Node counting in wireless ad-hoc networks

Joep Evers[*] *Demeter Kiss*[†] *Wojtek Kowalczyk*[‡]
Tejaswi Navilarekallu[‡] *Michiel Renger*[*] *Lorenzo Sella*[§]
Vincent Timperio[§] *Adrian Viorel*[¶] *Sandra van Wijk*[*]
Albert-Jan Yzelman[||]

Abstract

We study wireless ad-hoc networks consisting of small microprocessors with limited memory, where the wireless communication between the processors can be highly unreliable. For this setting, we propose a number of algorithms to estimate the number of nodes in the network, and the number of direct neighbors of each node. The algorithms are simulated, allowing comparison of their performance.

KEYWORDS: wireless, unreliable network, graph, algorithm, simulation.

1 Introduction

Wireless networks are part of our everyday lives. The most commonly known wireless networks enable communication between computers or PDA's. Recently there has been an increased interest in wireless networks of smaller microprocessors with limited processing power. It is the possible large deployment of these small wireless devices, that makes them applicable to many diverse situations. By equipping nodes with sensors, networks of many such nodes can, for example, gather information on bird flocking [2], monitor social behavior and map crowd dynamics [7], and detect forest fires [12].

The technology that is currently being developed at CHESS [1] enables the nodes to communicate with each other by broadcasting short messages through the air. However, not all nodes will actually receive such broadcasts. For nodes that are too far apart, the signal may be too weak to establish a link. But even for nodes that are physically close, the wireless communication may be asymmetric or temporarily failing. Moreover, the topology of the network is inherently dynamic due to the possible physical movement of the nodes. These factors add up to highly unreliable networks.

[0] Corresponding author: Michiel Renger, Eindhoven University of Technology, P.O. Box 513, 5600 MB, The Netherlands, + 31 40 247 2685, d.r.m.renger@tue.nl

[*] Eindhoven University of Technology, The Netherlands

[†] CWI, The Netherlands

[‡] VU University Amsterdam, The Netherlands

[§] University of Leiden, The Netherlands

[¶] Universitatea Babes-Bolyai Cluj Napoca, Romania

[||] Utrecht University, The Netherlands

Previous experiments at CHESS have been successful in synchronizing the internal clocks of all nodes in a network with each other. To save the battery lifetime of each node, one would like to reduce as much as possible the time interval in which a node listens to incoming messages. However, a node can only receive a message from one other node at a time. Thus, a trade-off has to be made between saving battery power and receiving sufficient messages. To this aim, each node needs to have information about the size of the network it is in. The goal of this study is to design robust algorithms that retrieve this information, using minimal processing and storage memory.

The paper is organized as follows. In Section 2 we explain the problem in more detail, and state the main assumptions for this research. We describe a number of algorithms to estimate the neighborhood size in Section 3, and to estimate the network size in Section 4. Then, in Section 5 we discuss the simulation of the algorithms and the corresponding results. Finally we draw conclusions and give suggestions for further research in Section 6.

2 Problem Description

For the goal of the research presented in this paper, we propose a number of algorithms that estimate, for each node:

1. the number of direct neighbors (also called the degree), and
2. the total network size.

We will assume that the algorithms proposed in this paper are much faster than the changes that occur in network topology. This motivates our choice to consider only networks that have fixed topology, although some random noise in data exchange is allowed. Naturally, this assumption restricts the robustness of the algorithms: the longer they take, the more likely the topology will change in practice, yielding unreliable estimates.

As mentioned, each node has its own clock; the available technology allows synchronization of these clocks. However, synchronization only works between nodes within a (sub-) network. In practice, it may take some time until all nodes in a network are synchronized, especially if two separated sub-networks merge. Because of the different time scales assumption, the nodes in the networks under consideration are synchronized.

Typically, the clock of each node has a period of $1s$, which is divided into an *active period* (of the order of 1% of the total period) and an *inactive period*. During the inactive period, the node cannot receive any messages, which saves its battery life considerably. The active period is subdivided into a number of *slots* (typically multiples of 4 or 8). Within one time slot, a node can receive one message from one other node only; if messages from multiple nodes arrive at another node within the same slot, the messages will interfere, and the receiving node can no longer distinguish the signal from noise (in this case we speak of a *collision*). See Figure 1.

Each node has a unique identifier; one trivial way to estimate the neighborhood and network size would be to simply let each node broadcast its identifier together with all identifiers it has received so far. However, this method requires too much bandwidth

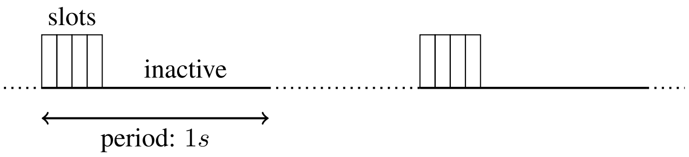

Figure 1: The periods of a node.

and local memory. Hence, we devise less demanding algorithms for estimating the neighborhood and network size.

3 Algorithms for neighborhood size estimation

In this section we study the neighborhood size (degree) of a given node. We present four estimation algorithms. The organization is the same in all cases: first we give an informal description, then we provide pseudocode for the algorithm.

Before presenting the algorithms, we should first note that the links between nodes are possibly directional. Since a node can not directly determine how many nodes receive its message (the *outgoing degree*), we focus on the neighbors from which a node can receive a message (the *incoming degree*). Sometimes we will refer to the incoming degree as the number of potential neighbors, to stress that we include neighbors whose message may not be received due to noise or collisions.

Secondly, we note that, given the number of messages received by a single node in each of the last X time periods, a simple lower and upper bound on the number of unique neighbors d can be constructed. Denote the number of received messages by $m_1, m_2, \ldots, m_X$. As a lower bound d_L on the neighborhood size, we take the maximum of received messages in one period:

$$d_L = \max_{1 \leq j \leq X} m_j.$$

The messages in one period are certainly coming from unique neighbors. Hence, the maximum is a lower bound: in the best case scenario, all messages originate from a neighbor already included in the lower bound. As an upper bound d_U, we take the sum of all messages received. In the best case scenario for d_U, all messages originate from different neighbors. Thus:

$$\max_{1 \leq j \leq X} m_j = d_L \leq d \leq d_U = \sum_{j=1}^{X} m_j.$$

Note that these bounds only incorporate the neighbors from which a message was actually received during at least one of the last X periods. It might be possible that it has more neighbors from which no messages have come through the last X periods.

3.1 Random ID: neighborhood size estimation

Here we describe an algorithm similar to the basic algorithm given at the end of the Section 2. The main difference is that we use randomized identifiers in stead of the built-in ones as follows.

Let L be the range of the random IDs, a parameter of the algorithm. Each node v picks a uniform random number from $\{1, 2, \dots, L\}$, which we refer to as random identifiers and denote by RID_v. Also each node v uses R_v a bit vector of length L, set to 0 at start.

