8 Supervised Overlay Networks II

In this section we present a general framework for constructing supervised overlay networks. The main ingredients of this framework are the hierarchical decomposition approach that we have already used for oblivious routing and distributed caching and the continuous-discrete approach of Naor and Wieder [4]. Afterwards, we show how to use these approaches to design supervised hypercubic networks and supervised de Bruijn networks. Up to this point we assume that all peers are honest and reliable. However, in reality, peers may fail or may behave in a selfish or adversarial way. To address these issues, we present a general approach for making supervised overlay networks robust against faulty and adversarial behavior.

8.1 A general framework for supervised overlay networks

We start with our general framework. First, we describe the hierarchical decomposition approach, then we describe the continuous-discrete approach, and after that we show how to glue these two approaches together to obtain our general framework.

The hierarchical decomposition approach

Consider any space $U = [0,1)^d$ 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 the 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. 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, t_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.

Our goal for the supervised peer-to-peer system will be to map the peers to nodes of $T(U)$ so that the following invariants are met:

Invariant 8.1 *1. The subcubes of the (nodes assigned to the) peers are disjoint,*

2. the union of the subcubes of the peers gives the entire set U, and

3. the peers are only distributed among nodes of two consecutive levels in $T(U)$.

To satisfy these invariants, we use the following mapping strategy:

Recall the supervised cycle network in the previous section. Let ℓ be the labeling function used in the cycle. We change the labels in a way that for every peer v in the cycle, its new label ℓ'_v is

$$\ell'_v = \ell_v \circ 0^k \quad \text{where } k = \max\{|\ell_{\text{succ}(v)}| - |\ell_v|, 0\}$$

("$\circ 0^k$" means "appended by k zeroes"). Every peer v is mapped to the node w in the decomposition tree with $t_w = \ell'_v$, with the only exception that if there is only one peer in the system, it is mapped to the root of the decomposition tree. This mapping strategy has the following property:

Lemma 8.2 *The mapping strategy satisfies Invariant 8.1.*

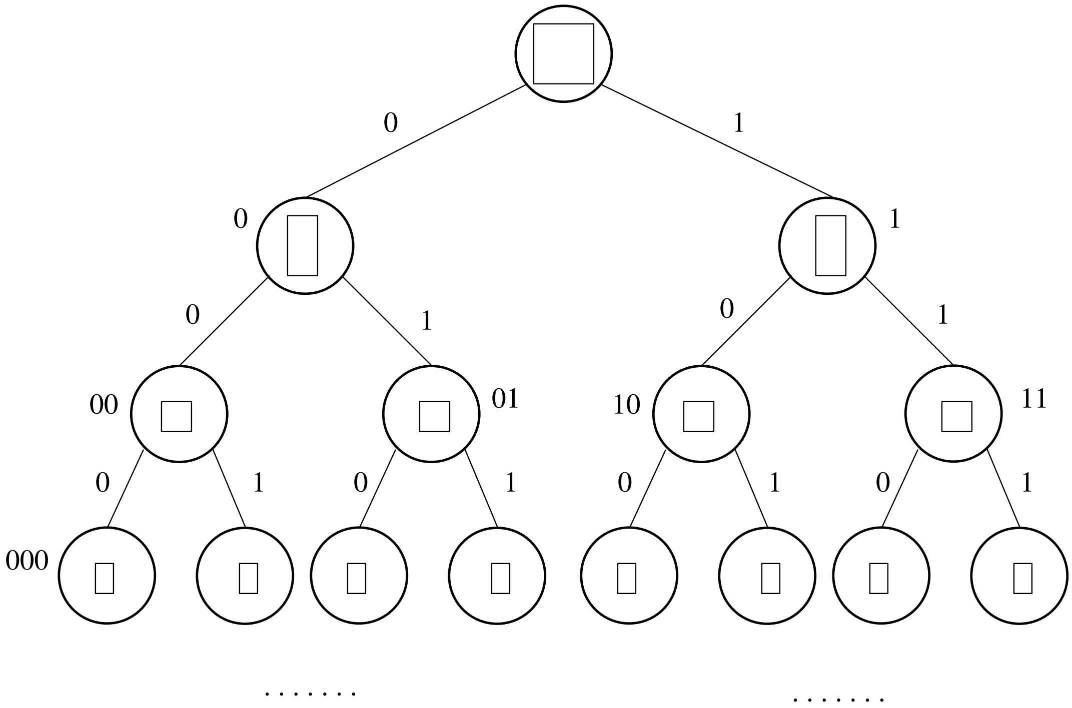

Figure 1: The decomposition tree for $d = 2$.

