Towards Robust Peer Counting

Alberto Montresor
University of Trento, Italy
alberto.montresor@unitn.it

Ali Ghodsi
Royal Institute of Technology, Sweden
aligh@kth.se

***Abstract*—This paper describes T-SIZE, a peer counting protocol that is based on gossip-based aggregation. Peer counting has become increasingly important as the size of the network is often a crucial parameter used to guarantee robustness, small diameter, load-balance, or to generally optimize the system. Our work improves the previous work by providing a protocol that is eventually accurate, i.e. the estimate will eventually converge to the true peer count in absence of churn. The protocol can handle extreme levels of churn, and automatically ensures that all participating nodes learn the outcome of the peer counting.**

I. INTRODUCTION

Overlay networks have received much attention in the past decade due to their ability to scale and handle dynamism. As these systems have been refined and become more sophisticated, it has become more common to use the peer count, n, as a key parameter in many of the protocols.

For example, the number of routing pointers are sometimes set to a function $f(n)$ of the peer count, to guarantee a maximum hop count [1], or to provide churn resilience [2]. Others let each node assume $f(n)$ identities to balance load [3], [4]. The performance of gossip-based broadcast protocols [5] can be optimized if n is known; groups of specified size can be built on the fly, using slicing protocols [6], [7]. In all above cases, n represents the peer count, which is to be accurately estimated at runtime.

In this short paper, we present T-SIZE, a simple gossip-based peer counting algorithm that has three advantages compared to previous work on the subject. First, it is *eventually accurate*, i.e. the estimate converges to the number of nodes in absence of churn. Second, the estimate is available on every node. Third, it can handle extreme levels of churn, which to our knowledge has not been possible previously.

A. Related work

A number of peer counting algorithms have appeared for overlay networks. Le Merrer et al. [8] provide an extensive comparative study of most of them by identifying three main classes of approaches: *probabilistic polling* [9], *random walks* [10], *gossip-based* [11]. Their comparison shows that the gossip approach based on aggregation [11] is the most accurate. Our approach is most similar to aggregation [11], but substantially improves its accuracy and resilience to failure as follows. The original aggregation approach, which is based on averaging, prescribes one node to start with its initial estimate set to one, and the rest setting their estimate to zero. Averaging will ensure that all nodes eventually receive $1/n$ as their estimate. This approach has the disadvantage that the estimate is initially exceedingly inaccurate, and failures in the beginning of the averaging process can considerably impair accuracy. Our approach does not suffer from these drawbacks.

Some related work use routing information already existing at every node as a statistical sample, and derive the number of nodes from it [12], [13], [14]. The advantage of these approaches is that they do not consume, or consume little, bandwidth, as the sample information is already present. The estimate provided might, however, never converge to the true peer count, even in the absence of churn. In contrast, our approach builds on aggregation, which is known to converge to the true statistic in absence of churn.

Certain approaches, such as Sample&Collide [10] and HopsSampling [9] will make the estimate available at the initiating node. If all nodes need to use the peer count, as is the case if n is a crucial parameter used at every node, the estimate has to be broadcast to all nodes. This bears extra bandwidth and time costs compared to our approach.

II. THE ALGORITHM

T-SIZE is a combination of two protocols, AVERAGE [11] and T-MAN [15]. The former allows the computation of the average of a collection of values distributed among the participant nodes, while the latter is capable to "bootstrap" (build from scratch) complex topologies defined through a distance function.

We provide a brief overview of the two protocols; for a complete description, please refer to the original papers [11], [15]. They are both epidemic protocols, following the generic scheme shown in Fig. 1. Nodes regularly exchange information in periodic, pairwise interactions. The scheme can be modeled by means of two distinct threads executed at each node: the active one takes the initiative to communicate, while the passive accepts incoming exchange requests.

The active thread is repeated periodically every δ_c time units (the *cycle length*). Each node selects a peer node q from the system population through function $getPeer()$; extracts a summary of the local state through function $extract()$; and finally, sends this summary to q. These operations are repeated forever. The other thread passively waits for incoming messages, replies in case of active requests, and modifies the local state through function $update()$. Note that a typical interaction consists of the active thread of node p sending a message to the passive thread of another node q, which replies with a message that is handled by the passive thread of p.

