RESEARCH ARTICLE

Memory and communication efficient algorithm for decentralized counting of nodes in networks

Arindam Saha[1]*, **James A. R. Marshall**[1], **Andreagiovanni Reina**[1,2]

1 Department of Computer Science, University of Sheffield, Sheffield, United Kingdom, **2** IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

* a.saha@sheffield.ac.uk

OPEN ACCESS

Citation: Saha A, Marshall JAR, Reina A (2021) Memory and communication efficient algorithm for decentralized counting of nodes in networks. PLoS ONE 16(11): e0259736. https://doi.org/10.1371/journal.pone.0259736

Editor: Gabriele Oliva, University Campus Bio-Medico of Rome, ITALY

Received: April 28, 2021

Accepted: October 25, 2021

Published: November 22, 2021

Data Availability Statement: All relevant data are within the manuscript and its Supporting information files.

Funding: All the authors (A. S., J. A. R. M. and A. R.) acknowledge funding from the Office for Naval Research Global under grant no. 12547352 (the "Swarm Awareness" project; url:https://www.onr.navy.mil/Science-Technology/ONR-Global.aspx). A. R.also acknowledges support from the Belgian F.R. S.-FNRS, of which he is a Charge de Recherches. The funders had no role in study design, data

Abstract

Node counting on a graph is subject to some fundamental theoretical limitations, yet a solution to such problems is necessary in many applications of graph theory to real-world systems, such as collective robotics and distributed sensor networks. Thus several stochastic and naïve deterministic algorithms for distributed graph size estimation or calculation have been provided. Here we present a deterministic and distributed algorithm that allows every node of a connected graph to determine the graph size in finite time, if an upper bound on the graph size is provided. The algorithm consists in the iterative aggregation of information in local hubs which then broadcast it throughout the whole graph. The proposed node-counting algorithm is on average more efficient in terms of node memory and communication cost than its previous deterministic counterpart for node counting, and appears comparable or more efficient in terms of average-case time complexity. As well as node counting, the algorithm is more broadly applicable to problems such as summation over graphs, quorum sensing, and spontaneous hierarchy creation.

Introduction

All decentralized systems share the common aspect of being comprised of a network of units (which can be considered as graph nodes) that rely on local and partial information which they can gather from the subset of devices in their communication range (communication links can be represented as graph edges). An open challenge is to allow the units of these large-scale decentralized systems to estimate properties of the entire group.

A fundamental property that is crucial for the design and the efficient functioning of several systems is the system size, that is, the number of units in the system. Computing the exact network size in finite time with a decentralized algorithm with finite complexity is proved to be impossible [1]. Previously proposed solutions are therefore stochastic algorithms that only give an approximation of the system size, providing the possible advantages of robustness and speed. Deterministic algorithms provide the exact solution in a finite time, however, they may rely on stringent assumptions on the communication network topology. An overview of the

collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

existing algorithms is provided in Section *State of the art*. We propose, in Section *The aggregate-and-broadcast algorithm*, a new decentralized deterministic algorithm, the *aggregate-and-broadcast* (AnB) algorithm, that iteratively aggregates the node counts into a small number of local hubs which finally broadcast the count throughout the whole network. The AnB algorithm allows the nodes to compute the exact network size in a finite time when an upper bound is provided. In other words, the network size computed by the AnB algorithm is exact up to a limit that is bounded by the algorithm's execution time, as proved in the S1 File. The algorithm relies on the only two assumptions of a connected network and uniquely identifiable units (i.e. unique id), and requires minimal computation and communication capabilities of the units. The algorithm performance is analyzed and when possible compared with previous algorithms in terms of time, communication, and memory costs (see Section *Analysis of the algorithm*). The results indicate that the AnB algorithm is scalable, efficient, and accurate, with better performance than the existing algorithms in terms of smaller memory and communication costs. Therefore, as discussed in the *Conclusion*, the AnB algorithm can be beneficial for systems with constrained memory and communication, and has the potential to be employed in numerous application cases and impact a large variety of decentralized systems.

The problem statement

Consider a connected network $\mathcal{G} = (\mathcal{V}, E)$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of nodes in the network and $E \subseteq \mathcal{V} \times \mathcal{V}$ is the set of the edges of the network. The edges describe undirected and unweighted communication links between nodes, i.e. $(u, v) \Leftrightarrow (v, u) \in E$. Each node can only communicate at synchronous timesteps with its neighbors, where the set of neighbors of the generic node v is defined as $\mathcal{N}_i = \{u \in \mathcal{V} | (v, u) \in E\}$. We assume $\mathcal{G}$ to be time-invariant. Each node is characterized by a unique identifier (id). Each node knows an upper bound $N_{\max}$ of the network size, such that $N_{\max} \geq N$. In this paper, we propose an algorithm to be executed by every node of the network to allow them to compute the network size N in a finite number of iterations $t_{\max} \leq 4N_{\max} + 1$ (and therefore, a finite amount of time). Note that knowledge about $N_{\max}$ is only necessary in order to bound the number of iteration steps required for the execution of the algorithm to $t_{\max}$. This is required due to the results reported by Hendrickx et al. [1] who have proved that it would be otherwise impossible for a finite complexity algorithm to correctly count the number of nodes (see discussion in Sec. Stopping criteria).

State of the art

Most of the algorithms proposed to estimate the size of the network rely on stochastic methods. The most common approach relies on executing variations of random walks on the network [2–5]. In particular, Ganesh et al. [2] used continuous time random walks to obtain a target number of redundant node samples. The time required to obtain such a sample was then used to estimate the network size. In a different study, Gjoka et al. [3] compared various weighted random walk techniques. The study identified efficient methods to identify various macroscopic properties of the network by simulating weighted random walks on the network (e.g. Metropolis-Hastings Random Walk and Re-Weighted Random Walk). Similarly, Katzir et al. [4] proposed a method based on simulating multiple simultaneous random walks in order to estimate the size of the network. Building upon this work, Musco et al. [5] proposed an algorithm where multiple nodes execute random walks and compute the network size based on the degrees of the nodes encountered. Notable stochastic algorithms which do not involve random walks rely on either average consensus [6] or on order statistics consensus [7–10].

One of the shortcomings of stochastic algorithms is that their run-times depend on the desired accuracy of the results. Therefore, for applications where the size of the network is required to a high degree of accuracy, stochastic algorithms might take a long time to converge. For instance, the number of dynamical attractors in Boolean networks and their periodicities depend on whether the network size is even or odd, prime or composite [11]. Since dynamics on such networks are crucial in studying social networks, neural networks and gene and protein interaction networks [12–16], accurate knowledge of the network size is crucial. In such scenarios, deterministic algorithms to estimate the network size are better suited.

To the best of our knowledge, the number of deterministic algorithms for decentralized network node counting is very limited. One of the most trivial algorithms is the *All-2-All* method, as alluded to in Ref. [17]. It consists in having each node broadcasting a unique id together with all ids that it has already received so far. This simple algorithm is the most efficient algorithm we are aware of for deterministic network node counting on general network topologies. Other algorithms for node counting have been proposed for networks with specific topologies. For example, an algorithm inspired by the Breadth-First-Search (BFS) algorithm can be used on a tree network. In 2003, Bawa et al. [18] generalized such an algorithm so that it could be implemented on a network with a general topology. In their paper, the authors propose three different algorithms which may be used for computing various aggregates across the network. While the proposed algorithms are efficient, they investigated a different problem. They focus on the situations when the network size or the other aggregate quantities are sought by a single node of the network. When every node requires the size information, repeating the algorithm of [18] on every node becomes less efficient than the All-2-All method, as described in Sec. Analysis of the algorithm. Notably, numerous algorithms have been proposed to create a spanning tree on a general network. However, they are constrained in a crucial aspect as underlined in the next section.

