ is reported in Figures 5.1-5.2.

It is intuitive that the code terminates correctly on a $I_{G'}$ such that $G' \in \mathcal{G}(\text{PD})_h$. It could be less intuitive that the code does not terminate incorrectly on an instance $I_{G''}$ such that $G'' \notin \mathcal{G}(\text{PD})_h$.

Correctness Proof

Lemma 41. *Let consider a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$, and its subgraph G'. Let consider the instance $I_{G'}$ of the counting algorithm on $G' \in \mathcal{G}(PD)_h$. We have that v_l will never consider $I_{G'}$ as* BAD.

Proof. The leader considers $I_{G'}$ bad at Line 19. The line is executed either (a) if a *failure* is detected by $\mathcal{FD}$, i.e. a node $v \in V_h$ at some round $r > h - 1$ is not neighbor of nodes in V_{h-1}, or (b) if some set MS contains a BAD element. The latter happens if a non leader-node v' executes Line 15-Figure 5.1, 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', the same holds for condition (b). Both conditions would implies that a node 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 42. *Let consider a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$, and its subgraph G'. Let consider the instance $I_{G'}$ of the counting algorithm on G', we have that if exists a set of nodes in V_h either (1) the leader eventually obtains the count V_h or (2) the leader will consider $I_{G'}$ as* BAD.

```
1: distance_count[]
2: procedure SENDING_PHASE
3:     send(< leader >)
4:
5: procedure RCV_PHASE(MultiSet MS :< distance, M, H >)
6:     i = 1
7:     if (distance_count[i] ≠ ⊥ ∧ distance_count[i] ≠ |MS|) ∨ (∃m ∈ MS|m.distance > 1) then
8:         BAD_INSTANCE:terminate
9:     distance_count[i] = |MS|
10:    i++
11:    while true do
12:        if MS ≠ ∅ ∧ (∀m ∈ MS : m.M = [⊥, ..., ⊥] ∧ size(m.M) > 1) then
13:            count = ∑_{∀j|distance_count[j]≠⊥} distance_count[j]
14:            output(count)
15:        MS =BUILDLASTNEXTDISTANCESET (MS)
16:        if MS = ⊥ then
17:            break
18:        if MS == failure then
19:            BAD_INSTANCE:terminate
20:        distance_count[i] = |MS|
21:        i++
22:
23: function BUILDLASTNEXTDISTANCESET(MS)
24:    MS_last = ⊥
25:    if ∃BAD ∈ MS then
26:        return failure
27:    if FDTREE(RECENT(MS, 0)) ≠ ⊥ then
28:        for r = MinRound(MS); r < MaxRound(MS); r++ do
29:            if FDTREE(RECENT(MS, r)) == failure then
30:                return failure
31:            if FDTREE(RECENT(MS, r)) ≠ ⊥ then
32:                MS_last = FDTREE(RECENT(MS, r_last))
33:            else
34:                break
35:    return MS_last
```

Figure 5.2: Counting algoritm for $\mathcal{G}(1-\text{IC})$: pseudocode for Leader Node

Proof. The nodes in V_h set their distance at round $r = h - 1$, see Line 9 of Figure 5.1. At the same round an instance of $\mathcal{FD}$ between nodes in V_{h-1}, V_h is started: the multiset of messages MS_h sent at round $h-1$ by node in V_h will be propagated from nodes in V_h to v_l during rounds $[h-1, \ldots, r_{count_h}]$. At each different distance this is done by using other instances of $\mathcal{FD}$: between nodes in V_{h-1}, V_{h-2}, nodes in V_{h-2}, V_{h-3} and so on. Now let us consider condition (A) that is instances of $\mathcal{FD}$ between nodes in $V_0, \ldots, V_h$ that are propagating towards the leader MS_h never detect a failure and no nodes in $V_0, \ldots, V_h$ has a BAD elements in its multiset of messages MS. If condition (A) holds we have that the leader will obtains the correct count of nodes in V_h by reconstructing the multiset of messages MS_h sent by node in V_h at round $h-1$. This is ensured by Lemma 14 of algorithm $\mathcal{FD}$ and by a simple induction on the count for each set V_i. Otherwise if condition (A) does not hold the instance $I_{G'}$ will be considered BAD either in line 25 or because some instance of $\mathcal{FD}$ fails before the leader is able to reconstruct MS_h.