A. Montresor was supported by the European Commission through the NAPA-WINE Project (Grant No. 214412).

loop
  wait(δ_c)
  $q = getPeer()$
  $m = extract(s_p, q)$
  send $\langle \text{REQ}, p, m \rangle$ to q
end loop

(a) active thread at p

on receive $\langle t, q, m \rangle$ from * **do**
  if $t = \text{REQ}$ **then**
    $m' = extract(s_p, q)$
    send $\langle \text{REP}, p, m' \rangle$ to q
  end if
  $s_p = update(s_p, m, q)$

(b) passive thread at p

Fig. 1. The generic gossip scheme.

AVERAGE customizes the generic scheme as follows. The input is represented by a numerical value possessed by each node, which is copied in the local state at initialization. The local state represents the current approximation of the global average. $getPeer()$ returns a random peer as returned by a peer sampling service [16]; $extract()$ returns the entire state, i.e. the current approximation; while function $update(s_p, m, q)$ returns $(s_p + m)/2$, which is the new approximated value. After each exchange, the global average does not change, while the empirical variance is reduced. It has been proved that the expected variance reduction is $1/2\sqrt{e}$ after each cycle.

In T-MAN, the local state s_p of p is composed of a collection of neighbors. Initially, it is initialized randomly (again, thanks to a peer sampling service). At the end, the neighbor set is composed of the nodes closest to p w.r.t. to a given distance function d. This goal is achieved by $getPeer()$ selecting one of the nodes from s_p which is closest to p, and $extract(s_p, q)$ returning the set of $msgsize$ nodes that are closest to q. $update(s_p, m)$ simply merges the local state with the received message, returning $s_p \cup m$. It has been proven that this mechanism converges to the topology described by the distance function in a logarithmic time.

Both protocols need a local method to decide when to terminate. In AVERAGE, the protocol is stopped after an $inactive$ number of cycles has passed where the local estimate has changed less than a $precision$ threshold (i.e., when the estimates differ for a minuscule amount). In T-MAN, the protocol is stopped after an $inactive$ number of cycles has passed without adding any new node to the neighbor set (i.e., when all the closest nodes have been discovered).

Having described our building blocks, we now introduce our "combined" protocol. We first describe it in a static system, with a fixed collection $\mathcal{P}$ of nodes, and then later discuss what happens in case of failures.

Step 1: Each node p is assigned a random value v_p taken from the circular space $\mathcal{I} = \{0, \ldots, M-1\}$. For two arbitrary identifiers $x, y \in \mathcal{I}$, the distance from x to y is denoted $d(x, y)$; i.e.

$$d(x, y) = (y - x) \bmod M$$

Step 2: T-MAN is used to build a sorted ring over the set of values, using the distance function d; i.e., at the end of the execution, each node knows the identifier p and the value v_p of the successor node on the ring. q is the *successor* of p if and only if there is no process r that follows p on the ring and is closer to p than q; i.e.,

$$succ(p) = q \Leftrightarrow \nexists r \in \mathcal{P} : d(v_p, v_r) < d(v_p, v_q)$$

Step 3: Each node p initializes an internal variable a_p with its distance to its successor: $a_p = d(v_p, v_{succ(p)})$. It is easy to see that in the absence of failures, $\sum_{p \in \mathcal{P}} a_p = M$.

Step 4: The AVERAGE protocol is run on the a_p values; at the end of the execution, each of the n nodes knows an approximation of the average $(\sum_{p \in \mathcal{P}} a_p)/n = M/n$. Given that M is known, each node can easily derive n.

Failures do not require any special handling. If a node p disappears during the execution of the protocol, T-MAN will still exchange the pair (p, v_p). In this way, nodes will learn about their closest neighbors anyway – irrespective of whether they have crashed or not. In AVERAGE the initial values are already normally distributed around the average – so the loss of any of them will have little impact. This informal claim will be confirmed by the experimental evaluation. Once an execution is concluded, the protocol can be easily restarted to get a more recent estimate of the size.

