Supervised Peer-to-Peer Systems

Kishore Kothapalli and Christian Scheideler*
Department of Computer Science
Johns Hopkins University
3400 N. Charles Street
Baltimore, MD 21218, USA
Email:{kishore,scheideler}@cs.jhu.edu

Abstract

In this paper we present a general methodology for designing supervised peer-to-peer systems. A supervised peer-to-peer system *is a system in which the overlay network is formed by a supervisor but in which all other activities can be performed on a peer-to-peer basis without involving the supervisor. It can therefore be seen as being between server-based systems and pure peer-to-peer systems. The supervisor only has to store a* constant *amount of information about the system at any time and only needs to send a* small constant *number of messages to integrate or remove a peer in a* constant *amount of time. Thus, with a minimum amount of involvement from the supervisor, peer-to-peer systems can be maintained, for example, that can handle large distributed computing tasks as well as tasks such as file sharing and web crawling. Furthermore, our concept extends easily to multiple supervisors so that peers can join and leave the network massively in parallel. We also show how to extend the basic system to provide robustness guarantees under the presence of random faults and also adaptive adversarial join/leave attacks. Hence, with our approach, supervised peer-to-peer systems can share the benefits of server-based and pure peer-to-peer systems without inheriting their disadvantages.*

1. Introduction

Peer-to-peer systems have recently attracted a significant amount of attention inside and outside of the research community. The advantage of peer-to-peer systems is that they can scale to millions of sites with low-cost hardware whereas the classical approach of using server-based systems does not scale well, unless powerful servers are provided. On the other hand, server-based systems can provide guarantees and are therefore preferable for critical applications that need a high level of reliability. The question is whether it is possible to marry the two approaches in order to share their benefits without sharing their disadvantages. We propose supervised peer-to-peer systems as a possible solution to this.

A *supervised peer-to-peer system* is a system in which the overlay network is formed by a supervisor but in which all other activities can be performed on a peer-to-peer basis without involving the supervisor. That is, all peers that want to join (or leave) the network have to contact the supervisor, and the supervisor will then initiate their integration into (or removal from) the network. All other operations, however, may be executed without involving the supervisor. In order for a supervised network to be highly scalable, two central requirements have to be fulfilled:

1. The supervisor needs to store at most a polylogarithmic amount of information about the system at any time (i.e. if there are n peers in the system, storing contact information about $O(\log^2 n)$ of these peers would be fine, for example), and
2. the supervisor needs at most a constant number of messages to include a new peer into or exclude an old peer from the network.

The second condition makes sure that the work of the supervisor to include or exclude peers from the system is kept at a minimum. Still, one may certainly wonder whether supervised peer-to-peer systems are really as scalable as pure peer-to-peer systems on the one hand and as reliable as server-based systems on the other hand.

1.1. Motivation

First of all, remember that even pure peer-to-peer systems need some kind of a "rendezvous point", such as a well-known host server [13] or well-known web address (like gnutellahosts.com), which allows new peers to join the system. The rendezvous point typically does not play any role in the overall topology of the network but just acts as

* Supported by NSF grants CCR-0311121 and CCR-0311795.

a bridge between new nodes and the existing network. This means that nodes have to self-organize to form an overlay network with good topological properties such as diameter, degree and expansion.

We show that allowing the supervisor to oversee the topology of the overlay network, apart from working as the rendezvous point, tremendously simplifies the problem of maintaining the above mentioned topological properties of the overlay network. Hence, as long as the communication effort of a supervisor for including or excluding a peer is only a low constant, supervised designs should compete well with pure peer-to-peer systems.

Our approach has many interesting applications in the area of grid computing [6, 16, 19], WebTV, and massive multi-player online gaming [9]. A supervisor may also serve, for example, as a reliable anchor for code execution rollback, which is important for failure recovery mechanisms such as those used in the Time Warp system [7]. This would make supervised peer-to-peer systems particularly interesting for grid computing. Though supervised peer-to-peer systems are not as stable as server-based systems with powerful servers, their advantage is that because the supervisor only takes care of the topology but may not be involved at all in peer-to-peer activities, it is from a legal point of view a much safer design than the server-based design.