Proof. We prove the lemma by complete induction on the number of nodes in the system. Initially, there is only one node in the system, which is mapped to the root of the decomposition tree. Hence, Invariant 8.1 is obviously true.

Suppose now that we already showed the lemma for n nodes. Then we will show that it is also true for $n+1$ nodes. Let v be the new node and w its predecessor on the cycle. Given that the old label of w was ℓ''_w, it holds for the new labels ℓ'_w and ℓ'_v that $\ell'_w = \ell''_w \circ 0$ and $\ell'_v = \ell''_w \circ 1$. Hence, given that w was mapped to node u in the decomposition tree, w and v are mapped to u's children afterwards. Since the subcubes of u's children are disjoint and the union of them is the subcube associated with u, one can easily check that all 3 conditions in Invariant 8.1 must still hold after v has been inserted into the cycle. □

The continuous-discrete approach

Again, consider any d-dimensional space $U = [0, 1)^d$ for some fixed d, 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 a directed 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. We say that (U, E_F) is *connected* if for every set $S \subset U$ it holds that $\Gamma(S) \neq \emptyset$.

Consider now any finite set of peers V, and let $R(v)$ be the region in U that has been assigned to peer v. Let $G_F(V)$ be the directed graph with node set V and edge set

$$E = \{(v, w) \in V^2 \mid \exists x \in R(v) \exists y \in R(w) : \; (x, y) \in E_F\}$$

Then it holds:

Lemma 8.3 *If (U, E_F) is connected and $\bigcup_{v \in V} R(v) = U$, then also $G_F(V)$ is connected.*

Proof. Suppose that (U, E_F) is connected 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. □

Our goal will therefore be to make sure that $\bigcup_{v \in V} R(v) = U$. When using the mapping strategy of our hierarchical decomposition approach, this property is satisfied.

Putting the pieces together

Now we are ready to describe our general framework. Consider any space $U = [0, 1)^d$ and collection of functions F. Recall that for a peer v, $R(v)$ is the subcube associated with the node w in the decomposition tree with $t_w = \ell'_v$. Our goal is to maintain the following invariants at any time.

Invariant 8.4 *Every peer v in the system is connected to*

- $\text{pred}(v)$ *and* $\text{succ}(v)$ *(the cycle edges) and*
- *all peers w with the property that $\Gamma(R(v)) \cap R(w) \neq \emptyset$ (the edges of the graph $G_F(V)$)*

Invariant 8.5 *The supervisor is connected to* $\text{pred}(v)$, v, $\text{succ}(v)$, *and* $\text{succ}(\text{succ}(v))$ *where v is the peer with label $\ell(n-1)$.*

It turns out that this is very easy.

Lemma 8.6 *For any join or leave request, the supervisor has sufficient information to maintain Invariant 8.4.*

Proof. First, consider the situation that a new peer v joins the system. From the proof of Lemma 8.2 we know that $\text{pred}(v)$ will cut its subcube, say R, in half, keep one half, say R_1, and give the other half, R_2, to v. According to Invariant 8.4, v needs to establish connections to all peers w with $\Gamma(R_2) \cap R(w) \neq \emptyset$. However, since $R_2 \subseteq R$, $\text{pred}(v)$ owns all of these connections before v joins the system. Hence, all the supervisor has to do is ask $\text{pred}(v)$ to move all those edges $(\text{pred}(v), w)$ with $\Gamma(R_2) \cap R(w) \neq \emptyset$ to v. Thus, the connections that the supervisor has according to Invariant 8.5 suffice to do all updates.

