Random Node Sampling in Kademlia

Zoltán Novák
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
H-1117, Magyar tudósok körútja 2, Budapest, HUNGARY
Phone: (36)1-463-2225, Fax: (36) 1-463-3107
Email: novak@tmit.bme.hu

Zoltán Pap
Ericsson Telecomm. Hungary,
H-1117, Irinyi J. u. 4-20, Budapest, HUNGARY
Email: zoltan.pap@ericsson.com

***Abstract*—We present a novel distributed method for selecting random nodes uniformly in Kademlia DHT networks. The algorithm does not require any extension to the Kademlia protocol and can be applied to any Kademlia-type network. It is easily tunable with one parameter to set an appropriate tradeoff between the evenness of the distribution and time – message – complexity. The method may be used as a building block for network algorithms in areas such as load balancing, Byzantine agreement, fault recovery and statistical measurements.**

I. Introduction

Peers of large distributed overlay networks have only a limited knowledge of the status of system-wide parameters such as size, average load, etc. The most viable approach to acquire and maintain such information in a distributed environment is the use of statistical methods. By providing statistical tools for participating peers to estimate global properties of the system, new possibilities will open for designing efficient, adaptive P2P clients.

Simple random sampling is one of the fundamental techniques used in statistics. By choosing entities from the population with equal probability, it is ensured that this sample will lack of any bias or correlation. Facilitating uniform random node selection is the first step to provide a general base for clients to make system-wide statistical estimations.

Random selection also plays an important role in algorithm design. [1] Random node selection can be used as an algorithmic building block in randomized network algorithms. Load balancing is one of the areas which could benefit most from the availability of uniform distributed random node selection. [2] Another field of application was presented by Scott Lewis et al. [3]. Their scalable Byzantine agreement algorithm is based on random node selection.

A. Motivation

Random selection is usually not a problem when a computable bijection exists between a set of numbers and the population. Without such a bijection the problem can be very hard. Several general solutions have emerged, mainly based on random walks. Although the aforementioned problem is solvable with random walks – thus it can be used to select random nodes even in unstructured P2P systems [4] – we will show that a more sophisticated solution may be elaborated for Kademlia.

The motivation for using Kademlia is twofold:

- Kademlia is one of the most widely used DHT system in practice today. Some examples: KAD network (eMule[1], aMule[2], MLDonkey clients[3]), BitTorrent Mainline DHT[4], Vuze (Azureus) DHT[5], RevConnect[6]
- Kademlia has a unique addressing and searching mechanism (see section III). This allows the creation of a simple and intuitive random selection algorithm, which is largely based on the special properties of the Kademlia address space.

B. Related Work

Vivek Vishnumurthy and Paul Francis examined the problem of random selection in unstructured networks for P2P applications. [4] Their solution is based on random walks. The properties of random walks in P2P systems were also analyzed by Christos Gkantsidis et al. [5], [6].

In DHT networks there is an opportunity to develop specialized algorithms, considering the distinctive properties of the network graphs. We have to mention the work of Valerie King, Scott Lewis, Jared Saia and Maxwell Young. [7] They present two similar random node selection algorithms for the Chord [8] DHT network. Their solutions are based on alternating a search to a random address, and walks on the successor nodes of the Chord ring. Although both algorithms have good accuracy and complexity values, their failure probability and latency critically depends on four distinct constant parameters. Unfortunately the optimal selection of these constants was proved to be *computationally intractable*. The only possible way to achieve a near optimal solution was to set the constants with the help of numerous simulations.

The results of [7] could not be applied directly to Kademlia networks. A main difference is that a search to a random address in Kademlia results in a completely different node distribution. Furthermore in Kademlia there is no equivalent to Chord successor nodes. Both differences are the consequence of the different distance metric used in Kademlia.

[1] http://www.emule.org
[2] http://www.amule.org
[3] http://mldonkey.sourceforge.net
[4] http://www.bittorrent.org
[5] http://www.vuze.com
[6] http://www.revconnect.com

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7239
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7239

C. Our Results

Our algorithm:

- only uses the built in Kademlia [9] primitives (see section II).
- exploits the special properties of the Kademlia address space (see section III).
- is *simple* and intuitive (see section IV).
- is *accurate*, selects every node with equal probability (see section IV and V-E)
- is easily *tunable*. (By setting two confidence parameters, our algorithm has *adjustable failure probability*, see section V-A1 and V-E)
- has a low complexity – comparable to [7] – thus usable in practice (see section VI and VII)

II. Kademlia

This section is a short introduction to the Kademlia [9] DHT.

