Estimating Network Size from Local Information

Keren Horowitz and Dahlia Malkhi

School of Computer Science and Engineering
The Hebrew University of Jerusalem
{dalia,kerenyh}@cs.huji.ac.il

Key words: algorithms, approximation algorithms, distributed computing, distributed systems

1 Introduction

We consider the problem of locally estimating the size of a network whose constituent nodes dynamically join and leave it. More specifically, let the system be a dynamic process $S(1), S(2), ...,$, such that each $S(i)$ indicates the set of existing nodes at time i. The transition between $S(i)$ to $S(i+1)$ signifies either the arrival or the departure of one node. Further assume that a joining node is assigned by some means (outside the scope of this work) a *contact point*, i.e., the address of one existing node. The goal is to provide all node with an estimate of the network size at each time step, with low communication cost per step.

In this paper, we provide a localized estimation scheme that has good worst case accuracy. The scheme requires each joining node to communicate only with two nodes, its initial contact and one additional node. The expected accuracy of our estimator is within the range $n/2..n^2$, provided that the joiners' contact points are assigned at random.

Motivation.

Knowing the network size is a single point of globalization and a key enabler in several network constructions. For example, several recent peer-to-peer overlay networks construct various routing topologies with localized routing and localized maintenance. For example, Chord [16] builds a dynamic emulation of a hypercube; Pastry [4] and Tapestry [17] build a randomized hypercube based on the algorithm of Plaxton et al. [14] for finding near-by cache copies; Viceroy [9] approximates a butterfly network; and several others maintain a dynamic version of the De Bruijn network [5,1,6,13]; And the Symphony system [10] is a construction that builds a network based on the small worlds analysis in [8,3]. In all of these systems, the only global knowledge needed for building and maintaining the network is the size n (or the logarithm of it).

This knowledge is particularly required when a node joins the system. For example, Chord [16] requires that each node in the system simulates $\log(n)$ virtual nodes to achieve load balancing in the number of keys each node is responsible for. In Viceroy [9], a joining node is required to choose a level between 1 to $\log(n)$ to determine its outgoing links. In Symphony [10], a joining node draws a random number from a probability distribution function that depends on $\log(n)$, also as part of the process of choosing a node's links. The performance of all these schemes, in aspects such as routing length, join and leave cost, and congestion control hinges on the accuracy of the network size estimation.

In other settings, the network size determines how far should one push information in order to publicize it, and how far to search in order to pull information. For example, in the dynamic probabilistic quorum systems of [2], processes read and write information by accessing randomly selected sets of size $O(\sqrt{n})$. The effectiveness of the publicizing and the probability of succeeding in searching data both depend on how well these sets are selected.

Therefore, our methods provide an important, albeit small, building block in these systems and others. It is a viable alternative to existing solutions, exhibiting high accuracy and high localization.

Simple solutions like passing a synchronization message for each change in the network are considered impractical in such networks, because of the scale and the frequency of the changes. Existing localized solutions work by assigning a random bit-string identifier to each node, and letting each node estimate the network size through the distance (e.g., in bits) to the next closest node (see, e.g., [16,9,10]). This estimator hinges on uniform dispersal of node identifiers. This assumption is rather restrictive if physical identifiers are used, e.g., as in [8,15]. Moreover, even if identifiers are virtual and chosen uniformly at random, as e.g., in [16,9]), we expect that over a long term, frequent node joins and leaves would impair the uniformity of initial random choices, and hence the accuracy of the above estimator may decay.

Technical approach.

Our technique maintains a logical ring among existing nodes, thus requiring each node to hold a single *successor* link. We denote the successor of x by $succ(x)$.

The main idea of our scheme is the following: Suppose that n nodes exist in the network, and their estimators are accurate. Let n additional nodes join the network. Intuitively, we wish the existing nodes, as well as the joining nodes, to all obtain $\log(2n) = \log(n) + 1$ as their estimate of the logarithm of the network size. By randomly assigning successors to joining nodes, we have each existing node succeed one new node, on average. Hence, we maintain the estimators by letting each joining node examine the estimator of its successor node, increment it by one, and adopt the (incremented) estimator as its own estimator. We do the reverse process upon node departure.

Our analysis proves that our estimator is in expectation at least $n/2$, and is asymptotically close to $n/2$ at a large fraction of the network. We further substantiate our analysis with extensive simulations, whose results are summarised in Section 5.