Significance of the work

In Ref. [18], the authors propose algorithms to create a spanning tree on the given network. Once a tree is constructed, any information can be aggregated in the root by following the edges of the tree. Due to its important applications, numerous other algorithms [19–30] have also been proposed to construct a spanning tree on a connected network. All these algorithms, in order to build a spanning tree, require that one node of the network assumes the role of the root of the tree. However, selecting one node to assume such a role through a decentralized algorithm running on a sparse network is a difficult problem on its own. In fact, the network nodes would need to invest resources (time and computation) to reach a consensus on a single root node and avoid duplicates.

In this paper, we present an algorithm to create a ‘tree-like’ network to span a general connected network without assuming any particular node as a root. Instead of generating a tree from the root, our algorithm removes edges consecutively based on the local neighborhood of each node. This results in the emergence of possibly multiple ‘root-like’ nodes (which we call ‘residue’ nodes). Any information which was initially distributed among all nodes of the network can therefore be concentrated in these residue nodes. Thereafter, the information can be broadcast throughout the network.

In addition to relaxing the restriction of a selected root, the AnB algorithm performs better than the other known algorithms in terms of communication and memory costs than the existing algorithms. In fact, typically, the AnB algorithm, by creating multiple root-like nodes, decentralizes the computation to different parts of the network and thus nodes use on average less memory and send fewer messages. Our empirical analysis shows that the time costs of the algorithm depend crucially on the network topology: the proposed algorithm performs better than the previous algorithms for large random geometric networks but worse than them for other types of network topology. Hence, the AnB algorithm may prove to be useful in networks where the required memory per node is the major limiting factor or the limited communication between nodes is desirable.

Additionally, the proposed algorithm can be used to perform other collective tasks where aggregation of information is required but a distinguished root node cannot be identified (see the Conclusion).

The aggregate-and-broadcast algorithm

We propose the aggregate-and-broadcast (AnB) algorithm, a deterministic algorithm for the simultaneous and decentralized determination of the size N of a finite connected network by all its nodes. We assume that each node of the network has a unique id, can communicate only with its immediate neighbors, and knows N_{max}, the upper bound of the network size. Other than that, we make no prior assumptions about the topology of the network nor prior knowledge of the node. The underlying idea of the AnB algorithm is inspired by the standard node-counting method on a tree by its root. In a tree, the counts of the leaves are assimilated by their respective parents and then the leaves are iteratively pruned. Applying such an algorithm on a graph with a general topology poses a challenge since a strict hierarchy does not exist among the nodes. To overcome this problem, we add a step in each iteration where, based on the degree of its neighbors, each node determines its local hierarchy which, in turn, determines whether it should be pruned or not.

In the next subsections, we describe the proposed AnB algorithm in detail. We start with an overview of the entire algorithm in the next subsection. In subsections Pre-iteration steps and Iteration steps, we describe the pre-iteration steps (which include variable initialization) and the iteration steps of the algorithm respectively. Finally, in subsection Remarks on the AnB algorithm we compare the AnB algorithm to the standard node counting algorithm in trees and make some further remarks about the proposed algorithm. The correctness of the AnB algorithm is proved in Sec. Theorems and Proofs of the S1 File.

An overview of the AnB algorithm

Prior to the iterative steps, the nodes of the network are initialized as follows. The behavior of a node with id i at any particular instant is determined by its state s_i which can take one of four values during the course of the algorithm: 'active' (A), 'leaf' (L), 'residue' (R), or 'inactive' (I). The state of each node is initialized to $s_i = A$. Each node also starts with a local node counter $c_i = 1$. Since, at the beginning of the algorithm, each node is aware only of its own existence, the counter is initialized to 1. As the algorithm progresses, the node gathers information about the changing state of nodes (equivalent to the nodes getting 'pruned') from its neighbors and updates the value in c_i. Additionally, each node also has the following other internal variables: the set of its neighbors $\mathcal{N}_i$, its effective neighborhood $\mathcal{E}_i$, effective degree e_i, the set of residues $\mathcal{R}_i$ and final node count n_i. Among these, the first three variables are initialized to be empty sets $\mathcal{N}_i = \mathcal{E}_i = \mathcal{R}_i = \emptyset$, and the effective degree and final count variable are initialized as $e_i = n_i = 0$. Note, that it is assumed that the nodes of the network are synchronised and have a common sense of time. In other words, the nodes are aware of the beginning and end of each iteration step of the algorithm. Therefore, implementation of the AnB algorithm on a distributed system, e.g., a robot swarm or sensor network, will also need a mechanism to guarantee that synchronization is achieved and maintained.

From the perspective of a node, the AnB algorithm is divided into two phases: 'pre-reduction' and 'post-reduction'. A node is said to be in pre-reduction phase when its state is either $s_i = A$ or $s_i = L$. As this phase progresses, a node in 'active' state updates its local counter c_i by locally accumulating information from 'leaf' neighbors getting 'pruned' until the node itself changes its state to $s_i = L$ and becomes a 'leaf' node. Note that, here the term 'leaf' is used to denote a node which is about to be 'pruned' from the network; and not necessarily a node with only one neighbor. In the next iteration, each leaf node, depending on their effective neighborhood $\mathcal{E}_i$, again changes its state to either (a) $s_i = I$ and gets 'pruned', or (b) $s_i = R$ and becomes a residue node.

At the end of pre-reduction phase, the nodes of the network are either in residue ($s_i = R$) or inactive ($s_i = I$) states. These states can be considered analogous to the 'root' and the 'pruned leaves' of a tree network respectively. The residue nodes contain parts of the total count of nodes in the network. This is similar to the root of a tree network which contains the total node count of the entire tree after all the nodes have been pruned. This information is then broadcast across all other nodes and assimilated to give the final node count of the network. To do this, each residue node constructs a 'broadcast message' b_i, sends it to all its neighbors and changes its state to $s_i = I$. This broadcast message is then relayed by all nodes—irrespective of their state s_i—across the network. A node that receives a broadcast message adds the partial count to its final count variable n_i, and keeps track of the residue nodes to avoid double counting. Thus, after iteration steps $t_{\max}$, the variable n_i gives the total count of all nodes in the network. Further details of the algorithm and the the stopping criteria are provided in subsections Iteration steps and Stopping criteria respectively.