Kademlia uses a 160 bit address space, to which both nodes and keys are mapped. Every node stores data with keys closest to its address, in terms of the bitwise binary xor operator (see section III). Every node maintains 160 tables to store routing information, in Kademlia terminology these tables are called k-buckets. The i-th k-bucket contains at most K nodes whose distance from the current node is between 2^{160-i} and $2^{160-i+1}$, where K is a pre-chosen system parameter.

The Kademlia protocol contains four RPC-s that all the functions are built on:

- PING, to check if a node is still connected;
- STORE, to store a key and corresponding data;
- FIND_NODE, with an address as its parameter, to look for the K closest values to the given address from the node's routing tables;
- FIND_VALUE, with a key as its parameter; if a node stores data corresponding to the key, the return value is the stored data; otherwise it behaves identically to FIND_NODE.

When node A looks for node B with address y, the search goes thus through the following steps:

1) Node A creates a list L containing the K closest addresses to y. It first fills this list from its own k-buckets. It also marks every node in the list on which it has already run the FIND_NODE RPC.
2) Node A selects α unmarked nodes from the list, and runs the FIND_NODE RPC on them (α is a system-wide parameter).
3) Node A updates the list using the return values of the FIND_NODE RPCs, always maintaining only the K closest addresses to y.
4) If A hasn't found the node it was looking for, or the list still contains unmarked nodes, it returns to step 2.

K-buckets are ordered lists of nodes, with the most recently seen node at the beginning of the list. If a node A receives a message from another node B, than A tries to insert B into the appropriate k-bucket, if there's still room. If the given k-bucket is full, A sends PING to the node from the end of the list; if it replies, A moves it to the head of the list; if it does not, A deletes it from the list, and replaces it with B. With adequate network traffic, k-buckets remain consistent thanks to the procedures above.

When a node leaves the network, it simply copies its data to the nearest node, and disconnects.

III. Definitions

Definition 1: Let A be the set of all addresses in the system. We assume that these addresses are binary integers. Let N be the set of occupied addresses in the system – actually this is the set of online nodes. We denote the number of online nodes $|N|$ with n.

We assume that $n > 0$, i.e., every system has at least one node online.

Definition 2: Let $X \in N$ be the address of a node in the system, then define $\mathrm{Near}(X) \subseteq A$ as the set of addresses where:

$$\mathrm{Near}(X) = \{Z \mid \forall Y \in N, Y \neq X, X \oplus Z < Y \oplus Z\}$$

and $\oplus$ is the bitwise xor operation.

In this definition the result of the $\oplus$ operation is interpreted as an integer number. $\mathrm{Near}(X)$ is actually the Voronoi cell of X: the set of all addresses that are closer to X – according to the xor metric – than to any other online node.

Definition 3: Let the territory $\mathrm{T}(X)$ of a node $X \in N$ be:

$$\mathrm{T}(X) = \frac{|\mathrm{Near}(X)|}{|A|}, \ (0 < \mathrm{T}(X) \leq 1)$$

The territory of X represents the portion of addresses (A) that are closer to X than to any other online node. For example if $\mathrm{T}(X) = 0.5$, than if a random address R from the address set A is chosen, the nearest online node to R will be X with probability 0.5.

Definition 4: For all $C \in A$, $\mathrm{Route}(C) = X$ if $X \in N$ and $C \in \mathrm{Near}(X)$. This Route function assigns one online node to every possible address.

Our definition above is actually similar to Kademlia's routing – see the routing algorithm described in Section II. While Kademlia finds the k nearest nodes to an arbitrary address, we are only interested in finding the nearest online node to an address. This route function is available to every node in the Kademlia system, and gives results in $\mathrm{O}(\log n)$ steps.

In Kademlia the address space is: $A = \{0, 1, \ldots, 2^{160} - 1\}$. Joining nodes choose a uniformly distributed random address independently from each other.

