Efficient and Secure Decentralized Network Size Estimation*

Nathan Evans[1], Bartlomiej Polot[2], and Christian Grothoff[2]

[1] Symantec Research Labs
nathan_evans@symantec.com
[2] Free Secure Network Systems Group
Network Architectures and Services
Technische Universität München
{bart,grothoff}@net.in.tum.de

Abstract. The size of a Peer-to-Peer (P2P) network is an important parameter for performance tuning of P2P routing algorithms. This paper introduces and evaluates a new efficient method for participants in an unstructured P2P network to establish the size of the overall network. The presented method is highly efficient, propagating information about the current size of the network to all participants using $O(|E|)$ operations where $|E|$ is the number of edges in the network. Afterwards, all nodes have the same network size estimate, which can be made arbitrarily accurate by averaging results from multiple rounds of the protocol. Security measures are included which make it prohibitively expensive for a typical active participating adversary to significantly manipulate the estimates. This paper includes experimental results that demonstrate the viability, efficiency and accuracy of the protocol.

Keywords: Peer-to-Peer, protocol design, network security

1 Introduction

Individual peers participating in unstructured networks, such as Peer-to-Peer (P2P) networks, ad-hoc wireless networks and sensor networks, can benefit from knowing the size (total number of participants) of the network. Peers in unstructured P2P networks which know the network size can make intelligent decisions with respect to content replication, message routing and forwarding and the overall cost of operations. Additionally, nodes in a sensor network can use such data to gauge the health of the overall network, calculate on/off time to save energy, selectively route messages, and generate alerts.

This paper describes the design, implementation and experimental results of a protocol that provides all peers in a structured or unstructured P2P network with an accurate estimate of the total number of peers in the network. The

* This technical report is an extended version of a paper with the same title published in "NETWORKING 2012 — 11th International IFIP TC 6 Networking Conference Proceedings Part I" (LNCS 7289), pages 304–317.

primary motivation for our work are the requirements of various P2P routing protocols [6,8,15]; these protocols explicitly require a network size estimate to tune parameters responsible for routing, path selection and content replication. Also, studies about deployed P2P netwoks [3] could benefit greatly from knowing a good estimate of the analyzed network.

The focus of our design is to provide security in the context of an open and completely decentralized network architecture. While it would be possible to strengthen the security of our design with trusted centralized services — for example by preventing a Sybil-attack with a centralized registration requirement — our design does not require a centralized authority and is intended to provide security in the presence of actively participating adversaries. A key difference to existing proposals is that in our design there are no peers with special roles in the process; this eliminates the possibility of malicious peers abusing such roles.

A central goal of our design — which is generally not satisfied by many other network size estimation algorithms [2,13,17] — is that all peers are supposed to participate in calculating a network size estimate at roughly the same time and obtain the same result. This is achieved by a controlled flood of the network with the size estimation information, costing $O(|E|)$ messages per round. In practice, the constant factor is typically between one and two; in other words, the algorithm can be expected to only generate $|E|$ messages per round.

The basic idea behind our algorithm is to flood the network with the identity of the peer whose identity is closest to a particular key T. Each peer's identity is generated when the peer starts the first time. The key T is not chosen by any peer but instead is generated from the start time S of the current round. Despite using time, we specifically do not assume that the clocks in the network are closely synchronized; our protocol ensures that in the worst case individual peers with significant clock skew only cause a bounded amount of additional network traffic.

Our protocol considers many other important networking issues as well. The protocol is very efficient as it requires only $O(1)$ state per peer and does not require peers to establish new connections (we assume the network graph is connected). The amount of work required by each node is based only on the number of edges of the node, so the load between peers is typically balanced. Given that our protocol floods the network with size estimation information, peers randomly delay messages to avoid spikes in network traffic. Finally, our design handles network churn well, and allows the system designer to trade-off computational efficiency for security and bandwidth for accuracy.

The remainder of the paper is structured as follows. In Section 2, we present related work on methods for network size estimation for P2P networks. Section 3 then presents our protocol. In Section 4 we analyze the security aspects of the protocol. Experimental results obtained using large-scale emulation are given in Section 5.

2 Related Work

Algorithms for estimating the size of a P2P network can be categorized into algorithms for structured overlays, which typically exploit statistical properties of an existing routing table from a DHT, and algorithms for unstructured overlays, which make no assumptions about the structure of the underlying network.

2.1 Network Size Estimation for Structured Overlays

Structured overlays construct routing tables at each peer according to particular rules that enable efficient routing of messages to the peer with the "closest" identifier with respect to a given key [18, 21]. In these structured overlays, the distance to the nearest neighbors in the routing table can be used as a first network size estimate as it correlates with the network size [20].

As node identifiers are often not perfectly uniform, searching the structured overlay for the closest node to various randomly selected keys can be used to get accurate network size estimates [20]. Given a DHT routing algorithm with a typical cost of $O(\log n)$, network size estimation for all nodes using this method would be $O(n \log n)$. When compared to the method presented in this paper, a key disadvantage of existing methods for structured networks is that they rely on the security of the underlying routing algorithm; actively participating malicious nodes have thus the potential to significantly skew the network size estimate. Furthermore, for any of the structured methods that we are aware of, different nodes will virtually always compute somewhat different network size estimates.

2.2 Network Size Estimation for Unstructured Overlays

Several algorithms for unstructured overlays are based on sampling. Examples include Sample & Collide, Random Tour and Hops Sampling. In Sample & Collide [17], each peer starts bounded random walks to sample random peers in the network and uses the collision information and the birthday paradox to estimate the size of the network. In Random Tours [17], a message tours the network until it reaches the initiator; the size estimate is then computed based on a counter in the message that was incremented by each peer on the tour. Hops Sampling [13] works by flooding the network with a message containing a hop count. Peers report back to the initiator with a probability inverse to the hop count they received. The network size estimate is then the sum over all distances of the number of replies received for a particular distance divided by the reply-probability for that distance.

As described, these methods generate results for just one peer in the network, resulting in a high amount of bandwidth used overall (assuming each peer requires an estimate). Also, different peers will have potentially significantly different network size estimates. While these approaches do not assume a particular network structure for routing, they do still make implicit assumptions about the structure of the overlay topology and may significantly underestimate the network size if the overlay topology happens to have a structure that is unfavorable to the algorithm. For example, a circular topology would result in a network size estimate of n^2 for Sample & Collide.

Other algorithms, such as Gossip-based Aggregation [9], achieve a somewhat more uniform estimate for all participating nodes at the cost of sensitivity to node failures. Gossip-based Aggregation starts with one peer setting a local state to 1 while all other peers set their local state to 0. Peers continuously connect to randomly selected peers, and exchange states in pairs. Each peer then replaces its state with the average of both values. After a predefined number of iterations, all peers are supposed to end up with a value close to $1/n$ where n is the size of the network. A method that addresses the problem of who should set the state to 1 has been proposed [23], but only works in certain structured networks and retains other shortcomings of this approach, including high vulnerability to malicious peers. In [**?**] a much more efficient gossip-based method is introduced which uses aggregation and beacons to achieve fast convergence, high precision and is able to handle churn; however, it still offers no security.

A special case is the method proposed in [2] which attempts to produce a network size estimate using only "local" information. The idea behind this algorithm is to observe the number of new neighbors discovered in a breadth-first search of the network and estimate the network size based on the growth of this function. The authors claim to obtain accurate results with a breadth-first search of depth three, which makes this a "local" method. However, the way they constructed the topologies for their experiments does not seem to properly model the structure of actual networks. We were unable to reproduce their results on other network topologies.