2 System Model

The system is described as a dynamic process $S(1), S(2), ...,$, where $S(i)$ denotes the set of existing nodes at time i. The transition between $S(i)$ to $S(i+1)$ signifies either the arrival or the departure of one node. Each node x maintains a single *successor* link, denoted $succ(x)$. If y is the successor of x, then x is the *predecessor* of y, denoted $pred(y)$. Each node x holds a positive integer denoted $x.\log n_0$, that represents the node estimator for the logarithm of the current network size. The estimator $x.\log n_0$ may change during runtime.

Upon arrival, a joining node x is notified of a contact point, $contact(x)$, for help in entering the system. We assume that contact points are assigned at random from among existing nodes, and that departures occur independently of this assignment. In practice, this means first, that the system need to be discovered by the joining node; a number of existing discovery services may satisfy this, e.g., the Domain Name System (DNS) and Jini (see `http://www.sun.com/jini`). Second, the contact point sould be randomized. In [9,10,16] this is done as follows. A joining node selects a random identifier for itself, and then the system knows how to route/connect it to the node with the closest identifier in bits, thus supplying it with a random access node. In the rest of the paper, the manner by which a random contact point is obtained is left out of the discussion.

3 Process arrivals

The joining of a new node s, using contact point $contact(s)$, is given below.

Join of s:

`if (`$contact(s)$` == null) {`   `; the first node`

  $s.succ \leftarrow null$

  $s.\log n_0 \leftarrow 1$

`} else {`

  $s.succ \leftarrow contact(s).succ$

  $contact(s).succ \leftarrow s$

  $s.\log n_0 \leftarrow succ(s).\log n_0 \ \ +1$

  $succ(s).\log n_0 \leftarrow s.\log n_0$

`}`

3.1 Analysis

Theorem 1 *Let $S(1), ..., S(n)$ be a process of n arrivals. Let nodes choose their* $\log n_0$ *estimator according to* Join *algorithm above. Then for all nodes x, we have* $\log(n/2) \leq E[x.\log n_0] \leq 2\log(n)$ *and w.h.p.*[1] $x.\log n_0 \leq 6\log(n)$.

We first analyze a simple version of the above *Join* algorithm - the same algorithm without the last line (the update of the successor estimation). We call it the *simple algorithm.*

Lemma 2 *Let $S(1), ..., S(n)$ be a process of n arrivals. Let nodes choose their* $\log n_0$ *estimator according to the simple algorithm. Then for the k'th joining node x, we have* $E[x.\log n_0] = \log(k)$ *and w.h.p.* $x.\log n_0 \leq 3\log(n)$.

Proof: Let $M = \{x_1, ..., x_k\}$ be the set of first k joining processes. Assume that the x_i's are enumerated according to successor order, with x_k last. I.e., at time k, $S(k)$ is organized so that $x_{(i+1)} = succ(x_i)$ for $i = 1..(k-1)$. To simplify notation, we say that x_i is *smaller* than $x_{(i+1)}$, denoted $x_i < x_{(i+1)}$, $i = 1..(k-1)$. Let n_i denote the i'th node to join from among $\{x_1, ..., x_k\}$. Let $N_i = \{n_1, ..., n_{(i-1)}\}$ denote the set of nodes that have arrived up until step i.

Consider the random process $N_1, ..., N_k$. In this process, a step from N_j to $N_{(j+1)}$ makes a difference on the final outcome of the process, i.e., on $x_k.\log n_0$,

[1] Throughout this paper, we use *w.h.p.* to denote probability $\geq 1 - 1/n^{1+\epsilon}$.

only if n_j is the largest process (according to the successor order defined above) to join so far. More precisely, let max_j be the maximal process in N_j, i.e., $max_j = \max\{x | x \in N_j\}$. If $n_j < max_j$, then it will not affect $max_j. \log n_0$, and we have $max_{(j+1)}. \log n_0 = max_j. \log n_0$. If $n_j > max_j$, then we will have $max_{(j+1)} = n_j$, and $max_{(j+1)}. \log n_0 = max_j. \log n_0 + 1$. At the end of the process, the value $max_{(k-1)}. \log n_0$ will determine the estimator $x_k. \log n_0$.

Therefore, we use a view that regards x_k as an endpoint beyond the series $x_1..x_{(k-1)}$, and we say that endpoint x_k *sees* step j if $n_j \geq \max\{x \mid x \in N_j\}$. See Figure 1 for a pictorial view of this random process. Our analysis is based on the fact, which is explained above, that only steps that are seen by x_k affect the final estimator that will be used by x_k. Hence, the estimator $x_k. \log n_0$ is determined by the number of steps j that are seen by x_k.