Each node in each round sends a signal containing its random ID. When a node v receives a signal from node w, then set the RID_wth bit of R_v to 1. The vector R_v stabilizes with a list of the random IDs of the neighbors of v. Then we use the estimator in (2) explained below. The pseudocode is given in Algorithm 1.

After R_v is stabilized, the probability that the first bit of R_v is equal to 0 is given by

$$P(R_v(1) = 0) = \left(1 - \frac{1}{L}\right)^d. \tag{1}$$

Replacing $P(R_v(1) = 0)$ in Eq. (1) with

$$V_v = \frac{\#\{i \mid R_v(i) = 0\}}{L},$$

the proportion of the bits with value 0 in R_v provides the estimate

$$\hat{d}_v = -\frac{\log V_v}{\log\left(1 - \frac{1}{L}\right)} \approx -L \log(V_v). \tag{2}$$

To conclude, we discuss the choice of the parameter L. Using Taylor expansion, Whang et al. [15] show that, the bias and standard error of $\frac{\hat{d}_v}{d}$ satisfies

$$E\left(\frac{\hat{d}_v}{d_v} - 1\right) / \frac{e^t - t - 1}{2d_v} \to 1, \tag{3}$$

$$StdError\left(\frac{\hat{d}_v}{d_v}\right) / \frac{\sqrt{L\left(e^t - t - 1\right)}}{d_v} \to 1 \tag{4}$$

as $d_v, L \to \infty$ with $\frac{d_v}{L} \to t$.

In [15], only the case $L = \mathcal{O}(d_v)$ was considered. Hence, they keep open the possibility to use the estimator $\hat{d}_v$ for smaller values of L. However, a simple computation based on the expected number of 0 bits of the stabilized R_v shows that we have to choose $L > c\frac{d_v}{\log d_v}$ for some constant c to be able use the estimator $\hat{d}_v$. Hence we have to use at least $\mathcal{O}\left(\frac{d_v}{\log d_v}\right)$ of memory to get a non-trivial estimate with this algorithm. This is not a significant improvement on the algorithm described at the end of Section 2 which uses $\mathcal{O}(d_v)$ memory. Nevertheless, we can use equations (3) and (4) to tune the parameter L for the desired accuracy, in the same way as in [15]. This results in a constant multiple (depending on the required accuracy) of memory decrease compared to the basic algorithm of Section 2.

3.2 Parameter estimation of binomial distribution

A given node has d potential neighbors, i.e. other nodes from which it might receive messages. Assume that with probability $p \in [0, 1]$ such a message from a potential

Algorithm 1 Random ID: neighborhood size estimation.

Parameters:
Range for random IDs: L

```
Initialise:
Set $R$ to an array of bits with length $L$ of 0-s.
Set $RID$ to $rand(1, L)$.
$R(RID) = 1$
Each round:
$bsp\_bradcast(RID)$
for $i = 1$ to $bsp\_nlsots$ do
    $r = bsp\_get(i)$ where $r$ is the sent random ID.
    Set $R(r) = 1$.
Compute $V_v$ and $-L\log(V_v)$ – that is the estimate
```

neighbor is indeed received in a period, independently and identically distributed for all messages, all potential neighbors, and all periods. Then, the number of messages that this nodes receives during a period, denoted by the random variable M, follows a binomial distribution with parameters d and p.

We are given the number of messages received by the node during the last X periods. Let these numbers be given by the vector $m = (m_1, \ldots, m_X)$, where each $m_j, j = 1, \ldots, X$, is a realization of M. From this information, we want to estimate d, as this gives an estimate for the number of unique neighbors the node has seen during the last X periods.

So, we reduce the problem to estimating the parameters d of a binomial distribution, when X observations are given and p is unknown. An option would be to use the maximum likelihood estimator (MLE), or use the method of moments estimator (MME), however, both are known to be unstable [3]. That is, if instead of some m_j one observed $m_j + 1$, this can result in a relatively large change in the estimator for d. Therefore, we chose a more stable estimator, more precisely, the one given in DasGupta and Rubin [5]. It uses three variables that are derived from the vector m, namely the mean, denoted by $\overline{m}$, the maximum, m_{max}, and the sample variance:

$$S_m^2 = \Big(\sum_{j=1}^{X} (m_j - \overline{m})^2\Big) / (X - 1).$$

Then, the estimator for d, $\hat{d}$, is given by (cf. [5, Eq. (8)])

$$\hat{d} = \frac{m_{max}^2 \, S_m^2}{\overline{m} \, (m_{max} - \overline{m})}. \tag{5}$$

The proposed estimator of [5] actually involves an extra parameter α, which can be used for fine tuning the estimator. We set $\alpha = 1$ here, which is claimed to be a good generic choice [5, p. 397]. A more complex estimator is proposed [5, Eq. (11)] as well, which uses a summation over inverse Beta functions. We, however, judge this

Algorithm 2 Parameter estimation of binomial distribution.

Parameters:
Number of previous periods taken into account: X

```
Initialise:
1: Let $m = (m_1, \dots, m_X)$ be the vector of received messages in each period,
   initialize $m$ to be zero-vector of length $X$.
Each round:
1: Count the number of received messages in round: $m_0$.
2: Update $m$ to be $m = (m_0, \dots, m_{X-1})$.
3: Calculate sample mean $\overline{m}$, sample variance $S_m^2$, and maximum $m_{max}$ based on
   this $m$.
4: Compute $\hat{d} = \frac{m_{max}^2 S_m^2}{\overline{m}(m_{max} - \overline{m})}$ (Equation (5)) - this is the estimate for number of
   unique neighbors.
```

one to be too complicated and time consuming to be executed on each node in every period.

Now, (5) gives an estimate for the number of (unique) neighbors of the node under consideration seen during the last X periods. In the next period, a number of messages is received by this node, say m_0, and the vector m is updated accordingly: m_X is discarded and m_0 is prepended to m. All steps of this estimator are given in Algorithm 2.

3.3 Future broadcast

The key idea is to broadcast a message consisting of a set of random numbers which are used by the receivers to avoid repetitions in counting the number of their neighbors.

We denote by k the size in bits of the random numbers that used generated by each node, by l the size of the set of random numbers that a node broadcasts in a single message and by X the number of active periods during which one wants to estimate the number of their neighbors. In any application, there will be an intrinsic relation between these variables. In particular, l should equal the average number of broadcasts by a node in X active periods.

A broadcast message from a node consists of a list of size l, whose elements are k-bit random numbers. Once the message is broadcast, the node removes the leading random number from the list and appends a newly generated random number to the end of the list. This new list is then used in its next broadcast.

