7 Decentralized overlay networks I

In the next two sections we present overlay networks that are completely decentralized, i.e., they do not depend on a supervisor. We assume that, in principle, every peer has the right to initiate the integration of new peers into the system and that every peer knows at least one peer currently in the system so that publicly available entry points such as a supervisor are not necessary any more. In this section, we will focus on overlay networks that are based on the continuous-discrete approach [5], and in the next section overlay networks are presented that are based on so-called skip graphs [2]. First, we will first assume that all peers are reliable and honest, and later we will show how to remove this assumption.

7.1 Virtual space management

Many decentralized peer-to-peer systems are based on the concept of a virtual space. That is, we are given a space U and every peer v is associated with a region $R(v) \subseteq U$ so that $\bigcup_{v \in V} R(v) = U$ for the current set of peers V. This property has to be maintained while peers join and leave the system. A very general concept for doing this is the hierarchical decomposition approach (recall Section 3 or 5).

Hierarchical decomposition

We assume that there is a generic way of recursively cutting U in half. In order to simplify the presentation, we assume that $U = [0, 1)^d$ for some fixed $d \geq 1$. The *decomposition tree* $T(U)$ of U is an infinite binary tree in which the root represents U and for every node v representing a subcube U' in U, the children of v represent two subcubes U'' and U''', where U'' and U''' are the result of cutting U' in the middle at the smallest dimension in which U' has a maximum side length. The subcubes U'' and U''' are closed, i.e., their intersection gives the cut. Let every edge to a left child in $T(U)$ be labeled with 0 and every edge to a right child in $T(U)$ be labeled with 1. Then the label of a node v, $\ell(v)$, is the sequence of all edge labels encountered when moving along the unique path from the root of $T(U)$ downwards to v. For $d = 2$, the result of this decomposition is shown in Figure 1.

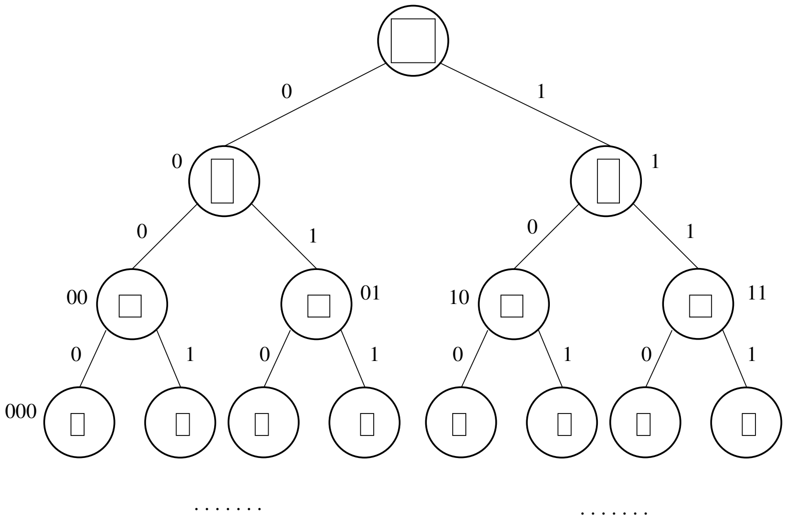

Figure 1: The decomposition tree for $d = 2$.

The goal is to map the peers to nodes in $T(U)$ so that the following conditions are met:

Condition 7.1

(1) *The interiors of the subcubes associated with the (tree nodes assigned to the) peers are disjoint,*

(2) *the union of the subcubes of the peers gives the entire set U.*

In order preserve this condition, the following is done when peers join or leave the system.

Joining the system

When a new peer p joins the system, it follows down the decomposition tree (which is simulated by a proper routing scheme in the given overlay network) according to its (random or pseudo-random) label $\ell(p)$ until it arrives at some node v that is currently occupied by peer q. Then q is moved to node $v0$ and p is moved to node $v1$ of the decomposition tree (i.e., q splits its region into two pieces and gives one of them to p) so that Condition 7.1 still holds.