The accuracy and performance of various size estimation methods for unstructured networks are compared in [19] using simulation. The authors identify the Sample & Collide method as the strongest algorithm and state that it requires about 50 million messages in a random-graph topology of 100,000 nodes for an accuracy of ±4%. It should be noted that this is the overhead for an individual node to obtain an estimate; if each of the 100,000 nodes were to run the Sample & Collide protocol, it would take 5 *trillion* messages to achieve this degree of accuracy.

None of these papers mention concrete implementations or discuss security concerns. Furthermore, all of them are clearly vulnerable to malicious participants. For example, in the case of sampling-based algorithms, malicious participants can manipulate walks that pass through them (allowing virtually unbounded manipulation of the network size estimates) or achieve a significant multiplier ($O(\sqrt{n})$) to their network bandwidth in a denial-of-service attack by continuously initiating size estimation requests. Similarly, an active adversary can manipulate the exchanged values in gossip-based methods to change the size estimate in any direction.

3 Our Approach

We generate node identifiers by hashing the public key of the respective node. Node identifiers for benign nodes should therefore be statistically equivalent to random numbers from a uniform distribution. Furthermore, nodes are able to cryptographically sign messages using their respective private key.

Similar to the network size estimation algorithms for structured overlays, our network size estimation approach is based on the largest number of leading overlapping bits between any node identifier and a random key:

Theorem 1. *Let $\overline{p}$ be the expected maximum number of leading overlapping bits between all n random node identifiers in the network and a random key. Then the network size n is approximately* $2^{\overline{p}-0.332747}$.

Proof. Let X be the random variable for all n identifiers and let X_i be the number of overlapping bits for an individual random node identifier i.

The probability that a single random node identifier i overlaps with at least α bits with a random key is

$$P(X_i \geq \alpha) = 2^{-\alpha}. \tag{1}$$

Then, the probability that a single random node identifier overlaps with less than α bits with a random key is

$$P(X_i < \alpha) = 1 - 2^{-\alpha}. \tag{2}$$

The probability that the maximum number of leading overlapping bits for all n random nodes is strictly less than α is

$$P_n(X < \alpha) :=: P\left(\bigwedge_i X_i < \alpha\right) = \left(P(X_i < \alpha)\right)^n = \left(1 - 2^{-\alpha}\right)^n. \tag{3}$$

Then $E_n(X)$, the expected maximum number of leading overlapping bits between n random node identifiers in the network is:

$$\begin{aligned}
E_n(X) :=: \sum_{\alpha=0}^{\infty} \alpha \cdot P_n(X = \alpha) &= \sum_{\alpha=1}^{\infty} P_n(X \geq \alpha) \\
= \sum_{\alpha=1}^{\infty} \left(1 - P_n(X < \alpha)\right) &= \sum_{\alpha=1}^{\infty} \left(1 - \left(1 - 2^{-\alpha}\right)^n\right) \\
= \sum_{\alpha=1}^{\log_2 n} \left(1 - \left(1 - 2^{-\alpha}\right)^n\right) &+ \sum_{\alpha=\log_2 n+1}^{\infty} \left(1 - \left(1 - 2^{-\alpha}\right)^n\right)
\end{aligned}$$

Suppose n is sufficiently large such that we can use $\lim_{n\to\infty}(1-\frac{x}{n})^n = e^{-x}$. By substituting $\beta := \alpha - \log_2 n$ and $\gamma := \log_2 n - \alpha$ we then get:

$$\begin{aligned}
E_n(X) &= \log_2 n - \sum_{\gamma=0}^{\log_2 n - 1} \left(1 - 2^{\gamma - \log_2 n}\right)^n + \sum_{\beta=1}^{\infty} \left(1 - \left(1 - 2^{-(\beta + \log_2 n)}\right)^n\right) \\
&= \log_2 n - \sum_{\gamma=0}^{\log_2 n - 1} \left(1 - \frac{2^\gamma}{n}\right)^n + \sum_{\beta=1}^{\infty} \left(1 - \left(1 - \frac{2^{-\beta}}{n}\right)^n\right) \\
&\approx \log_2 n - \sum_{\gamma=0}^{\log_2 n - 1} e^{-2^\gamma} + \sum_{\beta=1}^{\infty} \left(1 - e^{2^{-\beta}}\right) \\
&\approx \log_2 n - 0.521865 + 0.854613 = \log_2 n + 0.332747
\end{aligned}$$

Thus, for sufficiently large values of n,

$$E_n(X) \approx \log_2 n + 0.332747. \tag{4}$$

□

Given Theorem 1, the key remaining challenge is thus to efficiently and securely find a closest node identifier (with distance measured in terms of leading overlapping bits) to a random key in an unstructured network.

In our design, all nodes in the network periodically participate in a global network size estimation operation at a frequency of f. Each round results in all peers learning a discrete approximation p (the number of overlapping leading bits for a particular random key) for $\overline{p}$ (the theoretically expected number of overlapping leading bits). The specific frequency f is chosen based on the expected level of network churn and the desired accuracy. f is a design parameter and fixed in the implementation. The results from the last k iterations are averaged locally by each peer to obtain an approximation $\tilde{p}$ for $\overline{p}$. A standard deviation can also be computed if an estimate for the error of the size estimate is desired. Furthermore, the current p value is used by the protocol as a parameter to (slightly) improve the performance for the next round. We will refer to the number of overlapping leading bits from the previous round as p'.

3.1 Generating a random "key"

Given a frequency f, the random target key T for each round is generated by hashing the start time S, which is the absolute UTC time at times that are zero modulo f. For example, if $f = 1h$, then a fresh key could be generated every hour by hashing "DD-MM-YYYY HH:00:00". Using this method, all peers will generate exactly the same key at (roughly) the same time. Generating the key this way has the advantage that it will be known to all peers without communication and that malicious participants cannot influence the process. However, it should be noted that while the keys satisfy the statistical properties of being random and uniform, it is trivial to compute them in advance.

Our method requires all peers to calculate the current key T at the respective start time S. The network size estimation protocol's goal is to communicate to all peers an identity I_T of a peer with the largest proximity p with respect to T. More specifically, all peers are supposed to learn one of the closest peer identities I_T between time S and time $S+f$. Given I_T, each peer can then calculate p, the average $\tilde{p}$ of the p-values from the last k rounds and finally the current network size estimate $2^{\tilde{p}-0.332747}$.

Note that p is a discrete value representing the number of leading matching bits between the key T and a peer's identity. As such, it is quite likely that many peers have identities with the same number of leading matching bits and hence the same proximity p. Our protocol deliberately ignores all bits after the first mismatch to improve performance; if multiple peers have the same proximity score, it does not matter which of these equivalent identities is propagated as they will all ultimately result in the same proximity estimate p.

3.2 Starting the Flood

Our protocol essentially floods the network with the identity of a closest peer I_T. If only the identities of closest peers are propagated, this operation would create less than $2|E|$ messages (up to two per edge in the network). The challenge is to avoid creating significantly more than $2|E|$ messages, which is difficult since in an unstructured network a peer with the closest identity I_T cannot be certain that there is no other peer that is closer to T. We address this problem by delaying the flood based on proximity.