□

Lemma 43. *Let consider a dynamic graph $G \in \mathcal{G}$(1-IC), if v_l terminates then it outputs the correct count.*

Proof. The Leader terminates at Line 12, that is the leader has reconstructed a multiset MS_h from nodes 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 42 we have that if this happen the leader has correctly counted nodes in $V_0, \ldots, V_h$, we have to show that when Line 12 is triggered we have $V \setminus V_0 \setminus \ldots \setminus V_h = \emptyset$. Let us assume Line 12 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 node in $V_0, \ldots, V_h$. If it is neighbor of a node $v_1 \in V_j$ then v_1 will put BAD 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 BAD message thus the Line 25 will be triggered, then the instance is considered BAD and the Line 12 will not be executed; (2) that v_1, or some other node that is sending the BAD message to v_l, is moved away. This triggers a failure in $\mathcal{FD}$ during the reconstruction of MS_h therefore Line 12 is not executed. If v is neighbor of nodes in V_h and $r' > h$ we have the same behavior of the previous case. The only possibility left is v neighbor of nodes in V_h and $r' = h$. In this case at least one node v' in V_h will execute Line 12 setting $v'.M(h) = [\perp, \neg\perp]$ therefore Line 12 cannot be executed.

□

Lemma 44. *Let consider a dynamic graph $G \in \mathcal{G}(1\text{-}IC)$, eventually v_l terminates and outputs the correct count.*

Proof. For Lemma 40 there exists $G' \subseteq G$ with $G' \in \mathcal{G}(\mathrm{PD})_h$. For Lemma 41 the instance $I_{G'}$ will never be considered as BAD by v_l. This means that, see Lemma 42, the leader will eventually obtain the count for each set $V_1, \ldots, V_h$ on instance $I_{G'}$. Since the set V_{h+1} is empty we have that eventually v_l will execute the terminating condition. Therefore for Lemma 43 the leader will eventually terminates correctly on $I_{G'}$. Moreover also for Lemma 43 if the leader terminates on another instance $I_{G''}$ with $G'' \subseteq G$ the counts is also correct. From these considerations the claim follows.

□

5.2 Conclusive Remarks

The algorithm in the worst case needs an exponential number of rounds: the adversary, in the worst case, cycles among all possible graphs of $|V|$ nodes, the length of this cycle is exponential in $|V|$. We also have that the number of not BAD instances grows exponentially with respect to the number of rounds, this implies an exponential number of messages exchanged. This exponential cost in rounds, memory and messages means that the algorithm is not usable in practice. We designed a simulator of a random adversary and even on small networks, less than 50 nodes, after few rounds the memory requirements are overwhelming.

Let us notice that a dynamicity governed by an random random adversary does not lead to good performance. The condition of stability on graph subsequence that we are looking is a rare event on a sequence of purely random graphs. Therefore the algorithms presented in Chapter 4, are more interesting, from a practical point of view despite their strong requirements on knowledge, the presence of a local degree detector oracle.

However the algorithm shows that on $\mathcal{G}$(1-IC) the only presence of the leader is enough to obtain an exact count. When the size $|V|$ of G is known for the property of $\mathcal{G}$(1-IC) we can construct a terminating broadcast algorithm.

Open Problems From the results presented in this Chapter we can identify the following open problem:

- It is unknown if counting in $\mathcal{G}$(1-IC) requires more than $\Omega(|V|)$ rounds. Therefore an open question is to quantify the gap, if it exists, between the algorithm proposed in this Chapter and the optimal counting time for $\mathcal{G}$(1-IC).

