Broose: a Practical Distributed Hashtable based on the De-Bruijn Topology

Anh-Tuan Gai[*]
INRIA Rocquencourt [†]
anh-tuan.gai@inria.fr

Laurent Viennot[*]
INRIA Rocquencourt [†]
laurent.viennot@inria.fr

Abstract

Broose is a peer-to-peer protocol based on the De-Bruijn topology allowing a distributed hashtable to be maintained in a loose manner. Each association is stored on k nodes to allow higher reliability with regard to node failures. Redundancy is also used when storing contacts avoiding complex topology maintenance for node departures and arrivals. It uses a constant size routing table of O(k) contacts for allowing lookups in O(log N) message exchange (where N is the number of nodes participating). It can also be parameterized for obtaining O(log N / log log N) steps lookups with a routing table of size O(k log N). These bounds hold with high probability. Moreover, the protocol allows load balancing of hotspots of requests for a given key as well as hotspots of key collisions. The goal is to obtain a protocol as practical as Kademlia based on the De-Bruijn topology.

keywords: peer-to-peer, ditributed hashtable, De Bruijn topology.

1. Introduction

Broose is a peer-to-peer protocol based on the De Bruijn topology allowing a distributed hashtable to be maintained in a loose manner. More precisely, conversely to previous distributed hashtables based on the De Bruijn topology [7, 5, 2] and similarly to Kadmelia [6], it stores an association on k nodes instead of one, for getting high reliability with regard to node failures. Similarly to other De Bruijn based hashtables it uses a constant size $O(k)$ routing table instead of $O(k \log N)$ (where N is the number of nodes) for Kademlia. Lookups are then performed by contacting $O(\log N)$ nodes. Similarly as Kademlia, the protocol can be tuned to obtain $O(\log N / \log\log N)$ steps lookups with a routing table of size $O(k \log N)$ instead of $O(k \log^2 N)$ for Kademlia (see Table 1 in Section 4 for a more detailed comparison).

Most protocols for distributed hashtables split the key space among nodes according to their identifiers. This results in a very strict topology which is hard to make reliable with regard to node failures. (This is the case for previous distributed hashtables based on the De Bruijn topology [7, 5, 2, 1].) The major breakthrough of Kademlia [6] is to select the nodes storing an association for a given key in a loose manner: on the k closest nodes to the key for some metric at the moment when the association is inserted. Contacts populating the routing table are also selected in a similar loose manner. The parameter k is tuned according to a probability p_n of node failure within the next hour. Some experiments [6] show that approximately 70 percents of nodes with uptime at least one hour stay connected one more hour. This suggest $p_n < 0.3$ if nodes participate in storing associations only after the first hour of uptime. The authors of Kademlia suggest $k = 20$. The probability that all the nodes storing an association quit the network is then less than 10^{-10}. An association is then republished every hour. (As a node stores all the contacts which are close to its identifier, republication is made locally.)

Kademlia uses a topology similar to the hypercube resulting in a routing table of $O(\log N)$ buckets for lookups in $O(\log N)$ steps. (A bucket stores k contacts which can equivalently participate in a given lookup.) However the same lookup efficiency can be achieved using a constant size routing table with the De Bruijn topology. The De-Bruijn graph over $N = 2^n$ nodes is defined as follows. Every node u with identifier $u[1, n]$ has two successors $s = u[2, n]0$ and $s' = u[2, n]1$ obtained by shifting u to the left ($u \ll 1$) and adding a bit on the right (one of these successors may be u itself), and thus two predecessors $p = 0u[1, n-1]$ and $p = 1u[1, n-1]$. This simply defined graph has the property of having constant in-degree and out-degree 2 and logarithmic diameter n. One can easily find a route from u to any node $v = v_1 \cdots v_n$: $u \rightarrow u[2, n]v_1 \rightarrow u[3, n]v_1v_2 \rightarrow \cdots \rightarrow u[n, n]v_1 \cdots v_{n-1} \rightarrow v$. Note that another route can be similarly found by following edges backward. This topology can be adapted for a varying number N of nodes for getting a very efficient distributed hashtable [7, 5, 2, 1]. Lookups can be made

[*] Research supported by the PairAPair project, ACI Masse de données.

[†] Domaine de Voluceau, B.P.105, 78153 Le Chesnay cedex, France.

by shifting bits to the right from predecessor to predecessor as in [7] or to the left from successor to successor as in [2]. However, these solutions split the key space very strictly among nodes, inducing complex topology maintenance when nodes join and quit the network.