Leaving the system

When a peer p at node v in the decomposition tree leaves, the peer at the lowest position in the decomposition tree that is reachable from the sibling of v, say q, is taken to replace the position of p (and thereby takes over its region). q must have a peer q' at its prior sibling position in the decomposition tree which is moved to its former parent to take over q's old region.

Given the two rules, one can show the following result for $U = [0, 1)^d$ [1].

Lemma 7.2 *Suppose that there are n peers in the system. If the join-leave activity of the peers is independent of their labels, then the level of every peer in the decomposition tree is within $\log n \pm (\log \log n + O(1))$, w.h.p.*

The lemma implies that the sizes of the regions assigned to the peers only differ by a factor of $O(\log n)$. Often, using the decomposition tree approach is overly complicated, especially when $U = [0, 1)$. In this case, a much simpler strategy is to use the consistent hashing approach in order to partition $[0, 1)$ among the peers:

- Every peer p is assigned to some random point $x_p \in [0, 1)$.
- Every peer p is responsible for the region $R_p = [x_p, \mathrm{succ}(x_p))$ where $\mathrm{succ}(x_p)$ is the closest point succeeding x_p in $[0, 1)$ that is occupied by a peer.

Using this rule, it is obvious that the regions are pairwise disjoint and that $\bigcup_{R_v} = [0, 1)$, which is necessary for applying the continuous-discrete approach. For this strategy, it holds:

Lemma 7.3 *Suppose that there are n peers in the system. If the join-leave activity of the peers is independent of their positions, then every peer is responsible for a region of size at least $\Omega(1/n^3)$ and most $O(\log n/n)$, w.h.p.*

Proof. We first prove the upper bound. Consider any interval I of size $(c \ln n)/n$ for some sufficiently large constant $c > 0$. The probability that none of the peers has its point in I is equal to

$$\left(1 - \frac{c \ln n}{n}\right)^n \leq e^{-((c \ln n)/n) \cdot n} = e^{-c \ln n} = n^{-c} .$$

Hence, when partitioning $[0,1)$ into $n/(c \log n)$ such intervals, every one of these has at least one point in them, w.h.p. Thus, a peer can be responsible for a region of size at most $O(\log n/n)$, w.h.p.

Next we prove the lower bound. The probability that any two peer positions have a distance of less than $1/n^3$ is at most

$$\binom{n}{2} \frac{1}{n^3} \leq \frac{1}{2n}$$

Hence, the probability is very low that such a case occurs, completing the proof. □

The upper bound is acceptable though much better results can be achieved when performing local load balancing. In fact, evenly balancing the region size among the $\Theta(\log n)$ closest peers in $[0,1)$ would ensure that the region size of every peer is $\Theta(1/n)$, w.h.p., which is implied by the following lemma.

Lemma 7.4 *Given n peers, every interval of size $\Theta(\log n/n)$ has $\Theta(\log n)$ peers in it, w.h.p.*

Proof. Consider some fixed interval I of size $(c \ln n)/n$ for some sufficiently large constant $c > 0$. For every peer v let the binary random variable X_v be 1 if and only if $x_v \in I$. Let $X = \sum_{v \in V} X_v$. It holds that

$$\mathrm{E}[X_v] = \Pr[X_v = 1] = \frac{c \ln n}{n}$$

and from the linearity of expectation it follows that

$$\mathrm{E}[X] = \sum_{v \in V} \mathrm{E}[X_v] = n \cdot \frac{c \ln n}{n} = c \ln n .$$

Hence, when using the well-known Chernoff bounds, we obtain that

$$\Pr[X \geq (1+\epsilon)\mathrm{E}[X]] \leq e^{-\epsilon^2 \mathrm{E}[X]/3} = e^{-\epsilon^2 c \ln n/3} = n^{-\epsilon^2 c/3}$$

and

$$\Pr[X \leq (1-\epsilon)\mathrm{E}[X]] \leq e^{-\epsilon^2 \mathrm{E}[X]/2} = n^{-\epsilon^2 c/3}$$

for all $0 \leq \epsilon \leq 1$. Thus, the probability is polynomially small in n that the bound in the lemma is violated. □

7.2 The continuous-discrete approach

The basic idea underlying the continuous-discrete approach [5] is to define a continuous model of graphs and to apply this continuous model to the discrete setting of a finite set of peers. A well-known peer-to-peer system that uses an approach closely related to the continuous-discrete approach is Chord [6].