Chapter 6

Conclusion

In this manuscript we have shown that counting in $\mathcal{G}(1\text{-IC})$ is possible once is provided the existence of an unique distinguished leader node. In the path towards this result we have proved that on anonymous dynamic networks the time needed to output the count is function of the network size for any dynamic diameter $D > 3$. Other existential results have been provided regarding the trade-off between the time needed to count and the accuracy of deterministic counting algorithms. An optimal algorithm for $\mathcal{G}(\text{PD})_h$ networks is given. In this kind of networks the adversary is strongly restricted, nevertheless these networks could model a setting where nodes are positional static and where the dynamicity of links among nodes is introduced by communication problems. The possibility of polynomial counting in $\mathcal{G}(\infty\text{-IC})$ leads to an efficient way to implement deterministic broadcast on top of these networks. In Chapter 4 we have introduced a different technique to attack the counting problem under the assumption of additional local knowledge. If a node knows its degree before the send phase we have that it is possible to implement a deterministic terminating algorithm that experimentally has been proved efficient, even if analytically we have only obtained an exponential upper bound on its time complexity. Summarizing we have used mainly two approaches to attack the counting problem, in the first one (used in Chapter 3,5) we have exploited a certain structure in the dynamic network in order to convey counting information towards the leader. In Chapter 3 the structure is always present in the dynamic network, due to adversarial restriction. In Chapter 5 we use a *"trick"* to find that structure, we simply consider all possible subsequences of rounds until we find the one that ensures the structure on which we can count. In Chapter 4 where we have never considered this structure, we used a leader-based convergent technique with deterministic terminating conditions.

These results are a small step towards a comprehensive picture of deterministic computability in dynamic anonymous networks. The restriction imposed by the presence of the leader is necessary. However, considering problems that are different from the counting, a possible trade-off for its presence would be the knowledge about the network size (see [50] and for a parallelism in anonymous static networks [68]), or the assumption of an eventual symmetry breaking that would allow leader election.

The following key problems are left opens:

- Considering the bounds presented in Chapter 2, it is easy to see that they are heavily based on the indecision introduced by the lack of knowledge about nodes degree. If a node knows its degree before the send phase then counting in $\mathcal{G}(\text{PD})_2$ is trivial, and it would require $\mathcal{O}(1)$ rounds. However it is not clear if this would allow to design faster algorithms on $\mathcal{G}(\text{1-IC})$. Therefore an open question is to quantify, if it exists, the advantage given by degree knowledge in the general model.

- The kernel based technique used in Chapter 2 has been fundamental to prove the trade-off lower bound. Once the local knowledge about the dynamic network is modeled as a succession of matrices the kernel space gives an exact characterization of all possible local-undistinguishable networks, since it implicitly considers all possible configurations that the adversary could create. We are looking forward the possibility of an extension to general unknown $\mathcal{G}(\text{1-IC})$ networks, but it is more likely to expect applications in dynamic networks where a certain structure is assumed, e.g. by having a recurrent set of edges as in [20, 60]. A structure in the dynamic network imposes a structure in the succession of matrices, structure that is necessary in order to have a closed characterization of the kernel space.

- In the whole manuscript we have never considered bandwidth or space constraints. Therefore an interesting open question is about the minimal requirements needed to count. Let us consider networks in $\mathcal{M}(\text{DBL})_k$, if we restrict our attention on the class of algorithms where nodes send their last c degrees it is easy to verify, by using the technique introduced in Chapter 2, that if c is not order of $\log(|V|)$ then they cannot solve count. A generalization of this observation to all possible algorithms would lead to a bound on space complexity. We conjecture that on $\mathcal{G}(\text{PD})_2$ messages of size $\Omega(\log^2 |V|)$ are necessary, proving this would lead to a non trivial lower bound for anoymous $\mathcal{G}(\text{1-IC})$: when IDs are present messages of size $\mathcal{O}(\log |V|)$ are sufficient.

