Gossip-based counting in dynamic networks

Ruud van de Bovenkamp, Fernando Kuipers, and Piet Van Mieghem

Network Architectures and Services
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
{R.vandeBovenkamp, F.A.Kuipers, P.F.A.VanMieghem}@tudelft.nl

Abstract. We propose GOSSIPICO, a gossip algorithm to average, sum or find minima and maxima over node values in a large, distributed, and dynamic network. Unlike previous work, GOSSIPICO provides a continuous estimate of, for example, the number of nodes, even when the network becomes disconnected. GOSSIPICO converges quickly due to the introduction of a beacon mechanism that directs messages to an autonomously selected beacon node. The information spread through the network shows a percolation-like phase-transition and allows information to propagate along near-shortest paths. Simulations in various different network topologies (ranging in size up to one million nodes) illustrate GOSSIPICO's robustness against network changes and display a near-optimal count time. Moreover, in a comparison with other related gossip algorithms, GOSSIPICO displays an improved and more stable performance over various classes of networks.

Keywords: Gossip-algorithms, network dynamics, node counting

1 Introduction

Several developments over the past years, such as the growing use of peer-to-peer overlay networks and sensor networks have led to the deployment of very large distributed networks. Although distributed networks are very scalable, they have no central point where information is stored, which makes it challenging to gather global network properties or perform coordinated actions.

One of the paradigms that has emerged to spread and gather information in a fully distributed network is that of gossip algorithms [1]. During the gossip process, a node periodically selects one of its neighbours and either sends, requests or exchanges information with that neighbour. This simple communication structure can be used to perform various tasks in distributed networks, ranging from overlay building and information location to calculating functions such as sums and averages.

In this paper we investigate averaging and summation over node values in large dynamic networks, and in particular the specific case of counting the number of nodes. Good estimates of the size of a distributed network can be valuable

in optimising the performance of protocols and services that run on top of it, such as topology building in a peer-to-peer network. Node counting can also provide information on how many sensors are still working in a sensor network or how large an ad-hoc network currently is. The latter can be useful in vehicular communication to estimate traffic conditions.

This paper is organised as follows. Related research is first discussed in Sec. 2, after which we propose our gossip-based counting algorithm GOSSIPICO in Sec. 3. In Sec. 4, we illustrate the algorithm's speed and robustness against network dynamics by a series of simulations. Sec. 5 presents our conclusions.

2 Related Work

Algorithms specifically designed to estimate or count the number of nodes in a network can be roughly divided into three groups: probabilistic polling algorithms, random walk based algorithms, and gossip-based algorithms [2]. We highlight two representative examples of the first two techniques, after which we overview gossip-based algorithms. The probabilistic polling technique proposed by Kostoulas *et al.* [3] uses a conditional reply to a request message to estimate the network size, where the condition is determined by the distance between the replying node and the initiating node. In the algorithm by Massoulié *et al.* [4], a node sends a message containing an initial counter value on a random walk, and each node that is passed on the walk adds a degree dependent value to the counter. When the random walk returns to the initiating node, it can estimate the network size based on the counter value.

The gossip-based algorithms in [5, 6] rely on averaging an initial value over all nodes to estimate the network size. Jelasity and Montresor [6] initialise the node values to 0, with the exception of a single "special" node that has value 1. Nodes find the estimate of the network size as the reciprocal of the average. Montresor and Ghodsi [5] first build an overlay structure based on random node values to determine an initial guess of the network size at each node. That initial guess is subsequently averaged to improve the estimate. In the work of Kempe *et al.* [7] each node has a weight-value pair, and periodically sends half of its weight and value to both a random neighbour and itself. Nodes sum the pairs they receive and estimate the network size by dividing the summed value by the summed weight. By assigning a 1 or 0 to the initial weight of one or all of the nodes, the algorithm either sums or averages. Finally, Guerrieri *et al.* [8] studied the performance of both an averaging-based gossiping algorithm and three variations on a token collecting algorithm in delay-tolerant networks.

A major drawback of the random walk and probabilistic polling strategies is the poor accuracy [2]. Size estimation based on averaging using gossip, however, currently also has its drawbacks. First of all, the process has to be started by a single node, which requires some sort of coordination in the initial phase of the algorithm. Moreover, the estimate of the network size at a specific node can exhibit a large overshoot, and joining or leaving nodes cause large local variations in the estimate that have to be spread out over the network. More

extreme network dynamics such as the joining of two networks will result in an estimate that is half the new network size, whereas splitting the network in two will lead each part to believe it consists of the original number of nodes.