A node can count its own territory size by counting its empty k-buckets. If a node X has e empty bucket, then the territory size of this node is:

$$\mathrm{T}(X) = \frac{2^e}{2^{160}}$$

IV. A Novel Random Sampling Algorithm

Before presenting our random sampling algorithm we will discuss two naive reference approaches first. A simple solution for the problem would be to route to a random address and to

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7239
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7239

accept the resulting node as a random sample. Unfortunately although addresses are distributed uniformly in the address space, the resulting territory sizes are highly uneven. (Figure 1). This approach would therefore violate the uniform property of the sampling.

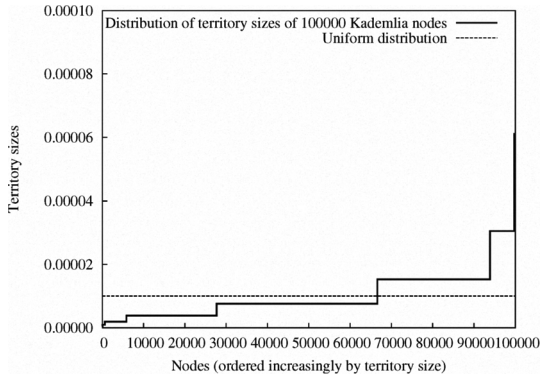

Figure 1. Territory sizes of 100000 Kademlia nodes ordered by magnitude

Another solution would be to repeatedly route to random addresses until by chance an occupied one is found. Using this method, every node would be selected with the same probability: $n/2^{160}$. Unfortunately this method is unusable in practice, as the probability of finding an occupied address is extremely low.

Our algorithm is the mixture of the previous two naive solutions, and is based on an intuitive idea:

1) Choose a random address R from the address space A
2) $X := \mathrm{Route}\,(R)$
3) Accept X with probability $\frac{T_{min}}{\mathrm{T}(X)}$ – where T_{min} is the minimal territory size in the system – otherwise repeat the whole procedure from step I.

In one cycle the algorithm selects a node with exactly the same T_{min} probability. So the resulting distribution is completely even, if the exact value of T_{min} is known. Unfortunately this information is typically not available by default in distributed systems such as Kademlia; therefore we need to provide a method for estimating T_{min}.

V. Estimating T_{min}

In Kademlia T_{min} is a random variable. The most straightforward method to estimate T_{min} is to estimate it with its expected value: $\mathrm{E}(T_{min})$. Distribution of the territories (and the minimal territory size) is a function of the number of nodes in the system: n.

Our approach for estimating T_{min}:

1) We provide the means to estimate n – the number of online nodes in the system – based on the density of the occupied addresses in the address space. (see subsection V-A)
2) We determine the exact distribution of the random variable T_{min} as a function of n (see subsection V-C).
3) We show that T_{min} can be estimated with $\mathrm{E}(T_{min})$ (see subsection V-D), or – as discussed in subsection V-E – interval approximation of T_{min} is also possible.

A. Estimating the Number of Online Nodes

Estimating the size of a DHT network is not a particularly hard problem. One of the first approaches to tackle it was given by the Viceroy [10] DHT, where the complete architecture is based on the possibility of estimating the actual size of the overlay. Another solution to the problem was given by Binzenhd'ż˝fer et al. in [11] assuming Chord networks.

Definition 5: Let p be the density of occupied addresses:

$$p = \frac{n}{|A|}$$

the number of online nodes divided by the address space size.

Because the address space size is constant in Kademlia, the density p is the function of the number of nodes n.

Addresses in Kademlia are selected by nodes with uniform distribution. Therefore the expected value (first moment) of node density in any subset of A is exactly p. Assume we have an S sample from the address space, in which the occupied addresses are denoted with 1 and free addresses are denoted with 0, then let $\hat{p}$ be the mean of our sample. As we can see, whether an address is occupied or not, has a Bernoulli distribution where p is the Bernoulli parameter.

Now the point estimation statistics of the number of nodes $\mathrm{St}(n)$ can be done easily, using the method of moments (replacing the first moment p with the mean of our sample $\hat{p}$):

$$n = |A|\,p$$

$$\mathrm{St}(n) = |A|\,\hat{p}$$

We can use Kademlia search to get our sample S fast. First route to a random address R. In Kademlia we get back the K nearest nodes to R according to the xor metric. Let F be the farthest address in the result. Then the result of the search is actually a sample of the following addresses: R, $R \oplus 1$, $R \oplus 2$, $R \oplus 3$,...,F. We can use this sample to count $\hat{p}$:

$$\hat{p} = \frac{K}{(R \oplus F) + 1}$$

By aggregating the results of multiple Kademlia searches the estimation can be further improved.