First, each peer evaluates its own proximity x with respect to T. How close the peer is to T is then used to determine how soon the peer will initiate the flood of the network with a message claiming that he is the closest peer. We use the previous network size estimate as a guide to time the release. Specifically, given a proximity of x leading overlapping bits and a network size estimate p' from the previous round, we use the following function to determine the time when a peer starts to flood the network:

$$r(x) := S + \frac{f}{2} - \frac{f}{\pi} \cdot \arctan\left(x - p'\right) \tag{5}$$

Using this function, if the peers proximity x is equal to the proximity of the last iteration, the peer floods the network at time $S + \frac{f}{2}$. If the peer's proximity is 0 bits, the peer floods the network close to time $S + f$, which is at the end of the time interval for the current round; if the peer's proximity is significantly higher than p', the peer floods at the beginning of the round, which is close to time S. It should be noted that since $\lim_{x \to p'} \frac{\partial r}{\partial x}(x) = 1$, Equation 5 maximizes the difference between release times for nodes with proximities that are close to the previous number of overlapping bits p' and as such minimizes the chance of two peers releasing floods for different network size estimates around the same time — assuming the network size estimate did not change significantly.

3.3 Processing the Flood

Peers that receive the resulting network size estimation messages first perform a series of validation steps before continuing to forward the message. First, each peer checks if a notification from a closer peer has already been received for round S. Messages with proximity scores equal to the currently known best score for the current round are simply discarded. Messages with lower proximity scores should only occur if there is significant clock skew, and are answered immediately with a message indicating the higher proximity score. If the message contains a higher proximity than what was previously known for the current round, the peer checks if the proximity p of the given message justifies receiving it at the current time. If not, further processing is delayed until the local peer's time is past $r(p)$. Finally, before forwarding and further processing, the format of the message is validated (this is discussed in more detail in Section 3.5).

Assuming the message validates, the peer then proceeds to forward it to all of its neighbors. For each neighbour, the message is forwarded with a peer-specific random delay. If a peer receives a message with an equivalent proximity score during the delay, the transmission is canceled. As a consequence, the delay helps to both avoid an explosion of messages on the network in a tiny amount of time, and to improve the chances of traversing each edge in only one direction per link (as it decreases the chances of equivalent messages being sent in both directions at the same time).

The permissible delay L is calculated using the time difference between $r(p)$ and $r(p-1)$ divided by an estimate of the network's diameter. The network diameter D is estimated using the maximum of the hop counters in the network size estimation messages from the previous k iterations. For each neighbor, the peer then applies a delay chosen uniformly at random from the interval $[0, L)$ where L is defined as

$$L := \frac{r(p-1) - r(p)}{D}. \tag{6}$$

As a result, each peer in the network is expected to receive a proximity notification with proximity p before any peer with notifications for proximity $p-1$ would even begin to flood the network. Naturally, there are various causes that could increase the number of messages above $|E|$ in a real-world network; for example, different system times between peers, high network latencies, and peers with unusually high distances to I_T can increase the total number of messages. However, all benign peers that form a connected component are guaranteed to eventually receive I_T. Furthermore, given that the number of bits in the key is a small constant, the total number of messages can never exceed $O(|E|)$ per round.

3.4 Joining the Network

A peer that is freshly joining the network lacks results from previous rounds for network size estimation. In order to bootstrap the protocol, each peer starts with a network size estimate based on its own key in relation to the key T

from the previous round, and a network diameter estimate of one. Whenever a connection between two peers is established, they exchange the network size estimation result from the previous round (and the current round if their local time is past $f(p)$). As a result, all nodes can always be expected to use the same value for p' in Equation (5).

If, as a result of this exchange, one side has to increase its network size estimate for the previous round, it floods its neighbors with the result from that round as well. If information about the previous round is flooded in this fashion, the artificial delay limit L is set to an implementation-defined small constant (we use 50 ms). Note that only flooding of information about the current round S and the previous round $S - f$ is permitted. As a result, all nodes can always be expected to use the same value for p' in Equation (5).

It is not true that all peers will calculate the same value for L based on Equation 6. The hop counters in the flooded messages can clearly differ between peers, potentially resulting in a different estimate for D. Furthermore, since the network size estimate given to applications is based on the last k values, application-level network size estimates may differ between established and recently joined peers as well. If the latter is unacceptable, peers establishing connections could propagate the last k network size estimates.

3.5 Proof of Work

The presented design is vulnerable to an adversary that creates fake identities (Sybil attack [4]). Such an adversary could create identities that are "close" to the respective key for each time $S + \mathbb{Z}r$. By flooding the network with the respective messages at the right time, the adversary can then make the network appear to be larger than it is.

Our design defends against this attack by requiring a proof of work [22] for the identity of the peer as part of the network size estimation message. Specifically, we require the originator to produce a value with a W-bit hash collision with the peer's identifier, and a cryptographic signature to demonstrate that the identifier was derived from a valid public-private key pair.

The complete message format for the network size estimation messages is described in Figure 1.

4 Security Analysis

For our security analysis, we assume that an active adversary is participating in the P2P network. The adversary is allowed to control a certain percentage of colluding malicious nodes in the network. Individual malicious and benign nodes are assumed to have the same amount of computational resources; all nodes are assumed to have sufficient bandwidth to participate in the protocol in the absence of an attack.

We can imagine three different high-level goals an adversary may pursue with an attack. First, an adversary may try to cause nodes to significantly underestimate the size of the network. Second, an adversary may try to cause nodes

Offset	Contents
0	Message header magic code
4	Hop-Count (updated at each peer)
8	Signed data header magic code
16	Time S of the round
24	Proximity p in bits
28	Public key (2048 bit RSA)
288	Proof-of-work
296	Signature (signing bytes 8–295)

Fig. 1: Message format for the network size estimation messages. The proof-of-work is a 64-bit number such that the hash of the concatenation of the public key and the proof ends with W bits of zeros. The claimed proximity p is redundant (it could be calculated from hashing the public key and S). However, by including p this moderately expensive cryptographic hash calculation can be avoided entirely if p is not larger than the current local estimate known to the peer for time S. A signature is included to ensure that the public key itself is well-formed and derived from a private key.

to significantly overestimate the size of the network. Finally, an adversary may want to use the protocol for a denial-of-service attack where the P2P network uses significantly more traffic for network size estimation, possibly causing other components of the system to be left with insufficient bandwidth.

The best method for an adversary to cause peers to underestimate the size of the network is to not participate in the protocol. If the adversary controls $X\%$ of the network, that will cause the protocol to underestimate the size of the network by $X\%$. Furthermore, if the adversary is able to control an ϵ-separator of the network graph [10, 11, 16] (removing an ϵ-separator from a graph reduces the size of the largest remaining connected component to ϵn), then the overall network size estimate would be reduced to less than ϵn for all nodes in the network. Given that in all of these cases the network size estimate would correspond to the size of the network after the removal of the adversaries' nodes, this attack is not particularly disruptive in relation to the strength of the adversary. Thus, an adversary cannot make the network appear significantly smaller than it is.

If the adversary wants to make the network look larger by M nodes, it needs to first compute (and store) M public-private key pairs. Then, at every time interval f, the adversary needs to compute collisions costing an additional $O(2^W)$ to generate the required W-bit collision. Actually joining the network with M "fake" peers is not required. If W is chosen so large that the adversary cannot solve the problem at frequency f, it is still possible for the adversary to cause an increase in the network size estimate by solving the problem every $c \cdot f$ (for an appropriate choice of c based on the adversaries computational resources), which would still affect the computed medium-term averages computed for subsequent intervals. Using such an attack, an adversary can make the network appear significantly larger than it is, as long as the adversary has access to sufficient computational resources.