Broose generalizes both approaches (shifting left or right) and offers a very simple routing table consisting of three buckets. As in Kademlia, no topology maintenance is needed with regard to node arrival and departure thanks to redundancy in contacts. Reenforcement of buckets through requests is also achieved by using both types of lookups.

The rest of the paper is organized as follows. Section 2 presents the Broose protocol in detail. Some simulations are presented in Section 3. Finally, the protocol is analyzed and its correctness is proved in Section 4.

2. Broose Protocol

2.1. The xor metric and identifiers

All node identifiers and hash table keys are n bits positive integers. Each node chooses its identifier randomly. n should be sufficiently large for making collisions very unlikely ($n = 128$ or $n = 160$ for example). A key, value association will be stored on the k *closest* nodes, i.e. on the k nodes with closest identifier to the key for the xor metric.

As in Kademlia, we use the xor metric because it measures whether identifiers have long common prefix: two identifiers are at xor distance less than 2^{n-l} if and only if they share at least the same l first bits. (Identifiers are read as positive integers as well as the distance $u \oplus v$ between two identifiers u and v.) The xor distance verifies the triangular inequality since $u \oplus w = (u \oplus v) \oplus (v \oplus w)$ for any u, v, w and $x \oplus y \leq x + y$ for any x, y. An interesting property of the xor metric is that there are exactly x identifiers at xor distance less than x from any given identifier.

For alleviating notations, we will equally denote by u a node u and its identifier $u = u[1, n]$. $u[1]$ is the high order bit of u. $u[i, j] = u[i] \cdots u[j]$ will denote the $j - i + 1$ bits portion of u beginning at position i. If x and y are two bit sequences, xy will denote the sequence obtained by concatenating them. If $x = x[1, i]$ is a bit sequence, $\overline{x} = \overline{x[1]} \cdots \overline{x[i]}$ will denote the sequence where each bit is negated ($x \oplus \overline{x} = 1 \cdots 1$). $x \ll d = x[d + 1, n]0 \cdots 0$ will denote the identifier obtained by shifting x by d bits to the left and padding with zeros.

2.2. Right shifting lookup

Each node maintains two buckets R_0, R_1 storing contacts with identifier close to the right shifted identifier of the node. More precisely, for a node with identifier u:

- R_0 stores the k' closest nodes to $0u[1, n - 1]$,
- R_1 stores the k' closest nodes to $1u[1, n - 1]$.

k' is a protocol parameter lying between $k/2$ and k. See Section 4 for more details. The reader may first assume $k' = k$.

To make a lookup for a key w, a node u first estimates the distance d in number of hops to a node storing w. The idea is to contact any node v_{d-1} in R_{w_d}, and then any node v_{d-2} in the $R_{w_{d-1}}$ bucket of v_{d-1}, and so on until finding some node v_0. If d was chosen sufficiently large, v_0 should have an identifier sufficiently close to w to store information associated to w. The intuition behind this process is that each v_i will have an identifier close to $w[i + 1, d]u[1, n - d + i]$. The bits of w are inserted on the left and shifted to the right until a node sharing a long common prefix with w is reached. (A more detailed proof of correctness is given in Section 4.1.)

As shown in Section 4.1, d can be estimated from R_0 or R_1: if l is the length of the longest common prefix of the identifiers contained in R_0, then $d = l + 1$ is almost surely sufficient. (Notice that l is an estimation of $\log_2 \frac{N}{k'}$.)

More precisely, the right shifting lookup procedure for key w by node u consists in the following process. u initializes a lookup bucket K with $\{u\}$ and estimated distance $d_K = d$ hops. u then repeats the following steps until d_K reaches zero:

- u contacts one to α nodes in K for a right lookup on w at d_K hops, each contacted node should reply with its $R_{w_{d_K}}$ bucket;
- if u receives a reply for d_K hops, K is replaced by the bucket contained in the message and d_K is decremented by one;
- if u receives a reply for $d_K + 1$ hops, the contacts contained in the message are added to K;
- if u receives a reply for more than $d_K + 1$ hops, it ignores the message.

α is a protocol parameter allowing speedup of lookups with regard to node failure. If no answer is received, u may contact α more nodes in K (each node in K should be contacted only once). Notice that the parameter k' is chosen such that it is unlikely that no node in K answers. (K always contains at least k' contacts except for the first iteration where u contacts itself.)

To find one of the k' closest node to the key w, the right shifting lookup should be sufficient. This is often sufficient for finding an existing association with key w. However, if no association has been stored or for storing an association, all the k closest nodes to w must be found.