Consider any space U, and suppose that we have a (possibly infinite) collection F of functions $f_i : U \to U$. Let

$$E_F = \{\{x, y\} \in U^2 \mid \exists i : \; y = f_i(x)\}$$

Then (U, E_F) can be seen as an undirected graph on an infinite number of nodes. For any set $S \subseteq U$ let $\Gamma(S) = \{y \in U \setminus S \mid \exists x \in S : \; \{x, y\} \in E_F\}$ be the neighbor set of S (i.e., all points y with $y = f_i(x)$ or $x = f_i(y)$ for some i since we consider undirected edges). If $\Gamma(S) \neq \emptyset$ for every $S \subset U$, then F is said to be *mixing*. If F does not mix, then there are disconnected areas in U.

Consider now any set of peers V, and let $R(v)$ be the subset in U that has been assigned to peer v (for example, by using the hierarchical decomposition approach). Then the following continuous-discrete condition has to be met:

Condition 7.5 *For every pair of peers v and w, v is connected to w if and only if there are two points $x, y \in U$ with $x \in R(v)$, $y \in R(w)$ and $(x, y) \in E_F$.*

Let $G_F(V)$ be the graph resulting from this condition. Then the following result holds.

Lemma 7.6 *If F is mixing and $\cup_v S(v) = U$, then $G_F(V)$ is strongly connected.*

Proof. Suppose that F is mixing and $\bigcup_{v \in V} R(v) = U$ but $G_F(V)$ is not connected. Then there must be a set $V' \subset V$ that has no edge leaving it. Let $R' = \bigcup_{v \in V'} R(v)$ and $R'' = \bigcup_{v \in V \setminus V'} R(v)$. Since $\Gamma(R') \neq \emptyset$ and $\Gamma(R') \subseteq R''$, there must exist an $x \in R'$ and a $y \in R''$ with $\{x, y\} \in E_F$. Hence, according to our definition of $G_F(V)$, there must exist a node $v \in V'$ and a node $w \in V \setminus V'$ with $(v, w) \in E$, contradicting our assumption. $\square$

Hence, it is important to make sure that F is mixing and that $\cup_v R(v) = U$. The continuous-discrete approach has the following advantage.

Fact 7.7 *When using the continuous-discrete approach together with consistent hashing or hierarchical decomposition, then for any set of functions F it holds: For each join request of some peer p, p only has to contact one old peer (namely, the one containing its region) for a region update and to learn about all of its connections in the system. For each leave request, at most two other peers have to be contacted in order to update their regions.*

This ensures that the continuous-discrete approach is highly scalable, as long as F is chosen so that the peers have at most polylogarithmic degree. Next we consider specific examples of decentralized overlay networks based on $U = [0, 1)$. We start with the dynamic hypercube, and then we consider the dynamic de Bruijn network.

7.3 The dynamic hypercube

Recall the definition of the d-dimensional hypercube. Let V be its node set and E be its edge set. All nodes $v \in V$ have labels $(v_1, \ldots, v_d) \in \{0, 1\}^d$, and two nodes v and w are connected if and only if $H(v, w) = 1$. When associating each node v with the point $x_v = \sum_{i=1}^{k} v_i / 2^i \in [0, 1)$ and letting $d \to \infty$, then $V = [0, 1)$ and E is determined by the set F_H of functions f_i for all $i \geq 1$ with $f_i(x) = x \oplus 1/2^i \pmod 1$ where $\oplus$ is the bit-wise XOR of the binary representations of x and $1/2^i$, i.e., the i-th bit in x is reversed. Let F be the set of all functions f_i^- and f_i^+ with $f_i^-(x) = x - 1/2^i \bmod 1$

and $f_i^+(x) = x + 1/2^i \bmod 1$. Then for every $x \in [0,1)$, either $f_i^-(x) = f_i(x)$ or $f_i^+(x) = f_i(x)$, so F can be seen as a superset of F_H. For simplicity, we will view $([0,1), E_F)$ as the continuous form of the hypercube. Our choice of F ensures that for each $(x,y) \in E_F$ also $(y,x) \in E_F$.