1) Interval Estimation of the Number of Nodes: We can ensure the correctness of our algorithm by overestimating the number of nodes (see subsecion V-E). For interval approximation of the Bernoulli parameter several approximate methods exists. [12]–[14]

According to [14], the following simple estimation using Chi-square distribution gives good results, because the sample size ($(R \oplus F) + 1$) is extremely large and p is extremely low:

$$n_{High} = \frac{|A|}{2(R \oplus F) + 2}\chi^2_{2(K+1),\alpha}$$

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7239
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7239

Where α is the confidence parameter.

A simple numeric example: $|A| := 2^{32}$ (the address space size), $K := 10$, and the result of our query gives $(R \oplus F) + 1 = 1000000$, then our simple estimated node number is

$$|A|\,\hat{p} = |A|\,\frac{K}{(R \oplus F) + 1} = 2^{32}\frac{10}{1000000} \approx 42950$$

And the number of nodes is lower than:

$$n_{High} = \frac{2^{32}}{2000000}\,\chi^2_{22,0.01} = \frac{2^{32}}{2000000}\,40.289 \approx 86520$$

with a probability of 0.99. Of course any other α parameter can be used easily with the χ^2 distribution. By repeating queries, the boundaries of this interval converge to the mean. If the Kademlia query is repeated two more times, and the aggregated success rate ($\hat{p}$) remains the same, then our new estimated upper bound using the same confidence becomes:

$$\frac{2^{32}}{6000000}\,\chi^2_{62,0.01} \approx 64998$$

B. Visualizing territory size

To estimate T_{min} we first have to understand the way territories in the XOR metrics build up.

- The sum of territories for all online nodes in the system is exactly 1.
- The sum of territories for online nodes whose addresses start with 0 or 1 are 0.5 and 0.5 respectively, if there is at least one online node in both the 0xxx...and the 1xxx...address space.
- Groups of nodes with prefix: 00, 01, 10, 11 share 0.25, 0.25, 0.25, 0.25 parts of the whole territory if all address prefixes contain at least one online node.
- and so on...

What happens if there isn't any node with prefix 10? Then nodes with prefix 11 will share 0.5 territory, as they are the only nodes with address prefix 1, and nodes with prefix 1 share 0.5 territory according to bullet 2 above.

The easiest way to picture the division of territories is to visualize it as a binary tree, where the leaves are the occupied addresses. At the root the territory is 1, and at every lower node the territory is divided by 2 if that node forks. Figure 2 shows a division of territories between nodes: 0000, 0001, 1001, 1100, 1111 in the four bit address space.

Now we can see the main difference from Chord or other DHTs such as Pastry [15] or CAN [16] that use Euclidean distance metrics. In the latter DHTs the territory size of a given node depends only on the distance to its direct neighbors, but in Kademlia the territory is affected by distant nodes - or their absence - as well.

C. Distribution of T_{min}

We have two conflicting assumptions:

1) Joining nodes choose their addresses from a finite address space independently and randomly with uniform distribution, thus address collision is possible
2) Every node has a unique address

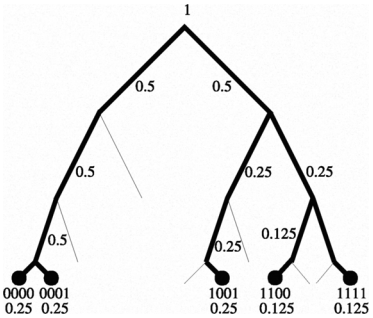

Figure 2. Division of territories in xor metrics

In the rest of the paper we address this inconsistency by assuming that the sizes of the addresses are unbounded. Note that this assumption can be made without loss of generality as the probability of address collision is negligible.