2.3. Brother lookup

Each node maintains a *brother bucket* B storing contacts with identifiers close to the identifier of the node (called *brothers*):

- B stores the δ closest nodes to u.

The size δ of B is chosen so that one of the k closest nodes to some key w will almost surely know all the k closest nodes to w. We will see in Section 4.1 that $\delta = 7k$ is sufficient.

To make a brother lookup for key w a node u must know a set of nodes K with identifiers close to w. The k closest nodes to w are queried. Each node should answer with the k closest nodes to w in its B bucket. If some node do not answer, u queries further nodes until k nodes answer.

To make a complete lookup, a node first makes a right shifting lookup and terminates with a brother lookup. Any query for a lookup at 0 hops should be considered as a brother lookup. Alternatively, it can begin with a left shifting lookup.

2.4. Left shifting lookup

To allow reenforcement of buckets through requests, a left shifting lookup is provided. It is precisely the reverse of a right shifting lookup.

Each node maintains a *left bucket* L storing contacts with identifier close to the left shifted identifier of the node. More precisely, for a node with identifier u:

- L stores any node v such that u is among the k' closest nodes to $u[1]v[1, n-1]$.

Notice that u can test whether it is among the k' closest nodes to some identifier by computing the k' closest nodes to the identifier in $B \cup \{u\}$. If the bucket is lexicographically sorted, the k' closest nodes to v can be found by scanning the bucket symmetrically around the insertion position of v.

The left shifting lookup procedure is very similar to the right shifting lookup procedure except that each contacted node for a left lookup on w at d_K hops replies with the k' nodes v with $v \ll d_K$ closest to w. As we will see in Section 3 and 4.1 a node should preferentially query the $k'' < k'$ closest nodes to $w \ll d_K$. k'' is a protocol parameter (typically, $k'' \approx k/2$). If these k'' nodes fail to answer, the node may query the α closest nodes to $w \ll d$ (resp. $v \ll -d$) among the remaining $k' - k''$ contacts at the risk of lookup failure with higher probability.

A node v is in the L bucket of u when u should be in one of the R buckets of v. This symmetry implies that right shifting lookups allow L buckets to be refreshed while left shifting lookups allow R buckets to be refreshed. Both lookups procedures should be used equally.

2.5. Unified lookup queries

All types of lookups described above can be unified with the same query format. Each node defines its *right bucket* R as $R = R_0 \cup R_1$. Each node has thus mainly three buckets: R, L and B. Each lookup query message should contain a key w and an estimated hop distance d which is positive, negative or zero. Such a query is called a lookup query for w at d hops. A node receiving a lookup query message should reply with:

- the k' closest contacts to w in B if $d = 0$,
- the k' closest contacts to $w \ll d$ in R if $d > 0$,
- the k' nodes v in L with $v \ll -d$ closest to w if $d < 0$.

A right shifting lookup consists in a sequence of lookup queries with decreasing hop distance and terminates with a lookup query at 0 hops. A left shifting lookup consists in a sequence of lookup queries with increasing hop distance and terminates with a lookup query at 0 hops.

2.6. Accelerated lookups : shifting more than one bit at a time

To minimize traffic and allow faster lookups, the protocol shifts more than one bit at a time. b, a parameter of the protocol, denotes the number of bits shifted. The R bucket of node u thus contains $2^b k'$ nodes: the k' closest nodes to each identifier u_p composed of any prefix p of b bits followed by $u[1, n-b]$. The size of bucket B does not depend on b. We define $u \ll_b i = u \ll bi$ to simplify notations. A node v is in the L bucket of u when u is among the k' closest nodes to $v \ll_b 1$. The average size of L will thus be $2^b k'$. We will see that it is very unlikely that L contains more than $O(2^b k')$ contacts. More precisely for $k' = 20$ and $b = 4$ or $b = 5$ it is very unlikely that a node will have more than $4.3 * 2^b k'$ contacts in its L bucket. (See Section 4.2 for more details.)

With $b = \log\log N$ the routing table size thus becomes $O(k \log N)$ and lookups are performed in less than $\frac{1}{b} \log_2 \frac{N}{k'} + 1 = O(\log N / \log\log N)$ steps as detailed in Section 4.1.

