9 Decentralized overlay networks

In this section 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. To simplify the presentation, we will assume in this section that all peers are reliable and honest. In the next section we will consider scenarios in which this is not the case.

This section consists of two parts. The overlay networks in the first part are based on the continuous-discrete approach [5], and the overlay networks presented in the second part are based on so-called skip graphs [1] defined later in this section. The difference between these two concepts is that the former concept only works well if the peers are given random or hashed names for some pseudo-random hash function, whereas the latter concept works well for arbitrary distinct peer names.

9.1 The continuous-discrete approach

First, we show how to maintain a dynamic hypercube, and then we show how to maintain a dynamic de Bruijn network.

A dynamic hypercube

Recall the definition of a hypercube. According to this definition, every node with label $(x_1, \ldots, x_d) \in \{0,1\}^d$ is connected to the nodes $(\bar{x_1}, x_2, \ldots, x_d)$, $(x_1, \bar{x_2}, x_3, \ldots, x_d), \ldots$, $(x_1, \ldots, x_{d-1}, \bar{x_d})$, where $\bar{x} = (1+x) \bmod 2$. Consider now the space $U = [0,1)$ and the collection $F = \{f_i^-, f_i^+ : U \to U \mid i \in \mathbb{N}\}$ of functions

$$f_i^-(x) = (x - 1/2^i) \bmod 1 \quad \text{and} \quad f_i^+(x) = (x + 1/2^i) \bmod 1 \ .$$

When interpreting every label $(x_1, \ldots, x_d)$ as $x = \sum_{i=1}^d x_i/2^i$, then for every neighbor x' of x in the hypercube there is a function $f \in F$ with $f(x) = x'$. More precisely, if x and x' only differ in the ith bit, then it holds:

- if $x_i = 1$ then $x' = f_i^-(x)$ and
- if $x_i = 0$ then $x' = f_i^+(x)$.

Hence, as long as the peer set V is assigned to regions whose union gives U and connections between the peers are established according to the continuous-discrete approach, the resulting graph $G_F(V)$ establishes a dynamic hypercube. The problem is, how to assign proper regions to the peers in V. Here, we use a very simple rule:

Every peer $v \in V$ chooses a random point $x_v \in [0,1)$ and is responsible for the region $R_v = [x_v, \text{succ}(x_v))$ where $\text{succ}(x_v)$ is the closest successor of x_v on $[0,1)$ among the points of the peers.

Using this rule, it is obvious that the regions are pairwise disjoint and that $\bigcup_{R_v} = [0,1)$. Moreover, the following lemmas hold.

Lemma 9.1 *Given n peers, 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. □

Lemma 9.2 *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. □

In order to interconnect the peers, we demand that every peer v must be connected to all peers w whose regions overlap with $f(R_v)$ for some $f \in F$ and to the two neighboring peers on the $[0, 1)$-ring. Since the size of these regions is equal to the size of R_v, it follows from Lemmas 9.1 and 9.2:

Lemma 9.3 *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:

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

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 maximally 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 9.1.

Theorem 9.4 *Using the continuous-discrete routing strategy, it takes at most $O(\log n)$ hops until a message is routing 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, 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 9.5 *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 9.2 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 9.1, 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. $\square$

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 9.4. 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.

Notice that $F = F^{-1}$, i.e., all inverse functions of functions in F are also in F. Thus, the peers do not have to worry about incoming connections because a peer v has an outgoing connection to peer w if and only if w has an outgoing connection to v. Hence, all connections can indeed be updated by just looking at the outgoing edges.

We know from Theorem 9.4 that the routing part only takes $O(\log n)$ hops. Furthermore, Lemmas 9.1 and 9.2 imply that every peer has at most $O(\log^2 n)$ incoming and outgoing edges. Hence, we obtain the following theorem.

Theorem 9.6 *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 5: data items are hashed to random values in $[0, 1)$ using a pseudo-random hash function h, 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.