Our goal is to develop a gossip algorithm that does not share the drawbacks that averaging-based algorithms show and can be used in continuously changing networks, even when the network becomes disconnected. In our design we aim for little processing and storage at each node. Also, all nodes are uniform in capabilities and function, i.e. no node is special. The only requirements for the nodes are that they have a unique identifier and can reliably pass messages to their direct neighbours.

3 Gossipico

We propose Gossipico, a gossip-based algorithm that does not rely on averaging to estimate the network size, but combines messages to count the number of nodes. Message combining offers more flexibility in algorithm design than pair-wise gossip interactions alone, and allows for the simultaneous execution of several functions such as summation, finding maxima, and averaging in a single algorithm. Gossipico consists of two parts: Count and Beacon. Count performs the actual counting of the nodes, while Beacon is used to speed up Count. We first introduce the basic ideas behind Count and Beacon in Sec. 3.1 and 3.2, respectively, and will further develop those in Sec. 3.3 where we add robustness against dying nodes and disconnecting networks. In both Count and Beacon, nodes initiate communication periodically. We define a gossip cycle as the time period during which all nodes have initiated communication with one of their neighbours. The time between cycles in a deployment scenario will be a trade off between bandwidth consumption and speed.

3.1 Count

The general idea behind Count is that information from every node is combined and then spread to all the nodes again. This idea is illustrated in Fig. 1, where in the left-hand side picture every node has a single token. Each node periodically selects a random neighbour and gives this neighbour all its tokens (the path of the tokens is indicated by arrows); when a node receives tokens from another node it spreads messages telling how many tokens it has. The centre picture shows the spreading messages as hexagons with the number of tokens inside. As can be seen, tokens start to accumulate at different points in the network and the first estimates of the network size start to spread. In the right-hand side picture all tokens have piled up and every node knows how many tokens there are.

In Count the exchanged messages are more complex than tokens to allow for greater functionality: counting both upwards and downwards, averaging and finding minima and maxima. A message $M = \{C, F, T\}$ contains a count value $C \in \mathbb{Z}$, a freshness value $F \in \mathbb{N}$, and a type value $T \in \{0, 1\}$. The type value

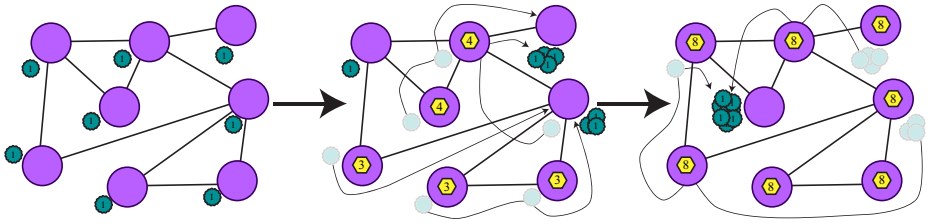

Fig. 1. Three snapshots in the counting process.

indicates whether the message is an Information Spreading (IS) ($T = 0$) message, corresponding to the hexagons in Fig. 1, or an Information Collecting (IC) ($T = 1$) message, corresponding to the tokens. The count value C is the current estimate of the network size, whereas the freshness value F is a measure for the estimate's recentness. The message with the highest value for F contains the most recent information.

Every node that joins the network is initialised with an IC message with count and freshness value 1, i.e. $M = \{1, 1, 1\}$, analogous to the single token in Fig. 1. Nodes send their message periodically and always create a new IS message afterwards. Messages are processed by the receiving node by following one of four rules, based on the type and the freshness of both the received message, M_r, and the receiving node's current message waiting, M_w (subscripts r and w stand for received and waiting, respectively). These four rules are explained below:

1 ($T_r = 0, T_w = 0$) If both messages are IS messages, the received message will replace the waiting message if the received message has a higher freshness value F than the waiting one. By keeping the message with the freshest information, only new information is spread through the network.
2 ($T_r = 1, T_w = 0$) A received IC message will replace a waiting IS message in order not to lose the collected information.
3 ($T_r = 0, T_w = 1$) A received IS message will be discarded if the waiting message is an IC message. This again ensures that no collected information is lost.
4 ($T_r = 1, T_w = 1$) If both messages are IC messages, a new message will replace the waiting message. The new message will be an IC message containing the sums of the waiting and received messages' C and F values, i.e. $M = \{C_w + C_r, F_w + F_r, 1\}$.[1] We call the process of replacing M_w and M_r by a new message combining messages.