On the receiving end, a node maintains a list of *recently* seen random numbers and a list of counters whose elements keep the count of number of neighbors in the *recent* active periods. In an active period, each received broadcast message is discarded if the leading random number in the message is an element in the list of random numbers. If not, the list of random numbers in the message is appended to that of seen random numbers. A new counter is added to the second list which keeps track of the number of new neighbors seen during the current active period. At the end of the active period, both the lists are updated by removing the *old* random numbers and the *old* counter of

Algorithm 3 Future broadcast.

```
Parameters:
Length of random numbers: $k$
Number of active periods to be considered: $X$
Average number of broadcasts by a node in $X$ active periods: $l$

Initialise:
  Set $M$ to an array of $l$ random numbers between 1 and $2^k - 1$.
  Set $RR$ to an array of length $X$ of arrays.
  Set $C$ to an array of length $X$.
Broadcast:
  $broadcast(M)$
  Remove the last element from the list $M$.
  Insert $(rand(1, 2^k - 1)$ at the beginngin of the list $M$.
Each active period:
  Set $c = 0$.
  Set $temp = [\,]$.
  for each received message do
    if $M[0] \notin RR$ then
      $append(M, temp)$.
      $c = c + 1$.
    Remove the last element from the list $RR$.
    Remove the last element from the list $C$.
    Insert $temp$ at the beginning of the list $RR$.
    Insert $c$ at the beginning of the list $C$.
Count the number of neighbors:
  Return($sum(C)$).
```

number of neighbors.

The broadcast message simply consists of a list M of l k-bit random numbers. Every node maintains two lists of length X each: a list RR of lists of random numbers seen during each of the last X active periods and a list C of number of neighbors seen during each of the last X active periods. In an active period, both these lists are appended at the end with the newly seen random numbers and the number of newly seen neighbors respectively and the leading elements are deleted.

The sum of all the elements from the list C gives an estimate of the number of unique neighbors seen during the last X active periods. All steps are given in Algorithm 3.

3.4 Counting neighbors of neighbors

We now focus on the average degree of a node in the network, using this average as a (rough) estimate for the number of neighbors. For this, we want a node to count its neighbors, the neighbors of its neighbors, the neighbors of the neighbors of its neighbors, and so on. Each node has to broadcast a small list containing these numbers. It also receives these lists from its neighbors, from which it updates its own list. Note that there is inevitable double-counting of nodes in this approach, but that is taken into account in this method, and hence it is not a problem.

Let $n_i^{(1)}$ be an estimate for the neighborhood size of node i, e.g. as in (5) or just the number of messages received during the last period. These are the 'first degree' neighbors. A message received from neighbor j includes node j in its estimate of its neighborhood size $n_j^{(1)}$. The number of neighbors of neighbors for node i is now simply the summation over these numbers, resulting in an estimate for its 'second degree' neighbors (all nodes at most two connections away): $n_i^{(2)} = \sum_{j \in \mathcal{N}_i} n_j^{(1)}$, where $\mathcal{N}_i$ is the set of all neighbors of node i in the last period. As these numbers are broadcast as well, node i gets the following estimate for its 'third degree' neighbors: $n_i^{(3)} = \sum_{j \in \mathcal{N}_i} n_j^{(2)}$. Theoretically, we can continue this to every desired degree:

$$n_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} n_j^{(k)},$$

where $k = 1, \dots, K$ for some arbitrary K. However, the information to be broadcast by every node increases in the length of the list $(n_i^{(1)}, n_i^{(2)}, \dots, n_i^{(K)})$. Therefore, we propose to send only the list $(n_i^{(1)}, n_i^{(2)}, n_i^{(3)})$, i.e. $K = 3$, that consists of three numbers, so a node can compute up to its fourth degree neighbors $n_i^{(4)}$. In a graph that satisfies a power law for the degrees at its nodes, typically all nodes are connected within six connections. Also, an Erdős-Renyi random graph typically has a small diameter. Hence, we argue that knowledge up to the fourth degree neighbors is a good balance between knowledge of the network and the amount of data that has to be broadcast.

We use these counts to come up with an estimate for the average number of neighbors (degree) of a node in the network. We denote this average by $\bar{d}$. Hence, a node has on average $\bar{d}$ neighbors, so $n_i^{(1)} \approx \bar{d}$. These $\bar{d}$ neighbors have each $\bar{d}$ neighbors ($n_i^{(2)} \approx \bar{d} \cdot \bar{d}$), and each of them have again $\bar{d}$ neighbors ($n_i^{(3)} \approx \bar{d}^3$), and so on. From this a list of estimates for $\bar{d}$ follows:

$$\left\{ \left(n_i^{(k)} \right)^{(1/k)} \right\}_{k=1}^{K+1}.$$

We expect the entries of this list to converge. We use the last element from this list in the estimator for $\bar{d}$, the average degree of a node. So, the estimator for $\bar{d}$, say $\hat{d}$, is given by:

$$\hat{d} = \left(n_i^{(K+1)} \right)^{1/(K+1)},$$

where we have chosen $K = 3$ here. We use $\hat{d}$ as an estimate for the number of neighbors of the considered node. All steps of the estimator are given in Algorithm 4. Moreover, as actually the average number of neighbors of an arbitrary node is estimated, we use it as well in Algorithm 8 of Section 4.4 for estimation of the network size.

Algorithm 4 Counting neighbors of neighbors

Parameters:
Maximum degree of neighbors taken into account: K

```
Initialise:
  1: Let $n^{(k)}$ be the number of $k$th degree neighbors, initialize $n^{(k)} = 0$ for
     $k = 1, \dots, K$.
Each round:
  1: Broadcast $(n^{(1)}, \dots, n^{(K)})$.
  2: Set $n^{(1)}$ to estimate number of unique neighbors.
  3: From the received vectors $(n_j^{(1)}, \dots, n_j^{(K)})$, calculate and update
     $n^{(1)}, n^{(2)}, \dots, n^{(K+1)}$ using $n^{(k+1)} = \sum_j n^{(k)}$, for $k = 1, \dots, K$.
  4: Compute $\hat{d} = \left(n^{(K+1)}\right)^{1/(K+1)}$ - this is the estimate for the average degree of
     an arbitrary node in the network.
  5: Let $\hat{d}$ also be the estimate for number of unique neighbors.
```

4 Algorithms for network size estimation

In this section we present a number of algorithms for the estimation of the network size, i.e. the total number of nodes in the network. In each case, we first give an informal description of the algorithm, followed by a pseudocode. Additional to the algorithms presented in this section, we have included in the Appendix one algorithm that needs more investigation of its possible implementation and performance.

4.1 Epidemic algorithm

The idea of the method is to start a diffusion-like process such that after enough time an initially concentrated (in one node) quantity will be uniformly distributed over all nodes. If the initial quantity is exactly 1, then the amount present at each node when equilibrium is reached should be $1/N$, where N is the size of the network.