A dynamic de Bruijn network

Recall the definition of a de Bruijn graph. In this definition, every node with label $(x_1, \ldots, x_d) \in \{0,1\}^d$ is connected to the nodes $(0, x_1, \ldots, x_d)$ and $(1, x_1, \ldots, x_d)$. Consider now the space $U = [0, 1)$ and the collection $F = \{f_0, f_1\}$ of functions

$$f_0(x) = x/2 \quad \text{and} \quad f_1(x) = (1 + x)/2 \ .$$

When interpreting every label $(x_1, \ldots, x_d)$ as $x = \sum_{i=1}^{d} x_i/2^i$, then these functions are a good approximation of the de Bruijn edges. In fact, for $d \to \infty$ they match the de Bruijn edges. In order to assign proper regions to the peers in V, we use the same strategy as for the dynamic hypercube.

Every peer $v \in V$ chooses a random point $x_v \in [0, 1)$ and is responsible for the region $R_v = [x_v, \mathrm{succ}(x_v))$ where $\mathrm{succ}(x_v)$ is the closest successor of x_v on $[0, 1)$ among the points of the peers.

In order to interconnect the peers, we use the classes F and F^{-1} where F^{-1} contains the inverse functions of the functions in F. That is, every peer v is connected to all peers w whose regions overlap with $f_0(R_v)$, $f_1(R_v)$, $f_0^{-1}(R_v)$ and $f_1^{-1}(R_v)$. The size of these regions differs only by a factor of 2 from the size of R_v. Furthermore, v connects to its successor and predecessor in $[0, 1)$. It follows from Lemmas 9.1 and 9.2:

Lemma 9.7 *Given n peers, the dynamic de Bruijn network has a maximum degree of $O(\log n)$, w.h.p.*

Routing in a dynamic de Bruijn network

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-discrete strategy to route the message from x to y:

The message starts at the peer v_0 responsible for x. If y is not in the region of v_0 or one of its neighbors, v_0 chooses a random bit z_1 and forwards the message to the peer v_1 owning $(z_1 x_1 x_2 \ldots)$. In general, if the message has reached a peer v_i and $(z_i \ldots z_1 y_1 y_2 \ldots)$ is not in the region of v_i or one of its neighbors, v_i chooses a random z_{i+1} and forwards the message to the peer owning $(z_{i+1} \ldots z_1 x_1 x_2 \ldots)$. Once the message has reached a peer w_i owning $(z_i \ldots z_1 y_1 y_2 \ldots)$, it is forwarded to the peer w_{i-1} owning $(z_{i-1} \ldots z_1 y_1 y_2 \ldots)$ and so on until it reaches the peer owning $(y_1 y_2 \ldots)$.

All of the hops in the routing strategy can be performed along edges of the dynamic de Bruijn graph. Also, notice that the maximum distance between $(z_i \ldots z_1 x_1 x_2 \ldots)$ and $(z_i \ldots z_1 y_1 y_2 \ldots)$ shrinks by a factor of 2 in each hop. Hence, once a distance equal to the smallest region is reached, the routing strategy can move from a peer v_i to a peer w_i. Thus, the following theorem immediately follows from Lemma 9.1.

Theorem 9.8 *Using the continuous-discrete routing strategy, it takes at most $O(\log n)$ hops until a message is routing from any point x to any point y in $[0, 1)$.*

The continuous-discrete routing strategy does not only have a low dilation but also a small congestion because it has Valiant's trick already built into it. Hence, the following theorem holds using similar arguments as in Theorem 9.5.

Theorem 9.9 *For every permutation routing problem, the congestion caused when using the continuous-discrete routing strategy above is at most $O(\log^2 n)$, w.h.p.*

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 Theorem 9.8. 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 Theorem 9.8 that the routing part only takes $O(\log n)$ hops. Furthermore, Lemmas 9.1 and 9.2 imply that every peer has at most $O(\log n)$ outgoing edges. Hence, we obtain the following theorem.

Theorem 9.10 *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.

9.2 Skip graphs

So far, we saw how to construct completely decentralized peer-to-peer systems with good topological properties if the peers are assigned to random locations in the $[0, 1)$-interval. However, there are several scenarios in which it would be much better if the peers are organized according to their real, user-defined names instead of just random names.