Finally, for a denial-of-service attack, an adversary would first generate additional identities and generally perform the same steps as for increasing the estimated network size. Now, suppose the adversary has created m identities that are closer to the current key than the closest actual peer in the network by $1 \dots m$-bits respectively. Then, just after the identity of the peer that is actually closest to the key has been broadcast to the network, the adversary can cause m additional broadcasts by transmitting its m "fake" identities in order of increasing proximity to the key. Each time, the network will presume that a closer peer was "late" with its transmission (for example, due to clock-skew or network latency) and broadcast the update. If the network is already of size n, the expected one-time cost for the adversary to create m such identities is $O(n2^m)$; the attack then requires an additional $O(m2^W)$ operations for the hash collisions at frequency f. Therefore, if we neglect the high one-time cost of computing identities, the adversary can cause $|E|$ traffic on the network at the cost of $O(2^W)$ computations.

Analytical Worst-Case Analysis

The following scenario describes the theoretical worst case in terms of bandwidth consumption by the protocol. Without loss of generality, suppose a 512-bit hash function is being used. Then, the worst-case network would for $\mu \in [1, 512]$ have exactly one peer with μ matching bits with the target key T (in each round); all other peers in the network would have zero matching bits. The peers that do have matching bits should be connected to the main network via a long chain (with larger distances for peers with larger μ), causing the network diameter D to be large. (As a result, the algorithm will calculate $L \approx 0$.) Peers with μ matching bits should furthermore have (or pretend to have) a late system time that causes them to transmit effectively at time $S + \mu\epsilon$; all other peers have fast clocks that cause them to accept any message at any time, causing 512 network-wide floods per round.

creating a total of $1024 \cdot |E| \in O(|E|)$ transmissions. Note that this scenario covers the worst-case and includes an adversary with infinite computing power and full control over the network topology.

5 Experimental Results

We have implemented the presented protocol in the GNUnet P2P framework[3], and evaluated the behavior of the proposed protocol using large-scale emulation [7].

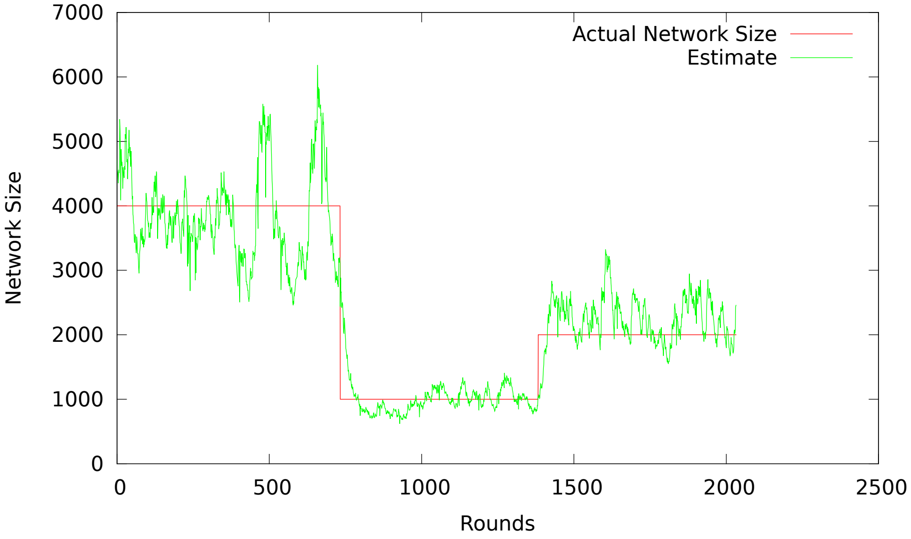


Fig. 2: Network size estimates as actually observed by a node in relation to the (changing) total network size over time for a random graph topology. All nodes arrive at the same estimate, except for nodes that recently joined the network.

5.1 Adaptivity to Churn

To evaluate the network size estimation quality under churn, we show the network size estimate based on an average of the previous 64 rounds. Figure 2 shows the evolution of the network size estimate for a random graph topology [5, 7] with approximately 10 edges per peer. The estimate is calculated with a weighted average over the last 64 readings. It should be noted that the shape of the network topology has no impact on the size estimate. The experiment was started with an initial network size of 4,000 nodes for 640 rounds. Then, we decreased the network size to 1,000 nodes for 640 rounds and finally increased it to 2,000 nodes for another 640 rounds.

5.2 Precision vs. Number of Iterations

The number of rounds used to calculate the result has an impact on the precision of the estimate. The trade-off between more measurements and the resulting precision is plotted in Figure 3. Precision is measured as $|\tilde{p} - \overline{p}|$. Averaging over four rounds gives results with a standard deviation of one. As the network size is calculated as $2^{\overline{p}-0.332747}$ (Theorem 1), a standard deviation of one means that the network size estimate is in an interval between half and double the actual

[3] https://gnunet.org/svn/gnunet/src/nse/

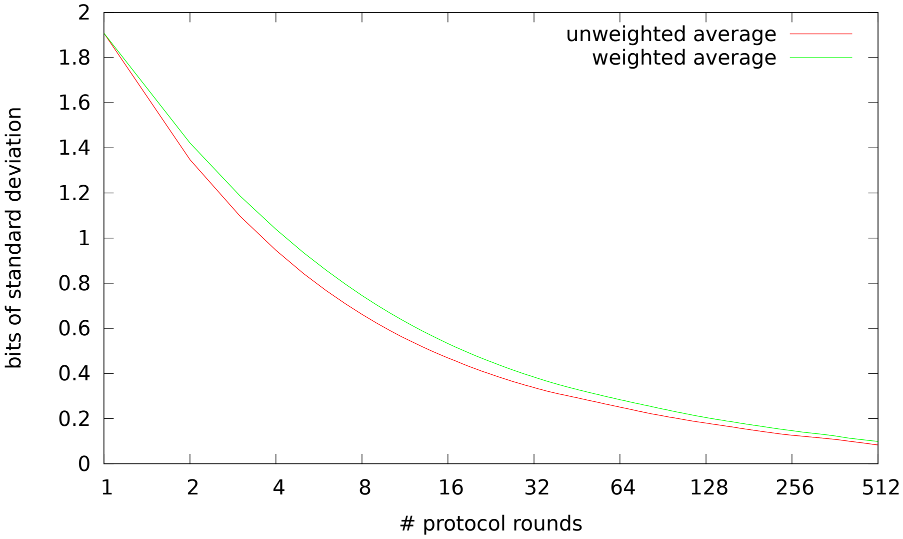


Fig. 3: Trade-off between the precision of the network size estimate vs the number of rounds used to calculate $\tilde{p}$. Naturally, this plot assumes that the network size does not change during the measurement.