This approach is not new and we follow ideas of Jelasity and Montresor [9] who apply the anti-entropy epidemic protocol introduced by Demers et al. in their seminal work [6] to the aggregation problem. Aggregation is defined as a ‘collective name of many functions that provide global information about the system’ [9].

Jelasity and Montresor prove that using this method in a fully connected network with reliable bidirectional links, all nodes estimate N continuously and adaptively: the estimates converge exponentially fast to the exact size of the network. Furthermore, the convergence rates and memory requirements are independent of N and the same for all nodes.

More precisely, let $V(j)$ denote the amount of the aforementioned quantity, present in node j. In a perfect network the algorithm is conservative in the sense that

$$\sum_{j=1}^{N} V(j) = 1$$

at all times, and has the convergence property

$$V(j) \to \frac{1}{N}, \text{ for all } j = 1, \ldots, N.$$

The question is if this algorithm can still be effective in an unreliable network. Because of asymmetric links undetectable information loss is possible as a node can not establish which other node has received its broadcasts. The effectiveness and accuracy of the algorithm in directional, unreliable networks will depend on the network under consideration, and can only be studied by extensive simulations.

Nevertheless, heuristically we can argue the following. If we restrict our analysis to one node, there are two ways in which its communications are affected due to the unreliability of the network:

1. some of the potentially incoming messages are lost which leads to an underestimation of the neighborhood size

2. some of the messages broadcasted by the node are also lost.

Thus, the presented algorithm may have a natural tendency to correct errors in estimating the size of the network due to 1 and 2, if the incoming error and the outgoing error would cancel themselves out.

The pseudocode of the algorithm is given in Algorithm 5.

4.2 Random ID: network size estimation

The set-up is the same as Section 3.1. The main difference from the algorithm in Section 3.1 is that now each node will not only broadcast its random ID, but also the list of random IDs which it already encountered. An additional difficulty arises, since usually the recurrence vector does not fit in the message, since the network size is usually much bigger than the length of the message which can be broadcasted in one round. Hence we divide the recurrence vector of pieces of l bits, and send the pieces separately, in random order. The algorithm is the following:

Let L be the range of the random IDs, and l is the length of the broadcasted piece, parameters of the algorithm. Then each node v picks a uniform random number from $\{1, 2, \ldots, L\}$, which we refer to as random identifiers and denote by RID_v. Also each node v uses R_v, a bit vector of length L, set to 0 at start.

Each node in each round picks a uniform random number k from $\{1, 2, \ldots, \lfloor L/l \rfloor\}$. It sends a signal containing RID_v, k and the kth part of its recurrence vector, which we denote by $R_{v,k}$. When the node v receives a message from w, then it sets the RID_wth bit of R_v to 1, takes $k = k_w$ and updates $R_{v,k}$ to $R_{v,k} \vee R_{w,k}$, where $\vee$ denotes the coordinate-wise maximum. Then we use the estimator of EQ. (2). The pseudocode is given in Algorithm 6.

Note that the vector R_v stabilizes with the list of random IDs in the network for all nodes v. Using the arguments of Section 3.1 we get results similar to those in Section 3.1 for the memory usage of the algorithm.

Algorithm 5 Epidemic algorithm.

Parameters:
None

```
Initialise:
  Assign to all nodes except one node called $i$ the value zero. To $i$ assign the value one:
      $V(j) := 0,\ \text{for } j \neq i;$
      $V(i) := 1.$
Each round:
  Each node $j$ estimates its outcoming degree by counting the number of incoming messages
  Each node sends a message containing
      $m(j) = \frac{V(j)}{d_{in}(j) + 1}.$
  Each node updates its value
      $V(j) := \frac{V(j)}{d_{in}(j) + 1} + \sum_{k \text{ incoming msg}} m(k).$
```

Algorithm 6 Random ID: network size estimation.

Parameters:
Range for random IDs: L
Length of the pieces recurrence vector: l

```
Initialise:
  Set $R_v$ to an array of bits with length $L$ of 0-s.
  Set $RID_v$ to $rand(1, L)$.
  $R_v(RID_v) = 1$
Each round:
  Set $k = rand(1, \lceil L/l \rceil)$.
  $bsp\_bradcast(RID_v, k, R_{v,k})$
  for $i = 1$ to $bsp\_nlsots$ do
      $g_1 g_2 g_3 = bsp\_get(i)$ where $g_1$ is the sent random ID, $g_2$ position, $g_3$ is the part of the recurrence vector
      Set $R_v(g_1) = 1$.
      Set $R_{v,g_2} = \max(g_3, R_{v,g_2})$.
  Compute $V_v$ and $-L \log(V_v)$ – that is the estimate
```

4.3 The MinTopK algorithm

Let us assume, as in the previous section, that each node keeps its private random identifier, RID_v, that is drawn uniformly from a set of identifiers that is much bigger than the network size N, so it is safe to assume that all identifiers are different. For the sake of simplicity of our presentation we will view these identifiers to be a uniform sample of size N, drawn from the interval [0,1]. The key idea behind our algorithm is based on an observation that the nodes can collectively create a list (shared by all the nodes) of K biggest values, where K is relatively small, for example $K = 100$, depending on the size of the available memory. The smallest element of this list, call it s, can be used to estimate the network size. Indeed, if the interval $[s, 1]$ of length $1 - s$ contains K values, then the whole interval $[0, 1]$ is expected to contain about $K/(1-s)$ values (they are uniformly distributed), so the network size can be estimated by $K/(1 - s)$.

In the rest of this section we work out some details of our method. First, we present an algorithm that each node has to execute in order to create the list of top K values that are known to network nodes. Second, we provide a slightly better formula for estimating the network size from the smallest value of this list and experimentally evaluate the accuracy of our approach as a function of K and N. Finally, we make some recommendations for further research.

To create a list of the top K values we propose the following algorithm. It is well known that calculating the maximum of values that are distributed along the nodes of the network is a relatively simple task: the simplest gossip-based algorithm which asks network nodes to propagate only the biggest value they have ever seen, converges, under some assumptions, exponentially fast to a configuration, where each node knows the maximal value. This style of propagating information through the nodes was successfully generalized to compute other aggregates, such as average, variance, count, see [11] or [10]. Our algorithm finds a list of k biggest values that are distributed along the nodes using the same principle. During the initialization, each node creates a local list of K values that are initially set to 0. The only value that is known to a node - its own identifier - is stored in the list. Then the nodes start to exchange information: each node broadcasts a randomly selected non-zero value from its local list and, whenever a new value is received, it is inserted into the list as long as there is a place for it - the smallest element of the list can be deleted to make a space for the new value, as long as this new value is bigger than the smallest one. Clearly, the network will eventually converge to a state, where each node keeps a list of the top K values. The pseudocode of the final procedure for estimating the network size is presented in Algorithm 7.