- A big question remains open about the solvability of deterministic terminating counting on $\mathcal{G}(\text{1-IC})$ in sub-exponential time. An exponential lower bound for the problem will show that counting is solvable from a theoretical perspective but it is not solvable in practice.

List of Figures

Bibliography

[1] S. Abshoff, M. Benter, M. Malatyali, and F. Meyer auf der Heide. On two-party communication through dynamic networks. In *Proceedings of 17th International Conference on Principles of Distributed Systems*, OPODIS '13, pages 11–22. Springer, 2013.

[2] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, FOCS '87, pages 358–370. IEEE Computer Society, 1987.

[3] D. Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, STOC '80, pages 82–93. ACM, 1980.

[4] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. On the power of anonymous one-way communication. In *Principles of Distributed Systems, 9th International Conference, OPODIS 2005*, pages 396–411, 2005.

[5] L. Aniello, R. Baldoni, C. Ciccotelli, G.A. Di Luna, F. Frontali, and L. Querzoni. The overlay scan attack: inferring topologies of distributed pub/sub systems through broker saturation. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based System, Mumbai, India, May 26-29, 2014*, DEBS '14, pages 107–117. ACM, 2014.

[6] G. Ateniese, R. Baldoni, S. Bonomi, and G.A. Di Luna. Oblivious assignment with m slots. In *Proceedings of 14th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, volume 7596 of *SSS '12*, pages 187–201. Springer, 2012.

[7] Giuseppe Ateniese, Roberto Baldoni, Silvia Bonomi, and Giuseppe Antonio Di Luna. Fault-tolerant oblivious assignment with m slots in synchronous systems. *Journal of Parallel and Distributed Computing*, 74(7):2648–2661, 2014.

[8] H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35(4):845–875, October 1988.

[9] J. Augustine, A. R. Molla, E. Morsy, G. Pandurangan, P. Robinson, and E. Upfal. Storage and search in dynamic peer-to-peer networks. In *Proceedings of the 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 53–62, New York, NY, USA, 2013. ACM.

[10] J. Augustine, G. Pandurangan, and P. Robinson. Fast byzantine agreement in dynamic networks. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 74–83, New York, NY, USA, 2013. ACM.

[11] C. Avin, M. Koucký, and Z. Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part I*, ICALP '08, pages 121–132, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, STOC '87, pages 230–240, New York, NY, USA, 1987. ACM.

[13] B. Awerbuch, B. Patt-Shamir, Da. Peleg, and M. E. Saks. Adapting to asynchronous dynamic networks (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, STOC '92, pages 557–570. ACM, 1992.

[14] R. Baldoni, G. A. Di Luna, and L. Querzoni. Collaborative detection of coordinated port scans. In *Proceedings of 14th International Conference on Distributed Computing and Networking*, volume 7730 of *ICDCN '13*, pages 102–117. Springer, 2013.

[15] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. *Journal of Computer and System Sciences*, 73(3):245–264, May 2007.

[16] J. Beauquier, J. Clement, S. Messika, L. Rosaz, and B. Rozoy. Self-stabilizing counting in mobile sensor networks. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '07, pages 396–397, New York, NY, USA, 2007. ACM.

[17] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In *Proceedings of 15th International Conference on Distributed Computing*, pages 33–47. Springer-Verlag.

[18] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 181–188. ACM, 1999.

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

[20] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Deterministic computations in time-varying graphs: Broadcasting under unstructured mobility. In *Theoretical Computer Science*, volume 323, pages 111–124. Springer Berlin Heidelberg, 2010.

[21] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *CoRR*, abs/1012.0009, 2010.

[22] J. Chalopin, E. Godard, and Y. Métivier. Election in partially anonymous networks with arbitrary knowledge in message passing systems. *Distributed Computing*, 25(4):297–311, 2012.

[23] J. Chalopin, Y. Métivier, and T. Morsellino. Enumeration and leader election in partially anonymous and multi-hop broadcast networks. *Fundamenta Informaticae*, 120(1):1–27, 2012.