We refer to the series of visible steps for a specific endpoint as a *chain*. The simple algorithm creates chains whose length determine the $\log n_0$ estimator of their last link (see Figure 2). Note that for every x in the system, there is a different chain, which is determined by nodes that are visible upon arrival by x.

We wish to analyze the length of the chain, i.e., the number of steps which are seen by the node x_k. Let y_i, $i = 1..(k-1)$, be an indicator random variable that is 1 if x_k sees step i, and 0 otherwise. Then $x_k. \log n_0 = 1 + \sum_{i=1..(k-1)} y_i$.

For the analysis, we compute the expected value of y_i *backwards* [2] That is, we will imagine a process starting from N_k and randomly deleting points from it one by one. Since the arrival order is completely random, this process is equivalent to our arrival process. The advantage of backward analysis is that now we can compute the expected value of y_i conditional on a given $N_{(i+1)}$, and move backward by removing n_i to obtain N_i. As we shall show below, the expected value of y_i will depend on i only, and not on $N_{(i+1)}$.

Now let $N_{(i+1)}$ be given. Since arrival order is completely random, any point in $N_{(i+1)}$ is equally likely to be the last one added to N_i, or, put differently, each point has equal probability of being removed in the (equivalent) backward process. There is only one removal that will let y_i equal 1, namely, the removal of the node whose value is highest in $N_{(i+1)}$. Therefore, $Pr[y_i = 1 | N_{(i+1)}] = 1/|N_{(i+1)}| = 1/i$. The crucial thing to observe is that $Pr[y_i | N_{(i+1)}]$ depends on i alone, and not on $N_{(i+1)}$. From this, it follows that $E[y_i] = 1/i$, and consequently, $E[\sum_{i=1..m} y_i] = \sum_{i=1..m} 1/i = \log(m) + o(1)$.

We now move on to the high probability result. By construction, the y_i's are independent, but they are not identically distributed. Hence, let z_i, $i = 1..m$,

[2] The *backward analysis* technique for randomized processes was offered, e.g., in [12].

be i.i.d. binary random variables with $E[z_i] = \log(m)/m$. It is a well known reduction to use the fact that $Pr[\sum_{i=1..m} z_i > T] \geq Pr[\sum_{i=1..m} y_i > T]$ in order to bound the probability that the sum of y_i's deviates from its mean, using the probability that sum of the i.i.d z_i's deviates from the (same) mean (see, e.g., [11]). Using Chernoff's bounds [3], we have that:

$$Pr[\sum_{i=1..m} y_i > 3\log(n)] \leq Pr[\sum_{i=1..m} z_i > 3\log(n)] \leq e^{-2^2 \log(n)/3} = 1/n^{(4/3)} .$$

□

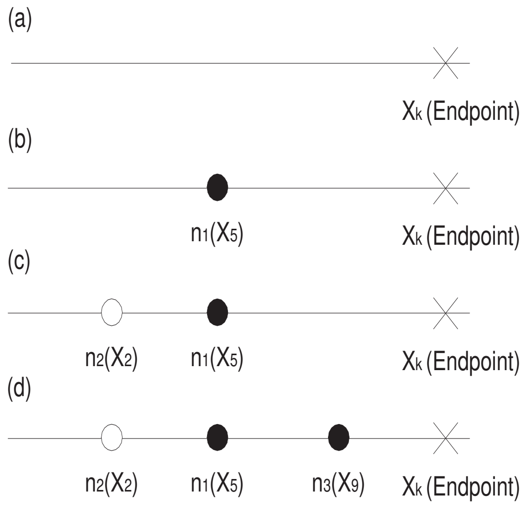

Fig. 1. Analysis of chains: x_k *sees* $n_1(x_5)$ and $n_3(x_9)$ but does not see $n_2(x_2)$. In this case, its initial estimator is 3.

Going back to the analysis of the full estimation algorithm, we can see that **lengths** of the chains are not affected by incrementing the estimator of the successor (last line of the *Join* algorithm). However, incrementing the successor estimator may change the value of the resulting estimation.

Lemma 3 *Let* $S(1), ..., S(n)$ *be a process of* n *arrivals. Let nodes choose their* $\log n_0$ *estimator according to the* Join *algorithm above. Then for the* k*'th node* x *to join the system, the expected number of times node* x *changes its initial* $\log n_0$ *estimator is* $\log(n-k-1)$ *and it is bounded w.h.p. from above by* $3\log(n)$.