In addition to the waiting message, every node has two state values: a count value $C_s \in \mathbb{Z}$, and a freshness value $F_s \in \mathbb{N}$ (subscript s stands for state).

[1] Nodes should be initiated with a message $M_w = \{V, 1, 1\}$ to sum over node values V instead. Finding a minimum or maximum value is achieved by not summing the values but keeping the highest or lowest value, whereas averaging requires an extra field in the message where the number of summed values are counted to allow the average to be calculated as V/C.

After processing the message, the receiving node updates its state values C_s and F_s if the resulting waiting message contains fresher information. Without separate state information, a node's estimate of the network size could go down temporarily when, for example, rule 2 forces a node to discard an IS message with fresher information, in order not to lose an IC message. After the node has passed on this IC message, it uses the state information to create a new spreading message with the most recent information it has.

All nodes are counted when there is only one IC message left. Rule 4 forces nodes to combine a received and waiting IC message, thereby reducing the number of IC messages in the network. At the same time, nodes cannot create new IC messages after initialisation; all new waiting messages are IS messages. The four rules described above ensure that if there is only one IC message left, it will contain the combined information of all the initial IC messages.

A message's freshness value will only differ from the count value in dynamic networks or when the algorithm is used to sum or average. When the algorithm is used in dynamic settings, the freshness value could run out of its range, but this can be fixed by allowing nodes to accept messages with a freshness value much lower than their F_s as fresher, if F_s is close to the maximum, or, alternatively, to trigger a recount using the mechanisms described in Sec. 3.3.

3.2 Beacon

The rules listed in Sec. 3.1 ensure that an IC message is passed on from one node to another, until it reaches a node that already has an IC message waiting, where it is combined. When nodes select a communication partner at random, IC messages perform random walks. Since a static network is counted when the last two remaining IC messages are combined, the count time is at least equal to the time it takes two independent random walks to meet at the same node, which increases prohibitively with the network size. Hence, IC messages should be forwarded in a less random fashion in order to speed up the counting process.

IC messages have a much higher chance of meeting each other when they are guided to a particular node in the network. We propose Beacon to guide IC messages towards each other by using a beacon. A beacon is a node whose location information spreads through the network by means of gossip.

Every node starts out as a beacon and competes for dominance with the other nodes to ensure that eventually there will be only one beacon. The competition resembles a battle between different armies, with the beacons as the army leaders. Every node j belongs to an army A_j that has a certain strength $S_j \in \mathbb{R}$; node j also knows which neighbour P_j is the first hop towards the beacon of its army, and the estimated hopcount D_j to that beacon.

Initially, every node forms its own one-node army with a randomly chosen strength, i.e. $A_j = j$, $P_j = j$, $D_j = 0$, and $S_j = \text{random}$.[2] Nodes periodically and randomly select one of their neighbours to skirmish with. The outcome of such a skirmish is determined by two rules:

[2] We assume perfect randomness, i.e. $\Pr[S_j = S_k] = 0$, for $j \neq k$.

1 If both nodes are of the same army, the shortest path to the beacon is updated.
2 If one node is stronger than the other, the weaker node is incorporated into the army of the stronger one. The losing node takes over the values for A and S from the winning node, sets the winning node as the next hop to the beacon, and sets the estimated hopcount to one more than the estimated hopcount of the winning node.

Following these rules, the strongest node in the network will defeat all other nodes and become the only beacon. At the same time, the hop sequences towards the beacon converge to shortest paths along which IC messages can combine.

3.3 Network dynamics

GOSSIPICO, as outlined so far, can cope very efficiently with a growing network: new nodes simply initialise their message at $M_w = \{1, 1, 1\}$ and start communicating. Node (and link) removals, however, are more challenging. A node can leave gracefully by sending an IC message with a count value of -1 ($M_w = \{-1, 1, 1\}$) to account for its departure, but when a node dies, it cannot send this message. Moreover, even when a node leaves the network gracefully, there is no guarantee that the network will remain connected. If the network indeed becomes disconnected, a restart of the counting process must be triggered.

In order for nodes to be able to trigger a recount, the counting process is restricted to work only within the BEACON armies. When a new army is built, the nodes that are a part of that army count the size of their *army* using GOSSIPICO as described above. As the growing army conquers the network, the network is counted. Triggering a restart comes down to building a new army that can defeat the reigning army.