III. EVALUATION

Our protocol has been evaluated using the event-driven version of Peersim [17], with a transport layer that emulates end-to-end delays based on the traces of the King data set [18]. Each experiment is repeated 50 times with different random seeds. When graphically feasible, individual experiments are displayed as individual dots; when too many dots overlap, we separate them with a small random translation on the x-axis (thus creating a "cloud effect").

Parameter	*Value*	*Range*
Size	2^{16}	2^{10}–2^{18}
δ_c	1s	0.1s – 1.0s
$msgsize$	10	2 – 20
$precision$	10^{-5}	10^{-3} – 10^{-8}
$inactive$	3	2 – 8

The main simulation parameters defined in Section II are listed here. Default values are shown; in each experiment, all of them are fixed apart from one or two which are varied in the specified range. To allow for reproducibility of our results, code and configuration files can be found here: http://peersim.sf.net/code/tsize.tgz.

A. Evaluation criteria

The following metrics are used to assess the protocol.

Error: The quality of the estimate, measured as the absolute difference between the actual system size and the estimate, reported as a percentage over the actual system size.

Convergence time: The speed to which the desired level of error is reached, measured as the number of seconds that passes from the beginning of the protocol to the time at which all nodes stop sending messages.

Overhead: The total amount of traffic generated by T-MAN and AVERAGE. The actual value is computed assuming the use of UDP, as TCP would be overkill for sporadic gossip contacts; we assume that each node descriptor in T-MAN is composed

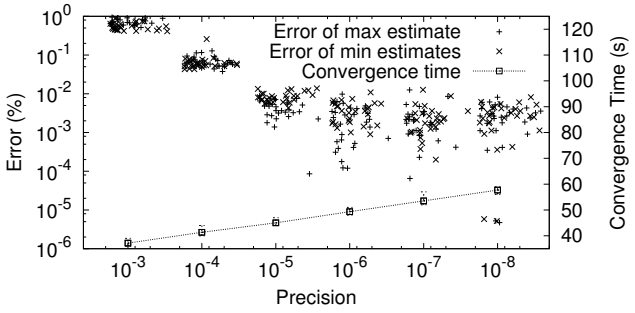

Fig. 2. Evaluation of parameter $precision$.

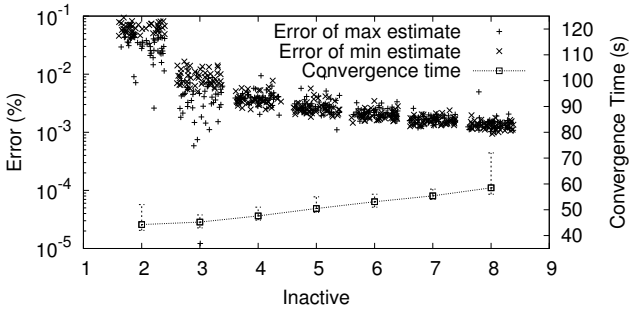

Fig. 3. Evaluation of parameter $inactive$.

of 12 bytes (64 bits for the random identifier and 32 bits for the IP address); thus, the total size of each T-MAN message is $20+12*msgsize$ bytes. Each message in AVERAGE is composed of just one 64 bit value, so the total size is $20+8=28$ bytes.

B. Failure-free experiments

The first set of experiments is meant to evaluate how the error can be reduced by varying $precision$ and $inactive$, in order to fix them for the subsequent simulations. In Fig. 2, $precision$ varies between 10^{-3} and 10^{-8}, while in Fig. 3, $inactive$ varies between 2 and 8. As expected, the error level (shown as a percentage on the size of the network in the figures) decreases in correspondence of larger values of $inactive$ and small values of $precision$; unfortunately, the convergence time grows linearly, so we fixed $precision = 10^{-5}$ and $inactive = 3$ as a trade-off between our metrics. Note that with these values, the error is around 0.01%.