To estimate the network size from MinTopK we proceed as follows. As mentioned earlier, if s denotes the smallest of the top K values then $K/(1 - s)$ is a good estimate of the network size. However, it may happen that the network size is smaller than K. In such a situation the constructed list of biggest values will contain exactly N non-zero elements, so the exact network size can be found by counting these non-zero values. However, it is clear that in case $K < N$, the estimate of the network size will be only an approximation of the true node count. Instead of quantifying the quality of this approximation in an analytical way, we have estimated this error by generating 100.000 uniform samples of size N and applying our formula to the minimum of the top K elements of each sample. The results are summarized in Table 1.

Obviously, the errors reported in the table provide only a lower bound on the actual errors made by the MinTopK algorithm. In practice, depending on the network topology, reliability of connections and the number of send messages, the actual errors might be bigger: the lists that are maintained by the nodes need some time to converge to correct values. One way of estimating these errors is by running extensive simulations. Some initial results of such simulations are described in Section 5.

	K=8	**K=16**	**K=32**	**K=64**	**K=128**	**K=256**
N=64	31.07	18.76	10.65	0.00	0.00	0.00
N=128	32.37	20.34	12.77	7.25	0.00	0.00
N=256	33.06	20.69	13.72	8.89	5.02	0.00
N=512	33.07	21.27	14.34	9.70	6.22	3.58
N=1024	33.45	21.37	14.37	9.69	6.64	4.38
N=2048	33.12	21.57	14.51	9.97	6.94	4.67
N=4096	32.74	21.34	14.75	10.04	7.06	4.84

Table 1: The average relative absolute error (in percents) of the MinTopK estimator for various values of N and K. Each estimate is based on the results of 100.000 runs. The standard errors, all smaller than 0.4%, are not shown.

There are several research problems that could be further investigated. Firstly, we have considered only the most elementary version of the gossiping algorithm. It is obvious that this algorithm can be improved in many ways. For example, local lists could be sorted and instead of propagating only values, nodes could propagate values together with their ranks. This information could be used to speed-up the convergence, by intelligently processing the incoming information. Also, instead of selecting randomly values to be broadcasted, nodes could put more attention to broadcasting values that just entered their local lists: these seem to be most informative.

Secondly, performance of the presented algorithm should be tested on various topologies and other properties of networks. Let us note that our network might be organized in such diverse structures as a 1-dimensional grid and a fully connected graph. Clearly, the convergence speed will strongly depend on network topology.

Thirdly, it would be interesting to modify our algorithm to handle situations when nodes join or leave the network or the network topology is dynamically changing over time.

4.4 Power law

The inspiration for this algorithm was provided by a paper by R. Cohen et al. [4], in which the internet is modeled as a random graph. A network is considered in which the nodes are connected randomly to each other. The probability that a node is connected to some other node depends on the connectivity of the two. That is, the number of edges connected to the node. Connectivity is also called degree. In the paper a particular type of such random graphs is analyzed, namely those graphs in which the connectivity is distributed according to a *power law*.

More specifically, the probability that a certain node is connected to exactly k other nodes is given by:

$$P(k) = ck^{-\tau}, \qquad k = m, m+1, \ldots, \tag{6}$$

Algorithm 7 The MinTopK algorithm.

```
Parameters:
Length of the list of values: $K$

Initialise:
create a list of $K$ entries, each storing 0
put on the top of the list your own value
set the current estimate of network size to 1

Each round:

Broadcast:
choose at random any non-zero element from the list and broadcast it
Receive:
for each received identifier $x$ do
    find the smallest element on the list, $s$
    if $s > x$ then
        do nothing
    else
        replace $s$ by $x$
Estimate network size:
find the smallest element on the list, $s$
if $s = 0$ then
    set the current estimate of network size to the number of non-zero elements on
    the list
else
    set the current estimate of network size to $K/(1 - s)$
```

where typically $2 < \tau < 3$ holds, and c is a normalization constant. m denotes the smallest possible degree (in many cases $m = 1$ will hold). For simplicity, integration is used instead of summation, such that c can be derived from the condition:

$$\int_m^\infty ck^{-\tau} dk = 1. \tag{7}$$

It follows that:

$$c = (\tau - 1)m^{\tau-1}. \tag{8}$$

A finite graph, consisting of N nodes, is considered, where the degree of each node is a random sample from the proposed power law. To estimate the maximal degree M in this graph, the assumption is made that only one node in the whole network is expected to have degree M. We denote by:

$$p := \int_M^\infty P(k)dk, \tag{9}$$

the probability that a particular node has degree greater than or equal to M. Now, the expected value of the number of nodes with degree greater than or equal to M is:

$$\begin{aligned} \mathbb{E}[\#\text{ nodes with degree } \geqslant M] &= \sum_{k=1}^{N} k \begin{pmatrix} N \\ k \end{pmatrix} p^k (1-p)^{N-k} \\ &= Np \sum_{j=0}^{N-1} \begin{pmatrix} N-1 \\ j \end{pmatrix} p^j (1-p)^{(N-1)-j} \\ &= Np. \end{aligned} \tag{10}$$

Setting this expected value 1, we obtain:

$$\int_M^\infty P(k)dk = \frac{1}{N}. \tag{11}$$

From the above one easily derives:

$$\left(\frac{m}{M}\right)^{\tau-1} = \frac{1}{N}, \tag{12}$$

and thus the network size can be deduced from the minimal and maximal degree, together with the parameter τ:

$$N = \left(\frac{M}{m}\right)^{\tau-1}. \tag{13}$$

We would like to apply the ideas of [4] to our network of broadcasting nodes. We assume that the connectivity of our nodes can also be modeled by a power law distribution. This implies that an estimate of the network size N can be obtained, given that we have (good) estimates for m, M, and the parameter τ. These estimates are to be derived from the information a node receives from its surroundings. In particular we want to derive the value of τ from the average number of neighbors of a node in