2.7. Physical proximity

Notice that any α nodes among k' (resp. $k/2$) for right (resp. left) shifting lookups are queried in each lookup step. Some heuristic for choosing physically close contacts should be used (for example by selecting contacts with longest common prefix of IP address). More sophisticated strategies as in [8] could eventually be used. A simple heuristic may reside in storing the minimum response time for each neighbor. As several nodes are queried at the same time in each lookup step, we can expect that the closest one answers first. Assuming that the physically closest neighbors from a close node are close too, querying the closest neighbors from the first answering node could be a good heuristic.

2.8. Balancing hotspots

As Kademlia, Broose uses caching for solving hotspots of requests for a given key. When a node performs a lookup for key w and gets an answer when querying node v_i at i hops, it should store the key, value association on the node v_{i-1} that answered for $i-1$ hops. (If a brother lookup was necessary, it stores the association on the node v_0 queried for the brother lookup.) The association is cached during a duration decreasing exponentially with i.

An advantage of the De Bruijn topology is that it also offers a solution for balancing hotspots of key collisions. It may happen that many associations have the same key w. Associations are supposed to be sorted according to some total order of the associated values. The k closest nodes to w will store the A first associations. A is a protocol parameter (for example $A = 1000$). The $2A$ next associations are stored on the k' closest nodes to $w[n-b+1,n]w[1,n-b]$ and the k' closest nodes to $\overline{w[n-b+1,n]}w[1,n-b]$ depending on the first bit of the associated value. Notice that all these nodes are in the R bucket of the k closest nodes to w. (These associations will be replicated k times at the first republication.) The $4k$ next associations are similarly stored on $4k'$ contacts found in the R buckets of these $2k'$ nodes according to the two first bits of the associated value and so on in a binary tree fashion. Notice that storing an association consists in descending this tree according to the first bits of the value. On the other hand, retrieving the $(2^i-1)A$ first associations requires to query all the tree up to depth i. However, this only consists in pushing further ahead a right shifting lookup. Porting this strategy on another topology than De Bruijn would result in (2^i-1) lookups, a cost which is prohibitive. If an importance of associations can be estimated, a good choice for ordering associations would be to place more important associations first.

Both strategies may cohabit. When the retrieval begins with a right shifting lookup, the binary tree is explored in the following way. If the A first associations are found in the cache of a node with an identifier sharing a long common prefix with $w[bi+1,n]$, the $2A$ next associations are searched on nodes with prefix close to $w[n-b+1,n]w[bi+1,n-b]$ and $\overline{w[n-b+1,n]}\,w[bi+1,n-b]$, and so on. If an association block is not found, the tree is searched with root prefix close to $w[b(i-1)+1,n-b]$ and a copy is stored on one of the k' corresponding nodes of the tree rooted at $w[bi+1,n]$. If a block is still not found, the tree rooted at $w[b(i-2)+1,n-2b]$ is searched, and so on until searching the original tree rooted at w. In any case, a copy is cached in the previous tree. When no association exists for some block (due to limited number of associations), an empty block is cached. Notice that the necessary contacts are always found in the R buckets of queried nodes.

A similar strategy can be used for exploring the binary tree when the retrieval first began with a left shifting lookup. If the A first associations are found in the cache of a node with identifier close to $u[1,bi]\ w[1,n-bi]$, the $2A$ next associations are searched on $w[n-b+1,n]u[1,bi]w[1,n-b(i+1)]$ and $\overline{w[n-b+1,n]}\,u[1,bi]w[1,n-b(i+1)]$, and so on. A similar tree searching is then performed as for right shifting lookups using R buckets. L buckets allow the selection of a tree closer to the original tree rooted at w.

2.9. Node insertion and bucket creation

A new node u must know an entry point: node v. R is constructed by performing 2^b complete lookups starting with a lookup bucket $K = \{v\}$, one for each identifier u_p composed of any prefix p of b bits followed by $u[1,n-b]$. B can then be constructed with L buckets of nodes in R and L can be constructed with L buckets of nodes in B.

Alternatively, B can be initialized with the k closest nodes to the own identifier of u. Let $u[1,l]$ be the longest common prefix of these k nodes. B is then further completed with the k closest nodes to $u[1,l-1]\overline{u[l]}u[l+1,n]$. Retrieving the B buckets of these $2k$ nodes should be sufficient.

As soon as a node has its B and R buckets initialized, it can participate to the network and let L be constructed online. However, L could be constructed from scratch by exploring the prefix trie of identifiers rooted at $u[b+1,l]$. The details of such construction cannot be included here due to space limitation.