For example, suppose that we want to implement a distributed name service such as the well-known domain name service (DNS). Then we would like to organize the peers in a peer-to-peer system so that a peer with a given name can be found quickly. If the names were well-spread in the name space so that we could interpret them as well-spread numbers in the $[0, 1)$-interval, then we could use the dynamic de Bruijn network to implement such a service. However, we cannot guarantee that the names will be well-spread, and therefore we need a different overlay network design.

As another example, consider the situation that we want to design a peer-to-peer system in which we can take locality issues into account. Locality is an important issue in the Internet. Using the dynamic de Bruijn network can mean that a message is sent $\log n$ times across the world before it reaches its destination. Instead, imagine that we knew the geographic location of every peer. One possible way of specifying such a location could be

North_America.USA.MD.Baltimore.Johns_Hopkins_Univesity.Computer_Science

If such information is available, we could organize the peers in an overlay network sorted according to this location information so that now messages will only be sent once across the world in the worst case. Instead of a geographical location, one could also use a hierarchically specified Internet location, starting with the backbone ISP, the local ISP, and so on (which may be determined via traceroute, for example).

In the following, we present overlay network designs that allow peers to be ordered according to arbitrary user-defined names. We first present (random) skip graphs [1, 4], and then we present deterministic skip graphs which are also known as hyperrings [3].

Skip graphs

Given an infinite bit string $b = x_1x_2x_3\ldots$, we define $\text{prefix}_0(b) = \epsilon$ (the empty word) and $\text{prefix}_i(b) = x_1x_2\ldots x_i$ for every $i \geq 1$. Suppose that we have a (pseudo-)random hash function h assigning to each node an ID representing an infinite bit string. Given a set of nodes V, we define for every $v \in V$ and $i \geq 0$:

- $\mathrm{succ}_i(v) = \mathrm{argmin}\{w \in V \mid \mathrm{Name}(w) > \mathrm{Name}(v)$ and $\mathrm{prefix}_i(h(v)) = \mathrm{prefix}_i(h(w))\}$, i.e. $\mathrm{succ}_i(v)$ is the node w whose name is the closest successor of v's name (with respect to lexicographical ordering) with the same i first bits in $h(w)$ as $h(v)$, and
- $\mathrm{pred}_i(v) = \mathrm{argmax}\{w \in V \mid \mathrm{Name}(w) < \mathrm{Name}(v)$ and $\mathrm{prefix}_i(h(v)) = \mathrm{prefix}_i(h(w))\}$.

Notice that we view the name space as a ring here. This means for $\mathrm{succ}_i(v)$ that if there is no node w with $\mathrm{Name}(w) > \mathrm{Name}(v)$ that fulfills the prefix condition, then we associate $\mathrm{succ}_i(v)$ with the node w with smallest name so that $\mathrm{prefix}_i(h(v)) = \mathrm{prefix}_i(h(w))$. If there is no other node w in the network with that property, then we set $\mathrm{succ}_i(v) = v$. In skip graphs, the following invariants have to be kept at any time.

Invariant 9.11 *For any set of nodes V currently in the system, it holds for every $v \in V$ that v is connected to $\mathrm{succ}_i(v)$ and $\mathrm{pred}_i(v)$ for all $i \geq 0$.*

Invariant 9.11 requires that the nodes are organized in a hierarchy of doubly linked cycles, where the node names have to be sorted in every cycle, and every node participates in exactly one cycle for every $i \geq 0$. A cycle at level i is called i-cycle or i-ring, and an edge in an i-ring is called an i-edge. Skip graphs have the following properties, where n is the current number of nodes in the network:

Theorem 9.12 *If Invariant 9.11 is true and h assigns random bit strings to nodes, then the skip graph has a maximum degree of $O(\log n)$, a diameter of $O(\log n)$, and a node expansion of $\Omega(1)$, with high probability.*