1.2. Our contribution

In Section 2, we show how to combine known techniques in the pure peer-to-peer world such as the hierarchical decomposition approach of CAN [14] and the continuous-discrete approach [12] in a novel way to obtain a general framework for the design of supervised peer-to-peer systems that only requires the supervisor to store a *constant* amount of information about the system at any time and to only send and receive a *low constant* number of messages in order to integrate or remove a peer from the system. We demonstrate our approach by showing how to maintain a supervised hypercube network and a supervised de Bruijn network with it. Our scheme can also be extended to allow concurrent join/leave operations or allow multiple supervisors as outlined in Section 3. In Section 4 we look at robustness issues and discuss how our supervised design can be extended to handle random or adversarial faults.

1.3. Related work

Special cases of supervised peer-to-peer systems have already been formally investigated in [13, 16, 17], but to the best of our knowledge a general framework for supervised peer-to-peer systems has not been presented yet. In [13], the authors consider a special node called the *host server* that is contacted by all new peers that join the system. The overlay network maintained by the host server is close to a random-looking graph. As shown by the authors, under a stochastic model of join/leave requests the overlay network can, with high probability, guarantee connectivity, low diameter, and low degree. Alternative designs were later proposed in [16, 17]. [17] shows how to maintain supervised trees for guaranteed broadcasting and [16] shows how to maintain a supervised de Bruijn graph for grid computing. In this work, we propose a unified model that enables one to create a large class of supervised overlay networks.

Most of the distributed systems are either server-based or peer-to-peer. For example, Napster is rather server-based because all peer requests are handled at a single location. Also systems like SETI@home [19], Folding@home [8], and distributed.net [6] are heavily server-oriented because they do not allow peer-to-peer interactions. Other systems such as the IBM OptimalGrid allow communication between peers but it still uses a star topology and therefore is still closer to being server-based than supervised.

The line of research that is probably closest to our approach is the work on overlay networks in the area of application-layer multicasting. Among them are SpreadIt [5], NICE [1], Overcast [10], and PRM [2], to name a few. However, these systems only focus on specific topologies such as trees, and they do not seem to be generalizable to a universal approach for supervised systems. Other protocols for application-layer multicasting such as Scribe [4], Bayeux [23], I3 [20], Borg [22], SplitStream [3], and CAN-Multicast [15] are rather extensions of a pure peer-to-peer system.

2. A general framework for supervised peer-to-peer systems

Our general framework for supervised peer-to-peer systems needs several ingredients, including a hierarchical decomposition technique [14], a continuous-discrete technique [12], and a recursive labeling technique. After presenting these techniques we show how to glue them together in an appropriate way so that we obtain a universal approach for supervised peer-to-peer systems. Afterwards, we give some examples that demonstrate how to apply this approach.

2.1. The hierarchical decomposition technique

Consider any space $U = [0,1)^d$ for some $d \geq 1$. 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, ℓ_v, is the sequence of all edge labels encountered when moving along the unique path from the root of $T(U)$ downwards to v.

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

- the subcubes of the (nodes assigned to the) peers are disjoint,
- the union of the subcubes of the peers gives the entire set U, and
- the peers are only distributed among nodes of two consecutive levels in $T(U)$.

Whereas CAN-based peer-to-peer systems usually satisfy the first two properties, they have problems with the third property. But as we will see, it will be easy for our supervised peer-to-peer approach to also maintain the third property.

2.2. The continuous-discrete technique

The basic idea underlying the continuous-discrete approach [12] is to define a continuous model of graphs and to apply this continuous model to the discrete setting of a finite set of peers.