Algorithm 1: The aggregate-and-broadcast (AnB) algorithm for network node counting

```
let $s_i \leftarrow A$, $c_i \leftarrow 1$, $\mathcal{N}_i \leftarrow \emptyset$, $\mathcal{E}_i \leftarrow \emptyset$, $\mathcal{R}_i \leftarrow \emptyset$, $n_i \leftarrow 0$;
send message $m_{i,\text{echo}}$;
foreach message $m_{j,\text{echo}}$ received do
    $\mathcal{N}_i \leftarrow \mathcal{N}_i \cup \{j\}$;
set $e_i \leftarrow |\mathcal{N}_i|$;
send message $m_{i,\text{degree}} = e_i$;
foreach message $m_{j,\text{degree}}$ received do
    $\mathcal{E}_i \leftarrow \mathcal{E}_i \cup \{(j, e_j)\}$;
for $t = 0$; $t < t_{max}$; $t = t + 1$ do
    if $s_i = A$ then ;                                  // Executed if node is in Active state

        foreach message $m_{j,h}$ received do
            if $h =$ count then
                let $c_i \leftarrow c_i + m_{j,\text{count}}$;
                let $\mathcal{E}_i \leftarrow \mathcal{E}_i \setminus \{(j, e_j)\}$;
                let $e_i \leftarrow e_i - 1$;
                send message $m_{i,\text{reduce}}$;
            if $h =$ reduce then
                let $\mathcal{K}_j \leftarrow (j, e_j - 1)$, where $\mathcal{K}_j \in \mathcal{E}_i$;
        if messages $m_{j,\text{count}}$ are not received and $e_i \leq e_j \ \forall (j, e_j) \in \mathcal{E}_i$ then
            send message $m_{i,\text{leaf}}$;
            let $s_i \leftarrow L$;

    else if $s_i = L$ then// Executed if node is in Leaf state
        foreach message $m_{j,\text{leaf}}$ received do
            let $e_i \leftarrow e_i - 1$;
        if $e_i = 0$ then
            let $s_i \leftarrow R$;
        else
            send message $m_{i,\text{count}} = \frac{c_i}{e_i}$;
            let $s_i \leftarrow I$;

    else if $s_i = R$ then ;                             // Executed if node is in Residue state

        let $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{i\}$ and $n_i \leftarrow n_i + c_i$;
        send message $m_{i,\text{broadcast}} = (i, c_i)$ ;
        let $s_i \leftarrow I$;

    foreach message $m_{j,\text{broadcast}} = (k, c_k)$ received do ;          // Executed at each
    iteration

        if $(k \notin \mathcal{R}_i)$ then
            let $\mathcal{R}_i \leftarrow \mathcal{R}_i \cup \{k\}$ and $n_i \leftarrow n_i + c_k$;
            send message $m_{i,\text{broadcast}} = m_{j,\text{broadcast}}$

```

Table 1. The different types of messages $m_{i,h}$ used in the AnB algorithm.

h	Content	Role of the message
echo	-	Indicates the presence of the sender i
degree	e_i	Sends the initial effective degree e_i of the sender i
leaf	-	Indicates the transition of the sender i to leaf state
count	c_i	Sends the local count c_i of the sender i
reduce	-	Indicates the reduction of effective degree e_i of the sender i
broadcast	(k, c_k)	Sends or relays the broadcast message

https://doi.org/10.1371/journal.pone.0259736.t001

Pre-iteration steps

We now describe the AnB algorithm in detail. The actions taken by a node i in a particular step are determined by its internal variables and the messages it receives from its neighbors, *i.e.* the nodes in $\mathcal{N}_i$.

Any message sent by a node is denoted as $m_{i,h}$, where i is the sender of the message and h is the 'type' of the message. The 'type' of the message determines the action to be taken by the receiver of the message. The various types of messages and their roles are summarized in Table 1. Note that every message is broadcast to the entire neighborhood $\mathcal{N}_i$ and thus, can be accessed by all nodes in $\mathcal{N}_i$.

After the initialization of all internal variables, each node of the network identifies its neighborhood. To do so, it sends a message $m_{i,\text{echo}}$ indicating its presence to all its neighbors. It then receives similar messages $m_{j,\text{echo}}$ from other nodes. The set of all nodes from which such a message is received is then identified as the neighborhood $\mathcal{N}_i$ (Line 4).

One of the most crucial internal variables for the node is its effective degree e_i which is the number of its neighbors which are in the active state ($s_i = A$). Since all nodes start in the active state, the initial effective degree of the node is the number of elements in its neighborhood: $e_i = |\mathcal{N}_i|$. In addition to its own effective degree, the node also needs to be aware of the effective degrees of those neighbors which are in active state. The node keeps track of this information in form of its effective neighborhood,

$$\mathcal{E}_i = \{(j, e_j) : j \in \mathcal{N}_i \text{ and } s_j = A\}. \tag{1}$$

Therefore, $\mathcal{E}_i$ is a set of tuples where the first element of the tuple is the id of an active neighbor of i and the second element is the effective degree of the neighbor.

The identification of neighborhood also allows the node to compute its initial effective degree $e_i = |\mathcal{N}_i|$ and to send it to its neighbors as $m_{i,\text{degree}}$. Thereafter, a node i receiving a message $m_{j,\text{degree}}$ updates its effective neighborhood $\mathcal{E}_i$ as described in Line 8.

Iteration steps

After the pre-iteration steps, the node i enters an iterative phase where its steps are determined by its state s_i. The details of these state-dependent steps are illustrated in the finite state machine of Fig 1 and are elaborated as follows.

- **Active nodes:** Each active node i with $s_i = A$ first detects any change in its neighborhood. This change can be of two types: (a) Either some of its neighbors are transitioning to inactive state (message with h = count); or (b) the effective degree of some of its neighbors is being reduced (message with h = reduce). Therefore, upon receipt of a message $m_{j,\text{count}}$, the node i

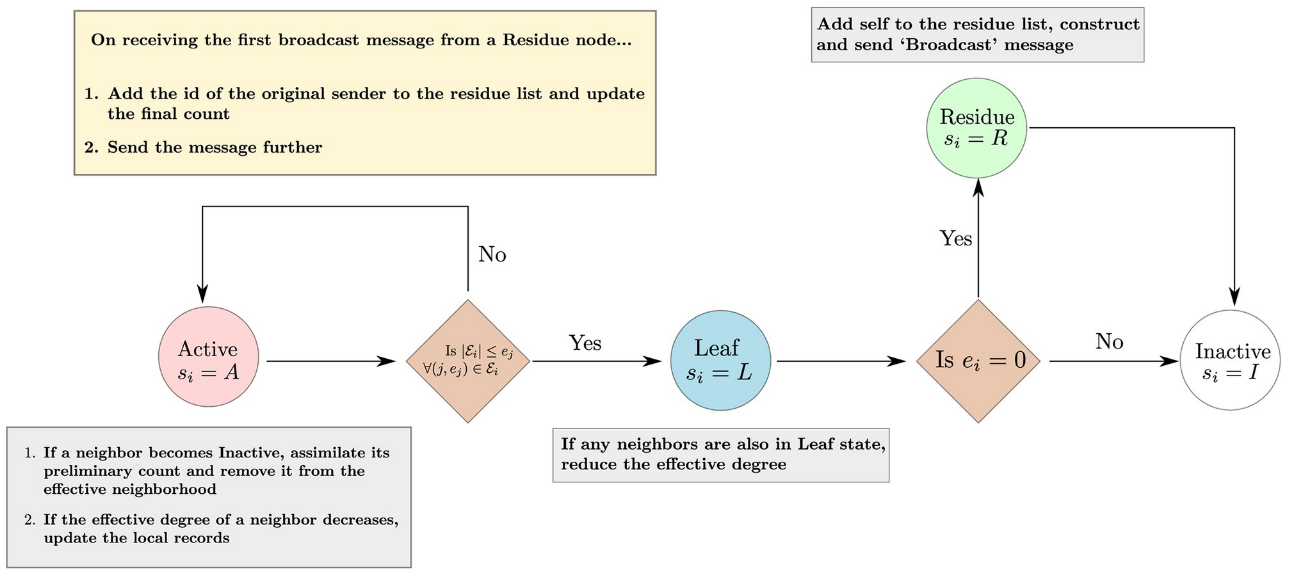
On receiving the first broadcast message from a Residue node...
1. Add the id of the original sender to the residue list and update the final count
2. Send the message further
Add self to the residue list, construct and send 'Broadcast' message
Residue $s_i = R$
Yes
No
Active $s_i = A$
Is $|\mathcal{E}_i| \leq e_j$ $\forall (j, e_j) \in \mathcal{E}_i$
Yes
Leaf $s_i = L$
Is $e_i = 0$
No
Inactive $s_i = I$
1. If a neighbor becomes Inactive, assimilate its preliminary count and remove it from the effective neighborhood
2. If the effective degree of a neighbor decreases, update the local records
If any neighbors are also in Leaf state, reduce the effective degree