Definition 6: Let $\mathrm{F}_n^{T_{min}}(x)$ be the cumulative distribution function (CDF) of T_{min} in a system with n independently and randomly chosen node addresses. Then, in a system with one node ($n = 1$):

$$\mathrm{F}_1^{T_{min}}(x) = \begin{cases} 0 & \text{if } x \leq 1, \\ 1 & \text{if } x > 1. \end{cases}$$

To define the cumulative distribution function of T_{min} for all n, we use the following statement: If X and Y are two independent random variables with cumulative distribution functions $\mathrm{F_x}$ and $\mathrm{F_y}$, then the CDF of $Z = \min(X, Y)$ is:

$$\mathrm{P(Z < x) = F_z}(x) = \mathrm{F_x}(x) + \mathrm{F_y}(x) - \mathrm{F_x}(x)\mathrm{F_y}(x)$$

Now we can write a recursive definition of $\mathrm{F}_n^{T_{min}}(x)$ using the law of total probability:

1) Let us visualize the n node addresses at the root of the territory tree (fig. 2). Every address begins with 0 or 1 with a probability of 0.5 respectively. The root node divides the n nodes into two sets.
2) The cardinality of these two sets has a binomial distribution, and they sum up to n:

$$P(\text{no address begins with } 0) = \binom{n}{0}2^{-n}$$
$$P(1 \text{ address begins with } 0) = \binom{n}{1}2^{-n}$$
$$\vdots$$
$$P(n \text{ addresses begin with } 0) = \binom{n}{n}2^{-n}$$

3) Let us assume that 5 addresses begin with 0 and $n - 5$ with 1. If $\mathrm{F}_5^{T_{min}}(x)$ and $\mathrm{F}_{n-5}^{T_{min}}(x)$ is known, the CDF of their minimum can be written. As each of the two branches has only 0.5 territory, we have to use $\mathrm{F}_5^{T_{min}}(2x)$

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7239
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7239

and $\mathrm{F}_{n-5}^{T_{min}}(2x)$ instead. Note that this would not be the case if the division was $(n;0)$ or $(0;n)$.

4) Using the law of total probability we can write the following recursive definition:

$$\mathrm{F}_n^{T_{min}}(x) = \frac{1}{2^n}\left(\binom{n}{0}F_n^{T_{min}}(x) + \binom{n}{n}F_n^{T_{min}}(x) + \sum_{k=1}^{n-1}\binom{n}{k}\left(\mathrm{F}_k^{T_{min}}(2x) + \mathrm{F}_{n-k}^{T_{min}}(2x) - \mathrm{F}_k^{T_{min}}(2x)\mathrm{F}_{n-k}^{T_{min}}(2x)\right)\right) \quad (1)$$

5) Finally – after rearranging occurrences of $\mathrm{F}_n^{T_{min}}(x)$ to the left – we reach the following formula:

$$\mathrm{F}_{pleasereferto1}^{T_{min}}(x) = \begin{cases} 0 & \text{if } x \leq 1, \\ 1 & \text{if } x > 1. \end{cases}$$

$$\mathrm{F}_n^{T_{min}}(x) = \frac{1}{2^n - 2}\sum_{k=1}^{n-1}\binom{n}{k}\left(\mathrm{F}_k^{T_{min}}(2x) + \mathrm{F}_{n-k}^{T_{min}}(2x) - \mathrm{F}_k^{T_{min}}(2x)\mathrm{F}_{n-k}^{T_{min}}(2x)\right) \quad (2)$$

D. The $\mathrm{E}_n(T_{min})$ function

With the help of the CDF of T_{min} we can calculate $\mathrm{E}_n(T_{min})$. For this we have to recognize that T_{min} is a discrete random variable, with the following possible values: $2^0, 2^{-1}, \ldots, 2^{1-n}$.

Therefore even though we only have a complex recursive definition of $\mathrm{F}_n^{T_{min}}(x)$ we have to count it only in those particular points where the CDF has step discontinuity. This means $\mathrm{E}_n(T_{min})$ can be written as:

$$\mathrm{E}_n(T_{min}) = \sum_{i=0}^{n-1}\frac{1}{2^i}\left(\mathrm{F}_n^{T_{min}}(2^{1-i}) - \mathrm{F}_n^{T_{min}}(2^{-i})\right)$$

The exact $\mathrm{E}_n(T_{min})$ function is presented on figure 3 and also on figure 5.