2.10. Refresh policy

Broose policy for refreshing buckets is similar to Kademlia policy for the k closest nodes. Intuitively, Broose only stores close contacts (allowing routing table size to be reduced) allowing few choice for contacts. On the other hand, Kademlia has a different policy (keeping contacts with long uptime) for long range contacts.

As soon as a new alive contact is discovered in one of the bucket range during any message exchange, it should be inserted in the corresponding bucket. This continuous process allows new contacts to be discovered. However, node departure is harder to detect. A possibility is to ping contacts when they have not been refreshed during a certain period of time. To reduce the ping traffic, each lookup query for a node v_i at hop distance i could contain the identifier of the node v_{i+1} that has responded at the previous step for hop distance $i+1$.

Alternatively, a node may periodically repeat the procedure for constructing its buckets from scratch.

Similarly to Kademlia, an association is republished every hour. This is done efficiently thanks to a brother lookup. To avoid redundant re-publications a node republishes an association if no other node has republished the association during the previous hour and it is still among the k closest nodes to the key. To allow association expiration, an association is republished at most 24 times, and the source of an association must republish it every day.

3. Simulation

The size of buckets R and L depends on b and k'. A large b ($b = 4$ or $b = 5$) is necessary for speeding up lookups. A small k' ($k' < k$) allows the size of buckets to be minimized. A large value for k' makes the system more robust. We propose some simulations for finding the best compromise for k'. (We will use $b = 4$.) A critical situation occurs when a large fraction of nodes changes during the refresh time period. We make simulations where nodes leave the network at the same rate as new nodes enter it. We start with a one million nodes network ($N = 10^6$) in a stable situation (i.e. buckets are accurate). Then rN nodes are deleted and rN inserted. r denotes the node renewal fraction . The arrival and departure process are continuous.

We distinguish three different types of nodes: dead, old and new. The rN first arrived nodes are dead, and the rN last arrived nodes are new.

- Old nodes are not aware of dead nodes departure. An old node u have a new node v in its buckets with probability $p_v = \frac{rN-a}{rN}$. a denotes the arrival position of v. (An old node has more chances to learn about new nodes with longer uptime.)

- A new node u considers a dead node v still alive if v leaves the system after the arrival of u. u knows new nodes inserted before itself in addition to old nodes. u also knows a new node v inserted after itself with probability p_v.

- A lookup fails if the k' nodes returned during a lookup step are dead nodes, or if none of the k' final nodes are among the k closest to the key. (No brother lookup is performed at the end.)

This model grabs the effects of inconsistency between nodes: each node has its own view of the network.

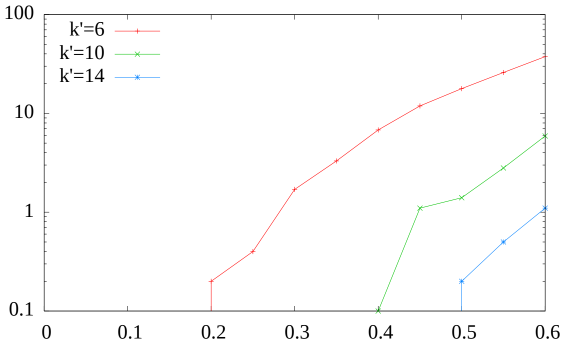

Figure 1: Percentage of lookup failures as a function of the node renewal fraction r for $k' = 6, 10, 12, 15$ when the furthest alive contact at hop distance i from the shifted key $w \ll_b i$ is selected at each lookup step.

Figure 1 shows the percent of right shifting lookup failures as a function of r for different values of k'. (1000 simulations of lookups are performed for each ratio and for each curve.) Notice that a log-scale is used for percentages. These simulations are further pessimistic since the worst alive contact with respect to the bits of the key is selected at each step among the k' known contacts. For left shifting lookups, the worst contact among the k'' best contacts is selected (if these k'' contacts are dead, the best one among known alive contacts is selected). The parameters have been tuned for a ratio $r < 0.3$ (yielding a probability $p_n = 0.3$ of node failure). However, a larger ratio is needed for being able to observe some failures. Note that a failure was observed for $k' = 15$ only for $r = 0.6$. As a comparison $r^{k'}$ becomes greater than $1/1000$ for $r = 0.3$ when $k' = 6$, $r = 0.5$ when $k' = 10$, $r = 0.63$ when $k' = 15$. This is consistent with the expected probability of failure $r^{k'}$ shown in Section 4.1. Much better results are obtained with a random choice of contacts and a final brother lookup.