Fig 1. Schematic flowchart depicting the finite state machine of each node of the network executing the AnB algorithm. Note that the colors of the circles correspond to the colors of the section in Algorithm An overview of the AnB algorithm. Also, the steps outlined in the yellow box are carried out by all nodes irrespective of their state.

https://doi.org/10.1371/journal.pone.0259736.g001

excludes the sender from its effective neighborhood $\mathcal{E}_i$, decreases its effective degree e_i by 1 and assimilates the contents of the message in its local count (Line 14),

$$c_i = c_i + m_{j,\text{count}}. \tag{2}$$

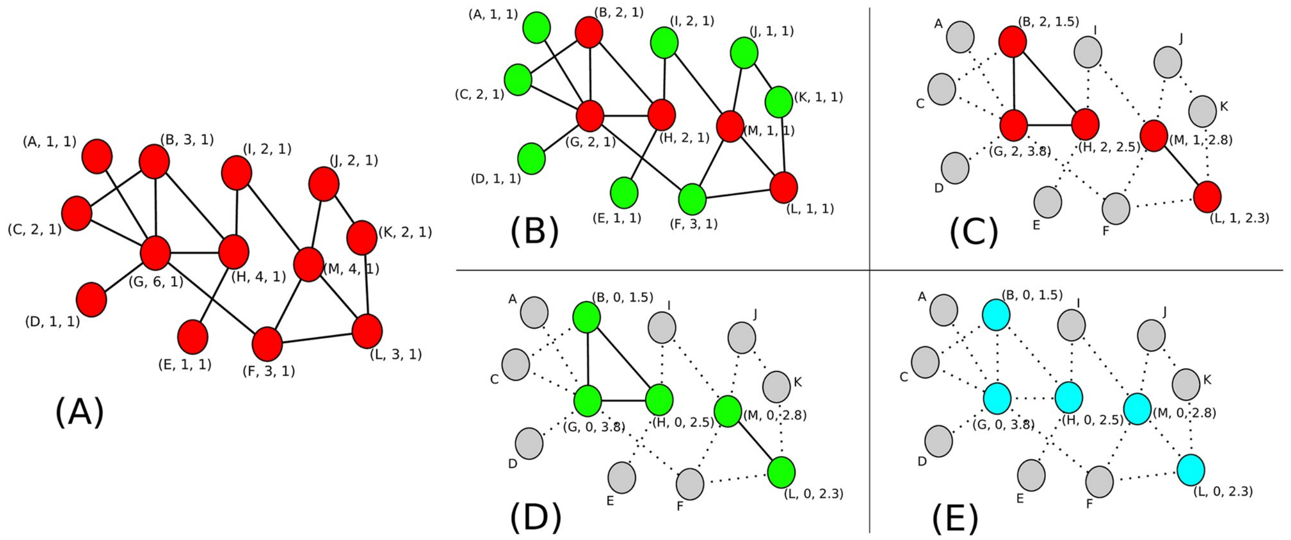

Fig 2. Demonstration of the AnB algorithm on a typical network. Panel (A) shows the initial state of a network of size 13. Panels (B) through (E) show the successive steps in the execution of the aggregate phase of the algorithm. The nodes in Active, Leaf, Residue and Inactive states are shown in red, green, cyan and grey colors respectively. The bracket beside each node shows the id i of the node, the number of its active neighbors e_i, and its current local count c_i, as an ordered tuple. For inactive nodes, only the id of the node is shown because the quantities e_i and c_i are not required in the inactive state. Once the network reaches a state where only residue and inactive nodes are present in the network (as seen in Panel E), the AnB algorithm enters its broadcast phase and the individual local counts of the residue nodes are broadcast throughout the network.

https://doi.org/10.1371/journal.pone.0259736.g002

Since the effective degree of node i is decreased by 1, it sends a message $m_{i,\text{reduce}}$ to its neighbors. For each message of type $h = 5$ received, the node updates the record of the effective degree corresponding to the sender of the message (Line 19).
After processing the incoming messages, the node i checks for the two conditions indicated in Line 20. If both conditions are met, the node sends a message $m_{i,\text{leaf}}$ and changes its state to $s_i = L$; otherwise, the node stays in the active state for the next iteration.

- **Leaf nodes:** The node i in state $s_i = L$ stays in this state for exactly one iteration and then changes its state to either $s_i = R$ or $s_i = I$. First, it processes any incoming message of the type h = leaf. The reception of any such message implies that some of its neighbors have transitioned to the leaf state in the same time step, and are therefore no longer in the active state. For each message $m_{j,\text{leaf}}$ received, the effective degree e_i of the node is reduced by one. After processing all incoming messages, the node i changes its state; if the effective degree $e_i = 0$, it change its state to $s_i = R$ otherwise, it sends the message
$$m_{i,\text{count}} = \frac{c_i}{e_i} \tag{3}$$
and changes state to $s_i = I$ (Lines 27–31).

- **Residue nodes:** Each node i in state $s_i = R$ updates its residue set $\mathcal{R}_i$ with its own id i and the total node counter n_i adding its local counter c_i. It then broadcasts a message $m_{i,\text{broadcast}} = (i, c_i)$ and changes its state to $s_i = I$.

- **All nodes:** While the previous steps are executed by nodes in a specific state, the following steps are executed by all nodes of the network at each iteration irrespective of their state. Whenever a node i receives a message $m_{j,\text{broadcast}} = (k, c_k)$ from any of its neighbors, it checks if node k is in the residue set $\mathcal{R}_i$. If $k \notin \mathcal{R}_i$, the node i adds k to its residue set $\mathcal{R}_i = \mathcal{R}_i \cup \{k\}$, adds the corresponding local count c_k to its final node count $n_i = n_i + c_k$ and finally relays the message forward by sending message $m_{i,\text{broadcast}} = m_{j,\text{broadcast}}$.

After a sufficient number of iteration steps $t_{\max}$, all nodes converge to the same final count n_i equal to the network size N. A detailed analysis of the convergence time is provided in Sec. Time Cost of the S1 File. An illustration of the working of the aggregate phase of the AnB algorithm is shown in Fig 2.

Stopping criteria

The AnB algorithm terminates when sufficient iteration steps, $t_{\max}$, has passed. This $t_{\max}$ should be sufficiently large so that each broadcast message reaches every node of the network. However, determining an exact value for $t_{\max}$ is impossible as reported by Hendrickx et al. [1] who have shown that it is impossible for a finite complexity algorithm to correctly estimate the size of a network with probability one. If $t_{\max}$ could be exactly determined for the network, we would be absolutely sure that each residue message has reached every node and hence, each node is aware of the size of the network. This would be in direct violation of the aforementioned result. However, depending on the prior knowledge about the network, various estimates of $t_{\max}$ can be made as follows. In Sec. Theorems and Proofs (Corollary 1) of the S1 File, we show that the maximum time required for all nodes to reach the final state, i.e., the inactive state, has the above boundary of $t_r = 3N + 2$. It is also trivial that the number of time steps required to broadcast a message across a network of size N is, in the worst-case, $t_b = N - 1$.