Since the Leave operation is just the reverse of a Join operation, also Leave operations can be executed with the connectivity information the supervisor has according to Invariant 8.5, under the assumption that the leaving node passes all of its connections to the node taking over its place in the cycle. □

From the lemma it follows that, besides the supervisor being able to handle join and leave requests with its limited information, the time and work the supervisor has to invest for these requests is constant.

8.2 Examples

First, we show how to maintain a supervised 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 $(\bar{x_1}, x_2, \ldots, x_d)$, $(x_1, \bar{x_2}, x_3, \ldots, x_d)$, ..., $(x_1, \ldots, x_{d-1}, \bar{x_d})$, where $\bar{x} = (1 + x)$ mod 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 $x' = f_i^+(x)$.

From Lemma 8.6 it immediately follows:

Theorem 8.7 *Using our framework, the supervisor can maintain a dynamic hypercube with work and time $O(1)$ for each join and leave request.*

Also a low degree and diameter and a high expansion can be maintained.

Theorem 8.8 *Using our framework, it holds that at any time, the dynamic hypercube has an outdegree of $\Theta(\log n)$, a diameter of $O(\log n)$ and an expansion of $\Omega(1/\log n)$, where n is the number of peers in the system.*

Proof. First, we bound the degree. According to Invariant 8.1, every peer v is responsible for an interval $R(v)$ of size in $\{1/\bar{n}, 2/\bar{n}\}$ where $\bar{n} = 2^{\lfloor \log n \rfloor}$. Using any function $f \in F$, any interval I is mapped to an interval $f(I)$ of length the length of I. Hence, when using the continuous-discrete approach for the edges, every peer has at most two outgoing edges for every function $f \in F$. Furthermore, once $i \geq \log \bar{n} + 1$, the interval $f(I)$ is contained in $R(\text{pred}(v)) \cup R(v) \cup R(\text{succ}(v))$. Hence, every peer has an outdegree of at most $2(\log \bar{n} + 1)$.

Next, we consider the diameter. Consider any two points $x, y \in [0, 1)$. Let $(x_1, x_2, \ldots), (y_1, y_2, \ldots) \in \{0,1\}^*$ be the binary representations of x and y. Using F, it takes at most k traversals of edges to adjust $(x_1, \ldots, x_k)$ to $(y_1, \ldots, y_k)$ in (U, E_F). Since for every edge $(x', y') \in E_F$ there is an edge (v', w') in G_F with $x' \in R(v')$ and $y' \in R(w')$, it takes at most $\log \bar{n} + 2$ edge traversals in G_F to get from the node v owning x (i.e., $x \in R(v)$) to the node w owning y (i.e., $y \in R(w)$). Hence, the diameter of the dynamic hypercube is $O(\log n)$.

Finally, it is not difficult to see that if the peers are assigned to a single level of the decomposition tree, say level d, then $G_F(V)$ contains as a subgraph the d-dimensional hypercube. Hence, because the peers are only spread across two consecutive levels of the decomposition tree, the expansion of $G_F(V)$ is at least half of the expansion of the $\log \bar{n}$-dimensional hypercube, which is $\Omega(1/\log n)$. □

Next, we show how to maintain a supervised 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. Hence, it follows:

Theorem 8.9 *Using our framework, the supervisor can maintain a dynamic de Bruijn network with work and time $O(1)$ for each join and leave request.*

Also, similar to Theorem 8.8 it holds:

Theorem 8.10 *Using our framework, it holds that at any time, the dynamic de Bruijn network has an outdegree of $O(1)$, a diameter of $O(\log n)$ and an expansion of $\Omega(1/\log n)$, where n is the number of peers in the system.*

Proof. It is not difficult to check that whenever the peers are assigned to nodes in a single level of the decomposition tree, say level d, then $G_F(V)$ is exactly equal to the d-dimentional de Bruijn graph. In the worst case, the peers can only be distributed across two consecutive levels of the decomposition tree, which implies the theorem. $\square$

Also dynamic versions of expander graphs can be constructed. Recall the Gabber-Galil graph in Section. Here, we can set $U = [0, 1)^2$ and $F = \{f_1, f_2\}$ with

$$f_1(x, y) = (x, (x + y) \bmod 1) \quad \text{and} \quad f_2(x, y) = ((x + y) \bmod 1, x)$$

to obtain a continuous version of this graph. Also this version can be maintained with constant work and time for the supervisor for each join and leave requests. Furthermore, the following result holds.