Figure 2 shows the percentage of left shifting lookup failures as a function of r for $k' = 15$. The simulation assumes that the worst alive contact among the k'' closest

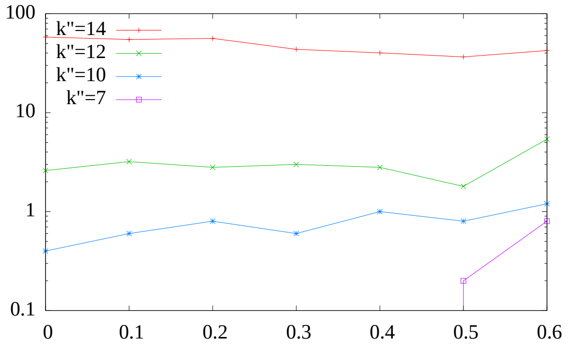

Figure 2: Percentage of lookups failures as a function of the node renewal fraction r for $k' = 15$ when the k''th furthest alive contact (with respect to the key) is selected for the next lookup step for $k" = 7, 9, 11$.

known nodes (with respect to the bits of the key) is chosen at each step. We observe that left shifting lookups are less reliable than right shifting lookups when the furthest contacts are chosen. Choosing the worst alive node among the k' closest nodes gives poor results. However, satisfying results are obtained when the k'' closest contacts are preferred with $k'' < k'$. The proof in Section 4.1 will give some hints about the reasons for that. For low r, this is the main reason for lookup failure. For large r, the probability of loosing the k'' closest contacts becomes preponderant.

As we are going to see in the following section, a larger value of b would also enhance reliability.

4. Protocol analysis

4.1. Correctness of lookup procedures

We are going to show that the probability that a lookup fails is in the same order of magnitude that the probability that the k nodes storing the information fail. If p_n is the probability that a node fails, the probability that k nodes fail is p_n^k. (We will give numerical values for protocol parameters assuming $p_n = 0.3$ and $k = 20$.)

However, Broose has two independent lookup procedures. We are going to show that a right shifting lookup fails with probability less than $p_n^{k'}$ and that a left shifting lookup fails with probability less than $p_n^{k-k'}$ (for appropriate value of k''). This avoids to double the size of routing tables and still ensures that the probability of failure of both lookup procedures is approximately less than p_n^k.

Our proofs are mainly based on the fact that nodes choose their identifier randomly and that the probability of choosing an identifier of n bits at xor distance less than x from a given identifier is $x/2^n$. (This is due to the fact that there are exactly x positive integers at xor distance less than x.) These properties allow the following lemma which is a direct implication of Chernoff bounds [4].

Lemma 1 *Consider a normalized distance $x = d_N(\mu)$ defined by $d_N(\mu) = \mu * 2^n/N$ and an identifier u (N is the number of nodes). Then the number N_x of nodes at distance less than x from u is $\Theta(\mu)$ with high probability. More precisely, there exist some increasing functions f_+ and f_- such that $P[N_x \geq m] < \exp(-\mu f_+(m/\mu - 1))$ and $P[N_x \leq m'] < \exp(-\mu f_-(1 - m'/\mu))$.*

Notice that the average number of nodes at distance x is $E[N_x] = \mu$ since the probability that a random identifier falls at distance less than x is μ/N. When $\mu = \log N$, the above probabilities thus get smaller than $1/N^r$ where r is a constant depending on the values of f_+ and f_-. (Note that $k = 20$ for example is greater than $\log N$ for $N \leq 10^8$.)

The Chernoff bounds state that there exist f_+ and f_- such that $P[N_x \geq (1+\epsilon)\mu] < \exp(-f_+(\epsilon)\mu)$ and $P[N_x \leq (1-\epsilon)\mu] < \exp(-f_-(\epsilon)\mu)$. The bounds of the lemma are obtained for $\epsilon = m/\mu - 1$ and $\epsilon = 1 - m'/\mu$ respectively. The classical Chernoff bounds use $f_+(\epsilon) = \epsilon^2/2$ and $f_-(\epsilon) = \epsilon^2/3$. However, we will use the sharper bounds obtained with $f_+(\epsilon) = (1+\epsilon)\log(1+\epsilon) - \epsilon$ and $f_-(\epsilon) = (1-\epsilon)\log(1-\epsilon) + \epsilon$. See [4] for more details about Chernoff bounds. Notice finally that the above probabilities fall down exponentionally as μ increases.