Algorithm 8 Power law.

```
Initialise:
    Perform the initializations of Algorithms 2 and 4.
Each round:
    Receive messages from neighbors $j$ from index set $\mathcal{N}$; extract $d^{(j)}_{\text{min}}$ and $d^{(j)}_{\text{max}}$.
    Perform Algorithms 2 and 4; extract $\hat{n}$ and $\bar{d}$.
    Set $m = M = \hat{n}$.
    for $j \in \mathcal{N}$ do
        $m = \min(m, d^{(j)}_{\text{min}})$
        $M = \max(M, d^{(j)}_{\text{max}})$
    Set $d_{\text{min}} = m$ and $d_{\text{max}} = M$.
    Calculate $\tau = \frac{d_{\text{min}} - 2\bar{d}}{d_{\text{min}} - \bar{d}}$.
    Calculate the estimate for the network size: $N = \left(\frac{d_{\text{max}}}{d_{\text{min}}}\right)^{\tau-1}$.
    Broadcast message containing information desired in Algorithms 2 and 4, and
        $d_{\text{min}}$ and $d_{\text{max}}$.
```

the network. First of all, note that the expected degree d of a particular node is given by (again using integration instead of summation):

$$\mathbb{E}[d] = \int_m^\infty kck^{-\tau} dk = \frac{\tau - 1}{\tau - 2} m, \tag{14}$$

provided that $\tau > 2$.
For each node we need an estimate of the average neighborhood size ($\bar{d}$, the average taken over all nodes), which can be obtained by the method described in Section 3.4. From $\bar{d}$ we derive an estimate for the parameter τ, by assuming $\bar{d} = \mathbb{E}[d]$:

$$\tau = \frac{m - 2\bar{d}}{m - \bar{d}}. \tag{15}$$

For estimating m and M the own neighborhood size of a node ($\hat{n}$) is needed. Again, this quantity can be obtained from the algorithm described in Section 3.4. In order to get a clue of what the minimum and maximum degree in the network are, we compare $\hat{n}$ to estimates of m and M which we receive from our neighbors. We make this more precise in the pseudocode for the algorithm, see Algorithm 8.

Note that this algorithm is based on the ideas presented in [4], but mathematical proofs for the applicability to our problem are lacking. For the moment we rely on simulations, as an indication of whether we are on the right track.

In case of a dynamic system, after some time, reset the system. This might be desirable, since the algorithm in its current form is incapable of capturing splitting-up of the network.

4.5 Random tours using convergence detection

This algorithm is based on a paper by L. Massoulié et al. [13], in which an estimate of the number of nodes in a peer-to-peer network is obtained using random walks. According to this algorithm, we start at a selected node, gather information on the estimated degree, randomly select a peer, and send the collected information there; the receiving node then repeats this process. By selecting random neighbors, we naturally construct a random walk, and this walk eventually returns at the initiating node, turning the random walk into a loop: this is also called a *random tour*. The information gathered at each node i in the tour is its estimate of the local degree d_i; the factor $1/d_i$ was added to the prior information on the degree and got sent to the next stop of the tour.

Upon completion of the tour, this data is multiplied by the current estimate of the degree which directly gives an estimate of the number of nodes in the entire network. The proof of convergence relies on modeling the random walk process as a Markov chain process, and specifically it should be a reversible chain with a unique stationary distribution. More global assumptions are that the graph is connected; but this is not restrictive: if not entirely connected, it estimates the size of the connected component of the initiating node, thus corresponding with our definition of a stationary network. Another assumption made is that individual neighbors can be selected and communicated to, and that the connectivity of the network is stable (messages over edges arrive with probability 1). Both of the last two assumptions surely do not apply to our situation, and need to be attended to.

We now describe how this algorithm is modified to apply to broadcasting systems. Instead of creating random walks, we broadcast a message starting at the initiator, and assign this message a randomized identification number (ID) and a path length (initially 0). Nodes receiving this message will store this message ID together with the current path length and re-broadcast this message. When a node receives a message of which the ID is already known, and the path length is large enough[1], two random broadcasts have converged and can be considered to have followed a specific path; the concatenation of these two paths then form a tour, and an estimate of the network size can thus be calculated (the degree information and tour lengths can be added to each other). Due to the network unreliability, however, there is no theoretical guarantee that the obtained estimate can be expected to be close to the network size when two tours meet. As such, initial simulations should be made to determine whether this method works as is or requires further adaptation. Algorithm 9 details this algorithm in pseudocode.

4.6 Message based counting

The main problem with broadcasting all node IDs in a network and keeping a list of all received IDs, is the memory usage involved with such a scheme. The idea described here attempts to save memory usage, while in principle still being able to obtain an exact network count (again assuming a static network topology). This is done by ID compression by grouping nodes in unique network *trails*, where a trail visits nodes in a network by traversing on arbitrary edges using each edge at most once. Note that

[1] this is a parameter to be experimented with

Algorithm 9 Random tours using convergence detection.

Parameters:
Minimum message ID length m_{ID} bits,
minimum tour length m_l,
a probability p_{init} of initiating a counting broadcast.

```
Main algorithm:
Set $d$ to the estimated neighborhood size
**if** If *rand*$() < p_{\text{init}}$ **then**
    Set $r_1 \dots r_{m_{\text{ID}}}$ to a random binary number
    Set $r_{m_{\text{ID}}+1} \dots r_{m_{\text{ID}}+1+\log_2 m_l}$ to the decimal number $m_l$
    Set the remainder of bits (up to $r_{27 \cdot 8}$) to $1/d$
    *bsp_broadcast*$(r_1 \dots r_{27 \cdot 8})$
Set $\mathcal{M}$ to $\emptyset$
Set $\mathcal{C}$ to $\emptyset$
Allocate an array $x = x_1 \dots x_{\textit{bsp\_nslots}()}$
Allocate an array $l = l_1 \dots l_{\textit{bsp\_nslots}()}$
**for** $l = 1$ to *bsp_nslots*$()$ **do**
    $g = g_1 \dots g_{27 \cdot 8}$ =*bsp_get*$(l)$
    **if** $g$ is a valid message **then**
        Set $i = g_1 \dots g_{m_{\text{ID}}}$
        Extract path length $l$ from $g_{m_{\text{ID}}+1} \dots g_{m_{\text{ID}}+1+\log_2 m_l}$
        Extract a floating point number $f$ from $g_{m+1} \dots g_{27 \cdot 8}$
        **if** $i \in \mathcal{M}$ **then**
            **if** $l = 0$ **then**
                Convergence: obtain a network size estimate of $d(x_i + f)$
                Store $i$ into $\mathcal{C}$
            **else**
                **if** $l < l_i$ **then**
                    Store $l$ into $l_i$
                    Store $f$ into $x_i$
        **else**
            Store $i$ in $\mathcal{M}$
            Store $f$ into $x_i$
            Store $l$ into $l_i$
Select a random $i$ in $\mathcal{M}$ s.t. $i \notin \mathcal{C}$
Construct a message $g$ with the first bits containing $i$, followed by $l - 1$ in $\log_2 m_l$ bits, and with the remaining bits set to $x_i + 1/d$
*bsp_broadcast*$(g)$
```

Algorithm 10 Message based counting.

```
Initialise:
Set $\mathcal{M}$ to $\emptyset$
Set $s = 0$

Main algorithm:
if *bsp_pid*() is an algorithm initiator then
    *bsp_broadcast*$((0, \{\text{bsp\_pid()}\}))$
    Set $s = 1$
for $l = 1$ to *bsp_nslots*() do
    if $g = (o, T)$ is a valid message then
        if $s = 0$ then
            if $|T| = M$ then
                *bsp_broadcast*$(o + 1, \{\text{bsp\_pid()}\})$
            else
                *bsp_broadcast*$(o, T \cup \{\text{bsp\_pid()}\})$
            Set $s = 1$
            Add the sent message to $\mathcal{M}$
        else
            if $g \notin \mathcal{M}$ then
                Add $g$ to $\mathcal{M}$
    if no message is sent yet then
        Select $b$ from $\mathcal{M}$ randomly
        *bsp_broadcast*$(b)$
```

nodes can thus be visited more than once; a trail thus differs from a path in this sense.