Therefore, $t_{\max}$ is bounded above by $t_r + t_b = 4N + 1$. Hence, if an overestimate $N_{\max}$ of the network size is known apriori, we can set $t_{\max} = 4N_{\max} + 1$ to know the exact size of the network in finite time.

Remarks on the AnB algorithm

As shown in Fig 1, a node spends exactly one iterative step as a leaf, and at most one iterative step as a residue node. Therefore, a typical node spends most of its iterative steps in either active or inactive states.

We can now elaborate on the similarities and differences between the proposed AnB algorithm and the standard node-counting method on a tree network which were indicated earlier. On comparion, we note the following points of interest.

1. Nodes in a tree network can also be classified into four categories analogous to those in the AnB algorithm: (a) the root (similar to $s_i = R$), (b) leaves (similar to $s_i = L$), (c) pruned leaves (similar to $s_i = I$) and (d) other nodes still in the network (similar to $s_i = A$).
2. In a tree network, leaves are easily identified as nodes with degree one. Since this is not true for a general network, we use the condition in Line 1 to identify, at each iteration step, the nodes which are to be labeled as leaves.
3. After a node has been identified as a leaf in a tree network, it passes on its local count to its parent and gets transformed to a pruned leaf. In a tree network, the parent of each node is unique. However, in a general network, a leaf node may have more than one parent. Therefore, in the AnB algorithm, the local count of each leaf is divided equally among all parents to avoid over-counting number of nodes.
4. Once the counts have been passed on, the leaf node becomes an inactive node, similar to the pruned leaves in a tree network. If there are no active neighbors ('parents') to which a node can pass on its local count, it becomes a residue node, which is similar to the root of the tree. While the structure of the tree implies that there can be only one root of a tree, there is no such restriction for a general network. Hence, the count of the size of a general network gets concentrated into the residue nodes which is then broadcast and recombined in the final stages of the AnB algorithm.

It is to be noted that each node checks for the reception of a message of type h = broadcast at each iteration. This is necessary because messages of type h = broadcast carry the node count of a part of the network as counted by a residue node. Therefore, all nodes which receive such a message should add it to their final count and send it further. This is in contrast with the other types of messages which are intended only for nodes in active or (as in case of h = leaf) leaf states.

Analysis of the algorithm

In this section, we demonstrate the correctness of the AnB algorithm and analyze the algorithm performance in terms of time, communication, and memory costs against the known node-counting algorithms. We do not compare AnB with stochastic algorithms which only compute an estimate of the network size that increases over time, but we limit our comparison against algorithms that return the exact node count in a finite time: the All-2-All algorithm and the Single Tree (ST) algorithm [18].

The All-2-All algorithm is, to the best of our knowledge, the only known deterministic algorithm for node counting which can work on any type of connected network regardless of its topology. In the All-2-All algorithm, each node broadcasts its id, and all received ids, to all its neighbors and every node counts the number of received unique ids.

The ST algorithm, instead, is the most efficient of the three algorithms proposed by Bawa et al. in [18]. Despite being stochastic, the ST algorithm is proved to return the exact network size in a finite time. The ST algorithm, similarly to AnB, relies on the construction of a tree-like hierarchy. However, in its original form, the ST algorithm allows only a single node to compute the network size. In order to allow all the nodes of the network to know the network size, the ST algorithm can be extended in the following two ways: (a) one randomly selected node executes the ST algorithm and then broadcasts the computed size to all other nodes; or (b) all the nodes of the network simultaneously execute the ST algorithm and compute the network size independently. Employing alternative (a) requires the nodes to be able to select in a decentralized way which node will execute the ST algorithm. Decentralized node-selection adds a new problem which may require further assumptions on the network topology or on the initial knowledge of the nodes [31]. Therefore, in our comparison against the ST algorithm, we employ alternative (b) by which every node makes an independent count of the network size.

We provide a comparison both as worst-case algorithm complexity and with generic analytical equations for each type of cost. When such analytical solutions are not possible, we provide the results of numerical simulations for specific graph topologies. In fact, the AnB algorithm is proved to work on any connected graph regardless on the graph topology. Through our analysis, we highlight the differences in performance for each topology.

Correctness of the AnB algorithm

In Sec. Theorems and Proofs of the S1 File, a detailed proof of correctness of the algorithm is provided. A brief sketch of the proof is as follows. We begin by identifying a sequence of time steps of the algorithm when the variables e_i and $\mathcal{E}_i$ correctly give correct information about the neighborhood of the node i (see Theorem 1). We say that, at these time steps, the network is in the *resting state*. We then show that, as the network progresses from one resting state to another, the number of active states decreases. During this process, the information about their local node counts c_i gets concentrated into the nodes which pass through the residue state (see Theorem 2). Therefore, when no active nodes are present in the the network, the information about the size of the network is concentrated in the nodes which passed through the residue state. This information is then broadcast throughout the network and is accumulated by each node (see Theorem 3).

Comparison with other algorithms in terms of complexity

We compare the efficiency of the AnB algorithm against the All-2-All and the Single Tree (ST, [18]) algorithms in terms of three aspects: (a) the time required to compute the network size by every node, (b) the number of messages sent by all nodes (i.e. the communication cost), and (c) the minimum amount of memory required by each node to execute the algorithm (i.e. the memory cost).

Note that, it is difficult to compare the efficiency of AnB against most other stochastic algorithms because their efficiency depends on the desired accuracy of the results. The more accurate we want the results to be, the longer the stochastic algorithms should run, at the cost of increased time and/or communication costs. On the other hand, deterministic algorithms like ours give accurate results in a finite time and make possible asymptotic performance analysis.

The efficiency results for the AnB algorithm are derived in Sec. Complexity Analysis of the S1 File and reported in Table 2. We derive exact results for the communication and memory

Table 2. Exact costs for the two algorithms for a general network with diameter *D*, average degree *d*, and *r* residue nodes. For memory cost, we indicate the individual degree d_i for the generic node *i*. The AnB algorithm is more efficient than the All-2-All and the ST methods in terms of memory and communication. Analytical solution for time is out of reach and we provide numerical results in Fig 3.

Algorithm	Time	Communication	Memory
AnB	numerically in Fig 3	$N(4 + r + d) - r$	$(2d_i + r + 5)\log(N)$
All-2-All	D	N^2	$N\log(N)$
ST	$2D$	$2N^2$	$2N\log(N) + d_i N$

https://doi.org/10.1371/journal.pone.0259736.t002

costs. Instead, computing a precise equation of the time cost is difficult, as it depends strongly on the topology of the network which evolves at every time step (see discussion in Sec. Time Cost). Through Theorem 3 in Sec. Theorems and Proofs, we computed the upper bound of the time complexity of AnB. To analyse the exact performance in terms of time, instead, we computed a set of numerical simulations on various graph topologies whose results are shown in Fig 3. In particular, we implemented and tested the AnB algorithm on four different types of random networks as listed in Table 3. The results of our analysis show a qualitative difference in algorithm performance as a function of the network topology. We employed these numerical simulations to compare the temporal performance of AnB with the All-2-All algorithm and to make general considerations on the execution time of the AnB algorithm (see also Sec. Time Cost).