Theorem 8.11 *Using our framework, it holds that at any time, the dynamic Gabber-Galil network has an outdegree of $O(1)$, a diameter of $O(\log n)$ and an expansion of $\Theta(1)$, where n is the number of peers in the system.*

Many more topologies are possible.

8.3 Robustness against random faults

So far we assumed that the peers leave gracefully, i.e., they announce their departures to the supervisor. In reality, however, also ungraceful departures can occur. In this section we show how to handle this case under the assumption that ungraceful departures are uniformly distributed over the cycle and the rate of ungraceful departures is slow enough for the supervisor to handle. Towards this goal we require the supervisor to maintain the following invariants for some k to be specified later.

Invariant 8.12 *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* $V' \subseteq V$, $R(V') = \bigcup_{v \in V'} R(v)$.

That is, now we have some redundancy in the system, since every peer takes care of all the edges in its k-neighborhood.

Invariant 8.13 *The supervisor maintains* join connections *and* repair connections.

- *Join connections are to v, the $2k$ closest predecessors of v and the $3k$ closest successors of v where v is the peer with label $\ell(n-1)$.*
- *Repair connections are to some (missing or fixed) peer w, the k closest predecessors of w and the k closest successors of w.*

The join connections are used for the peers that want to join the system, and the repair connections are used to refill the positions of peers that have left the system in an ungraceful manner.

The join operation works as for graceful departures. Whenever a new peer joins the system, the supervisor introduces it to existing peers so that its and the existing peers' connections satisfy Invariant 8.4. Afterwards, the supervisor requests new connections so that it can maintain its own invariant. The only difference to the graceful case is that whenever a peer is missing, it is ignored by the supervisor. Missing peers are exclusively handled by its repair connections.

The leave operation works in a way that the gracefully leaving peer w tells the supervisor all of its connections so that the supervisor can reverse the last join operation and can use the peer in that operation to fill the position of w. Missing peers are ignored and left to the repair connections.

The repair connections work as follows. Suppose the repair connections are currently centered around some intact peer w. If at least one of the k closest successors of w is missing, then choose the closest one as the new peer w, request the missing successors of it, and execute the Leave operation for w (which can be executed by the supervisor because all connectivity information of w is available). Otherwise, wait until the first successor of w leaves and then do the same as above.

These rules have the following property.

Theorem 8.14 *If $k = \Theta(\log n)$ is sufficiently large and ungraceful departures are distributed uniformly at random on the cycle and the rate of departures is low enough so that they can be handled by the supervisor, then for any (U,F) in our framework the supervisor can maintain a graph in which every $(x,y) \in E_F$ has at least one pair of working peers (v,w) so that $x \in R(N_v)$, $y \in R(N_w)$ and v has an edge to w.*

Proof. (Sketch) We need to show two lemmas.

Lemma 8.15 *In one round through the cycle, the repair connections handle all ungraceful departures that happened up to the beginning of that round.*

Proof. At any time, the supervisor knows exactly which positions in the cycle should be occupied, which are those positions with labels $\ell(0),\ldots,\ell(n-1)$. Hence, the supervisor can determine the positions with missing peers and repair them. Since the supervisor always repairs the closest missing successor, on its tour around the cycle, it will not leave out any position the a peer that left in an ungraceful way before the beginning of the tour. □

In the following, a position on the cycle is called *valid* if it is associated with a label in $\{\ell(0), \ldots, \ell(n-1)\}$.

Lemma 8.16 *Suppose that every peer fails with a constant probability* $0 < p < 1$. *Then there is an* $s = \Theta(p^{-1}\log n)$ *so that for every sequence of* s *consecutive valid positions, the probability that all or none of them have a failed peer is polynomially small in* n.