The algorithm works by sending messages of the following type: (o, T) with $0 \leq o < M - 1$ an integer and T a collection of at most $M - 1$ IDs; these messages thus have a size of M integers. Each node furthermore can store received messages in a collection $\mathcal{M}$, and has a local boolean s keeping track whether it has modified an incoming message. We assume the existence of an initiator, which starts with broadcasting a message $(0, \{bsp\_pid()\})$, storing this message in its local $\mathcal{M}$, and settings its s to `true`. Other nodes start with $\mathcal{M} = \emptyset$ and s set to `false`.

Each node, when a message is received and s is `false`, adds its ID to T, sets s to `true`, and re-broadcasts. It can happen that T becomes too large after addition of the local ID; in this case, the other, older, IDs are first removed from T, and o is incremented by one. The message variable o thus counts the number of *overflows*; while the message ID, now set to the node at which it overflows, remains unique; this is by virtue of the local s variable which prevents adding a node to a message more than once. Thus the message size is compressed: the re-broadcasted message still contains enough information to obtain the number of nodes its trail visited.

If a message is received but s is set to `true` already, if it is not already in $\mathcal{M}$, it is added to $\mathcal{M}$; otherwise it is not added. In all cases, a random entry of $\mathcal{M}$ is re-broadcasted so that in time, nodes in the network may converge to having the same set of trails. Algorithm 10 shows an implementation of this scheme.

5 Simulation and results

In this section we first describe the graphs for which we simulated the algorithms, then we describe how we have simulated the algorithms, and finally present the results of the simulations.

5.1 Random graphs

The graphs that we use in the simulation are generated such as to mimic the conditions of sensor node networks in practice. They depend on three parameters: the given network size N, the (expected) average degree $\bar{d}$, and an additional probability α (which we fix as 0.8) that a directed edge will be created. Although only the topological properties of the network are relevant, these are often related to geometric properties. Therefore, we first choose random positions for each node within a two-dimensional box of size 1. Next, we calculate the radius R that models the distance that messages may cover, such that the expected degree will be as given:

$$\bar{d} = \alpha(N-1)\pi R^2.$$

Between each pair of nodes that are within radius R of each other, a directed edge then is created with probability α. Moreover, we add a small number of completely random directed links between any two nodes with probability $N/\bar{d}$. These are added because experiments at CHESS have occasionally shown some unexpected links between nodes that were geographically far apart. Finally, we add to each directed edge a (uniformly) random number in $[0.4, 0.8]$ that models a probability that will be received.

We remark that the expected average degree will deviate slightly from the requested degree because of the latter random added links, and because we do not take into account that nodes may be close to the border of the box.

5.2 Simulation

A natural way to simulate the nodes is by a Bulk Synchronous Parallel (BSP) model [8, 14]. In BSP, each iteration cycle consists of two steps. During the first step (also called a superstep), each nodes performs only local calculation and processing, i.e. without communicating with the other nodes. In the second step (also called synchronization), communications starts: all nodes broadcast and their receive messages.

We simulate the dynamic topology as follows. Messages can only be transmitted along directed edges of the graph, but have a edge-specific reception probability that is fixed in advance. The order in which the nodes broadcast their messages is random; but when a node has received 32 messages, all new messages will be ignored. Previous experiments at CHESS show that this a realistic way to model the time slots and possible collision of messages in a slot.

5.3 Results

All algorithms were simulated for several given network sizes and average degrees, yielding estimates for the actual values of the network and neighborhood size in each node. From these estimate we calculated the absolute errors, normalized by the actual values, and averaged over all nodes in the network. The results are shown in Table 2.

Some remarks about these results need to be made. First of all, most algorithms for network size estimation make use of an estimate of the (average or local) neighborhood size. The Power Law algorithm uses the Neighbors-of-Neighbors estimator for the average neighborhood size; the Epidemic algorithm uses an explicit (deliberately undershooting) estimate for the outgoing degree. For the other algorithms that needed such estimates, we used the exact values, possibly leading to better results.

Secondly, most algorithms have one or more specific parameters to improve the convergence and precision. We did not study what the optimal values for such parameters are, and choose constant values while running the simulations on different networks sizes and average degrees. Most of these parameters are related to the values that are estimated by the algorithms, which are of course unknown beforehand.

Thirdly, some algorithms like the Power Law and the Parameter Estimation algorithms assume various things about the network under consideration or the occurrences of message loss. The resulting estimates are expected to be more accurate for networks that precisely meet these assumptions.

We paid a special attention to the MinTopK algorithm that seems to be very promising. As discussed in subsection 4.3, the accuracy of this algorithm strongly depends on the value of K, see Table 1. By playing with this value one can control the trade-off between the memory used by network nodes and the accuracy of the algorithm. In our experiments, we used three values of K: 8, 32, 128. For each of these values we performed 10 simulations and averaged the results, see Table 2. The results are consistent with our expectations and are close to the estimates that are provided in Table 1. The observed deviations can be attributed to the fact that not all simulation runs converged to the "correct" configuration–this is certainly the case for networks with 1000 nodes and the number of iterations limited to 30.

The Random Tours algorithm did not yield any estimates, which is denoted in the table by a dash. This is due to the fact that, in order to get a good random tour, the minimal path length must be high enough; but due to message loss, the higher this parameter, the lower the probability that two broadcasts meet, resulting in no estimates. Moreover, even if a random tour would be completed and estimates were calculated, we expect these to be inaccurate since the assumptions of the original algorithms for bidirectional graphs are not met: the Markov process corresponding to one of the random paths generated is not necessarily reversible, recurrent, nor does it necessarily have a stationary distribution.

We observed that the Epidemic algorithm sometimes diverges, especially for larger networks with a higher connectivity. This results from too much mass being lost by failing links, such that the estimates for the network size in each node (taken as the inverse of the mass in that node) will blow up.