Consider using the consistent hashing approach in order to partition $[0,1)$ among the peers. The dynamic hypercube for the decentralized case is based on the continuous-discrete approach together with a cycle connecting each peer to its predecessor and successor in $[0,1)$. If the peers are assigned to random points in $[0,1)$, the topological properties of hypercubes together with Lemmas 7.3 and 7.4 imply the following result.

Lemma 7.8 *Given n peers, the dynamic hypercube has a maximum degree of $O(\log^2 n)$, w.h.p.*

Routing in a dynamic hypercube

Suppose that we want to route a message from point x to point y in $[0,1)$. Let $(x_1, x_2, \ldots)$ be the binary representation of x and $(y_1, y_2, \ldots)$ be the binary representation of y. Then we use the following continuous strategy to route the message from x to y:

$$(x_1, x_2, \ldots) \rightarrow (y_1, x_2, \ldots) \rightarrow (y_1, y_2, x_3, \ldots) \rightarrow \ldots$$

If x and y have infinite binary representations, then this strategy may take an infinite amount of hops, but in the discrete world with a finite number of peers, this is not the case with the following discrete variant of the continuous routing strategy above:

The message starts at the peer v_0 responsible for x. Peer v_0 forwards the message to the peer v_1 responsible for $(y_1, x_2, \ldots)$, peer v_1 forwards it to the peer v_2 responsible for $(y_1, y_2, x_3, \ldots)$, and so on, until the message reaches a peer v_ℓ whose region or whose neighboring region contains y. From this peer the message is forwarded to the peer responsible for the region containing y.

Notice that the maximal remaining distance to y shrinks by a factor 2 in each step. Hence, once a distance equal to the smallest region is reached, the routing terminates. Thus, the following theorem immediately follows from Lemma 7.3.

Theorem 7.9 *Using the continuous-discrete routing strategy, it takes at most $O(\log n)$ hops until a message is routed from any point x to any point y in $[0,1)$.*

Besides having a small dilation, it is also important to have a small congestion, i.e., when routing multiple messages, the maximum number of messages to be handled by a peer should be as close to optimal as possible. In order to achieve a low congestion, the following routing strategy may be used by any peer, which is a continuous version of Valiant's trick:

Suppose that a peer with position x wants to send a message to position y. Then it chooses a random point $z \in [0,1)$, first routes the message from x to z and then from z to y using the continuous-discrete routing strategy above.

With this strategy, we obtain the following theorem.

Theorem 7.10 *For every permutation routing problem, the congestion caused when using the extended continuous-discrete routing strategy above is at most $O(\log^2 n)$, w.h.p.*

Proof. Consider any permutation routing problem π, and consider cutting $[0,1)$ into n' intervals of size $1/n'$ starting at integral multiples of $1/n'$ where n' is chosen so that n' is a power of 2 and $1/n' = (c \ln n)/n$ for some suitably chosen constant c. It follows from Lemma 7.4 that every interval has $O(\log n')$ packets starting at it and $O(\log n')$ packets aiming for it. Viewing these intervals as the nodes of a $\log n'$-dimensional hypercube, it follows from the analysis of Valiant's trick that at most $O(\log^2 n')$ packets pass every node, w.h.p. Since, according to Lemma 7.3, every peer is responsible for an interval of size at most $O(\log n/n)$, this implies that every peer is passed by at most $O(\log^2 n)$ packets, w.h.p., which proves the theorem. □

Joining and leaving a dynamic hypercube

We only consider isolated executions of join and leave requests because otherwise it can be quite tricky to correctly update the network.