Proof. The probability that some fixed node pair v and w fulfills $\mathrm{prefix}_i(h(v)) = \mathrm{prefix}_i(h(w))$ is equal to $1/2^i$. Hence, for $i \geq 3\log n$, it holds that

$$\begin{aligned} &\Pr[\text{there is a node pair } v,w \text{ with } \mathrm{prefix}_i(h(v)) = \mathrm{prefix}_i(h(w))] \\ &\leq \sum_{v,w} \Pr[\text{node pair } v,w \text{ fulfills } \mathrm{prefix}_i(h(v)) = \mathrm{prefix}_i(h(w))] \\ &= \sum_{v,w} \frac{1}{2^{3\log n}} \leq n^2 \cdot \frac{1}{n^3} = \frac{1}{n}\ . \end{aligned}$$

Hence, with high probability there is no ring of level $3\log n$ or higher. Thus, every node has a degree of at most $2(3\log n + 1)$.

Next, we bound the diameter. Consider any node $v \in V$. Our aim is to move from v to a node w with largest i so that $\mathrm{prefix}_i(w) = 00\ldots0$. To do this, we move from $u_0 = v$ to the closest successor u_1 on the 0-ring with $\mathrm{prefix}_1(u_1) = 0$, and in general from node u_i to the closest successor u_{i+1} on the current i-ring with $\mathrm{prefix}_{i+1}(u_{i+1}) = 00\ldots0$. For any $i \geq 0$, it holds:

$$\Pr[\text{the distance to } u_{i+1} \text{ is } \delta] = \left(\frac{1}{2}\right)^{\delta+1}\ .$$

Hence,

$$\begin{aligned} \mathrm{E}[\text{distance to } u_{i+1}] &= \sum_{\delta\geq 0} \delta \cdot \left(\frac{1}{2}\right)^{\delta+1} = \sum_{\delta\geq 0} (\delta+1) \cdot \left(\frac{1}{2}\right)^{\delta+2} \\ &= \frac{1}{4}\left(\sum_{\delta\geq 0}\left(\frac{1}{2}\right)^{\delta}\right) = \frac{1}{4}\cdot 4 = 1\ . \end{aligned}$$

Since we know from the degree proof that there are at most $3 \log n$ levels with high probability, the expected number of hops we need to perform to get from v to w is $O(\log n)$. Furthermore, going from w to the node w' with smallest bit string also takes just $O(\log n)$ hops on expectation. Hence, every node in V can reach the node w' with smallest $h(w')$ in $O(\log n)$ steps on expectation, and this can also be shown to hold with high probability. Thus, the diameter is $O(\log n)$, with high probability.

The expansion proof is involved and will not be shown here. See [2] for details. □

Routing in skip graphs

Consider the following routing strategy:

Suppose that node u is the current location of a message with destination Name. As long as $\mathrm{Name} \notin (\mathrm{Name}(\mathrm{pred}_0(u)), \mathrm{Name}(u)]$ (i.e. the message has not yet reached a node u that is the closest successor of Name), u sends the message to the node $\mathrm{succ}_i(u)$ with maximum i so that $\mathrm{Name}(\mathrm{succ}_i(u)) \leq \mathrm{Name}$ (treating the name space as a ring).

One can show the following result:

Lemma 9.13 *For any node $v \in V$ and any name Name, it takes at most $O(\log n)$ hops, with high probability, to send a message from v to the node whose name is the closest successor to Name.*

Joining and leaving the network

Suppose that a new node v contacts node $w \in V$ to join the system. Then w will forward v's request to $\mathrm{succ}_0(v)$ using the routing strategy above with $\mathrm{Name} = \mathrm{Name}(v)$. $\mathrm{succ}_0(v)$ will then integrate v between $\mathrm{succ}_0(v)$ and $\mathrm{pred}_0(v)$. Afterwards, v sends out two requests along the 0-ring to find $\mathrm{succ}_1(v)$ and $\mathrm{pred}_1(v)$. Once they are found, v integrates itself into its 1-ring. v then uses the 1-ring to find $\mathrm{succ}_2(v)$ and $\mathrm{pred}_2(v)$, and then integrates itself into the 2-ring. This continues until v has integrated itself into the highest possible ring containing at least 2 nodes.