E. Interval Approximation

According to our algorithm (section IV), a node accepts the result of a random route with probability $T_{min}/\mathrm{T}(X)$ in every iteration. Overestimation of T_{min} results in probability values greater than one; these values must be clipped to 1. This means that nodes with a territory size smaller than the estimated T_{min} are selected with lower probability. Therefore it is preferable not to overestimate T_{min}. Although underestimation of T_{min} results in higher complexity, it also ensures the uniform selection of nodes: If the estimated T_{min} is less than or equal to the minimal territory size of the system, then our algorithm selects each node with equal probability $(1/n)$ always and without exception.

Fortunately with the help of the CDF function our estimation is easily tunable as shown in the following example.

Let's assume we would like to underestimate T_{min} with 95% confidence. That means that the actual T_{min} will be greater than our estimation with probability 0.95.

Knowing the CDF of T_{min} it is easy. We only have to find the greatest x where:

$$\mathrm{F}_n^{T_{min}}(x) < 0.05$$

again we only have to check $\mathrm{F}_n^{T_{min}}$ in its stepping points. The resulting estimations are depicted on figure 3.

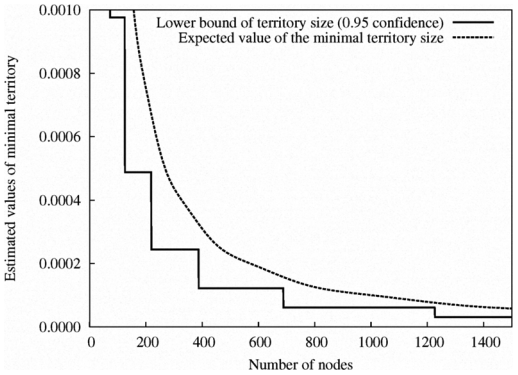

Figure 3. $\mathrm{E}_n(T_{min})$ and underestimation of T_{min} with confidence 0.95

If interval approximations are used, accuracy will be completely tunable. An example in numbers: with an underestimation of T_{min} with 95% confidence, and overestimation of node numbers with 99% confidence (subsection V-A1), our algorithm returns a completely uniform distribution with probability of:

$$0.95 \cdot 0.99 = 0.9405$$

at least.

Note that the probabilty value above is a lower limit because the two estimated random variables are not independent.

F. Approximation of $\mathrm{E}_n(T_{min})$

Although the accurate value of $\mathrm{E}_n(T_{min})$ is computable using the CDF, there is also a need for an approximate formula for several reasons: Our $\mathrm{F}_n^{T_{min}}(x)$ formula is recursive and d'ž˝therefore rather slow for larger n values. The computation of the large binomial coefficient requires high precision arithmetic. Furthermore an approximate formula is essential for understanding the asymptotic behavior of $\mathrm{E}_n(T_{min})$.

To find the best approximation, trial and error method was used, and numerous functions were examined in the domain:

$$\mathrm{E}_n(T_{min}) \approx \frac{1}{n}\mathrm{f}(x)$$

We achieved the best asymptotic results with:

$$f(x) = \frac{1}{\ln n \ln\log_c n}$$

Where c is a constant. On figure 4 the deviation of this approximation is depicted, first the constant was set to e, then to the best value $c = 4.9$ – we computed by curve fitting using the least squares method. The deviation is computed with the following formula:

$$\left|\mathrm{E}_n(T_{min}) - \frac{1}{n\ln n \ln\log_c n}\right|$$

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7239
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7239

On figure 5 the best approximation is depicted together with the accurate values. Further on – especially in section VI (Complexity) – we use the approximate function:

$$\mathrm{E}_n(T_{min}) \approx \frac{1}{n \ln n \ln \log_{4.9} n}$$

In section VII-A we evaluate this approximation using simulation results.