Proof. Consider some fixed sequence of s consecutive valid positions. The probability that all of them have a failed peer is equal to

$$p^s \leq n^{-c}$$

if $s \geq (c \log n)/\log p$, and the probability the none of them have a failed peer is equal to

$$(1-p)^s \leq n^{-c'}$$

if $s \geq (c' \log n)/\log(1-p)$. Since $0 < p < 1$, there is an $s = \Theta(\log n)$ so that both probabilities are polynomially small in n. In this case, the probability is also polynomially small in n that there is any sequence of s consecutive valid positions where all or none of them have a failed peer, proving the lemma. □

Suppose that in the time it takes for the supervisor to perform a complete tour around the cycle with its repair connections *without* waiting for failures, at most an ϵ fraction of the peers can fail for some sufficiently small constant $\epsilon > 0$. Then Lemma 8.16 implies that every repair tour through the cycle with waiting for failures takes an amount of time in which at most a constant fraction of the peers fail, with high probability. Since all previous failures have been handled by previous repair tours, failures can be kept at a constant fraction so that the supervisor can keep up with the failures, with high probability. In this case, it holds that for any sequence of k consecutive valid positions at least one of them is occupied by a work peer, which implies that every $(x, y) \in E_F$ has at least one pair of working peers (v, w) so that $x \in R(N_v)$, $y \in R(N_w)$ and v has an edge to w.

(A more formal proof will be given here in the future.) □

8.4 Robustness against adversarial behavior

When considering adaptive adversarial attacks, it does not suffice that the supervisor maintain information as in the previous subsection as the adversary can place nodes at critical positions to effectively disconnect the supervisor from the network or disrupt routing.

Formally, we allow the adversary to own up to ϵn of the n nodes in the system for some sufficiently small constant $\epsilon > 0$. These nodes are also called *adversarial* nodes and the rest are called *honest* nodes. The supervisor and the honest nodes are oblivious to adversarial nodes, i.e., there is no mechanism to distinguish at any time whether a particular node is honest or not. To achieve robustness in the presence of an adaptive adversary, we use the following scheme.

In the following, a *region* is an interval of size $1/2^i$ in $[0, 1)$ starting at an integer multiple of $1/2^i$ for some $i \geq 0$, and a node v belongs to a region R if $r(\ell_v) \in R$. Recall that $\overline{n} = 2^{\lfloor \log n \rfloor}$. The supervisor organizes the nodes into non-overlapping *quorum regions* so that each region contains between $c \log \overline{n}$ and $2c \log \overline{n}$ nodes for some constant $c > 1$. Whenever these bounds are violated in a quorum region, the supervisor splits it or merges it with a neighboring region. The n nodes are also organized into 5 sets S_1 to S_5 and the following invariant is maintained for these sets.

Invariant 8.17 *At all times,*

1. *S_1 has $\bar{n}/8$ nodes with labels $\ell(0), \cdots, \ell(\overline{n}/8 - 1)$.*
2. *S_2 has $\overline{n}/8$ nodes with labels $\ell(\overline{n}/8), \cdots, \ell(\overline{n}/4 - 1)$.*
3. *S_3 has $\overline{n}/4$ nodes with labels $\ell(\overline{n}/4), \cdots, \ell(\overline{n}/2 - 1)$.*
4. *S_4 has $\overline{n}/2$ nodes with labels $\ell(\overline{n}/2), \cdots, \ell(\overline{n} - 1)$.*
5. *S_5 has the remaining $n - \overline{n}$ nodes with labels $\ell(\overline{n}), \cdots, \ell(n - 1)$.*

The following invariants describe the connections maintained by the nodes in the various sets and the connections maintained by the supervisor. To simplify notation, for a real number $x \in [0, 1)$, $R(x)$ is the quorum region that x belongs to and $S_i(R)$ is the set of S_i-nodes belonging to R. For every quorum region R, let $S_R = S_1(R) \cup S_2(R)$ and $\bar{S}_R = S_3(R) \cup S_4(R) \cup S_5(R)$ if R precedes $R(r(\ell(n-1)))$, and otherwise, $S_R = S_1(R)$ and $\bar{S}_R = S_2(R) \cup S_3(R) \cup S_4(R) \cup S_5(R)$.