As a first application of Lemma 1, consider the nodes at distance less than $d_N(ck) = ck * 2^n/N$ from some identifier for some constant $c > 1$. The probability p_c that there are less than k such nodes is bounded by $\exp(-ckf_-(1 - 1/c))$. For $k = 20$, we get $p_c < 0.3^{20}$ for $c = 3.5$. In the sequel, let c denote the constant such that $p_c < p_n^k$ (we will use $c = 3.5$ for numerical applications).

Let $l = \lceil \log_2 \frac{N}{ck} \rceil$ denote the prefix length such that $\frac{1}{2}\frac{N}{ck} < 2^l \leq \frac{N}{ck}$. There exists almost surely at least k nodes whose identifier shares the l first bits of any given key w. (It is equivalent to share the l first bits of w and to be at xor distance less than $2^{n-l} \geq ck * 2^n/N = d_N(ck)$ from w.)

Brother lookup. First consider the B bucket of a node u. We have to prove that knowing the δ closest nodes to u is sufficient for knowing the k closest nodes to some key w when u is one of the k closest nodes to w. Almost surely, the k closest nodes to w share the same prefix $w[1,l]$. If there are less than δ nodes with prefix $w[1,l]$, then B contains all the nodes sharing this prefix including u and the k closest nodes to w.

Now suppose that more than δ nodes have prefix $w[1,l]$. Then we can show that almost surely, more than k nodes have prefix $w[1,l+1]$. Consider δ nodes with prefix $w[1,l]$.

As their $l+1$th bit is random they have prefix $w[1,l+1]$ with probability $1/2$. Applying the lower Chernoff bound, the probability that less than k nodes have prefix $w[1,l+1]$ is less than $\exp(-\mu f_-(1-k/\mu))$ with $\mu=\delta/2$. It is typically very low for $\delta=2ck$. (At least, it is less than 0.3^{20} for $\delta=7k$ with $k=20$.)

Now suppose that more than k nodes have prefix $w[1,l+1]$. Again if there are less than δ nodes with prefix $w[1,l+1]$, then B contains all the nodes sharing this prefix. The probability that B does not contain the k closest nodes to w is thus bounded by the probability that there are more than δ nodes at distance less than $2^{n-l-1} < d_N(ck) = ck * 2^n/N$ from w. This probability is again bounded thanks to Lemma 1. It is also very low for $\delta=2ck$. (It is less than 0.3^{20} for $\delta=7k$ with $k=20$.)

In any case, we have proved that the probability that B does not contain the k closest nodes to w is very low. (Take $\delta=7k$ when $k=20$ and $p_n=0.3$.)

Right shifting lookup. Now consider a right shifting lookup procedure from u for a key w. With probability greater than $1-p_n^{k'}$, a node v_i will answer for each iteration at an estimated distance of i hops. Consider this sequence $u=v_d,\dots,v_0$ of nodes that answer.

Let us first show that v_i shares the $b(d-i)$ first bits of $w \ll_b i$ as long as at least k' nodes share this prefix. This is true for v_d (empty matching prefix). Suppose $v_i[1,b(d-i)] = (w \ll_b i)[1,b(d-i)] = w[bi+1,bd]$. The R bucket of v_i contains the k' closest nodes to $w[b(i-1)+1,bi]v_i[1,n-b]$ and v_{i-1} is one of them. If there are at least k' nodes with prefix $w[b(i-1)+1,bd] = w[b(i-1)+1,bi]v_i[1,b(d-i)]$, then the k' closest nodes to $w[b(i-1)+1,bi]v_i[1,n-b]$ must share this prefix implying that v_{i-1} has prefix $w[b(i-1)+1,bd]$. The above property is thus true by induction.

Consider the first index i_m such that less than k' nodes have prefix $w[bi_m,bd-1]$. If d was chosen sufficiently large, $i_m \geq 1$. Indeed, the probability that less than k' nodes have prefix $w[1,l]$ (where $l=\lceil\log_2\frac{N}{ck}\rceil$) is less than p_n^k. As there are at least k nodes with a given prefix of l bits with rather high probability, v_{i_m} shares the l first bits of $w \ll_b i_m$. For the same reason, v_i will share the l first bits of $w \ll_b i$ for $i_m \geq i \geq 1$. v_1 thus shares the l first bits of $w \ll_b 1$ with high probability.

Finally, with probability less than $1-O(p_n^k)$, v_0 is among the k closest nodes to $w[1,b]v_1[1,n-b]$ which shares $l+b$ bits with w. We can then use the following arguments of the proof concerning brother lookups. If v_0 shares at least $l+1$ bits with w, then the B bucket of v_0 almost surely contains the k closest nodes to w. As the k closest nodes to w almost surely share the prefix $w[1,l]$, the final step may thus fail only if there are less than k nodes with prefix $w[1,l+1]$ and more than δ nodes with prefix $w[1,l]$ which happens again with probability less than p_n^k. This achieves the proof of right shifting lookups correctness.