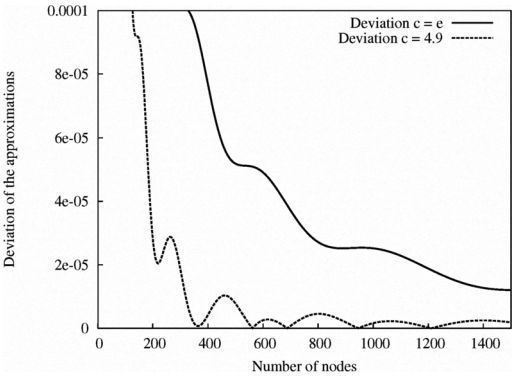

Figure 4. The error of our two approximations

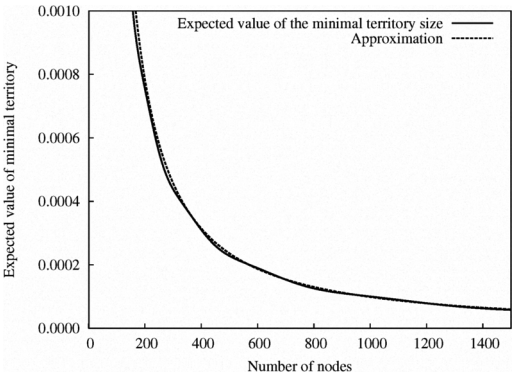

Figure 5. $\mathrm{E}_n(T_{min})$ and approximation: $\frac{1}{n \ln n \ln \log_{4.9} n}$

VI. Complexity

The complexity depends on the number of Kademlia searches used by our algorithm.

We consider the case when T_{min} is estimated using $\mathrm{E}_n(T_{min})$. We analyze the number of required Kademlia searches therefore we can first assume that our node number estimation is exact. This can be done because our estimation of n is an unbiased estimator (see [14]).

In this case the number of searches has a geometric distribution with parameter p:

$$p = \frac{\mathrm{E}_n(T_{min})}{1/n}$$

Thus the estimated value of repetitions is:

$$\frac{1}{p} = \frac{1}{n\mathrm{E}_n(T_{min})}$$

According to our approximation (section V-F) this has a complexity of:

$$O(\log n \log \log n)$$

As routing in Kademlia has a complexity of $O(\log n)$, The overall complexity of our method is:

$$O(\log^2 n \log \log n)$$

A. Cost of Overestimating n

Overestimation of n results in better failure probability. The cost of overestimation is easily computable.

Let's assume n is overestimated by a factor of c. That means our algorithm will use approximation:

$$\mathrm{E}_n(T_{min}) \approx 1/(cn \ln cn \ln \log_{4.9} cn)$$

Instead of the correct:

$$\mathrm{E}_n(T_{min}) \approx 1/(n \ln n \ln \log_{4.9} n)$$

So the estimated number of repetition is larger by a factor of:

$$\frac{c \ln cn \ln \log_{4.9} cn}{\ln n \ln \log_{4.9} n}$$

This has a limit of c if n converges to infinity.

VII. Evaluation

A. Evaluation of our $\mathrm{E}_n(T_{min})$ approximation using simulation results

We examined the accuracy of our algorithm using our approximation (see section V-F) of $\mathrm{E}_n(T_{min})$ by running 1000 simulations with different system sizes. We measured the mean number of nodes with smaller territory size than the approximated $\mathrm{E}_n(T_{min})$.

In table I we can see the mean number of nodes that our algorithm does not select correctly. The table shows that on average there is approximately one node only, for which $\mathrm{E}_n(T_{min}) > T(X)$.

That means that most of the time our approximation of $\mathrm{E}_n(T_{min})$ is a correct lower bound of the actual territory sizes.

Table I
Accuracy using approximate $\mathrm{E}_n(T_{min})$

Number of nodes	Mean number of nodes with territory smaller than $\mathrm{E}_n(T_{min})$
1000	0.87
5000	0.498
20000	1.79
30000	1.
100000	1.04

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7239
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7239

B. Evaluation of complexity

Our complexity result – $O(\log^2 n \log\log n)$ – is comparable to, but a bit worse than the complexity result of [7]: $O(\log^2 n)$. This is a very good result considering the difficulties introduced by the use of xor metric.

The practical values of our algorithm and [7] are also comparable. The authors made simulations with 10000 nodes, using their two different algorithms and their mean complexity result was $10.01 \log n$ and $20.02 \log n$ respectively. Our algorithm has an expected repetition number 16.18 in a system of the same size, where each repetition involves a Kademlia search with $O(\log n)$ complexity.

As we can see on figure 6 the estimated repetition number is below 30 even in a system of 1000000 nodes.