Using a probabilistic analysis, one can show the following result:

Theorem 9.14 *Inserting a new node requires $O(\log n)$ time and work with high probability.*

If a node wants to leave the system, it does this by simply connecting $\mathrm{pred}_i(v)$ with $\mathrm{succ}_i(v)$ for every $i \geq 0$. This gives the following result:

Theorem 9.15 *Deleting a node requires $O(\log n)$ time and work with high probability.*

Searching

When a node v searches for a node with name Name, then it simply uses the routing strategy described above. Once a node w with $\mathrm{Name}(w) = \mathrm{Name}$ has been found, w reports its IP address back to v. This strategy has the following performance:

Theorem 9.16 *Any search operation requires $O(\log n)$ time and work with high probability.*

Hyperring

Next we consider the hyperring. Like the skip graph, also the hyperring consists of a hierarchy of rings. However, here we are much more strict about how the rings are maintained.

Suppose that we have a hyperring with n nodes. Then it consists of approximately $\log n$ levels of rings, starting with level 0. Each level $i \geq 0$ consists of approximately 2^i directed cycles of approximately $n/2^i$ nodes, which we call *rings*. All rings have the same orientation, and we require the nodes in every ring to be ordered according to their names. For every ring R at level i, two rings of level $i+1$ share its nodes in an intertwined fashion. As before, a ring at level i will be called an *i-ring*, and a level i edge will be called an *i-edge*. Consider some i-ring R and let (u, v, w, x) be four consecutive nodes on R. We say that (u, v, w, x) form an *i-bridge* (or simply a *bridge* if i is clear from the context) if there is an $(i+1)$-edge from u to x and an $(i+1)$-edge from v to w. An $(i+1)$-edge is called *perfect* if it bridges exactly two i-edges.

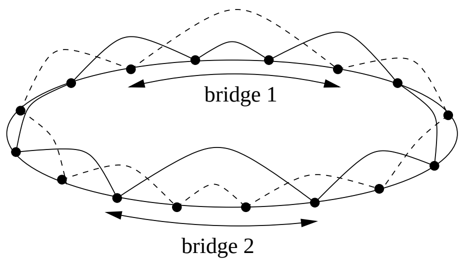

Figure 1: An example of a hyperring. The bridges have a distance of 5 from each other.

It is possible to maintain a hyperring with at most one bridge in every ring. However, in this case we would create too much update work for JOIN or LEAVE operations. Instead, we only demand that i-bridges are sufficiently far apart from each other. A hyperring is called *k-separated* if in every i-ring R the i-bridges on R are at least k nodes apart from each other, which means that there are at least $k-1$ nodes between the quadruples of nodes forming a bridge. We start with a few properties of hyperrings which are easy to prove.

Lemma 9.17 *For every $k \geq 0$, the k-separated hyperring has a maximum degree of at most $2(1+2/(k+1))\log n$ and a diameter of at most $3\log n$.*

Proof. First, we bound the maximum degree. Consider some i-ring R. In order to minimize the size of an $i+1$-ring R' on top of R without violating k-perfectness, the best one can do is using a repetitive sequence of $\lceil k/2 \rceil + 1$ edges, where one edge bridges three edges in R and the remaining $\lceil k/2 \rceil$ edges bridge two edges in R. Hence,

$$\begin{aligned} |R'| &\geq \frac{|R|}{3+2\lceil k/2\rceil} \cdot (1+\lceil k/2\rceil) = \left(\frac{1}{2} - \frac{1}{4(\lceil k/2\rceil+1)+2}\right) \cdot |R| \\ &\geq \frac{1}{2}\left(1-\frac{1}{k+3}\right) \cdot |R| \end{aligned}$$

This also implies that $|R'|$ can be at most $\frac{1}{2}(1+\frac{1}{k+3})|R|$. Hence, an i-ring R can have a size of at most

$$\frac{1}{2^i}\left(1+\frac{1}{k+3}\right)^i \leq \frac{1}{2^i} \cdot e^{i/(k+3)} \leq 2^{-i(1-2/(k+3))} .$$

This is at most 1 if $i \geq (\log n)/(1-2/(k+3))$. Since each node has 2 edges in each level in which it participates, the maximum node degree is $2(\log n)/(1-2/(k+3)) = 2(1+2/(k+1))\log n$.