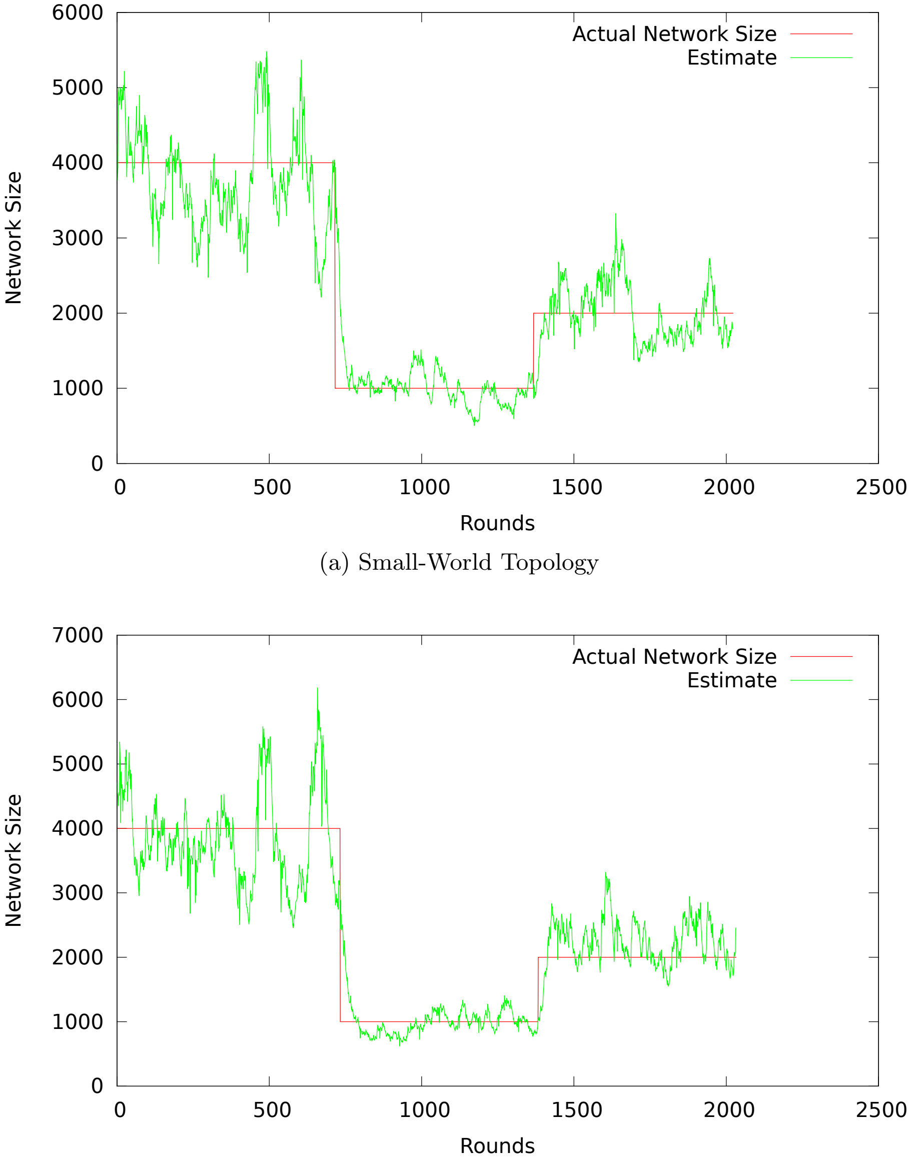
network size 68% of the time and between a quarter and four times the actual network size 95% of the time. The 64 rounds we used for Figure 2 correspond to a standard deviation of under 0.3. This means that 95% of the time the network size estimate is accurate up to a factor of ≈ 1.5.

5.3 Impact of the Network Topology

Figure 4 shows the evolution of the network size estimate for various topologies in relation to the actual network size for a Small-World topology [7, 12] (Figure 4a) and a random graph topology [5, 7] (Figure 4b). Our experiments show that the topology has no significant impact on the result. As we are only using $k = 8$ rounds for the averaging, the results between rounds still differ widely. This is the natural cause of using the counter of a discrete number of matching bits in the exponent: while the standard deviation is typically about 1 bit, this translates to a factor of two for the range of the 68%-confidence interval for the network size estimate.

5.4 Impact of Clock Skew

Figure 5 compares network size estimates from a network of 1,000 peers with and without clock skew. The clock skew can affect the protocol if the closest peer to the key has a clock so far back that other peers discard that information



(a) Small-World Topology

(b) Random Graph Topology

Fig. 4: Average network size estimate in relation to actual network size over time for different topologies.

as too old. This would cause an underestimate of the size of the network. For these tests, we created a Small-World network with approximately 5,000 total edges. We used a network size estimate interval of 30 seconds and ran the test for 96 minutes, averaging over 64 measurements. We skewed the clocks of peers by ±30 seconds, which is a significant clock skew in light of the short network size estimation interval of $f = 30s$. The results from Figure 5b generally show similar results in the size estimate, with minimum estimates around 600 peers. The experiment with skew has more data points. This is due to the multiple notifications per round that happen when peers receive delayed messages.

5.5 Network-Wide Agreement (under Churn)

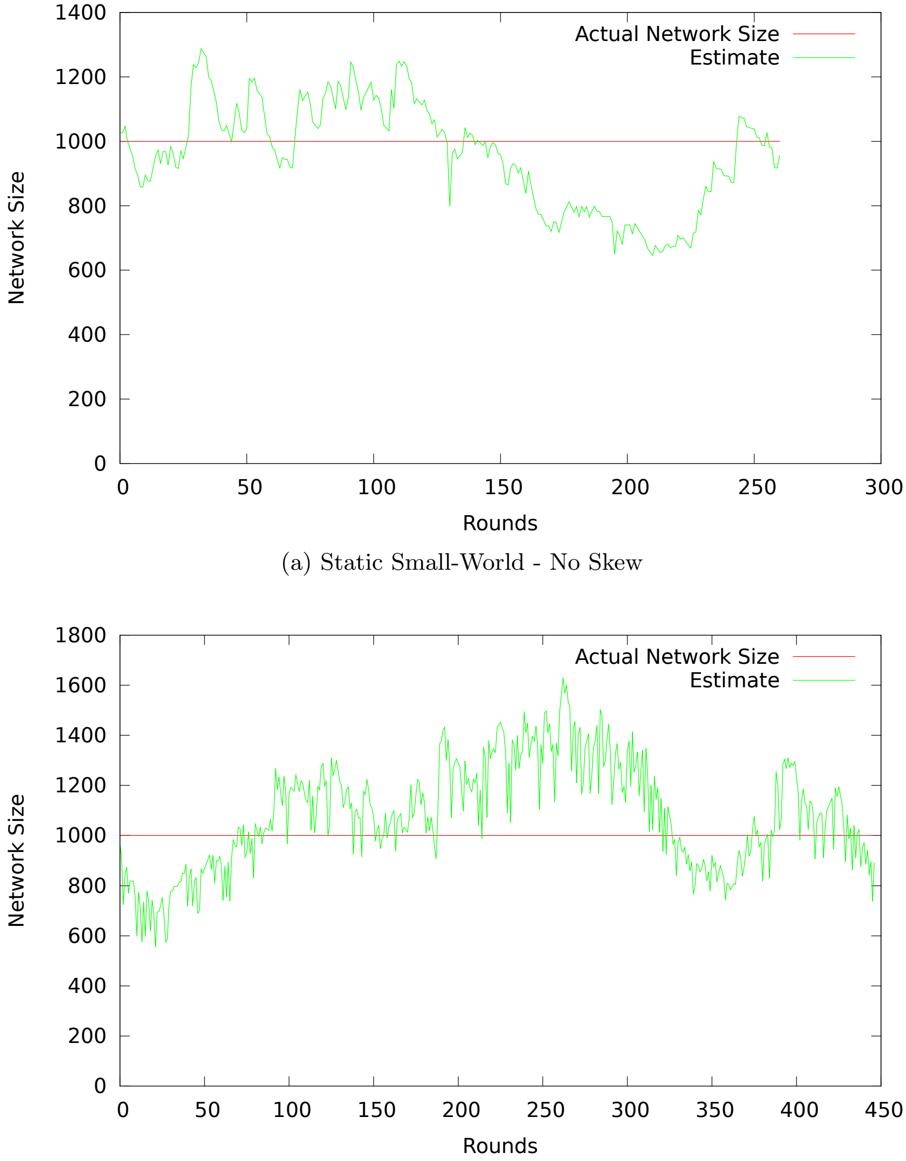
Figure 6 plots each of the individual estimates calculated by the various peers in an experiment with clock skew and churn. While the protocol guarantees that all peers within the same connected component will converge to the same network size estimate, this is not true for peers that just joined the network, thus lack some of the previous k proximity values and hence will arrive at a different average. Furthermore, different proximities and clock skew can cause some disagreement between peers, especially early in the experiment where the diameter has not yet been established.