GOSSIPICO copes with dying nodes and disconnecting networks by following a single rule: every time a link is removed, the nodes adjacent to the link rebuild their armies to trigger a recount. In case the link removal did not disconnect the network, the two new armies will compete for dominance and the network is recounted. In case the link removal disconnects the network, the two revived armies will both survive, but cannot reach each other to compete. The algorithm is unaffected by network dynamics such as node and link addition or removal, because every node can trigger a recount in response to node or link removals.

Recounts are possible in COUNT if nodes will only accept messages from neighbours that are in the same army, and return any received messages to the sender otherwise. When a node is incorporated in a new army, it will create a new IC message with all values set to one ($M_w = \{1, 1, 1\}$) to be counted in its new army.

Two concepts are added to BEACON to support recounts: army revival and immunity. Army revival allows a node j that is conquered by another army ($A_j = k$) to build its own army again by selecting a new random strength S_j and setting itself with a zero distance as the first hop to the beacon ($P_j = j, D_j = 0$). The newly revived army of node j ($A_j = j$) will also be immune ($I_j = k$) to

Algorithm 1 COUNT

Init: $M_w \leftarrow \{1, 1, 1\}$

Sending
if $T_w = 1$ & $A \neq id$ **then**
  select peer P
else
  select random peer
end if
send message M_w to peer
$M_w \leftarrow \{C_s, F_s, 0\}$

Receiving
if $A_r \neq A$ **then**
  return M_r to sender
  return
end if
if $T_r = 0$ & $T_w = 0$ **then**
  if $F_r > F_w$ **then**
    $M_w \leftarrow M_r$
  end if
end if
if $T_r = 1$ & $T_w = 0$ **then**
  $M_w \leftarrow M_r$
end if
if $T_r = 1$ & $T_w = 1$ **then**
  $M_w \leftarrow \{F_w + F_r, C_w + C_r, T_w\}$
end if
if $F_w > F_s$ **then**
  $F_s \leftarrow F_w$
  $C_s \leftarrow C_w$
end if

resetCount
$C_s \leftarrow 1$
$F_s \leftarrow 1$
$M_w \leftarrow \{1, 1, 1\}$

A message consists of three fields: C, F, and T, i.e. $M = \{C, F, T\}$. Subscripts r, w and s indicate received, waiting and state, respectively.

its previous army k. The immunity ensures that all nodes of army j will win all skirmishes with nodes of army k. Without immunity the recount will be suppressed if the new army j is weaker than the reigning army k.

Although GOSSIPICO is robust against dying nodes and other network dynamics using recounts as described above, a node's count value might drop during the recount process. Dips do not last long, and can even be avoided at the expense of a slight delay if nodes estimate the network size based on their estimate before the recount and the current count value. A node's estimate X could be calculated as $X = (1 - f(t))X_{old} + f(t)C_s$, where X_{old} is the node's estimate at the time it was incorporated into a new army and $f(t)$ is a logistic function that shifts the current estimate from the previous estimate to the current count value. The logistic function is given by $f(t) = 1/(1 + \exp(-t + 2D + 5))$ where t is the number of times the node created the same IS message. The constant 5 is used to make sure that $f(t)$ starts at a value close to 0. The distance D to the beacon ensures that high diameter (and therefore slower converging) networks automatically adjust to the recount time without the need of a tuning parameter. If the value of C_s is larger than X_{old}, nodes accept C_s as the best estimate. In all simulations we avoid dips.

3.4 Interaction between Count and Beacon

The interactions between COUNT and BEACON are illustrated in Fig. 2. COUNT uses the next hop to the beacon in BEACON when an IC message is sent and the army information when receiving messages; BEACON resets the count process when a node is conquered by another army. The precise interactions will be discussed using the pseudo-codes of both COUNT and BEACON.

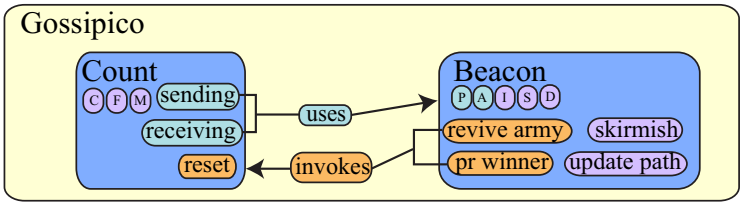

Fig. 2. Interactions between COUNT and BEACON.