Proof: The initial estimator of a node changes every time a new node becomes its predecessor. We can view this process as a chain in the reverse direction: The node estimator is changed upon arrival of a predecessor node. Figure 3 depicts this situation.

Let x_m be the m'th node to join. Let $x_1 > ... > x_{n-m-1}$ be the nodes that arrive after x_m, ordered according to successor order. Their arrival order determines

[3] The Chernoff bound we use, taken from [7], is $Pr[Y_m > (1+\delta)\mu] \leq e^{-(\delta^2)\mu/3}$.

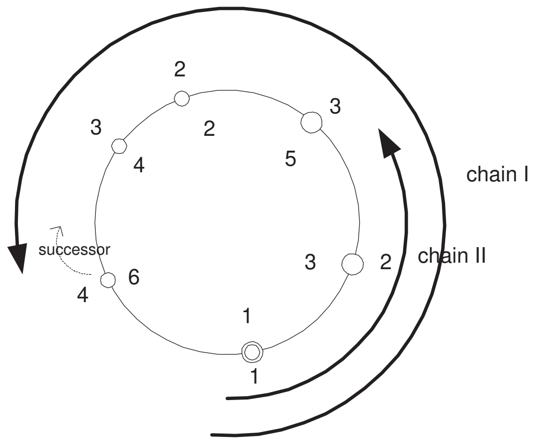

Fig. 2. Chains created by the simple algorithm. The inner numbers indicate the arrival order, the outer numbers are the nodes' $\log n_0$ estimations. The small cycles denote nodes that belong to the first chain, the bigger cycles denote nodes that belong to the second chain.

the number of times that x_m increments its $\log n_0$ estimator. By an identical analysis to Lemma 2 above, we get that the expected number of increments is $\log(n - m - 1)$, and w.h.p. it does not exceed $3 \log(n)$. $\square$

We are now ready to prove our main theorem:

Proof: (of Theorem 1) Consider the k'th node to join. Since $\max\{k, n-k\} \geq n/2$, then either its initial estimator is in expectation $\geq \log(n/2)$, from Lemma 2, or it undergoes an expected $\geq \log(n/2)$ increments, from Lemma 3.

The upper bound on the expectation follows by adding the maximal expectation value on both the initial estimator and on the number of increments, from lemmata 2 and 3, respectively. The high probability upper bound follows by adding the upper bounds of lemmata 2 and 3. $\square$

Theorem 4 *Let $S(1), ..., S(n)$ be a process of n arrivals. Let nodes choose their $\log n_0$ estimator according to the* Join *algorithm above. Then in expectation, at least $n(1 - (1/e^2))$ nodes have estimators at least $\log(n/2)(1 - (2/\sqrt{\log(n/2)}\ )\ )$.*

Proof: Let x be the k'th node to join. If $k \geq n/2$, then by Lemma 2 we have that upon joining, $E[x.\log n_0] = \log(k) \geq \log(n/2)$. Similarly, if $k \leq n/2$, then by Lemma 3 we have that the expected number of increments to $x.\log n_0$ is

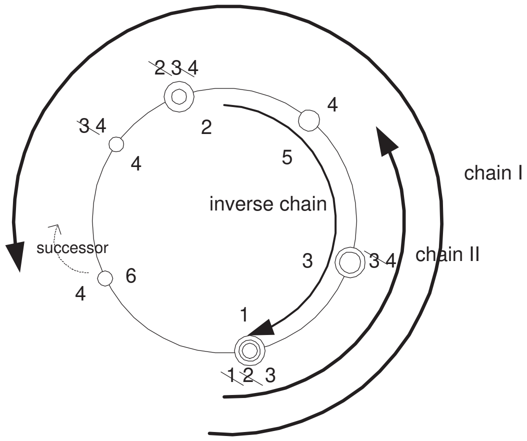

Fig. 3. Chains created by the $\log n_0$ estimation algorithm. The inner numbers indicate the joining order, the outer numbers are the nodes' $\log n_0$ estimations. The small cycles denote nodes that belong to the first chain, the medium cycles denote nodes that belong to the second chain and the biggest cycles denote the inverse chain nodes.