Suppose that a new peer v contacts some peer w already in the system to join the system. Then v's request is first sent to the peer u owning x_v using the continuous-discrete routing strategy, which only takes $O(\log n)$ hops according to Theorem 7.9. u forwards information about all of its outgoing edges to v, deletes all edges that it does not need any more, and informs the corresponding endpoints about this. Because $R(v) \subseteq R(u)$ for the old $R(u)$, the edges reported to v are a superset of the edges that it needs to establish. v checks which of the edges are relevant for it, informs the other endpoint for each relevant edge, and removes the others.

If a peer v wants to leave the network, it simply forwards all of its outgoing edges to the peer at $\mathrm{pred}(x_v)$. That peer will then merge these edges with its existing edges and notify the endpoints of these edges about the changes.

We know from Theorem 7.9 that the routing part only takes $O(\log n)$ hops. Furthermore, Lemmas 7.3 and 7.4 imply that every peer has at most $O(\log^2 n)$ incoming and outgoing edges. Hence, we obtain the following theorem.

Theorem 7.11 *Join and leave require at most $O(\log^2 n)$ work and $O(\log n)$ communication rounds, w.h.p.*

Data management

Suppose that we want to store data in the dynamic hypercube. Here we can simply use the consistent hashing strategy in Section 4: data items are hashed to random values in $[0,1)$ using a pseudo-random hash function h (which is known to all peers), and every data item d is stored in the peer v with $h(d) \in R_v$.

Using this strategy, data will, on expectation, be evenly distributed among the peers, and on expectation, at most a factor of 2 more data than necessary has to be replaced if a node joins or leaves. (Recall the section on hashing.)

7.4 The dynamic de Bruijn network

Recall the definition of the continuous de Bruijn graph. According to this definition,

- $U = [0,1)$ and

- $F = \{f_0, f_1\}$ with $f_0(x) = x/2$ and $f_1(x) = (1+x)/2$.

The dynamic de Bruijn graph for the decentralized case is based on the continuous-discrete approach together with a cycle connecting each peer to its predecessor and successor in $[0, 1)$. It follows from Lemmas 7.3 and 7.4:

Lemma 7.12 *Given n peers, the dynamic de Bruijn network has a maximum degree of $O(\log n)$ and a diameter of $O(\log n)$, w.h.p.*

Routing in a dynamic de Bruijn network

This is done in the same way as described for the supervised de Bruijn graph.

Joining and leaving a dynamic de Bruijn network

Suppose that a new peer v contacts some peer w already in the system to join the system. Then v's request is first sent to the peer u owning x_v using the continuous-discrete routing strategy above, which only takes $O(\log n)$ hops according to Section 6.6. u forwards information about all of its (incoming and) outgoing edges to v, deletes all edges that it does not need any more, and informs the corresponding endpoints about this. Because $R(v) \subseteq R(u)$ for the old $R(u)$, the edges reported to v are a superset of the edges that it needs to establish. v checks which of the edges are relevant for it, informs the other endpoint for each relevant edge, and removes the others.

If a node v wants to leave the network, it simply forwards all of its outgoing edges to the peer at $\mathrm{pred}(x_v)$. That peer will then merge these edges with its existing edges and notifies the endpoints of these edges about the changes.

We know from Section 6.6 that the routing part only takes $O(\log n)$ hops. Furthermore, Lemmas 7.3 and 7.4 imply that every peer has at most $O(\log n)$ outgoing edges. Hence, we obtain the following theorem.

Theorem 7.13 *Join and leave take at most $O(\log n)$ work and $O(\log n)$ communication rounds, w.h.p.*

Also the dynamic de Bruijn network can be used for data management with the help of the consistent hashing approach.

Dynamic Gabber-Galil graph

Recall the Gabber-Galil graph with parameter n. For this graph, $V = [n]^2$ and E consists of all edges $\{(x, y), (x', y')\}$ with

$$(x', y') \in \{(x, x+y), (x, x+y+1), (x+y, y), (x+y+1, y)\} \pmod{n}$$

When transforming V into $\{i/n \mid i \in \{0, \ldots, n-1\}\}^2$ and letting $n \to \infty$, then we obtain an node set $V = [0, 1)^2$ and an edge set E_F specified by the functions $f_1(x, y) = (x, x+y) \bmod 1$, $f_2(x, y) = (x+y, y) \bmod 1$, $f_3(x, y) = f_1^{-1}(x, y) = (x, y-x) \bmod 1$ and $f_4(x, y) = f_2^{-1}(x, y) = (x-y, y) \bmod 1$. Thus, $([0, 1)^2, E_F)$ can be seen as a continuous version of the Gabber-Galil graph. One can show the following result:

Theorem 7.14 *Suppose that we have n peers. When using random labels together with the hierarchical decomposition to assign each peer v to a subcube in $[0,1)^2$, the graph $G_F(V)$ has a degree of $O(\log n)$, diameter of $O(\log n)$ and expansion of $\Omega(1/\log n)$, with high probability.*

Routing in the dynamic form of the Gabber-Galil graph is very difficult since expanders tend to have a very irregular structure. So we do not describe how to do that here. Joining and leaving is done following the hierarchical decomposition approach. Since whenever a peer joins, a subcube is split into two, a new peer v can get all of its connections from the peer u whose subcube is split. Whenever a peer v leaves, we either merge to subcubes or take one peer w to take over the subcube of v and merge the subcubes of w and its sibling w' in the decomposition tree into one that is assigned to w'. In any case, we obtain the following result.

Theorem 7.15 *It takes a routing effort of $O(\log n)$ hops and an update work of $O(\log n)$ messages that can be processed in a logarithmic number of communication rounds in order to execute a join or leave operation in the dynamic Gabber-Galil graph.*

7.5 Robustness against random faults

In order to protect against random faults in dynamic networks based on $U = [0,1)$, each peer v aims at preserving the following condition (which is similar to the supervised case).

Condition 7.16 *Every peer v in the system is connected to*

- $\mathrm{pred}_i(v)$ *and* $\mathrm{succ}_i(v)$ *for every* $i \in \{1,\ldots,k\}$ *and*
- *all peers w with the property that* $\Gamma(R(N_v)) \cap R(N_w) \neq \emptyset$

where $N_v = \{v\} \cup \{\mathrm{pred}_i(v) \mid i \in \{1,\ldots,k\}\} \cup \{\mathrm{succ}_i(v) \mid i \in \{1,\ldots,k\}\}$ *and for any set* $V' \subseteq V$, $R(V') = \bigcup_{v \in V'} R(v)$.

When doing this, the following result can be shown.

Theorem 7.17 *If faults of peers are independent of their positions and only happen at a (sufficiently small) constant rate, then the peers can maintain Condition 7.16 everywhere, w.h.p.*

The result holds since for a sufficiently small constant fault rate and a sufficiently large k, no entire sequence of k consecutive peers in $[0,1)$ will become faulty within a constant number of steps, w.h.p., and every peer only needs a constant number of communication rounds to update its connectivity information as peers among its predecessors and successors become faulty as long as this is in principle possible (i.e., not all of its predecessors and successors fail).

7.6 Robustness against adversarial join-leave behavior

Finally, we consider the problem of protecting an overlay network against adversarial join-leave behavior. More precisely, we consider the following scenario. There are n blue (or honest) nodes and ϵn red (or adversarial) nodes for some fixed constant $\epsilon < 1$. There is a rejoin operation that, when applied to node v, lets v first leave the system and then join it again from scratch. The leaving is done by simply removing v from the system and the joining is done with the help of a join operation to be specified by the system. We assume that the sequence of rejoin requests is controlled by an adversary, which is a typical assumption in the analysis of online algorithms. The adversary can only issue rejoin requests for the red nodes, but it can do this in an arbitrary adaptive manner. That is, at any time it can inspect the entire system and select whatever red node it likes to rejoin the system. Our goal is to find an *oblivious* join strategy, i.e., a strategy that cannot distinguish between the blue and red nodes, so that for *any* adversarial strategy above the following two conditions can be preserved for every interval $I \subseteq [0, 1)$ of size at least $(c \log n)/n$ for a constant $c > 0$ and any polynomial number of rounds in n:

- *Balancing condition:* I contains $\Theta(|I| \cdot n)$ nodes.
- *Majority condition:* the blue nodes in I are in the majority.