The pseudo-code for COUNT is given in Algorithm 1. Lines 1-6 ensure that if a node wants to send an IC message ($T = 1$), and it is not the beacon of its own army, it will send the message to the next hop towards the beacon of its army. In all other cases the node will send its waiting message to a random neighbour. After sending a message a new waiting message is created in line 7. Lines 8-11 make sure that nodes only accept messages from within their army. If the message is indeed from within the army, it is processed according to the four rules in Sec. 3.1: lines 12-16 execute rule 1, rules 2 and 3 are executed in lines 17-19, and finally lines 20-22 form rule 4. The node's state values are updated in lines 23-26 if the new information is fresher than the current information. When a node is incorporated into a new army, it has to reset its state and message as shown in lines 27-29.

Algorithm 2 shows the pseudo-code for BEACON. The first thing that is checked during a skirmish is whether one of the nodes is immune to the army of the other node, as can be read from lines 1-8. If this is the case, the node that is immune will win the skirmish and the losing node takes over the strength, army and immunity from the winning node (ll. 26-30), sets the winning node as the first hop towards the beacon (ll. 29-30), and resets its COUNT process (l. 31). If both nodes are of the same army (ll. 9-12) the shortest path to the beacon is updated by checking whether either node can reach the beacon quicker via the other (ll. 18-25). If both nodes are from different armies, the stronger one will incorporate the weaker one (ll. 13-17). When a node revives its army it will determine a new random strength, set its army id equal to its node id, set its immunity equal to its present army, set itself at distance zero from the beacon and reset its COUNT process (ll. 32-37).

3.5 Convergence of Gossipico

In this discussion we divide the node counting process into three stages, although in reality these stages happen at the same time. In the first stage, the armies fight for dominance to establish a single beacon. In the second stage, the nodes send their IC messages around until they have all been combined. Finally, in the third stage, there is only one IC message left and the freshest count value spreads through the network. All three stages can be bounded by the spreading time of a single rumour in a network, which Giakkoupis [9] bounds as $O(\phi^{-1}\log(N))$,

Algorithm 2 BEACON

```
    Skirmish(Node i,j)
 1: if I_i = A_j then
 2:     prWinner(i,j)
 3:     return
 4: end if
 5: if I_j = A_i then
 6:     prWinner(j,i)
 7:     return
 8: end if
 9: if A_i = A_j then
10:     updatePath(i,j)
11:     return
12: end if
13: if S_i > S_j then
14:     prWinner(i,j)
15: else
16:     prWinner(j,i)
17: end if

    updatePath(Node i,j)
18: if D_i < D_j + 1 then
19:     P_i ← j
20:     D_i ← D_j + 1
21: end if
22: if D_j < D_i + 1 then
23:     P_j ← i
24:     D_j ← D_i + 1
25: end if

    prWinner(Node i,j)
26: A_j ← A_i
27: S_j ← S_i
28: I_j ← I_i
29: D_j ← D_i + 1
30: P_j ← i
31: resetCount          ▷ in Count

    reviveArmy(Node i)
32: S_i ← random integer
33: D_i ← 0
34: P_i ← i
35: I_i ← A_i
36: A_i ← i
37: resetCount          ▷ in Count
```

where N is the number of nodes in the network and ϕ the conductance.[3] It is important to note that ϕ depends on the network type and can depend on N.

During the first stage, the beacon information of the strongest army spreads unhindered through the network, just as a rumour in Giakkoupis' work. During the second stage, the IC messages are routed towards the beacon. Most messages combine on their way to the beacon, but in the worst case they combine at the beacon. Since the beacon information was spread in at most $\phi^{-1}\log(N)$ rounds, the longest path to the beacon is also $O(\phi^{-1}\log(N))$. During the third stage of the algorithm, a single piece of information, i.e. the count value of the last IC message, spreads to all other nodes by means of gossip; this is exactly the same process as rumour spreading. Simulations in the following section show that GOSSIPICO has an $O(\log(N))$ convergence in random and scale-free graphs.

4 Simulation results

We have performed various simulations to test the convergence and dynamic behaviour of GOSSIPICO. Time is measured in gossip cycles. During each gossip cycle, the nodes are processed sequentially and in a random order. The count time of the network is defined as the number of gossip cycles that pass until every node in the network reaches a count value equal to the network size, and is averaged over a number of runs. When we use Erdős-Rényi random graphs, the probability p that a link exists is set to $p = 2\log(N)/N$; scale-free graphs are

[3] $\phi = \min_{S\subseteq\mathcal{N},\sum_{i\in S} d_i \leq |\mathcal{L}|} \frac{\text{cut}(S,S^c)}{\sum_{i\in S} d_i}$, where $\text{cut}(S,S^c)$ is the set of links crossing a partition of a graph $G = (\mathcal{N},\mathcal{L})$ in S and S^c, with $\mathcal{N}$ the set of nodes, $\mathcal{L}$ the set of links, and d_i the degree of node i.