Notice that we can deduce from this proof an estimation of d: $b(d-i_m) \geq l$ and $i_m \geq 1$ allow initiation of the proof. $d \geq 1+\frac{l}{b}$ is thus sufficient. Notice also that the length of the longest prefix of the k closest nodes to a node u is greater or equal to l with high probability. u may thus obtain an upper bound of l from its B bucket. A better bound can even be obtained from the R bucket: as $k' \leq k$, the k' closest nodes to some identifier share the same l first bits with high probability. For each prefix of b bits, an estimation of an upper bound of l can be obtained. d can be computed from the smallest estimation.

Left shifting lookup. Due to space limitations, the proof of correctness of left shifting lookups is omitted here. However, the arguments are similar to previous proofs and the interested reader may consult [3]. For $k''=9$ it can be shown that the appropriate value of k' is $k'=18$ when $b=3$, $k'=15$ when $b=4$ and $k'=14$ when $b=5$. Again, the hop distance from the first node u performing the lookup can be estimated from the B bucket : d should be sufficiently large so that u is among the k'' closest nodes to $u[1,d]w[1,n-d]$. We have $d \leq \lceil\frac{1}{b}\log_2\frac{N}{k''}\rceil+1$ with high probability.

4.2. Routing table size

b	$7k$	$2^b k$	k'	$2^b k'$	Broose	Kademlia
1	140	40	20	40	180	400
2	140	80	20	80	220	600
3	140	160	18	144	428	933
4	140	320	15	240	620	1500
5	140	640	14	448	1036	2480

Table 1: Average number of contacts for Broose and Kademlia for various values of b with $k=20$ and $k''=9$.

The L bucket will contain an average of $2^b k'$ contacts. With similar arguments as before we can prove that it will contain more than $c2^b k'$ contacts (for some small c) with probability less than $p_n^{k'}$. More precisely, for $k'=15$ and $b=4$ (resp. $k'=14$ and $b=5$), we get $c=4.3$ (resp. $c=4$). and less than one percent of the nodes will have more than $2.4 * 2^b k'$ contacts. The R bucket contains exactly $2^b k'$ contacts.

Table 1 compares the average number of contacts for Broose and Kademlia for various values of b. Kademlia uses a parameter similar to b (identifiers are considered by chunks of b bits) allowing similar lookup complexity. For $b < 3$, Broose should only use the R buckets since the L bucket becomes reliable for $b \geq 3$. For $b \geq 3$, the best values of k' according to the proof of correctness of left shifting lookup are used for $k'' = 9$.

5. Conclusion

With it's novel symmetricized De Bruijn topology and its optimized contact list, Broose improves the loose framework for distributed hashtables introduced by Kademlia. We have shown how Broose can obtain significantly smaller routing tables than Kademlia. It has been proven that lookups succeed with high probability under the model of constant node failure probability and bucket consistency. This proof is confirmed by simulations for some degree of bucket inconsistency between nodes with a continuous node arrival and departure model.

Broose allows physical proximity to be taken into account. However, further work is needed to estimates how much it can gain from this flexibility with regard to physical proximity. Finally, Broose is the first peer-to-peer system introducing a solution for balancing key collision hotspots and is thus a good candidate for peer-to-peer file sharing with keyword indexing.

References

[1] I. Abraham, B. Awerbuck, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversial scenarios. In *17th International Parallel and Distributed Processing Symposium (IPDPS'2003)*, april 2003. Nice.

[2] P. Fraigniaud and P. Gauron. The content-addressable network d2b. Technical Report LRI 1349, Univ. Paris-Sud, 2003.

[3] A.T. Gai and L. Viennot. Broose: A practical distributed hashtable based on the de-brujin topology. Technical report, INRIA, june 2004.

[4] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Inform. Process. Lett.*, 33(6):305–308, Feb. 1990.

[5] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[6] P. Maymounkov and D. Mazieres. Kademlia: A peerto -peer information system based on the xor metric. In *1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[7] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, 2003.

[8] Keith W. Ross, Ernst W. Biersack, Pascal Felber, Luis Garces-Erice, and Guillaume Urvoy-Keller. Topology-centric look-up service. In *Proceedings of COST264 Fifth International Workshop on Networked Group Communications*, 2003.