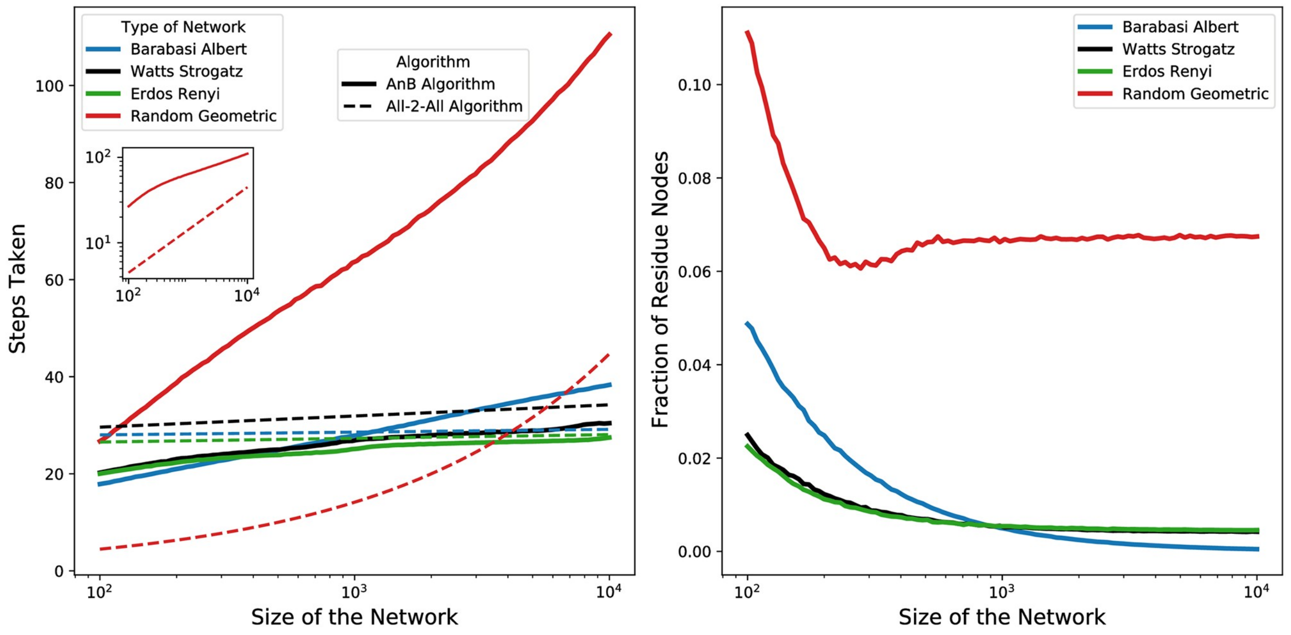

Fig 3. Numerically estimated time costs of the AnB algorithm. The left panel shows, on a log-linear scale, the total number of iterative steps taken by the AnB algorithm for different random networks (solid lines). The dashed lines show the scaling of the time for the All-2-All method, which corresponds to the network diameter *D* from Table 3. The diameter is known up to a scaling factor, here we report curves scaled to values comparable to AnB's execution time to ease the comparison. In fact, the intersection of same-colour curves indicates that for large networks, the AnB algorithm is asymptotically slower than the All-2-All method. This is the case for all the analyzed network topologies but the Random Geometric networks. In RG networks, All-2-All shows a steeper curve that would slow down the process for very large networks (see inset on a log-log scale). The right panel shows the fraction of residue nodes $x = \frac{r}{N}$ in the network. Low *x* implies low *r* and hence better performance of AnB algorithm in terms of memory and communications cost (see Table 2). For each network size, we report the average results for the simulation of 1,000 independent random networks. (95% confidence intervals are reported in the left panel as shades but often are smaller than the line width).

https://doi.org/10.1371/journal.pone.0259736.g003

Table 3. The analyzed networks.

Type of network	Constructing algorithm	Parameters	Diameter *D*
Scale-free	Barabasí Albert [32]	$m = 10$	$\frac{\log N}{\log \log N}$, [33]
Random	Erdös Renyí [34]	$p_e = \frac{20}{N}$	$\frac{\log N}{\log (p_e N)}$, [35]
Small-world	Watts Strogatz [36]	$k = 20, p_r = 0.5$	$\log N$, [33]
Random Geometric	Penrose [37]	$r = \sqrt{\frac{10}{N}}$	$\frac{\sqrt{2}}{r}$, [38]

Description of the internal parameters: m: Number of edges with which a new node attaches to existing nodes; p_e: Probability of forming an edge; k: Number of nearest neighbors to which the node initially connects; p_r: Rewiring probability; r: Threshold distance unto which two nodes are connected.

https://doi.org/10.1371/journal.pone.0259736.t003

The time, communication, and memory costs for All-2-All algorithm are relatively easy to compute. In terms of time, the algorithm ends when the messages created by every node (containing its id) reach every other node. Therefore, the time steps required for this to happen is equal to the diameter D of the network. In terms of communication, since each node broadcasts the id of every node to its neighborhood, the number of messages sent by each node is N and hence the total number of messages sent in the whole network is N^2. Finally, in terms of memory, each node needs to store the id of every node in the network. Therefore, the minimum memory required by each node is $N \log(N)$, by assuming that each id needs at least $\log(N)$ bits.

The time and communication efficiency of the ST algorithm has been outlined by Bawa et al. in [18]. We updated their efficiency measures in order to include the changes required to allow all nodes to compute the network size. Additionally, we derived the memory cost which was not originally indicated in [18]. The details of the complexity analysis are reported in Sec. Complexity Analysis of the S1 File; the results are reported in Table 2.

The results in Table 2 show that the AnB algorithm has the lowest costs in terms of memory and computation compared with the All-2-All and ST algorithms (see also Fig 4). The efficiency of the AnB algoritms is higher for networks which have the number of 'residue' nodes r much smaller than N. This is the case for most random networks as shown in Fig 3 (right panel). Our analysis also shows that the largest share of communication messages are typically sent by the residue nodes and the largest memory is typically required to store the ids of the residue nodes. Since the fraction of residue nodes is low for all the analyzed network classes, with the AnB algorithm the nodes send comparatively fewer messages and have lower memory requirements than with the All-2-All and ST algorithms. The only cases where the All-2-All and ST algorithms might perform better than AnB in terms of memory and communication are completely connected networks, almost completely connected networks, and networks with specific topologies (such as ring networks; see detailed discussion in Sec. Performance on Ring and Complete Networks of the S1 File). In terms of time, Fig 3 (left panel) shows that the All-2-All method scales as the network diameter D and the AnB algorithm has comparable, or slightly worse, time performance. Finally, in terms of all three complexity aspects (time, communication, and memory), in the worst case (i.e., when $d_i = N - 1$ and $r = N$), the AnB algorithm has an asymptotically complexity equal to the other algorithms (see Table 4 in the S1 File). Therefore, we conclude that the AnB algorithm is advantageous for applications with constrained or high-cost communication and memory, as confirmed by the results reported in Table 2 and Fig 4.