Next, we bound the diameter. Consider any two nodes v and w on a ring R, and let R_0 and R_1 be the two intertwined rings on top of R. Furthermore, let v_0 be the node in R_0 nearest to v and w_0 be the node in R_0 nearest to w. Define v_1 and w_1 in the same way for R_1. First of all, $d_R(v, v_0)$, $d_R(v, v_1)$, $d_R(w, w_0)$, and $d_R(w, w_1)$ are all at most 1. Hence, $d_R(v_0, w_0) \leq d_R(v, w) + 2$ and $d_R(v_1, w_1) \leq d_R(v, w) + 2$. Since the nodes used by R_0 and R_1 are disjoint, it must hold that either $d_{R_0}(v_0, w_0) \leq d_R(v, w)/2 + 1$ or $d_{R_1}(v_1, w_1) \leq d_R(v, w)/2 + 1$. Hence, if we always take the ring of lower distance in each layer, then for each layer i we obtain the recursion $d_{i+1} \leq d_i/2 + 3$ with $d_0 = d_R(v, w)$. Therefore, the total number of edges used is at most $3 \log n$. $\square$

Unfortunately, hyperrings with constant separation can have a bad expansion.

Theorem 9.18 *For every $k \geq 0$, the k-separated hyperring has, in the worst case, an edge expansion of*

$$O(1/n^{1/(2(3(k+4))^2)}) .$$

The proof of this theorem is quite involved and can be found in [3]. Unfortunately, Theorem 9.18 implies that no k-separated hyperring with $k = O((\log n)^{1/2-\epsilon})$ for some constant $\epsilon > 0$ can guarantee an expansion of $\Omega(1/\log^c n)$ for some constant c depending on ϵ. Hence, in order to have a good expansion, we need $k = \Omega(\sqrt{\log n})$. However, notice that when k depends on the size of the hyperring, node insertions and deletions that have been performed in the past might have used a k that significantly differs from the k used by current insertions and deletions. Hence, parts of the hyperring may be out of date. So the question is whether it is necessary to revisit these parts in order to bring the hyperring up to date. Fortunately, as one of the main results in [3], it was shown that *this is not necessary*. One can simply use as the current k the degree of any node currently in the system when executing a JOIN or LEAVE operation, and old JOIN or LEAVE operations never have to be revisited, to show the following result. ($|R|$ denotes the number of nodes in a ring R, and $|e|$ denotes the number of node on the 0-ring bridged by edge e.)

Proposition 9.19 *At any time it holds:*

1. *the* ring distortion is low, *i.e. for every i-ring R, $|R| \in [\frac{1}{2} \cdot n/2^i - 1, 2 \cdot n/2^i + 1]$ and*
2. *the* edge distortion is low, *i.e. for every i-edge e, $|e| \leq 4 \cdot 2^i$.*

The proof for this is quite complicated and can be found in [3]. For simplicity, we assume for the rest of this section that k is fixed. We start with describing how to route in the hyperring.

Routing in the hyperring

We use the same routing strategy as for skip graphs:

Suppose that node u is the current location of a message with destination Name. As long as $\mathrm{Name} \notin (\mathrm{Name}(\mathrm{pred}_0(u)), \mathrm{Name}(u)]$ (i.e. the message has not yet reached a node u that is the closest successor of Name), u sends the message to the node $\mathrm{succ}_i(u)$ with maximum i so that $\mathrm{Name}(\mathrm{succ}_i(u)) \leq \mathrm{Name}$ (treating the name space as a ring).

Since this routing strategy prefers edges of higher level and every $i+1$-edge bridges at most 3 i-edges for every i, we obtain the following fact.

Fact 9.20 *Any message moves along a sequence of edges of non-increasing level and uses at most two edges in each level.*

Combining this with Lemma 9.17, which says that there are at most $(1 + 2/(k+1)) \log n$ levels, we achieve the following result.