Invariant 8.18 *For all quorum regions R, every S_R-node is connected to all nodes in $S_R \cup \bar{S}_R$. Every S_R-node is also connected to all nodes in the predecessor and successor regions of R, denoted* $\mathrm{pred}(R)$ *and* $\mathrm{succ}(R)$, *and for every $u \in S_R$ that has a connection to a node $v \in S_{R'}$ according to the continuous-discrete approach, all S_R-nodes are connected to all $S_{R'}$-nodes.*

Invariant 8.19 *The supervisor maintains* join connections *and* mixing connections.

- *Join connections are to all the nodes in S_R in the regions $R(r(\ell(n-1)))$, $\mathrm{pred}(R(r(\ell(n-1))))$ and $\mathrm{succ}(R(r(\ell(n-1))))$.*
- *Mixing connections are to all nodes in $S_1(R)$, $S_2(R)$ and $S_3(R)$ for some quorum region R. The supervisor also has some special connection to a marked node $\hat{v} \in S_1(R) \cup S_2(R) \cup S_3(R)$.*

The set S_1 is referred to as the *stable* set. The goal of the supervisor is to have the honest nodes in the majority in every set $S_1(R)$, with high probability, since then quorum strategies can be used to wash out adversarial behavior. The set S_2 is in a stage called the *split-and-merge* stage because S_2-nodes are merged into the stable set or removed from it as nodes join or leave the system. The set S_3 is in a stage called *mixing* stage in which the supervisor performs random transpositions to ensure that the nodes are well-mixed before being integrated into the stable set. The set S_4 is in a *reservoir* stage. S_4 is used to fill departed positions in the sets S_1 to S_3 by selecting random nodes in S_4 and filling their positions with the last nodes in S_5. Finally, the set S_5 is in a *filling* stage where new nodes are added by assigning them the label $\ell(n-1)$.

The join and leave operations are extended as follows.

Join operation

The supervisor assigns to the new node v the label $\ell(n)$ and integrates it so that the Invariants 8.17 and 8.18 are satisfied. Afterwards, the supervisor takes the closest successor of $\hat{v}$ among the nodes in $S_1(R) \cup S_2(R) \cup S_3(R)$ of the mixing region R and calls this node the new $\hat{v}$. If there is no such node, the supervisor moves to the quorum region R' succeeding the region R and assigned $\hat{v}$ to the first node in $S_1(R') \cup S_2(R') \cup S_3(R')$. Suppose that the new $\hat{v}$ belongs to S_i. Then the supervisor chooses a node w with position between $\hat{v}$ (inclusive) and 1 (exclusive) uniformly at random and exchanges (or more precisely, asks the current mixing region to exchange) the positions of $\hat{v}$ and w. The supervisor finally updates its connections so that Invariant 8.19 is satisfied. Each time a new node causes the supervisor to switch from a join region R to $\mathrm{succ}(R)$, the nodes in $S_2(R)$ are merged into $S_1(R)$ as prescribed by Invariant 8.18.

Leave operation

If a node v leaves with $v \in S_4 \cup S_5$, the supervisor simply replaces it by the last node in S_5. Otherwise, the supervisor replaces v by a random node in S_4 (resp. orders the current join region to do this) which is replaced by the last node in S_5, and the supervisor performs a mixing operation in the current mixing region in the same way as for a join operation. (The supervisor initiates the leave operation for v only if a majority of S_1-nodes in v's region notify it about that. In this case, the supervisor can be sure that v has indeed left so that it correctly initiates v's replacement.) Each time a departure causes the supervisor to switch from a region R to $\mathrm{pred}(R)$, the nodes in $S_2(\mathrm{pred}(R))$ are split away from $S_1(R)$ as prescribed by Invariant 8.18.

These operations yield the following result.

Theorem 8.20 *For a sufficiently small constant $\epsilon > 0$ it holds that as long as the adversary owns at most ϵn nodes, the above scheme guarantees that in every region R, the honest nodes are in the majority in $S_1(R)$, with high probability.*

Proof. (Sketch) The following lemma is crucial for the theorem.