5.6 Traffic Cost

Figure 7 shows the amount of network traffic generated by the protocol for each round in relation to the number of edges in the network. As before, the network size is cut to a fourth in round 10 and is doubled in round 20. Given the small amount of bandwidth required, it is clearly possible to run this protocol with small values of f and large values of k in cases where precise network size estimates are required.

A second round of traffic measurements was performed to demonstrate the impact of clock skew on network bandwidth. For this measurement, the local times of the different peers were desynchronized; specifically, the local times at the different peers were offset from the actual time using a triangular distribution with a maximum deviation of one minute.

Finally, we evaluated the effect of randomly delaying messages. If we disabled random delays, the overall number of messages exchanged in the network increased by 25% (in a network with 1,000 peers in a small-world topology with 18,000 edges). This demonstrates that the random delays reduce the number of cases where an edge is traversed in both directions at the same time. Globally, the protocol also ensures that the network load is reasonably spread out over time. Figure 9 shows the maximum number of messages generated by the entire network in 1 ms intervals during the experiment. The shape of the first spikes is different because the network was just started and peers begin their transmissions starting from very diverse initial network size estimates. The second spike is significantly offset from the typical period because the initial estimate (based only on the first round) is far from the typical average for the network. Without the random delays, spikes in traffic for this network could theoretically increase by a factor of 180.



(a) Static Small-World - No Skew

(b) Static Small-World Topology - Skew

Fig. 5: Data showing effect of clock skew on average network size estimates for a static network of 1,000 peers. The differences between Figure 5a (without skew) and Figure 5b (with skew) are, as expected, small.

5.7 Accuracy of Approximations in Theorem 1

In the proof for Theorem 1 we made an approximation that is valid if “n is sufficiently large”. However, what constitutes a sufficiently large n in practice is

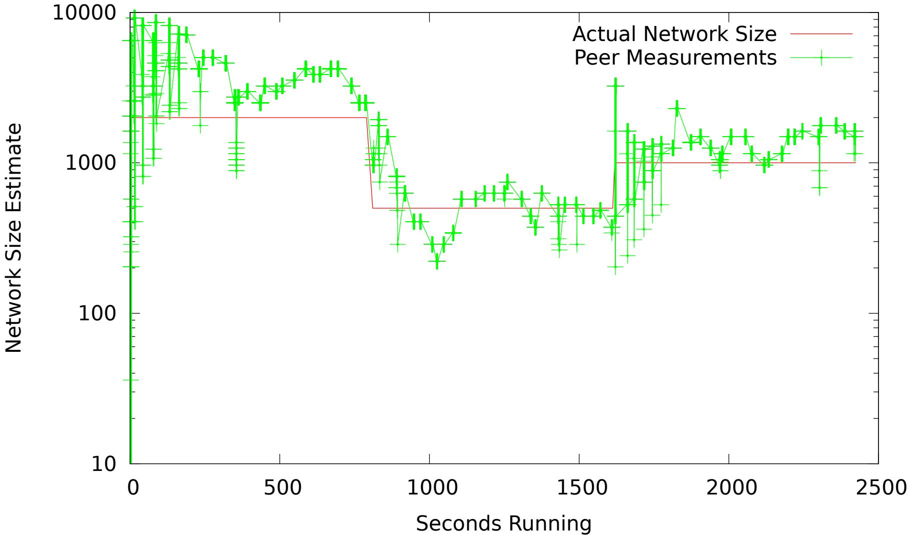


Fig. 6: This figure plots the results of all peer's network size estimates during the course of a single experiment. While clock skew causes more estimates, disagreement among peers is minimal.

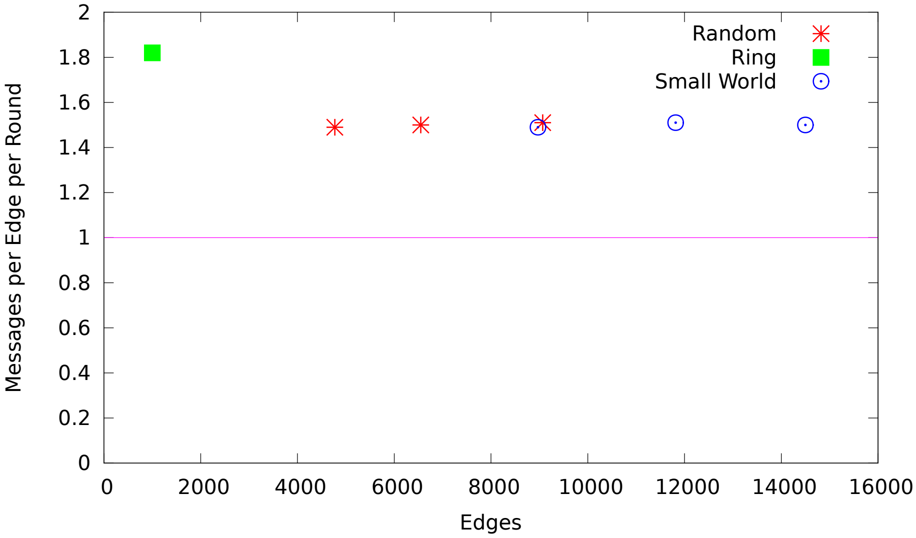


Fig. 7: Number of messages exchanged in relation to the number of edges in the network.

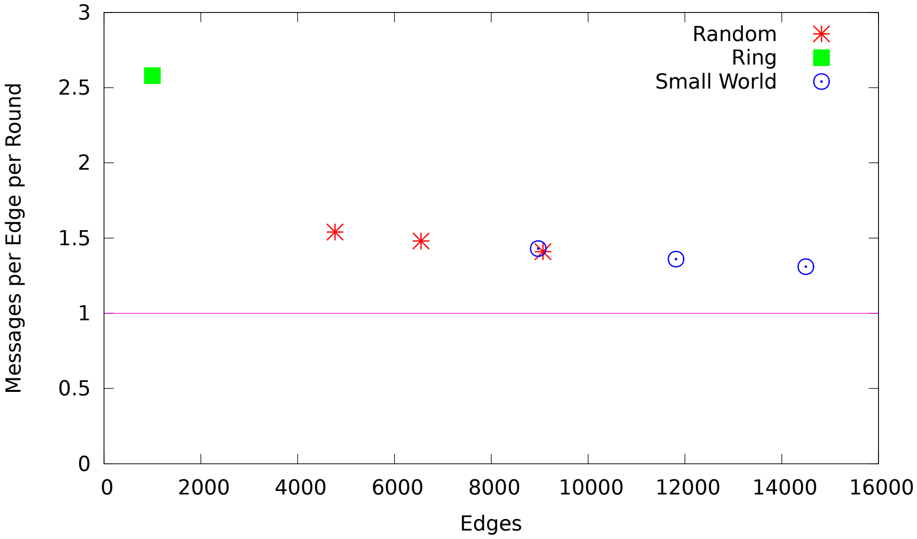


Fig. 8: Number of messages exchanged in relation to the number of edges in the network if the peers are suffering from clock-skew.

not obvious. Figure 10 shows the results of a simulation that determined $\tilde{p}$ from 50,000 rounds for networks of size $n \in [1, 2^{24}]$. The difference $\tilde{p} - \log_2 n$ quickly converges to the constant calculated in Theorem 1 (0.332747). It should be noted that even with 50,000 rounds the values for $\tilde{p}$ still exhibit some visible fluctuation. Figure 10 shows that for a reasonable number of rounds of measurement ($\leq 50,000$), the "sufficiently large values of n" are values larger than 2^5.