Lemma 9.21 *For any node $v \in V$ and any name Name, it takes at most $O(\log n)$ hops to send a message from v to the node whose name is the closest successor to Name.*

Joining and leaving the network

First, we introduce some notation. Let $\mathrm{succ}_i(v)$ be the successor of v in its i-ring and $\mathrm{pred}_i(v)$ be the predecessor of v in its i-ring. For every node v on R, its $> i$-endpoints represent all endpoints of edges in v with level more than i. Notice that each node has two endpoints in each level. By "moving" the i-endpoints from u to v, we mean that we replace the i-edges $(\mathrm{pred}_i(u), u)$ and $(u, \mathrm{succ}_i(u))$ by the i-edges $(\mathrm{pred}_i(u), v)$ and $(v, \mathrm{succ}_i(u))$. By "permuting" the i-endpoints of u and v, we mean that we move the i-endpoints of u to v and the i-endpoints of v to u.

Suppose now that a new node u contacts some node $v \in V$ to join the system. Then v will forward u's request to $\mathrm{succ}_0(u)$ using the routing strategy above with $\mathrm{Name} = \mathrm{Name}(u)$. $\mathrm{succ}_0(u)$ will then integrate u between $\mathrm{succ}_0(u)$ and $\mathrm{pred}_0(u)$. Afterwards, u is integrated into the hyperring level by level, starting with level 0. In each level i, we integrate the node by either removing an already existing bridge in its $k+2$-neighborhood or by creating a new bridge. A bridge is removed by first dragging it over to u by permuting $> i$-endpoints (see Figure 3). Then case (b) or (c) in Figure 2 is applied. Otherwise, we just apply case (a). JOIN terminates once we reach a ring of size in $\{4, \ldots, 7\}$ (for larger rings, two new subrings are created).

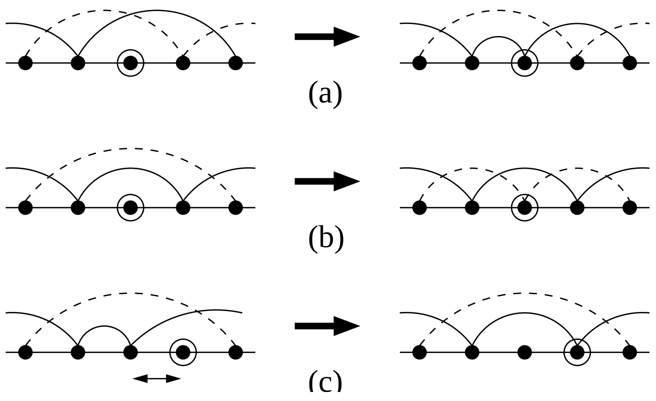

Figure 2: The three cases when adding a node. Case (c) reduces to case (b).

Theorem 9.22 JOIN *locally preserves the k-separation of the hyperring and requires $O(k \log^2 n)$ work and $O(\log k \cdot \log n)$ time.*

Proof. JOIN locally preserves the k-separation property because it only creates a bridge if there is no other bridge in the $k+2$-neighborhood. Otherwise, it removes a bridge. Thus, it remains to prove the work and time bounds.

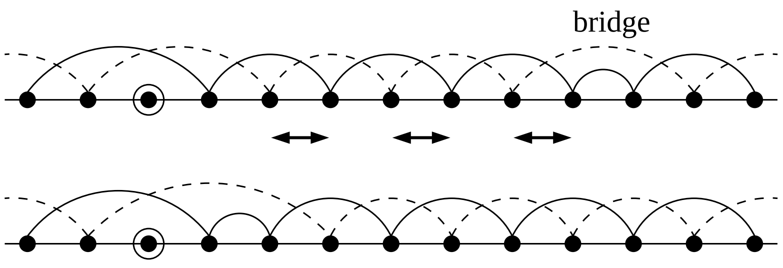

Figure 3: Permuting $> i$-endpoints drags the bridge over to obtain, e.g., case (c) in Figure 2.

In each level, only a $O(k)$-neighborhood is investigated. In the worst case, a bridge has to be moved to u (resp. to the node to be integrated into that level in place of u). This requires $O(k \log n)$ message transmissions. Since the hyperring has $O(\log n)$ levels, the total work is $O(k \log^2 n)$.