In the second set of experiments, we want to test how overhead and convergence time are related to each other. Fig. 4 has been obtained by varying two parameters, $msgsize$ between 2 and 20, and δ_c between 0.1s and 1.0s. By plotting the convergence time on the x-axis and the overhead per node on the y-axis, we have been able to highlight the trade-off between these two metrics - the faster you want to go, the larger overhead you have to pay. To match dots with their parameter setting, the same set of experiments is plotted twice with different gray-scale coding. On the top figure, the coding shown on the right bar corresponds to the cycle length; the smaller the cycle-length, the faster the speed (as expected). But note that the smallest simulated cycle length (0.1s) is

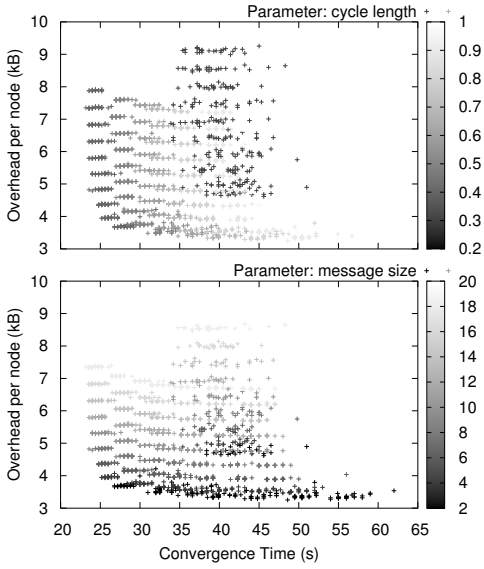

Fig. 4. Trade-off between overhead and convergence time

not shown, because none of the experiments have been able to converge in less than 200 cycles (the time limit of our simulations); furthermore, even 0.2s is not a good choice, given that the associated dark dots are scattered between 35 and 45 seconds. On the bottom figure, the coding corresponds to the message size; small message sizes tend to reduce the total overhead, but not the very small ones. In the rest of our experiments, we selected two fairly conservative values, i.e. $\delta_c = 1$s and $msgsize = 10$.

The log-log plot of Fig. 5 shows the scalability of T-SIZE and compare it with AVERAGE. T-SIZE scales logarithmically for sizes included in $[2^{10}, 2^{18}]$, and the convergence time scales better with T-SIZE rather than AVERAGE; the price to be payed is a larger (but still reasonable) overhead.

C. Robustness

We are particularly interested in evaluating these metrics in an environment subject to churn and message loss; our protocol has proven to be extremely robust in these cases. Fig. 6 show the behavior of the system under a disruptive scenario where up to 1% of the nodes leave/crash at each second. This corresponds to an expected lifetime of 99s – an insane level of churn, much larger than measured churn levels which are around 0.01%. Still, our protocol manages to keep the error below 7%, with most of the dots below 2%. Compared with AVERAGE, we note a strong error reduction.

Fig. 7 shows the behavior of the protocol in case of message losses. The protocol is sensitive to high levels of losses; this is because the loss of the reply message in an aggregation exchange causes an asymmetric update of the local values: one is changed, the other not. Nevertheless, for limited amounts of message loss (less than 5%), the protocol remains adequately accurate and its behavior is better than AVERAGE, which performs fairly bad even with small levels of message losses.

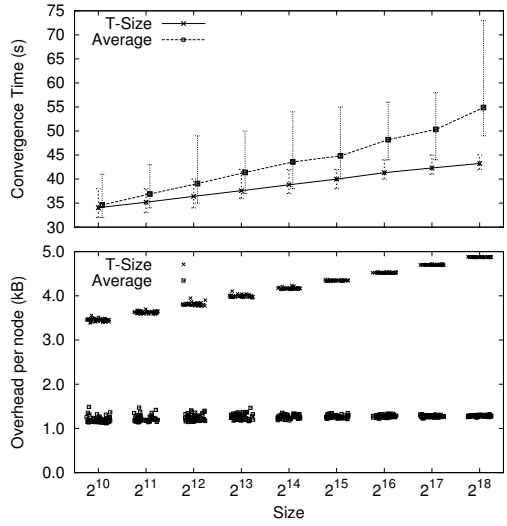

Fig. 5. Scalability

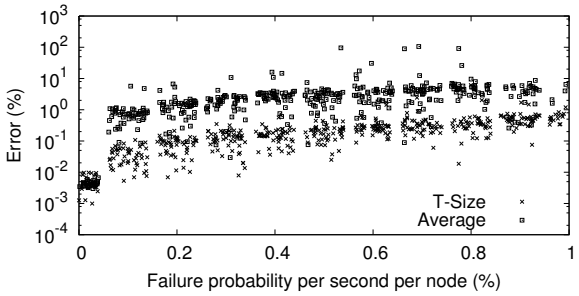

Fig. 6. Accuracy under churn.