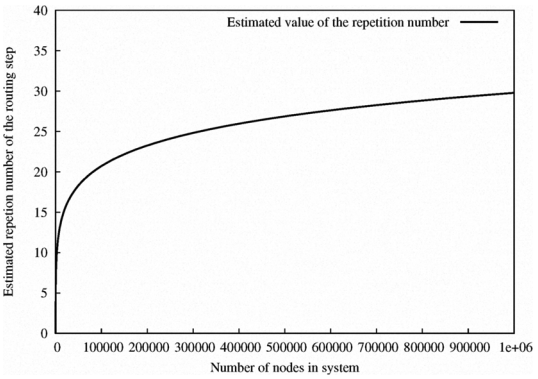

Figure 6. Estimated number of repetition of the Kademlia search ($f(x) = \ln x \ln \log_{4.9} x$)

The good complexity results of our algorithm are the direct consequence of the Kademlia territory distribution. For example in the Chord DHT the estimated value of the minimal territory size – according to [17] – is:

$$\mathrm{E}_n^{\mathrm{Chord}}(T_{min}) = \frac{1}{n^2}$$

Thus the complexity of our algorithm in Chord would be $O(n)$. This means that in Chord our algorithm would be unusable. An interesting question would be to examine the distribution of minimal territory size and the usability of our algorithm in these DHTs using different load – territory – balancing techniques [17] [18].

VIII. Conclusions and Future Work

We have presented an algorithm for selecting a peer uniformly at random in Kademlia, one of the most widespread DHT systems in practice. The algorithm has an expected complexity of $O(\log^2 n \log\log n)$. We showed that in practice the complexity of our algorithm is comparable to [7].

Some possible improvements:

- Kademlia search has a complexity of $O(\log n)$. Instead of using independent Kademlia searches in every step of our algorithm, we could replace some searching steps with shorter random walks.
- Load balancing algorithms – for example [18] – can cause a different, more balanced territory distribution in Kademlia. Although our base algorithm remains correct in this case too, with some tuning of the parameters, better complexity values are achievable.

References

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

[2] M. D. Mitzenmacher, "The power of two choices in randomized load balancing," Ph.D. dissertation, 1996, chair-Alistair Sinclair.

[3] C. Scott and L. J. Saia, "Scalable byzantine agreement," Tech. Rep., 2004.

[4] V. Vishnumurthy and P. Francis, "On heterogeneous overlay construction and random node selection in unstructured p2p networks." in *INFOCOM*. IEEE, 2006.

[5] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks." in *INFOCOM*, 2004.

[6] ——, "Random walks in peer-to-peer networks: algorithms and evaluation," *Perform. Eval.*, vol. 63, no. 3, pp. 241–263, 2006.

[7] V. King, S. Lewis, J. Saia, and M. Young, "Choosing a random peer in chord," *Algorithmica*, vol. 49, no. 2, pp. 147–169, 2007.

[8] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *ACM SIGCOMM 2001*, San Diego, CA, September 2001.

[9] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 53–65.

[10] D. Malkhi, M. Naor, and D. 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*. New York, NY, USA: ACM, 2002, pp. 183–192.

[11] A. Binzenhöfer, D. Staehle, , and R. Henjes, "On the fly estimation of the peer population in a chord-based p2p system," in *19th International Teletraffic Congress (ITC19)*, Beijing, China, sep 2005.

[12] T. D. Ross, "Accurate confidence intervals for binomial proportion and poisson rate estimation," *Computers in Biology and Medicine*, vol. 33, no. 6, pp. 509–531, November 2003.

[13] A. Agresti and B. A. Coull, "Approximate is better than 'exact' for interval estimation of binomial proportions," *The American Statistician*, vol. 52, no. 2, pp. 119–126, 1998.

[14] L. M. Leemis, K. S. Trivedi, H. C. F. P. Burch, H. Multhaup, B. Schmeiser, and L. Vignati, "A comparison of approximate interval estimators for the bernoulli parameter," *The American Statistician*, vol. 50, pp. 63–68, 1996.

[15] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001, pp. 329–350.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*, vol. 31, no. 4. ACM Press, October 2001, pp. 161–172.

[17] J. Cichon, M. Klonowski, L. Krzywiecki, B. Rozanski, and P. Zielinski, "Equalizing chord," Tech. Rep., 2004.

[18] D. R. Karger and M. Ruhl, "Simple efficient load balancing algorithms for peer-to-peer systems," in *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*. New York, NY, USA: ACM, 2004, pp. 36–43.

Digital Object Identifier: 10.4108/ICST.BROADNETS2009.7239
http://dx.doi.org/10.4108/ICST.BROADNETS2009.7239