When using edges in higher levels, we can investigate the $O(k)$-neighborhood of a node in $O(\log k)$ steps. Thus, in $O(\log k)$ steps we can update the endpoints necessary to proceed with the next higher level. Since there are $O(\log n)$ levels, this results in $O(\log k \cdot \log n)$ time. □

We also remove a node u from the hyperring level by level, starting with level 0. In each level, we remove the node by either removing an already existing bridge in its $k + 2$-neighborhood or by creating a new bridge. A bridge is removed by first dragging it over (Figure 3) and then applying case (b) or (c) in Figure 4. Otherwise, we just apply case (a). LEAVE terminates once we reach a ring of size in $\{4, \ldots, 7\}$ (rings smaller than 4 are removed).

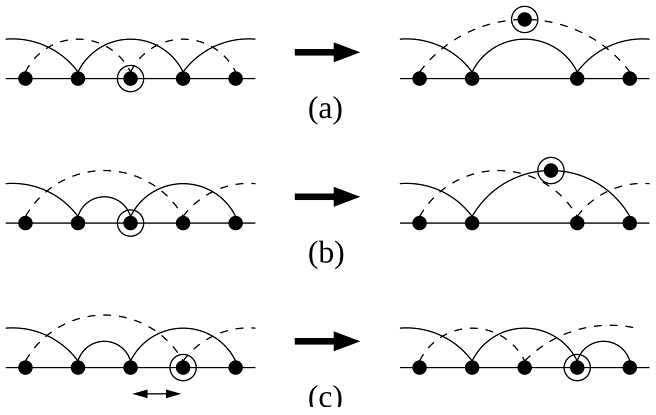

Figure 4: The three cases when removing a node. Case (c) reduces to case (b).

Theorem 9.23 LEAVE *locally preserves the k-separation of the hyperring and requires $O(k \log^2 n)$ work and $O(\log k \cdot \log n)$ time.*

The proof is similar to the proof of Theorem 9.22.

Searching

When a node v searches for a node with name Name, then it simply uses the routing strategy described above. Once a node w with $\mathrm{Name}(w) = \mathrm{Name}$ has been found, w reports its IP address back to v. This strategy has the following performance:

Theorem 9.24 *Any search operation requires $O(\log n)$ time and work with high probability.*

Furthermore, we can show the following result, demonstrating that not only the dilation but also the congestion of search requests can be kept low in the hyperring.

Theorem 9.25 *The congestion caused by n SEARCH requests, one per node, with random destinations is $O(\log n)$, with high probability.*

Proof. Fact 9.20 implies that every i-ring R can only receive requests from rings on top of it. Thus, it can only receive requests from its own nodes. Consider now an arbitrary node v in R. It is easy to check that only those requests will be sent to v whose destination is bridged by the i-edge e leaving v in R. From Proposition 9.19 we know that e bridges at most $4 \cdot 2^i$ nodes and that R consists of at most $3 \cdot n/2^i$ nodes. Since every node is the starting point of one request and every request has a random destination, the expected number of requests that want to reach v in R is at most $(4 \cdot 2^i/n) \cdot (3 \cdot n/2^i) = 12$. Combining this with the fact that every request only uses at most 2 edges in R (see Fact 9.20), the expected number of requests that traverse v in R is at most 24. Because every node participates in at most $\log n + O(1)$ levels, the overall expected number of search requests passing through v is $O(\log n)$. Using the fact that every request picks a random destination independently from other requests, one can also show that the congestion caused by SEARCH is $O(\log n)$ with high probability. $\square$

References

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

[2] J. Aspnes and U. Wieder. The expansion and mixing time of skip graphs with applications. In *Proc. of the 17th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 126–134, 2005.

[3] B. Awerbuch and C. Scheideler. The Hyperring: A low-congestion deterministic data structure for distributed environments. In *Proc. of the 15th ACM/SIAM Symp. on Discrete Algorithms (SODA)*, 2004.

[4] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[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.