It is not difficult to see that the brute-force strategy of giving every node a new random place whenever a node rejoins will achieve the stated goal, with high probability, but this would be a very expensive strategy. The challenge is to find a join operation that needs as little randomness and as few rearrangements as possible to satisfy the two conditions. Fortunately, there is such a strategy, called the *cuckoo rule*. We first introduce some notation, and then we describe the strategy.

In the following, a *region* is an interval of size $1/2^r$ in $[0, 1)$ for some integer r that starts at an integer multiple of $1/2^r$. Hence, there are exactly 2^r regions of size $1/2^r$. A *k-region* is a region of size (closest from above to) k/n, and for any point $x \in [0, 1)$, the k-region $R_k(x)$ is the unique k-region containing x.

Cuckoo rule: If a new node v wants to join the system, pick a random $x \in [0, 1)$. Place v into x and move all nodes in $R_k(x)$ to points in $[0, 1)$ chosen uniformly and independently at random (without replacing any further nodes).

See Figure 2 for an illustration of the cuckoo rule. The following result can be shown [3]:

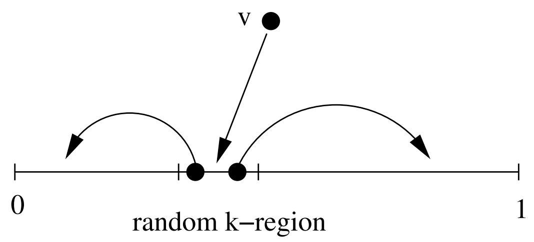

Figure 2: The cuckoo rule. (a) A new node is placed at a random point x. (b) Old nodes in $R_k(x)$ are evicted and moved to new, random places.

Theorem 7.18 *For any constants ϵ and k with $\epsilon < 1 - 1/k$, the cuckoo rule with parameter k satisfies the balancing and majority conditions for a polynomial number of rounds, with high probability, for any adversarial strategy within our model. The inequality $\epsilon < 1 - 1/k$ is sharp as counterexamples can be constructed otherwise.*

We just sketch the proof of the theorem. Let $\hat{R}$ be any fixed region of size $(c \log n) \cdot k/n$, for some constant c, for which we want to check the balancing and majority conditions over polynomial in n many steps. Thus, $\hat{R}$ contains exactly $c \log n$ many k-regions. The *age* of a k-region is the difference between the current round and the last round when a new node was placed into it (and all old nodes got evicted), and the age of $\hat{R}$ is defined as the sum of the ages of its k-regions. A node in $\hat{R}$ is called *new* if it was placed in $\hat{R}$ when it joined the system, and otherwise it is called *old*. The first lemma follows directly from the cuckoo rule because every k-region can have at most one new node at any time.

Lemma 7.19 *At any time, $\hat{R}$ contains at most $c \log n$ new nodes.*

In order to bound the number of old nodes in $\hat{R}$, we first have to bound the age of $\hat{R}$ (Lemma 7.20). Then we bound the maximum number of nodes in a k-region (Lemma 7.21) and use this to bound the number of evicted blue and red nodes in a certain time interval (Lemma 7.22). After that, we can combine all lemmas to bound the number of old blue and red nodes in $\hat{R}$ (Lemma 7.23).

Lemma 7.20 *At any time, $\hat{R}$ has an age within $(1 \pm \delta)(c \log n)\ n/k$, with high probability, where $\delta > 0$ is a constant that can be made arbitrarily small depending on the constant c.*

Lemma 7.21 *For any k-region R in $\hat{R}$ it holds at any time that R has at most $O(k \log n)$ nodes, with high probability.*

Next we bound the number of blue and red nodes that are evicted in a certain time interval.

Lemma 7.22 *For any time interval I of size $T = (\gamma/\epsilon) \log^3 n$, the number of blue nodes that are evicted in I is within $(1 \pm \delta)T \cdot k$, with high probability, and the number of red nodes that are evicted in I is within $(1 \pm \delta)T \cdot \epsilon k$, with high probability, where $\delta > 0$ can be made arbitrarily small depending on γ.*

Combining Lemmas 7.20 to 7.22, we obtain the following lemma.