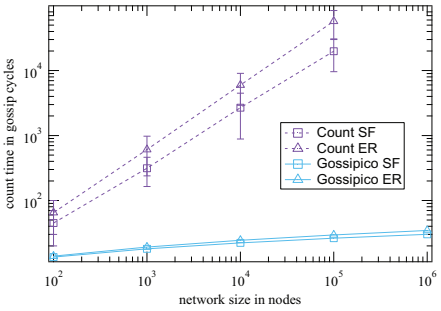

Fig. 3. Count time for Gossipico and Count. ER indicates an Erdős-Rényi graph, SF a scale-free graph.

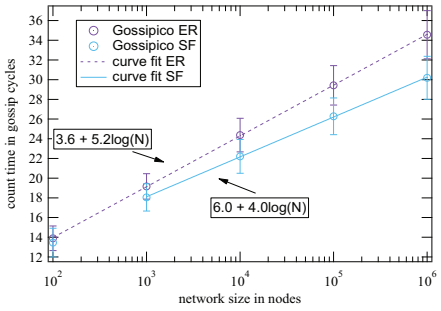

Fig. 4. Count time for Gossipico.

generated following the preferential attachment model in Batagelj and Brandes [10] with such a number of links per new node that the total number of links is closest to that in an Erdős-Rényi graph with the same number of nodes. We will first discuss the rate of convergence in static networks. We then discuss the dynamic behaviour of Gossipico.

4.1 Counting in static networks

To illustrate the impact of Beacon on the performance of Count, we compare the performance of Gossipico to that of Count alone. Since Count can be seen as an extension of randomized token forwarding algorithms, these simulations also illustrate by how much Beacon might speed up other algorithms. Simulated network sizes range up to 10^6 nodes, and for every size a hundred different networks were generated.

Fig. 3 shows the count time for both Gossipico and Count as a function of the network size for the two graph types. It is clear from Fig. 3 that Gossipico greatly outperforms Count on its own. The count time of Count scales as a power function of N, whereas the count time of Gossipico scales logarithmically. The logarithmic scaling of Gossipico is illustrated in Fig. 4. This plot is based on an average over 500 runs per size and shows an error band at $\pm$ one standard deviation. The count time can be fitted with $3.6 + 5.2 \log_{10}(N)$ for the random graphs and $6.0 + 4.0 \log_{10}(N)$ for the scale-free graphs. For random graphs the fit goes through all data points from $N = 100$ nodes and upwards, whereas for scale-free graphs the fit starts at $N = 1000$ nodes, because the scale-free degree distribution only emerges for larger networks. Both Count and Gossipico perform better in scale-free graphs than in random graphs; the hub nodes in scale-free graphs function as meeting places where IC messages can be combined. Because Beacon ensures that Gossipico scales logarithmically in random and scale-free networks, Count and similar algorithms are made suitable for use in very large networks.

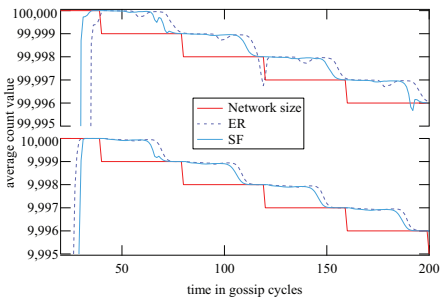

Fig. 5. Average estimate of the network size during a targeted beacon node attack for both ER and SF graphs and two different initial sizes.

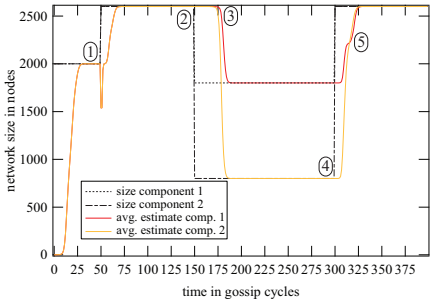

Fig. 6. Average estimate of the network size after random node additions (1), targeted link removals leading to disconnectedness (2,3), and reconnecting (4,5).