Consider any d-dimensional space $U = [0,1)^d$, and suppose that we have a set F of functions $f_i : U \to U$. Then we define E_F as the set of all pairs $(x, y) \in U^2$ with $y = f_i(x)$ for some i. Given any subset $S \subseteq U$, let $\Gamma(S) = \{y \in U \setminus S \mid \exists x \in S : \ (x,y) \in E_F\}$. The *expansion* α of F is defined as $\alpha = \min_{S \subseteq U,\ |S| \leq |U|/2} \frac{|\Gamma(S)|}{|S|}$ where $|S|$ denotes the volume of a set S. F is called *mixing* if $\alpha > 0$ and *rapidly mixing* if α is a constant. 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 region in U that has been assigned to peer v. Let $G_F(V)$ be the graph with node set V that contains an edge (v, w) for every pair of nodes v and w for which there is an edge $(x, y) \in E_F$ with $x \in R(v)$ and $y \in R(w)$. It can be seen that if F is mixing and $\cup_v R(v) = U$ then $G_F(v)$ is connected. Moreover, the following theorem holds implying that the expansion in the continuous case can be preserved in the discrete case.

Theorem 2.1 *If F has an expansion of α and regions have been assigned to the peers so that all three demands stated in the hierarchical decomposition approach are satisfied, then $G_F(V)$ has an expansion of $\Omega(\alpha)$.*

2.3. The recursive labeling technique

In the recursive labeling approach, the supervisor assigns a *label* to every peer that wants to join the system. The labels are represented as binary strings and are generated in the order: $0, 1, 01, 11, 001, 011, 101, 111, 0001, 0011, \ldots$. Formally, consider the mapping $\ell : \mathbb{N}_0 \to \{0,1\}^*$ with the property that for every $x \in \mathbb{N}_0$ with binary representation $(x_d \ldots x_0)_2$ (where d is minimum possible), $\ell(x) = (x_{d-1} \ldots x_0 x_d)$. Then ℓ generates the sequence of labels displayed above. In the following, it will also be helpful to view labels as real numbers in $[0,1)$. Let the function $r : \{0,1\}^* \to [0,1)$ be defined so that for every label $\ell = (\ell_1 \ell_2 \ldots \ell_d) \in \{0,1\}^*$, $r(\ell) = \sum_{i=1}^{d} \frac{\ell_i}{2^i}$. Then the sequence of labels above translates into $0,\ 1/2,\ 1/4,\ 3/4,\ 1/8,\ 3/8,\ 5/8,\ 7/8,\ 1/16,\ 3/16, \ldots$. When using the recursive approach, the supervisor aims to maintain the following invariant at any time:

Invariant 2.2 *The set of labels used by the peers is $\{\ell(0), \ell(1), \ldots, \ell(n-1)\}$, where n is the current number of peers in the system.*

This above invariant can be preserved when using the simple strategy of assigning the label $\ell(n)$ to a new peer that wants to join the system and increasing n by 1. Similarly, when a peer w with label ℓ is leaving the system, the supervisor asks node with label $\ell(n-1)$ to take over the role and label of w and decrements n by 1.

2.4. Putting all pieces together

Now we are ready to put the pieces together. We assume that we have a single supervisor for maintaining the overlay network. In the following, the label assigned to some peer v will be denoted as ℓ_v. Given n peers with unique labels, we define the *predecessor* $\mathrm{pred}(v)$ of peer v as the peer w for which $r(\ell_w)$ is closest from below to $r(\ell_v)$, and we define the *successor* $\mathrm{succ}(v)$ of peer v as the peer w for which $r(\ell_w)$ is closest from above to $r(\ell_v)$ (viewing $[0,1)$ as a ring in both cases). Given two peers v and w, we define their *distance* as $\delta(v,w) = \min\{(1 + r(\ell_v) - r(\ell_w)) \bmod 1,\ (1 + r(\ell_w) - r(\ell_v)) \bmod 1\}$. In order to maintain a doubly linked cycle among the peers, we simply have to maintain the following invariant:

Invariant 2.3 *Every peer v in the system is connected to* $\mathrm{pred}(v)$ *and* $\mathrm{succ}(v)$.