Lemma 7.23 *At any time, $\hat{R}$ has within $(1 \pm \delta)(c \log n) \cdot k$ old blue nodes and within $(1 \pm \delta)(c \log n) \cdot \epsilon k$ old red nodes, with high probability, where the lower bound on the red nodes holds if none of the red nodes has rejoined.*

Combining Lemmas 7.19 and 7.23, we can now prove when the balancing and majority conditions are satisfied.

- **Balancing condition:** From Lemmas 7.19 and 7.23 it follows that every region R of size $(c \log n)k/n$ has at least $(1-\delta)(c \log n) \cdot k$ and at most $(1+\delta)(c \log n + (c \log n)k + (c \log n)\epsilon k) = (1 + \delta)(c \log n)(1 + (1 + \epsilon)k)$ nodes, where the constant $\delta > 0$ can be made arbitrarily small. Hence, the regions are balanced within a factor of close to $(1 + \epsilon + 1/k)$.

- **Majority condition:** From Lemmas 7.19 and 7.23 it also follows that every region of size $(c \log n)k/n$ has at least $(1-\delta)(c \log n) \cdot k$ blue nodes and at most $(1+\delta)(c \log n + (c \log n) \cdot \epsilon k)$ red nodes, w.h.p., where the constant $\delta > 0$ can be made arbitrarily small. These bounds are also tight in the worst case, which happens if the adversary focuses on a specific region R of size $(c \log n)k/n$ and continuously rejoins with any red node outside of R. Hence, the adversary is not able to obtain the majority in any region of size $(c \log n)k/n$ as long as $(c \log n)(\epsilon k + 1) < (c \log n) \cdot k$ which is true if and only if $\epsilon < 1 - 1/k$.

Hence, for $\epsilon < 1 - 1/k$ the balancing and majority conditions are satisfied, w.h.p., and this is sharp, which proves Theorem 7.18.

The cuckoo rule has the drawback that it only works if only the red nodes show adversarial join-leave behavior. What if both kinds of nodes show adversarial join-leave behavior? Then we need to extend the cuckoo rule in the following way in order to main the balancing and majority conditions.

The join operation works in the same way as the cuckoo rule. But whenever a peer wants to leave the network, we use the following leave operation:

Leave(v): If a peer v leaves the system, then a k-region R is chosen uniformly at random among the k-regions of $R_{kc \log n}(x)$ for some (sufficiently large) constant c, where x is the position of v. R is flipped with a k-region R' chosen uniformly at random in $[0, 1)$, and then all peers in R (as well as v) have to rejoin the system from scratch using the cuckoo rule.

Hence, the departure of a peer may spawn several join operations. We call this algorithm the *cuckoo&flip strategy*. With this strategy the balancing and majority conditions can be kept, with high probability. More precisely, one can show the following result [4]:

Theorem 7.24 *For any constants ϵ and k with $\epsilon < 1/4 - (2 \log k + 1)/k$, the cuckoo&flip strategy satisfies the balancing and majority conditions for any polynomial number of rejoin requests, with high probability, for any adversarial strategy within our model.*

Hence, as long as $\epsilon < 1/4$, only a constant factor overhead has to be paid (on average) compared to standard join and leave operations without any additional replacements of peers.

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 *Proc. of the 2ndIntl. Workshop on Peer-to-Peer Szstems (IPTPS)*, 2003.

[2] J. Aspnes and G. Shah. Skip graphs. In *Proc. of the 14th ACM/SIAM Symp. on Discrete Algorithms (SODA)*, pages 384–393, 2003.

[3] B. Awerbuch and C. Scheideler. Towards a scalable and robust DHT. In *Proc. of the 18th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2006.

[4] B. Awerbuch and C. Scheideler. Towards scalable and robust overlay networks. In *Proc. of the 6thIntl. Workshop on Peer-to-Peer Szstems (IPTPS)*, 2007.

[5] M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. of the 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 50–59, 2003.

[6] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of the SIGCOMM '01*, pages 149–160, 2001.