5.8 Comparison with other Methods

A naive version of gossiping using randomly chosen edges with 1,000 peers in ring topology takes about 70 million interactions to get all 1,000 peers within factor of two for size estimate (compared to about 4 rounds with 1,000 interactions each for our protocol). For a 2D torus, naive gossiping still took 180,000 interactions. For a random graph, small world or clique, only about 10–20 thousand interactions are required (comparable to the protocol presented in this paper); thus Gossip efficiency is somewhat dependent on topology and not as efficient in some cases. A much more efficient and precise gossiping protocol is presented in [**?**], offering high precision at a cost comparable to the approach presented in this paper. However, their protocol is more complex, offers no security and has not been implemented.

Sample & collide showed to be quite dependent from the topology in our simulations, as the algorithm is very dependent on a good sampling method and the topology affects the sampling heavily. A clique of 1,000 nodes provided good estimations, although at a cost of 40,000 messages per round per peer with a

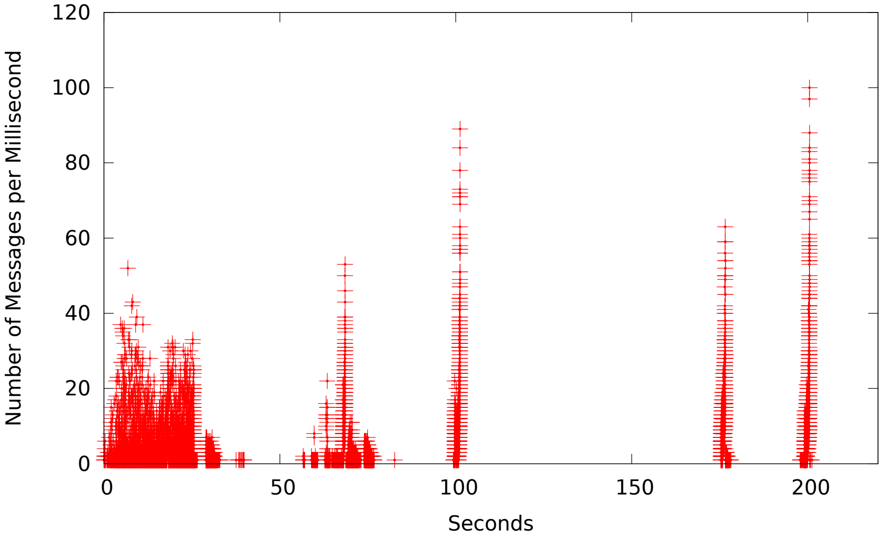


Fig. 9: Maximum number of messages received per millisecond globally in a network with 1,000 nodes and 18,000 edges. The result seen here is that while spikes exist as the nearest peer in a round sends a message, the total number of messages received is spread out over time due to the built-in message delays. The peaks are somewhat lower initially (while at the same time generating more traffic overall) due to the lack of previous network size and diameter estimates.

parameter $T = 1$ (the higher the degree of the nodes, the longer each sample message has to travel). After 16 rounds the results are within a factor of 2 of the real size. If every peer in the network would be to obtain an estimate, the traffic would amount to 64 million messages, compared to 64,000 messages with our algorithm.

On more restricted topologies the precision was not as good, although the traffic was also lower, due to the lower average degree of the nodes. On a 1000 nodes random topology with 5000 edges the algorithm showed a tendency to underestimate the size, converging to an estimate of under 900 even averaging over 100,000 rounds. The traffic generated was 1,600 messages per peer per round. To obtain an estimate for every peer would require to transmit 2,5 million messages, compared to 320,000 using our approach.

In our experiments, hops sampling gives an accurate (within 10%) result with a single round (for random graph with 1,000 nodes and 10,000 edges). However, already for this small graph hops sampling takes always significantly more than 10 million messages regardless of parameter choices and, compared to the method proposed in this paper, also requires more state to be kept by all participating nodes. Given that for a 1,000 node network hops sampling would thus have to be compared in precision and cost to the algorithm presented in this paper averaged

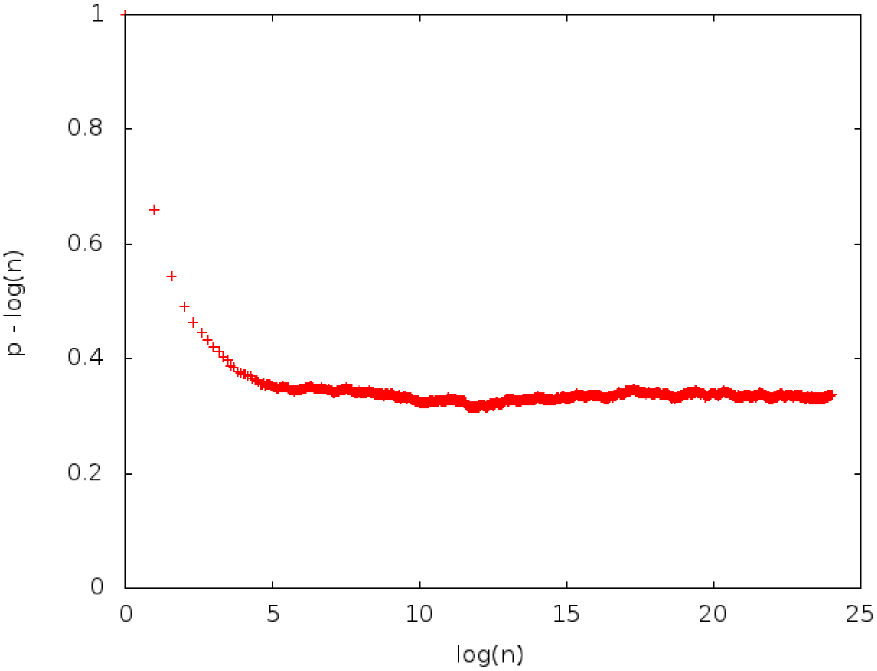


Fig. 10: Differences observed between $\log_2 n$ and the average observed value for p over 50,000 iterations in relation to the network size. The average difference was $0.33 \approx 1/3$. Since peer-to-peer networks smaller than $2^5 = 32$ peers are not really relevant for network size estimation techniques, we use a uniform correction of $1/3$ to compensate for the observed difference when estimating n from $\overline{p}$.

over 1,000 rounds, it is still not competitive in performance. Naturally, compared to hops sampling the network traffic of the protocol described in this paper is also more evenly distributed. Hops sampling is also much more vulnerable to message loss compared to other methods.

Finally, methods focused on individual estimates are prone to be abused to perform DoS attacks on the network, allowing a malicious participant to multiply its bandwidth by several orders of magnitude.