In general, we can see that all algorithms become more inaccurate for larger networks and increasing average degree. This is expected since in larger networks, algorithms must run longer before reaching consensus, and depending on the algorithm, too much information may become lost in the process.

	size	100	100	1000	1000
	av. degr.	10	30	30	100
algorithm	steps	neighborhood size estimation			
Rand. ID nbh.	30	1.5%	4.1%	16.1%	50.9%
	300	1.5%	4.1%	58.5%	59.6%
Parameter est.	30	55.3%	178.0%	252.2%	942.3%
	300	15.6%	20.7%	30.0%	95.6%
Future bc.	30	119.1%	138.5%	136.9%	33.9%
	300	116.2%	127.0%	136.6%	39.6%
algorithm	steps	average neighb. size estimation			
Counting n. of n.	30	29.9%	25.9%	27.5%	53.3%
	300	29.5%	25.0%	27.4%	53.0%
algorithm	steps	network size estimation			
Epidemic	30	76.0%	91.3%	80.0%	89.0%
	300	72.8%	91.5%	93.5%	94.0%
Rand. ID nw.	30	85.5%	55.4%	90.5%	85.4%
	300	37.1%	2.4%	5.5%	84.5%
MinTop8	30	22.7%	33.9%	23.3%	77.6%
	300	28.3%	30.4%	22.8%	39.8%
MinTop32	30	17.2%	14.0%	12.5%	72.9%
	300	11.3%	10.1%	17.4%	49.9%
MinTop128	30	0.3%	0.3%	8.8%	84.5%
	300	1.0%	0.5%	5.4%	36.5%
Power law	30	1550.2%	567.5%	98.6%	94.5%
	300	82.8%	8190.6%	98.6%	94.8%
Random tours	30	-	-	-	-
	300	-	-	-	-
Msg.b. count.	30	30.1%	22.8%	84.9%	91.7%
	300	36.4%	55.8%	77.4%	90.0%

Table 2: Simulation results: mean relative absolute errors.

6 Conclusion and Further Research

Although most algorithms did converge to a stable estimate, the relative errors were still significant, especially for larger networks. As mentioned, large network sizes are generally difficult to estimate in unreliable networks. The results of our simulations seem to suggest that the MinTopK algorithm is the most promising. However, a fair comparison between the algorithms is difficult because of the trade-off between memory and accuracy, and because all algorithms use different parameters. More research needs to be done to find smart and adaptive choices for the parameters of the algorithms, which is expected to improve the estimates considerably. Such results can be used in more extensive simulations to decide which of the algorithms are the fastest, robust and the most accurate.

Acknowledgments

We would like to thank Bert Bos and Daniela Gavidia from CHESS, The Netherlands, for useful discussions during the week. Furthermore, we would like to thank Sander Dommers (Eindhoven University of Technology) for sharing his knowledge of power law graphs and Adrian Muntean (Eindhoven University of Technology) for his help during the early stages of this research.

A Appendix: Halving algorithm

In this section we describe the strategy of an algorithm which computes the exact size of a network. This strategy can be applied to networks with both static and dynamic topologies.The main idea of this algorithm is to select one half of the population of the network a consecutive number of times, at the end the nodes which represent the remainder of the halving at some stage represent some digit 1 of the binary expression of the number of node in the network.

The core of the algorithm consists of a loop with three steps:

1. in the first step a node send messages to find a partner and check incoming messages to find pairing requests of other nodes,

2. in the second step (when a node either has received a pairing request or his pairing requests has been accepted by another node) the node decides whether itself or the partner is going to be included in the selected half of the population (upgrade). The level of the upgraded node is increased by one (the initial levels of all nodes are zero),

3. if a node can not find a partner considers itself a digit (at the position of its level) of the binary expression of the size of the network and spreads this information around

The algorithm eventually converges to the size of the network, nevertheless at any time it is undecidable if the current estimation of the size of the network is the final one (except when the estimation is less than the number of neighbors) this because the problem is semicomputable. There are a number of issues to take in account in order to ensure that the algorithm will converge.

First of all the way a node finds a partner needs to be implemented in a way which does not generate deadlocks. The most simple implementation is to set states for each stage of the pairing process, one for the following situations:

1. a node looking for a partner
2. a node who accepted the request of pairing of another node and he is waiting for confirmation
3. a paired node

To ensure the absence of deadlocks the second state needs to be split in two states, according to the role which a node has during the pair process. The role can be the one of asking the pairing, or the one of being asked the pairing, for instance the role can be decided according who has the greatest id. The same criteria could also decide which of the two nodes is going to be upgraded.

The pairing request is communicated by messages. A node is more likely to get paired with a neighbor. Nevertheless it may happen that all neighbors are paired and the only node available is distant and can not be reached by the messages because their lifespan is too short.

To cope with this issue every time a node needs to wait too long to find a partner doubles the life of his messages.

References

[1] CHESS. http://www.chess.nl.

[2] J.A. Carrillo, M. Fornasier, J. Rosado, and G. Toscani. Asymptotic flocking dynamics for the kinetic Cucker–Smale model. *SIAM Journal on Mathematical Analysis*, 42(1):218–236, 2010.

[3] R.J. Carroll and F. Lombard. A note on N estimators for the binomial distribution. *Journal of the American Statistical Association*, 80(390):423–426, 1985.

[4] R. Cohen, K. Erez, D. ben Avraham, and S. Havlin. Resilience of the internet to random breakdowns. *Physical Review Letters*, 85(21):4626–4628, Nov 2000.

[5] A. DasGupta and H. Rubin. Estimation of binomial parameters when both n, p are unknown. *Journal of Statistical Planning and Inference*, 130(1-2):391–404, 2005.

[6] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database management. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, PODC 87, pages 1–12. ACM, 1987.

[7] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, May 1995.

[8] J.M.D. Hill, B. McColl, D.C. Stefanescu, M.W. Goudreau, K. Lang, S.B. Rao, T. Suel, T. Tsantilas, and R.H. Bisseling. BSPlib: The BSP programming library. *Parallel Computing*, 24(14):1947–1980, 1998.

[9] M. Jelasity and A. Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, ICDCS '04, 2004.

[10] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23:219–252, 2005.

[11] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 482–491. IEEE Computer Society, 2003.

[12] P. Korteweg, M. Nuyens, R.H. Bisseling, T. Coenen, H. van den Esker, B.J. Frenk, R. de Haan, B. Heydenreich, R.W. van der Hofstad, J.C.H.W. in 't Panhuis, L. Spanjers, and M.H van Wieren. Math saves the forest: Analysis and optimization of message delivery in wireless sensor networks. In *Proceedings of the 55th European Study Group Mathematics with Industry*, pages 117–140. Technische Universiteit Eindhoven, Jan-Febr 2006.

[13] L. Massoulié, E. Le Merrer, A. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, PODC '06, pages 123–132, New York, NY, USA, 2006. ACM.

[14] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33:103–111, August 1990.

[15] Kyu-Young Whang, Brad T. Vander-Zanden, and Howard M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 15(2):208–229, June 1990.