Conclusion

In this paper, we propose the AnB algorithm, a deterministic algorithm by which all nodes of a network can become aware of its size. The AnB algorithm assumes no inherent hierarchy

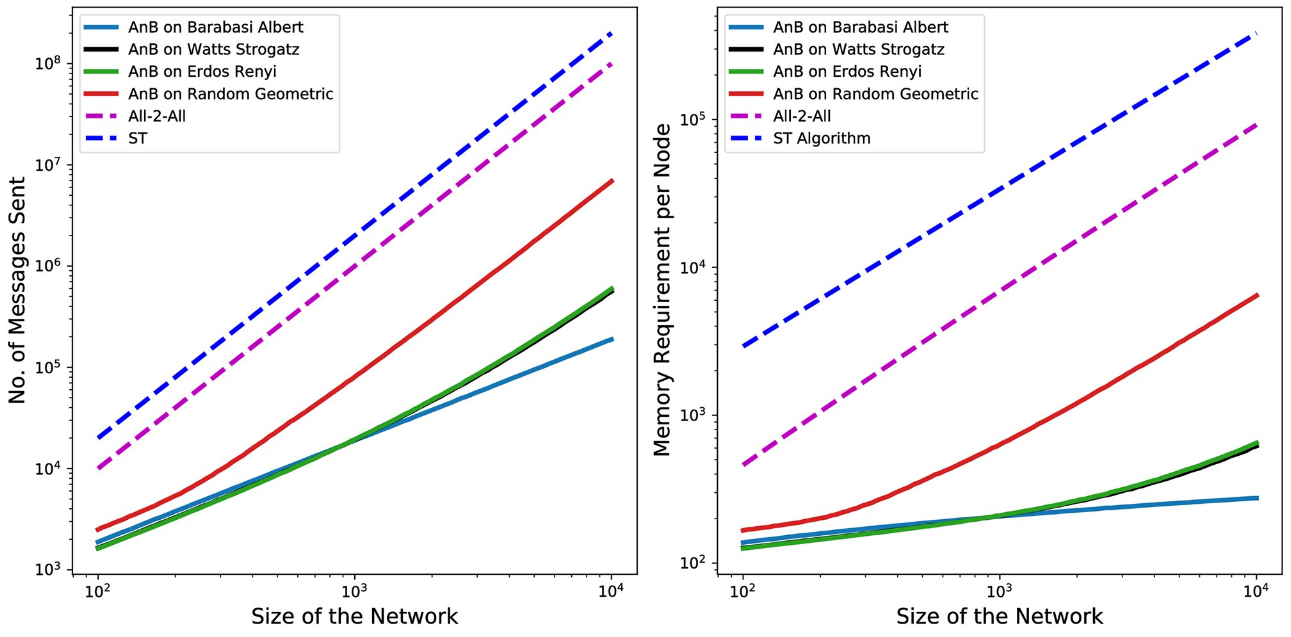

Fig 4. The AnB is the most efficient algorithm in terms of communication and memory costs, compared with the All-2-All and ST algorithms. The left panel shows the total number of messages sent by the nodes. The right panel shows the corresponding memory requirements per node with average connectivity degree *d*. In both panels, the dashed lines show the scaling for the All-2-All and ST algorithms, whereas the solid lines of various colors show the scaling for the AnB algorithm. Note that the number of messages sent and the memory requirements depends only on the network size for All-2All and ST algorithms and hence, are independent of the network topology. However, the number of messages sent and the memory requirements for AnB algorithm depends on the number of residue nodes which in turn depends on the topology of the network. Therefore, their dependence on the network topology is also explicitly shown.

https://doi.org/10.1371/journal.pone.0259736.g004

among the nodes and no prior knowledge of the network topology. Instead, it depends on (a) the nodes having unique ids and (b) the nodes being able to communicate with its immediate neighbors. We also analyze the efficiency of the AnB algorithm and compare it against the known algorithms. We conclude that the AnB algorithm is significantly better than the known deterministic algorithms on average in terms of memory and communication costs. This has potential benefits in engineering where decentralized systems composed of a large number of units that operate without a central controller are spreading in various application domains, since they can offer scalable, cost-effective, robust solutions. Three examples of such domains are swarm robotics [39], internet of things [40], and wireless sensor networks [41].

In this concluding section, we outline some of the salient features of the AnB algorithm and the ways in which it can be extended and applied to various physical systems.

1. **Quorum sensing:** It is notable that the local node counter c_i and the final count variable n_i are monotonic functions of time. Since both variables are aggregates of the size of the network, max(c_i, n_i) gives a lower bound of the network size at any point in time. This can be useful in systems which are trying to determine if a quorum is present on not [42]. Since in these cases the system is trying to determine if the network size is above a certain threshold or not, a node i can enter the broadcast phase as soon as c_i is greater than the threshold and inform the other nodes of the quorum being reached.

2. **Spontaneous hierarchy creation:** While the AnB algorithm assumes no hierarchy among the nodes, the progression of the algorithm can be used to create it depending on the time when a node enters the broadcast phase. If a node enters the broadcasting phase late, it is more likely to be connected to nodes with high degrees, and hence be more 'central'.

Conversely, if a node enters the broadcasting phase earlier, it is more likely to be 'peripheral'. While various other centrality measures exist for such classification of nodes in a network (for instance, closeness centrality [43] and betweenness centrality [44]), they generally require the computation and ordering of a measure by a centralized agency. In the proposed AnB algorithm, the nodes can spontaneously organize themselves into a hierarchy.

3. **Computation of other aggregate quantities:** Similar to other previously known algorithms of network size estimation [5, 18], the AnB algorithm can also be used to compute other global properties across networks. For example, if each node i is associated with a property s_i, they can compute the sum $\sum s_i$ by simply setting $c_i = s_i$ and executing the AnB algorithm. Similarly, other aggregate quantities such as averages and maximums/minimums can also be computed by suitably adopting the AnB algorithm.

Supporting information

S1 File.
(PDF)

Author Contributions

Conceptualization: Arindam Saha.

Formal analysis: Arindam Saha.

Funding acquisition: James A. R. Marshall, Andreagiovanni Reina.

Methodology: Arindam Saha, Andreagiovanni Reina.

Supervision: James A. R. Marshall, Andreagiovanni Reina.

Validation: Arindam Saha.

Visualization: Arindam Saha.

Writing – original draft: Arindam Saha.

Writing – review & editing: James A. R. Marshall, Andreagiovanni Reina.

References

1. Hendrickx JM, Olshevsky A, Tsitsiklis JN. Distributed Anonymous Discrete Function Computation. IEEE Transactions on Automatic Control. 2011; 56(10):2276–2289. https://doi.org/10.1109/TAC.2011.2163874
2. Ganesh AJ, Kermarrec AM, Le Merrer E, Massoulié L. Peer counting and sampling in overlay networks based on random walks. Distributed Computing. 2007; 20(4):267–278. https://doi.org/10.1007/s00446-007-0027-z
3. Gjoka M, Kurant M, Butts CT, Markopoulou A. Walking in facebook: A case study of unbiased sampling of OSNs. In: IEEE International Conference on Computer Communications. IEEE; 2010. p. 1–9. Available from: http://ieeexplore.ieee.org/document/5462078/.
4. Katzir L, Liberty E, Somekh O, Cosma IA. Estimating sizes of social networks via biased sampling. Internet Mathematics. 2014; 10:335–359. https://doi.org/10.1080/15427951.2013.862883
5. Musco C, Su HH, Lynch NA. Ant-inspired density estimation via random walks. Proceedings of the National Academy of Sciences. 2017; 114(40):10534–10541. https://doi.org/10.1073/pnas.1706439114 PMID: 28928146
6. Jelasity M, Montresor A. Epidemic-style proactive aggregation in large overlay networks. In: 24th International Conference on Distributed Computing Systems; 2004. p. 102–109.

7. Brambilla M., Pinciroli C., Birattari M., Dorigo M. A reliable distributed algorithm for group size estimation with minimal communication requirements In: 2009 International Conference on Advanced Robotics; 2009.

8. Lucchese R, Varagnolo D. Networks cardinality estimation using order statistics. In: 2015 American Control Conference (ACC). IEEE; 2015. p. 3810–3817. Available from: http://ieeexplore.ieee.org/document/7171924/.