6 Discussion

The security of the presented scheme is partially based on the assumption that the adversary cannot calculate W-bit collisions frequently. However, in practice, we must expect that a dedicated adversary may use specialized hardware such as GPUs to achieve significantly higher computational power than normal nodes [14]. Using a value of W that requires GPUs would exclude many “normal” users (which may not have access to properly configured modern GPUs) from participation. So instead of using bit collisions in cryptographic hash functions — the search for which is easily accelerated with GPUs — one might want to use computational puzzles that are memory-bound [1] and hence less suitable for GPU-based acceleration.

Another key issue is that the adversary can pre-compute the solution to the next round ahead of time, which makes it trivial for an attacker to parallelize the computation across multiple systems. Such pre-computations could be prevented if the key of future rounds was unpredictable. This would effectively force the adversary to either calculate the proof-of-work in an unrealistically short period of time or to pre-calculate proofs of work for each of the M "fake" peer identities. Consequently, unpredictable keys would seriously limit the effectiveness of such an attack.

However, creating and distributing an unpredictable key for the next round cheaply and securely is difficult. One approach for making the key harder to predict is to tie the key for the next round to the result from the previous round. This would require changing the protocol to propagate a unique "closest" peer identity to all peers. The protocol described herein allows for the possibility that different areas of the network determine different "closest" peers — as described, propagation stops if the number of leading matching bits is identical to that from a previously received message for the current round.

Changing the implementation to flood the entire network with the globally closest peer would be trivial; however, this has other disadvantages. Without adversaries, this change increases the expected overall communication cost since multiple peers with small differences in proximity may start to flood the network at the same time, ultimately causing many edges to be traversed many times. Furthermore, adversaries that are able to predict the target key for the next round could then efficiently generate many more "closer" node identifiers, increasing the effectiveness of denial-of-service attacks: since proximity changes would no longer be counted in leading bits, causing m rounds of broadcasts would require pre-computing m closer identifiers (at a cost of $O(nm)$). In our current implementation, for m rounds of broadcasts, $O(2^m)$ closer identifiers would have to be pre-computed (at a cost of $O(n2^m)$). Another drawback to such a design change is that an adversary with "unlimited" computational power could cause virtually unlimited rounds of broadcasts, whereas the described method , only allows small constant number of rounds to be possible.

While this change would make it much more difficult for the adversary to predict the key, it is still conceivable that the adversary may at some point successfully predict the closest key. At that point, the adversary might be able to pre-compute proofs-of-work in parallel into the future, dominate the closest peer estimate for some amount of time, and during that time have the ability to perform an exponentially more effective denial-of-service attack (due to the required global consensus on the result of the previous round).

For our implementation, we felt that this choice — making it initially harder for the adversary to make the network seem bigger vs. reduced traffic costs in normal operation and an exponentially less effective denial-of-service attack — should be made in favor of the method described in Section 3.

Naturally, in a setting that does not have to deal with the possibility of malicious participants trying to drive up the network size estimate, the entire proof-of-work, the cryptographic signature, and the public key would not be needed at all. This might, for example, be the case where developers try to get approximate usage metrics but are at the same time too concerned about leaking too much information and are thuse unwilling to keep detailed centralized records. The Tor Metrics Portal [4] is an example of such a service where approximate usage counts are actually desired and where it might be unlikely that participants would deliberately provide false information. Calculating these metrics based on Theroem 1 might thus serve the privacy-sensitive nature of the project.

7 Conclusion

We have presented the first protocol for securely and efficiently estimating the size of a P2P network. Our protocol combines proximity to a deterministic sequence of (pseudo-)random values, staggered triggering of messages and a proof-of-work component. The scheme works for structured and unstructured networks, is inexpensive in terms of bandwidth, perfectly distributed imposing equal requirements in terms of computation and bandwidth on all nodes, and is quite accurate even for networks under churn. The protocol is secure against adversaries trying to make the network appear smaller and makes it computationally expensive (based on a parameter W) to make the network appear larger or to flood the network with unwarranted traffic.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) under ENP GR 3688/1-1. We thank Mikhail Atallah for his help proving Theorem 1 and Christopher Wolf for an insightful discussion on an earlier draft of this paper.

References

1. Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. *ACM Trans. Internet Technol.*, 5:299–327, May 2005.
2. Javier Bustos-Jimenez, Nicolas Bersano, Satu Elisa Schaeffer, Jose Miguel Piquer, Alexandru Iosup, and Augusto Ciuffoletti. Estimating the size of peer-to-peer networks using lambert's w function. In Sergei Gorlatch, Paraskevi Fragopoulou, and Thierry Priol, editors, *Grid Computing*, pages 61–72. Springer US, 2008.
3. Thibault Cholez, Isabelle Chrisment, and Olivier Festor. Efficient dht attack mitigation through peers' id distribution. In *HOTP2P'10 - International Workshop on Hot Topics in Peer-to-Peer Systems*, Atlanta, Georgia, USA, 04/2010 2010.

[4] http://metrics.torproject.org/

4. John R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.
5. P. Erdős and A. Rényi. On random graphs. I. *Publ. Math. Debrecen*, 6:290–297, 1959.
6. P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21:341–374, November 2003.
7. Nathan Evans and Christian Grothoff. Beyond simulation: Large-scale distributed emulation of p2p protocols. In *4th Workshop on Cyber Security Experimentation and Test (CSET 2011)*. USENIX Association, 2011.
8. Nathan Evans and Christian Grothoff. R5n: Randomized recursive routing for restricted-route networks. In *5th International Conference on Network and System Security*, Milan, Italy, 2011. IEEE.
9. Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23:219–252, August 2005.
10. B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 49(1):291–307, 1970.
11. Jon Kleinberg, Mark Sandler, and Aleksandrs Slivkins. Network failure detection and graph connectivity. *SIAM J. Comput.*, 38:1330–1346, August 2008.
12. Jon M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000.
13. Dionysios Kostoulas, Dimitrios Psaltoulis, Indranil Gupta, Ken Birman, and Al Demers. Decentralized schemes for size estimation in large and dynamic groups. In *Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, pages 41–48, Washington, DC, USA, 2005. IEEE Computer Society.
14. Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro*, 28(2):39–55, 2008.
15. Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, page 183–192, New York, NY, USA, 2002. ACM, ACM.
16. Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351:394–406, February 2006.
17. Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi 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.
18. Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *1st International Workshop on Peer-to Peer Systems*, pages 53–65, Cambridge, March 2002.
19. Erwan Le Merrer, Anne-Marie Kermarrec, and Laurent Massouli. Peer to peer size estimation in large and dynamic networks: A comparative study. In *15th IEEE International Symposium on High Performance Distributed Computing 2006*, pages 7–17, 2006.
20. Bartlomiej Polot. Adapting blackhat approaches to increase the resilience of whitehat application scenarios. Master's thesis, Technische Universität München, 2010.
21. Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the*

IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, pages 329–350, London, UK, 2001. Springer-Verlag.

22. Andrei Serjantov and Stephen Lewis. Puzzles in p2p systems. In *8th CaberNet Radicals Workshop, Corsica*, 2003.
23. Tallat M. Shafaat, Ali Ghodsi, and Seif Haridi. A practical approach to network size estimation for structured overlays. In *Proceedings of the 3rd International Workshop on Self-Organizing Systems*, IWSOS '08, pages 71–83, Berlin, Heidelberg, 2008. Springer-Verlag.