4.2 Counting in dynamic networks

In order to illustrate GOSSIPICO's robustness against node failure and to show that no node is indispensable, we track the average estimate of the network size while the node currently functioning as beacon is killed every $40th$ cycle. By killing a node it is meant that all links adjacent to the node are removed and that the node's state information and current message are lost to the network. We start out with both Erdős-Rényi random graphs and scale-free graphs ranging from $N = 1000$ to $N = 100,000$ nodes. In Fig. 5 the average estimate of the network size over time is shown. As can be seen from Fig. 5, the average estimate of the network size follows the real network size. During the 25 cycles after the beacon dies, the network is recounted and the nodes shift their estimate from the previous count value to the new one to prevent a drop in the estimate. Even when the beacon node dies before the network is recounted, the average count value does not fluctuate too much as can be seen in the case of the $100,000$ node network (the upper graph in Fig. 5). This shows that GOSSIPICO can keep up with dynamics with a rate of change close to, or marginally over, its own speed of convergence without introducing large fluctuations in the estimate.

We next show GOSSIPICO's behaviour in fully dynamic and disconnecting networks. We start by creating two random graphs, one of 1500 nodes (component 1) and one of 500 nodes (component 2), and connect them by adding 10 links between randomly selected nodes in each component. After fifty cycles we add 300 nodes to each component, all new nodes only link to nodes in their own components. After another 100 cycles the 10 links connecting the two components are removed, and are added again after another 150 cycles.

The average estimate of the network size over time is plotted in Fig. 6. The figure shows that the maximum count time for the initial 2000-node network over all the runs in this simulations is 33 cycles, regardless of the bottleneck of 10 links connecting the two components.

The addition of the 300 nodes per component (event 1 in Fig. 6) shows that new IC messages combine along near shortest paths. All IC messages combine, on average, within 7 cycles, while the average hopcount in these bridged networks is 4.2. After another 14 cycles, the spreading process finishes. The narrow dip (which only lasts for 1 cycle) in the estimated network size that is visible immediately after the addition of the new nodes is the result of the initial estimate of 1 of all new nodes.

Removing the bridging links between the two components (event 2) has a similar impact on the average estimate as a single dying node. The estimate falls (event 3) rapidly after an initial delay of about 25 cycles, without fluctuating. Regardless of the fact that 20 nodes trigger a recount at the same time, the estimate of the component size smoothly approaches the correct value in both components.

After reconnecting the two components (event 4), the estimate of the network size is updated just as quickly as a normal count of the network would be, as can be seen by comparing the first 25 cycles in Fig. 6 to the first 25 cycles after the components are reconnected in cycle 300. The increase in the smaller component is continuous, while the estimate in the larger component displays a bend (event 5) half way. This bend can be explained by observing that in half of the cases the army in the smaller component will dominate the network, and in the other half of the cases the army in the larger component will do so. When the larger component assimilates the smaller, the nodes in the smaller component will immediately accept the new count value as the new estimate because it is higher than their previous count value. In the opposite case, the nodes in the larger component will hang on to their previous count value a little longer. In either case, the counting process continuous without a restart as soon as the two components are connected.

Figs. 5 and 6 illustrate that Gossipico smoothly tracks all dynamics in the network, ranging from node addition to dying nodes and network components disconnecting and reconnecting again. Instead of relying on a periodic recount, Gossipico reacts to changes in the network. This ensures that in periods of network stability the estimate of the network size is also stable, while during periods of network dynamics, the algorithm tracks the changing network size.

4.3 Maximum count value over time

In this section we discuss the maximum count value in the network as a function of the normalised number of sent messages. The product of the number of nodes and the expected hopcount $E[H]$ is taken as the normalisation constant. We simulated four different network topologies, an Erdős-Rényi random graph, a scale-free graph, a square grid, and a path. For each network type the highest count value in the network after every interaction between two nodes is averaged over 500 different realisations, both for $N = 1,000$ and $N = 10,000$. As an approximation of the average hopcount of both an Erdős-Rényi graph and a scale-free graph we use $\log(N)$, while for the grid and path

graph we use the approximation for a regular lattice $E[H] \simeq \frac{d}{3}N^{1/d}$, where d is the dimension of the lattice (for a grid $d = 2$, for a path $d = 1$) [11].