at least $\log(n/2)$. In either case, by applying Chernoff bounds we get

$$P\left[x.\log n_0 < \left(1 - \frac{2}{\sqrt{\log(n/2)}}\right)\log(n/2)\right] \le e^{-\left(\frac{2}{\sqrt{\log(n/2)}}\right)^2 \log(n/2)/2} \le e^{-2} .$$

Hence, in expectation, at least $n(1-(1/e^2))$ nodes x have each estimator $x.\log n_0 \ge \left(1 - \frac{2}{\sqrt{\log(n/2)}}\right)\log(n/2)$. $\square$

4 Process departures

According to the previous section, the $\log n_0$ estimator of a node changes if it becomes a successor of a joining node. As a result, we saw that the estimators of existing nodes are automatically adjusted when the number of nodes on the ring increases. What about decreasing the estimation when the number of nodes in the system decreases? In order for the network to adjust to a decrease in size, when a predecessor of any node s departs, its $\log n_0$ estimator is adjusted downward as indicated in the *Leave* algorithm below.

Leave of s:
$pred(s).succ \leftarrow s.succ$
$s.succ.\log n_0 \leftarrow \min\{s.\log n_0 - 1, 1\}$

The effect of decrementing the estimators is roughly the reverse of the incrementation, as argued in the following lemma:

Lemma 5 *Let $S(1), ..., S(m)$ be a process of arrivals or departures, such that n nodes exist in $S(m)$. Let nodes choose their* $\log n_0$ *estimator according to the* Join *algorithm above and update them upon departure according to the* Leave *algorithm. Then for every node x, if k nodes depart after it has joined, then the expected number of times node x decrements its initial* $\log n_0$ *estimator is* $\log(n) - \log(n-k)$.

Proof: The estimator of x changes every time its predecessor departs. We can view this process as a chain in the reverse direction: The node estimator is changed upon departure of a precedecessor node.

In an analogous manner to the proof of Lemma 2, we can view the backward process of moving from n nodes to $n - k$ nodes. Then the probability that x sees the i'th removal is $1/(n - i)$, and the expected sum of departures seen by x is $\sum_{i=1..k} 1/(n-i) \approx \log(n) - \log(n-k)$. $\square$

Remark:

For intuition, consider a concrete example: the departure of half of the system, i.e., $n/2$ nodes. Lemma 5 indicates that this causes existing nodes to decrease their logarithmic estimates of the system by 1. Though this indicates that departures are handled correctly, and additionally, our simulations below indicate good behavior, analyzing the effects of joining and leaving together in a closed form remains an open challenge.

5 Simulation Results

We examined our estimation algorithm in different network sizes and with different construction sequences. Some of the networks were built from a sequence of joining nodes. Other networks were built in an iterative process: First a group of nodes joined the network, then a group, which is half the size of the joining group, left the network, and so on, until the desired network size was reached. Our results are summarized in Figure 4 and Figure 5.

We see in Figures 4 and 5 that the majority of the nodes have $\log n_0$ estimators in the range of $1.5 \log(n)$ to $2.2 \log(n)$. Therefore the results confirm the theoretical proof. We remark that in networks that were constructed with departures of nodes, as well as arrivals, the tails of the $\log n_0$ estimators distribution are longer than for constructions with joins only. It is an open research direction to better stabilize the departure algorithm.

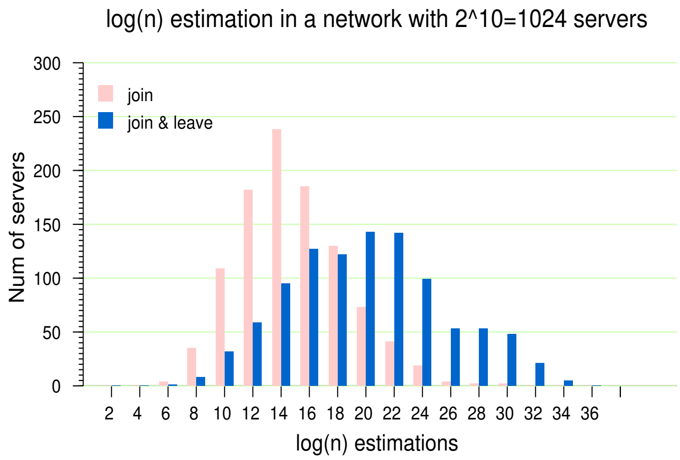

Fig. 4. $\log n_0$ estimation histogram in networks with 2^{10} nodes. The group size for joins and leaves is $2^6 = 64$.