Lemma 8.21 *Once a pass has been made through all positions of S_i, $i \in \{1, 2, 3\}$, the positions in S_i form a random permutation.*

Proof. Given m positions, consider the random experiment of first switching the first position with a position in $\{1, \ldots, m\}$ chosen uniformly at random, then switching position 2 with a position in $\{2, \ldots, n\}$ uniformly at random, and so on, until position m is reached. This random experiment creates a random permutation of the m positions for the following reasons:

- Every permutation has exactly one outcome of the random experiment.
- Every outcome of the random experiment is equally probable.

□

Hence, it takes at most n join and leave operations (where n is the initial number of nodes) until all nodes in S_1, S_2 and S_3 that were initially around are randomly permuted. For every new node in S_1, S_2

and S_3 a random node in S_4 is picked, which is adversarial with probability at most $(\epsilon n)/(\bar{n}/2) \leq 4\epsilon$. In the worst case, all honest nodes in S_3 leave and the adversarial nodes stay. In this case, a simple probability analysis can show that as long as $\epsilon > 0$ is a sufficiently small constant, the adversarial nodes will not be in the majority in any quorum region.

If in every quorum region the honest nodes in S_1 are in the majority, then adversarial behavior can be washed out in all operations so that the supervised overlay network works correctly. $\square$

8.5 Applications

Finally, we discuss some applications of the supervised overlay networks that arise in the area of distributed computing.

Grid Computing

Recently, many systems such as SETI@home [6], Folding@home [2], and Distributed.net [1] have been proposed for distributed computing. A main drawback of such systems is that the topology of the system is a star graph with the central server maintaining a direct connection to each client. Such a topology imposes heavy demands on the central server. Instead, we can use our general framework for supervised overlay networks to maintain an overlay network for distributed computing. Peer-to-peer connections allow subtasks to be spawned without the involvement of the supervisor so that the demands on the server can be significantly reduced. This is particularly interesting for distributed branch-and-bound computations as was discussed in [5].

WebTv

Our approach can also be used in Internet applications such as WebTv. In such an application, there are typically various channels that users can browse or watch while being connected to the Internet. The number of channels ranges in the scale of hundreds while the number of users can range in the scale of millions. Such a system should allow users to quickly zap through channels. Hence, such a system should allow for rapid integration and be scalable to a large number of users. Our supervised overlay networks can easily achieve such a smooth operation. Suppose that every channel has a supervisor, each supervisor maintains its own broadcast network, and the supervisors form a clique. Then it follows from our supervised approach, which can handle join and leave operations in constant time, that users browsing through channels can be moved between the networks in a very fast way, comparable to server-based networks, so that users only experience an insignificant delay.

Massive multi-player online gaming

Distributed architectures for massive multi-player online gaming (MMOG) have only recently been studied formally (see e.g., [3]). The basic requirements of such a system includes authentication, scalability, and rapid integration. Traditionally, such systems have been managed by a central server that takes care of the overall system with limited communication between the users. Certainly, such a system will not be scalable and also might experience heavy congestion at the central server. Hence, distributed architectures are required at a certain scale. A supervised overlay network approach can help here. For example, in a large virtual world, every supervisor may be responsible for a certain part of the world, and the supervisors may be interconnected like a cellular network to allow a fast handover process between them. Each supervisor then takes care of the peers currently exploring its part of the world. Since in our supervised approach peers can quickly be integrated and removed from a network, the handover process can be realized in a very fast way so that even fast moving peers can be handled. Additional supervisors may also be used for load balancing purposes in a sense that whenever a supervisor is heavily loaded, other supervisors may help out by taking over some of its peers and/or parts of the virtual world. In this way, it should be possible to create new generations of games in very complex worlds.

References

[1] Distributed.net. Available at http://www.distributed.net/.

[2] Folding@home. Available at http://folding.stanford.edu/.

[3] C. GauthierDickey, D. Zappala, and V. Lo. A fully distributed architecture for massively multiplayer online games. In *ACM Workshop on Network and System Support for Games*, 2004.

[4] 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.

[5] C. Riley and C. Scheideler. A distributed hash table for computational grids. In *18th Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[6] SETI@home. Available at http://setiathome.berkeley.edu/.