Fig. 7 shows that the normalised number of sent messages before the maximum count value is reached, is of the same order of magnitude for all four networks. The main difference between the networks lies in the slope of the curves. In random and scale-free graphs, the maximum count value follows an S-curve that is more pronounced for larger networks, while in the grid and path graphs the maximum count value increases more steadily, and shows little influence of the network size. The difference between the grid and path graphs on the one hand and the random graphs on the other, is probably best explained by the speed at which the winning army can grow. The winning army in the path graph will grow at a fixed rate because the configuration at the borders of the army never changes. A growing army will always have two nodes at its endpoints that are neighbours with different armies. In the case of a square grid, the nodes in the winning army roughly form a circle and the army grows at an increasing rate proportional to the circumference of that circle. The increase in growth rate, however, is stable. For the random and scale-free graphs the situation is quite different. With every addition of a node to the winning army, the percentage of nodes not in the winning army that can be reached by the winning army can increase rapidly, thus allowing the army to grow explosively. After a brief initial phase the number of nodes in the winning army undergoes a phase-transition and covers almost the entire network. Similar phase transitions have been observed in processes where connections are randomly formed and have been explained in the context of percolation theory (e.g., [12]). Although the average hopcount is the most important factor in the count time for all four network types, the speed at which an army can grow determines the slope of the graph.

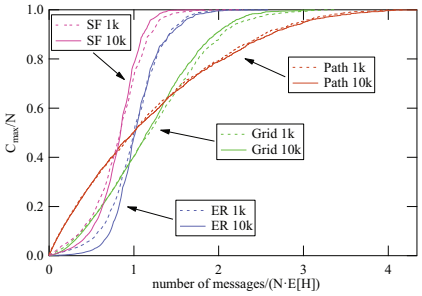

Fig. 7. Maximum count value as a function of the normalised number of sent messages for four graph types.

5 Conclusions

We have proposed a gossip algorithm, GOSSIPICO, to count the number of nodes in a network (or sum/average over node values). Our algorithm works by combining messages, which has an advantage over averaging-based counting algorithms in that the estimate approaches the network size quicker and more smoothly. GOSSIPICO uses only $O(\log(N))$ messages per node to count Erdős-Rényi random graphs and scale-free graphs, while randomised token forwarding based counting algorithms use $O(N^{\alpha})$ messages per node, with a power exponent $\alpha > 0$. The

count time closely follows the average hopcount for grids and path graphs, which matches the lower bound $O(D)$, with D the network diameter.

A major improvement, besides speed, over previous algorithms is that GOSSIPICO automatically restarts the counting process when a change is detected that could lead to disconnectedness in the network. When two components are joined, the algorithm converges to the correct count without a restart, which is impossible for algorithms based on averaging. Simulations show that GOSSIPICO is a very robust algorithm that provides nodes with a continuously updated estimate of the network size at a speed of convergence that equals that of rumour spreading, which is known to be very fast.

References

1. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proc. of the 6th annual ACM Symp. on Principles of Distributed Computing. (1987) 1–12
2. Le Merrer, E., Kermarrec, A.M., Massoulie, L.: Peer to peer size estimation in large and dynamic networks: A comparative study. In: Proc. of the 15th IEEE Int. High Performance Distributed Computing Symp. (2006) 7–17
3. Kostoulas, D., Psaltoulis, D., Gupta, I., Birman, K., Demers, A.: Decentralized schemes for size estimation in large and dynamic groups. In: Proc. of the 4th IEEE Int. Network Computing and Applications Symp. (2005) 41–48
4. Massoulié, L., Le Merrer, E., Kermarrec, A.M., Ganesh, A.: Peer counting and sampling in overlay networks: random walk methods. In: Proc. of the 25th annual ACM Symp. on Principles of Distributed Computing. (2006) 123–132
5. Montresor, A., Ghodsi, A.: Towards robust peer counting. In: Proc. of the 9th IEEE Int. Conf. on Peer-to-Peer Computing. (2009) 143–146
6. Jelasity, M., Montresor, A.: Epidemic-style proactive aggregation in large overlay networks. In: Proc. of the 24th Int. Distributed Computing Systems Conf. (2004) 102–109
7. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. of the 44th IEEE Symp. on Foundations of Computer Science. (2003) 482–491
8. Guerrieri, A., Carreras, I., De Pellegrini, F., Miorandi, D., Montresor, A.: Distributed estimation of global parameters in delay-tolerant networks. Computer Communications **33** (2010) 1472–1482
9. Giakkoupis, G.: Tight bounds for rumor spreading in graphs of a given conductance. In: Proc. of the 28th Symp. on Theoretical Aspects of Computer Science. (2011) 57–68
10. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Phys. Rev. E **71**(3) (2005) 036113
11. Van Mieghem, P., Hooghiemstra, G., van der Hofstad, R.: A scaling law for the hopcount in internet. Technical report, Delft University of Technology (2000)
12. Achlioptas, D., D'Souza, R.M., Spencer, J.: Explosive percolation in random networks. Science **323**(5920) (2009) 1453–1455