9. Varagnolo D, Pillonetto G, Schenato L. Distributed cardinality estimation in anonymous networks. IEEE Transactions on Automatic Control. 2014; 59(3):645–659. https://doi.org/10.1109/TAC.2013.2287113

10. Lucchese R, Varagnolo D, Delvenne JC, Hendrickx J. Network cardinality estimation using max consensus: The case of Bernoulli trials. In: 2015 54th IEEE Conference on Decision and Control (CDC). IEEE; 2015. p. 895–901. Available from: http://ieeexplore.ieee.org/document/7402342/.

11. Drossel B, Mihaljev T, Greil F. Number and length of attractors in a critical Kauffman model with connectivity one. Physical Review Letters. 2005; 94(8):1–4. https://doi.org/10.1103/PhysRevLett.94.088701

12. Green DG, Leishman TG, Sadedin S. The Emergence of Social Consensus in Boolean Networks. In: IEEE Symposium on Artificial Life; 2007. p. 402–408.

13. Cheng D. Input-State Approach to Boolean Networks. IEEE Transactions on Neural Networks. 2009; 20(3):512–521. https://doi.org/10.1109/TNN.2008.2011359 PMID: 19224735

14. Davidich MI, Bornholdt S. Boolean Network Model Predicts Cell Cycle Sequence of Fission Yeast. PLOS ONE. 2008; 3(2):1–8. https://doi.org/10.1371/journal.pone.0001672 PMID: 18301750

15. Kauffman S, Peterson C, Samuelsson B, Troein C. Genetic networks with canalyzing Boolean rules are always stable. Proceedings of the National Academy of Sciences of the United States of America. 2004; 101(49):17102–17107. https://doi.org/10.1073/pnas.0407783101 PMID: 15572453

16. Kauffman S. Homeostasis and differentiation in random genetic control networks. Nature. 1969; 224 (5215):177. https://doi.org/10.1038/224177a0 PMID: 5343519

17. Evers J, Kiss D, Kowalczyk W, Navilarekallu T, Renger M, Sella L, et al. Node counting in wireless ad-hoc networks. Proceedings of the 79th European Study Group Mathematics with Industry. 2011; p. 49–73.

18. Bawa M, Garcia-Molina H, Gionis A, Motwani R. Estimating Aggregates on a Peer-to-Peer Network. Stanford InfoLab; 2003. 2003-24. Available from: http://ilpubs.stanford.edu:8090/586/.

19. Arora A, Gouda M. Distributed reset. In: Nori KV, Veni Madhavan CE, editors. Foundations of Software Technology and Theoretical Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg; 1990. p. 316–331.

20. Dolev S, Israeli A, Moran S. Self-stabilization of dynamic systems assuming only read/write atomicity. Distributed Computing. 1993; 7(1):3–16. https://doi.org/10.1007/BF02278851

21. Afek Y, Kutten S, Yung M. Memory-efficient self stabilizing protocols for general networks. In: van Leeuwen J, Santoro N, editors. Distributed Algorithms. Berlin, Heidelberg: Springer Berlin Heidelberg; 1991. p. 15–28.

22. Awerbuch B, Kutten S, Mansour Y, Patt-Shamir B, Varghese G. Time optimal self-stabilizing synchronization. In: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing; 1993. p. 652–661.

23. Burman J, Kutten S. Time Optimal Asynchronous Self-stabilizing Spanning Tree. In: Pelc A, editor. Distributed Computing. Berlin, Heidelberg: Springer Berlin Heidelberg; 2007. p. 92–107.

24. Collin Z, Dolev S. Self-stabilizing depth-first search. Information Processing Letters. 1994; 49(6):297–301. https://doi.org/10.1016/0020-0190(94)90103-1

25. Cournier A. A New Polynomial Silent Stabilizing Spanning-Tree Construction Algorithm. In: Kutten S, Žerovnik J, editors. Structural Information and Communication Complexity. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010. p. 141–153.

26. Cournier A, Devismes S, Petit F, Villain V. Snap-Stabilizing Depth-First Search on Arbitrary Networks. The Computer Journal. 2006; 49(3):268–280. https://doi.org/10.1093/comjnl/bxh154

27. Cournier A, Devismes S, Villain V. A Snap-Stabilizing DFS with a Lower Space Requirement. In: Tixeuil S, Herman T, editors. Self-Stabilizing Systems. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 33–47.

28. Cournier A, Devismes S, Villain V. Light Enabling Snap-Stabilization of Fundamental Protocols. ACM Trans Auton Adapt Syst. 2009; 4(1). https://doi.org/10.1145/1462187.1462193

29. Datta AK, Larmore LL, Vemula P. Self-Stabilizing Leader Election in Optimal Space. In: Kulkarni S, Schiper A, editors. Stabilization, Safety, and Security of Distributed Systems. Berlin, Heidelberg: Springer Berlin Heidelberg; 2008. p. 109–123.

30. Kosowski A,Kuszner Ł. A Self-stabilizing Algorithm for Finding aÂ Spanning Tree in a Polynomial Number of Moves. In: Wyrzykowski R, Dongarra J, Meyer N, Waśniewski J, editors. Parallel Processing and Applied Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg; 2006. p. 75–82.

31. Patterson S, Bamieh B. Leader selection for optimal network coherence. In: 49th IEEE Conference on Decision and Control (CDC). IEEE; 2010. p. 2692–2697.

32. Barabási AL, Albert R. Emergence of Scaling in Random Networks. Science. 1999; 286(5439):509–512. https://doi.org/10.1126/science.286.5439.509 PMID: 10521342

33. Cohen R, Havlin S. Scale-Free Networks Are Ultrasmall. Phys Rev Lett. 2003; 90:058701. https://doi.org/10.1103/PhysRevLett.90.058701 PMID: 12633404

34. Erdős P, Rényi A. On the evolution of random graphs. Publ Math Inst Hung Acad Sci. 1960; 5(1):17–60.

35. Chung F, Lu L. The Diameter of Sparse Random Graphs. Advances in Applied Mathematics. 2001; 26 (4):257–279. https://doi.org/10.1006/aama.2001.0720

36. Watts DJ, Strogatz SH. Collective dynamics of'small-world' networks. Nature. 1998; 393(6684):440–442. https://doi.org/10.1038/30918 PMID: 9623998

37. Penrose M. Random geometric graphs. vol. 5. Oxford University Press; 2003.

38. Ganesan G. Stretch and Diameter in Random Geometric Graphs. Algorithmica. 2018; 80(1):300–330. https://doi.org/10.1007/s00453-016-0253-5

39. Hamann H. Swarm Robotics: A Formal Approach. Cham: Springer International Publishing; 2018. Available from: http://link.springer.com/10.1007/978-3-319-74528-2.

40. Atzori L, Iera A, Morabito G. The Internet of Things: A survey. Computer Networks. 2010; 54(15):2787–2805. https://doi.org/10.1016/j.comnet.2010.05.010

41. Rawat P, Singh KD, Chaouchi H, Bonnin JM. Wireless sensor networks: a survey on recent developments and potential synergies. The Journal of Supercomputing. 2014; 68(1):1–48. https://doi.org/10.1007/s11227-013-1021-9

42. Marshall JAR, Kurvers RHJM, Krause J, Wolf M. Quorums enable optimal pooling of independent judgements in biological systems. eLife. 2019; 8(i):1–14. https://doi.org/10.7554/eLife.40368 PMID: 30758288

43. Bavelas A. Communication Patterns in Task-Oriented Groups. The Journal of the Acoustical Society of America. 1950; 22(6):725–730. https://doi.org/10.1121/1.1906679

44. Freeman LC. A Set of Measures of Centrality Based on Betweenness. Sociometry. 1977; 40(1):35–41. https://doi.org/10.2307/3033543