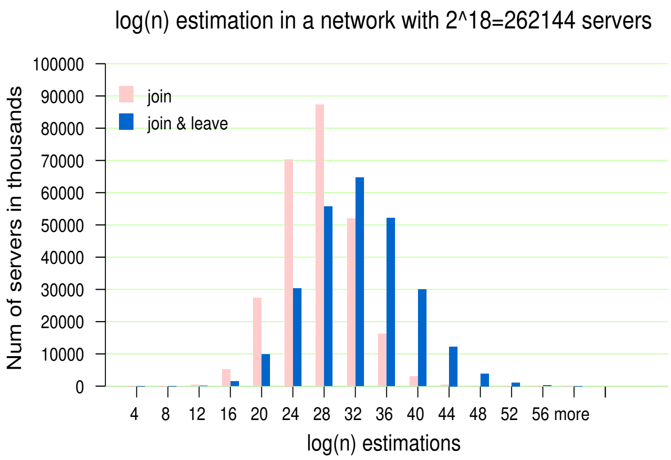

Fig. 5. $\log n_0$ estimation histogram in networks with 2^{18} nodes. The group size for joins and leaves is 2^{10}.

Acknowledgements

We are thankful to Alex Samorodnitsky for many useful discussions.

References

[1] I. Abraham , B. Awerbuch , Y. Azar , Y. Bartal D. Malkhi and E. Pavlov. "A Generic Scheme for Building Overlay Networks in Adversarial Scenarios". In *International Parallel and Distributed Processing Symposium* (IDPDS 2003), April 2003, Nice, France.

[2] I. Abraham and D. Malkhi. "Probabilistic Quorums for Dynamic Systems". In the *Proceedings of the 17th International Symposium on DIStributed Computing (DISC 2003)*, Sorento, Italy, October 2003.

[3] L. Barriére, P. Fraigniaud, E. Kranakis and D. Krizanc. "Efficient routing in networks with long range contacts". *15th International Symposium on Distributed Computing (DISC '01)*, Lecture Notes in Computer Science vol. 2180, Springer, 2001 pp. 270–284.

[4] P. Drushel and A. Rowstron. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". *Proceeding of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware),* November 2001.

[5] P. Fraigniaud and P. Gauron. "The Content-Addressable Network D2B". Technical Report 1349, LRI, Univ. Paris-Sud, France, January 2003.

[6] F. Kaashoek and D. R. Karger. "Koorde: A Simple Degree-optimal Hash Table". In *2nd International Workshop on Peer-to-Peer Systems* (IPTPS '03), February 2003, Berkeley, CA.

[7] M. J. Kearns and U. V. Vazirani. "An Introduction to Computational Learning Theory". The MIT Press, August 1994.

[8] J. Kleinberg. "The small world phenomenon: An algorithmic perspective". Proceedings of the 32nd ACM Symposium on Theory of Computing, May 2000, pp. 163–170.

[9] D. Malkhi, M. Naor and D. Ratajczak. "Viceroy: A scalable and dynamic emulation of the Butterfly". In *Proceeding of the 21 st ACM Symposium on Principles of Distributed Computing (PODC'02)*, July 2002.

[10] G.S Manku, M. Bawa, P. Raghavan. "Symphony: Distributed Hashing In Small World". In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[11] R. Motwani and P. Raghavan. "**Randomized Algorithms**". Cambridge University Press, 1995.

[12] K. Mulmuley. "Computational Geometry: An Introduction through Randomized Algorithms". Prentice Hall, 1994.

[13] M. Naor and U. Wieder. "Novel Architectures for P2P Applications: the Continuous-Discrete Approach". In *The Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '03)*, 2003.

[14] C. Plaxton, R. Rajaram, and A. Richa. "Accessing nearby copies of replicated objects in a distributed environment". *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA 97), pp. 311–320, June 1997.

[15] S. Ratnasamy, P.Francis, M. Handley, R. Karp and S. Shenker. "A scalable content-addressable network". In *Proceeding of the ACM SIGCOMM 2001 Technical Conference.* August 2001.

[16] I. Stoica, R. Morrise, D. Karger, M. F. Kaashoek and H. Balakrishnan. "Chord: A scalable peer-to-peer lookup service for Internet applications". In *Proceedings of the SIGCOMM 2001*, August 2001.

[17] B.Y. Zhao, J. D. Kubiatowicz and A.D. Joseph. "Tapestry: An infrastructure for fault-tolerant wide-area location and routing". U.C. Berkeley Technical Report UCB/CDS-01-1141, April, 2001.