[24] J. Chalopin, Y. Métivier, and T. Morsellino. On snapshots and stable properties detection in anonymous fully distributed systems (extended abstract). In *proceedings of the 2012 Structural Information and Communication Complexity*, pages 207–218. Springer Berlin / Heidelberg, 2012.

[25] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P.G. Spirakis. Passively mobile communicating machines that use restricted space. *Theoretical Computer Science*, 412(46):6469–6483, October 2011.

[26] A. E. F. Clementi, F. Pasquale, A. Monti, and R. Silvestri. Communication in dynamic radio networks. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, PODC '07, pages 205–214, New York, NY, USA, 2007. ACM.

[27] A. E.F. Clementi, C. Macci, A. Monti, F. Pasquale, and R. Silvestri. Flooding time in edge-markovian dynamic graphs. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, PODC '08, pages 213–222, New York, NY, USA, 2008. ACM.

[28] D.A. Cooper and K.P. Birman. Preserving privacy in a network of mobile computers. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, May 1995.

[29] G. Di Luna and R. Baldoni. Brief announcement: Investigating the price of anonymity in dynamic networks. In *To Appear in Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, http://arxiv.org/abs/1505.03509*, PODC '15. ACM, 2015.

[30] S. Dolev. *Self-stabilization*. MIT Press, Cambridge, MA, USA, 2000.

[31] C. Dutta, G. Pandurangan, R. Rajaraman, Z. Sun, and E. Viola. On the complexity of information spreading in dynamic networks. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 717–736. SIAM, 2013.

[32] Y. Emek, C. Pfister, J. Seidel, and R. 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.

[33] P. Flocchini, B. Mans, and N. Santoro. Sense of direction in distributed computing. *Theoretical Computer Science*, 291(1):29 – 53, 2003.

[34] P. Fraigniaud, A. Pelc, D. Peleg, and S. Pérennes. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00. ACM.

[35] E.M. Gafni and D.P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29:11–18, 1981.

[36] B. Haeupler and F. Kuhn. Lower bounds on information dissemination in dynamic networks. In *Proceedings of 26th International Symposium on Distributed Computing*, DISC '12, pages 166–180. Springer, 2012.

[37] D. Ilcinkas, R. Klasing, and A.M. Wade. Exploration of constantly connected dynamic graphs based on cactuses. In *Proceedings of 21st International Colloquium on Structural Information and Communication Complexity*, SIROCCO '14, pages 250–262, 2014.

[38] D. Ilcinkas and A.M. Wade. Exploration of the t-interval-connected dynamic graphs: The case of the ring. In *Proceedings of 20st International Colloquium on Structural Information and Communication Complexity*, volume 8179 of *SIROCCO '13*, pages 13–23. Springer International Publishing, 2013.

[39] 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.

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

[41] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *FOCS*, pages 482–491, 2003.

[42] J. Kong, X. Hong, and M. Gerla. An identity-free and on-demand routing scheme against anonymity threats in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 6(8):888–902, 2007.

[43] F. Kuhn, C. Lenzen, T. Locher, and R. Oshman. Optimal gradient clock synchronization in dynamic networks. *CoRR*, abs/1005.2894, 2010.

[44] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, STOC '10, pages 513–522. ACM, 2010.

[45] F. Kuhn, Y. Moses, and R. Oshman. Coordinated consensus in dynamic networks. In *Proceedings of 31th ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 1–10. ACM, 2011.

[46] F. Kuhn and R. Oshman. The complexity of data aggregation in directed networks. In *Proceedings of 25th International Symposium on Distributed Computing*, DISC' 11, pages 416–431, 2011.

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

[48] F. Kuhn, S. Schmid, and R. Wattenhofer. Towards worst-case churn resistant peer-to-peer systems. *Distributed Computing*, 22(4):249–267, 2010.

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

[50] G.A. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Counting the number of homonyms in dynamic networks. In *Proceedings of 15th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, volume 8255 of *SSS '13*, pages 311–325. Springer, 2013.