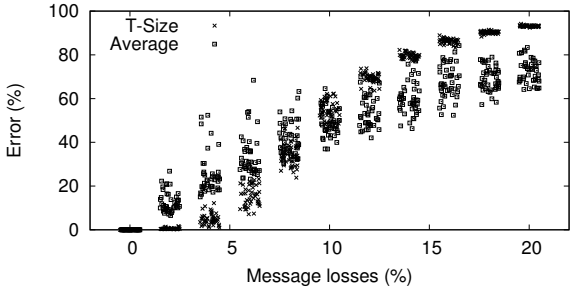

Fig. 7. Accuracy under message losses.

IV. Discussion and Conclusions

Epidemic protocols for peer counting have been proposed by Jelasity et al. in 2005 [11]; the idea was to use AVERAGE, initializing all nodes to 0, apart from one node set to 1 (the initiator). The network size n could be easily derived from the computed average $1/n$. The problem of this approach was its sensitivity to failures; in the first phases of the computation, the failure of the initiator or its neighbors nodes could easily double the estimate. Furthermore, the initial estimate at the nodes were exceedingly inaccurate. T-SIZE is more robust to failures and the initial estimates have the same expected error. This is because no node is more important than another.

While T-SIZE is an important improvement over state-of-the-art, we believe that it could further be improved by making it continuous – i.e. able to continuously provide the estimate without periodic restarting. This is the subject of future work.

References

[1] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek, "Bandwidth-efficient management of DHT routing tables," in *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*. Boston, MA: USENIX, May 2005.

[2] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. on Networking (TON)*, vol. 11, no. 1, pp. 17–32, 2003.

[3] P. B. Godfrey and I. Stoica, "Heterogeneity and Load Balance in Distributed Hash Tables," in *Proceedings of the 24th Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM'05)*, Miami, FL, Mar. 2005, pp. 596–606.

[4] A. Rao, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, ser. LNCS, vol. 2735, Berkeley, CA, 2003, pp. 68–79.

[5] A. J. Demers *et al.*, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing Systems (PODC'87)*, 1987, pp. 1–12.

[6] A. Fernandez, V. Gramoli, E. Jimenez, A.-M. Kermarrec, and M. Raynal, "Distributed slicing in dynamic systems," in *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS'07)*. Toronto, Ontario, Canada: IEEE Computer Society, 2007.

[7] A. Montresor and R. Zandonati, "Absolute slicing in peer-to-peer systems," in *Proceedings of the 5th International Workshop on Hot Topics in Peer-to-Peer Systems (HotP2P'08)*, Miami, FL, Apr. 2008.

[8] E. L. Merrer, A.-M. Kermarrec, and L. Massoulié, "Peer to peer size estimation in large and dynamic networks: A comparative study," in *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, 2006, pp. 7–17.

[9] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. J. Demers, "Decentralized schemes for size estimation in large and dynamic groups," in *Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA'05)*, 2005, pp. 41–48.

[10] L. Massoulié, E. L. Merrer, A.-M. Kermarrec, and A. J. Ganesh, "Peer counting and sampling in overlay networks: random walk methods," in *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing (PODC'06)*, 2006, pp. 123–132.

[11] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 1, pp. 219–252, Aug. 2005.

[12] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*. New York, NY: ACM Press, 2002.

[13] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," in *Proceedings of the ACM SIGCOMM 2004 Symposium on Communication, Architecture, and Protocols*. Portland, OR: ACM Press, March 2004, pp. 353–366.

[14] K. Horowitz and D. Malkhi, "Estimating network size from local information," *Inform. Process. Lett.*, vol. 88, pp. 237–243, 2003.

[15] M. Jelasity, A. Montresor, and O. Babaoglu, "T-Man: Gossip-based fast overlay topology construction," *Comput. Netw.*, 2009, to appear.

[16] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, p. 8, 2007.

[17] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," `http://peersim.sf.net`.

[18] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proceedings of the Internet Measurement Workshop (SIGCOMM IMW)*, 2002.