[51] G.A. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Conscious and unconscious counting on anonymous dynamic networks. In *Proceedings of the 15th International Conference on Distributed Computing and Networking*, volume 8314 of *ICDCN '14*, pages 257–271. Springer, 2014.

[52] G.A. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Counting in anonymous dynamic networks under worst-case adversary. In *Proceedings of the IEEE 34th International Conference on Distributed Computing Systems*, ICDCS '14, pages 338–347. IEEE Computer Society, 2014.

[53] G.A. Di Luna, S. Bonomi, I. Chatzigiannakis, and R. Baldoni. Counting in anonymous dynamic networks: An experimental perspective. In *Proceedings of 9th International Symposium on Algorithms and Experiments for Sensor Systems*, volume 8243 of *ALGOSENSORS 2013*, pages 139–154. Springer, 2013.

[54] G.A. Di Luna, P. Flocchini, S. Gan Chaudhuri, F. Poloni, N. Santoro, and G. Viglietta. Mutual visibility by luminous robots without collisions. `http://giovanniviglietta.com/papers/obstruction2.pdf`. Submitted to International Journal.

[55] G.A. Di Luna, P.Flocchini, S. Gan Chaudhuri, N. Santoro, and G. Viglietta. Robots with lights: Overcoming obstructed visibility without colliding. In *Proceedings of 16th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, volume 8756 of *SSS '14*, pages 150–164. Springer, 2014.

[56] G.A. Di Luna, P.Flocchini, F. Poloni, N. Santoro, and G. Viglietta. The mutual visibility problem for oblivious robots. In *Proceedings of the 26th Canadian Conference on Computational Geometry*, CCCG '14. Carleton University, Ottawa, Canada, 2014.

[57] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: Random walk methods. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 123–132, New York, NY, USA, 2006. ACM.

[58] E. Le Merrer, A.-M. Kermarrec, and L. Massoulie. Peer to peer size estimation in large and dynamic networks: A comparative study. In *Proceedings of the 15th*

IEEE International Symposium on High Performance Distributed Computing, HPDC '06, pages 7–17. IEEE, 2006.

[59] O. Michail, I. Chatzigiannakis, and P. Spirakis. Naming and counting in anonymous unknown dynamic networks. In *procedings of the 2013 International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 281–295, 2013.

[60] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. In *Proceedings of 16th International Conference on Principles of Distributed Systems*, OPODIS '12, pages 269–283, 2012.

[61] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Naming and Counting in Anonymous Unknown Dynamic Networks. *ArXiv e-prints*, August 2012.

[62] O. Michail, I. Chatzigiannakis, and P.G. Spirakis. Brief announcement: Naming and counting in anonymous unknown dynamic networks. In *Proceedings of 26th International Symposium on Distributed Computing*, volume 7611 of *DISC '12*, pages 437–438. Springer Berlin Heidelberg, 2012.

[63] O. Michail, I. Chatzigiannakis, and P.G. Spirakis. Causality, influence, and computation in possibly disconnected synchronous dynamic networks. *Journal of Parallel and Distributed Computing*, 74(1):2016–2026, 2014.

[64] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 113–122, New York, NY, USA, 2006. ACM.

[65] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing*, DIALM-POMC' 05, pages 104–110, 2005.

[66] J. O'Madadhain, D. Fisher, S. White, and Y. Boey. The JUNG (Java Universal Network/Graph) Framework. `http://www.datalab.uci.edu/papers/JUNG_tech_report.html`, October 2003.

[67] B. Ribeiro and D. Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 390–403, New York, NY, USA, 2010. ACM.

[68] N. Sakamoto. Comparison of initial conditions for distributed algorithms on anonymous networks. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, PODC '99, pages 173–179. ACM, 1999.

[69] M. Yamashita and T. Kameda. Computing on an anonymous network. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing*, PODC '88, pages 117–130, New York, NY, USA, 1988. ACM.

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

[71] M. Yamashita and T. Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Transactions on Parallel and Distributed Systems*, 10(9):878–887, September 1999.