Now, suppose that the labels of the peers are generated via the recursive strategy above. Then we have the following properties:

Lemma 2.4 *Let n be the current number of peers in the system, and let $\bar{n} = 2^{\lfloor \log n \rfloor}$. Then for every peer $v \in V$, $|\ell_v| \leq \lceil \log n \rceil$ and $\delta(v, \mathrm{pred}(v)) \in \{1/(2\bar{n}), 1/\bar{n}\}$.*

So the peers are approximately evenly distributed in $[0,1)$ and the number of bits for storing a label is as low as it can be without violating the uniqueness requirement.

Now, recall the hierarchical decomposition approach. The supervisor will assign every peer p to the unique node v in $T(U)$ at level $\log(1/\delta(p, \mathrm{pred}(p)))$ with ℓ_v being equal to ℓ_p (padded with 0's to the right so that $|\ell_v| = |\ell_p|$).

As an example, if we have 4 peers currently in the system, then the mapping of peer labels to node labels is $0 \rightarrow 00,\ 1 \rightarrow 10,\ 01 \rightarrow 01,\ 11 \rightarrow 11$. With this strategy, it follows from Lemma 2.4 that all three demands formulated in the hierarchical decomposition approach are satisfied.

Consider now any family F of functions acting on some space $U = [0,1)^d$ and let $C(p)$ be the subcube of the node in $T(U)$ that p has been assigned to. Then the goal of the supervisor is to maintain the following invariant at any time.

Invariant 2.5 *For the current set V of peers in the system it holds that*

1. *the set of labels used by the peers is $\{\ell(0), \ell(1), \ldots, \ell(n-1)\}$, where $n = |V|$,*
2. *every peer v in the system is connected to $\mathrm{pred}(v)$ and $\mathrm{succ}(v)$, and*
3. *there are bidirectional connections $\{v, w\}$ for every pair of peers v and w for which there is an edge $(x, y) \in E_F$ with $x \in C(v)$ and $y \in C(w)$.*

2.5. Maintaining Invariant 2.5

Next we describe the actions that the supervisor has to perform in order to maintain Invariant 2.5 during an isolated join or leave operation. For simplicity, we assume that all nodes are reliable and trustworthy and also that peers depart gracefully. We also assume that each message sent by the supervisor can contain up to a constant number of node addresses and labels. We start with the following important fact which can be easily shown.

Fact 2.6 *Whenever a new peer v enters the system, then $\mathrm{pred}(v)$ has all the connectivity information v needs to satisfy Invariant 2.5(3), and whenever an old peer w leaves the system, then it suffices that it transfers all of its connectivity information to $\mathrm{pred}(w)$ in order to maintain Invariant 2.5(3).*

Thus, if the peers take care of the connections in Invariant 2.5(3), the only part that the supervisor has to take care of is maintaining the cycle. For this we require the following invariant.

Invariant 2.7 *At any time, the supervisor stores the contact information of $\mathrm{pred}(v)$, v, $\mathrm{succ}(v)$, and $\mathrm{succ}(\mathrm{succ}(v))$ where v is the peer with label $\ell(n-1)$.*

We now describe how to maintain the invariant during any join or leave operation. In the following, S denotes the supervisor.

Join: If a new peer w joins, in order to satisfy Invariant 2.7, the following actions are performed.

- S informs w that $\ell(n)$ is its label, $\mathrm{succ}(v)$ is its predecessor, and $\mathrm{succ}(\mathrm{succ}(v))$ is its successor.
- S informs $\mathrm{succ}(v)$ that w is its new successor.
- S informs $\mathrm{succ}(\mathrm{succ}(v))$ that w is its new predecessor.
- S asks $\mathrm{succ}(\mathrm{succ}(v))$ to send its successor information to the supervisor, and
- S asks v which is now $\mathrm{pred}(w)$ to send the connectivity information according to F to node w.
- S sets $n = n + 1$.

Leave: If an old node w reports ℓ_w, $\mathrm{pred}(w)$ and $\mathrm{succ}(w)$ before it leaves, then the following actions can be performed in order to maintain Invariant 2.7.

- S informs v (the node with label $\ell(n-1)$) that ℓ_w is its new label, $\mathrm{pred}(w)$ is its new predecessor, and $\mathrm{succ}(w)$ is its new successor.
- S informs $\mathrm{pred}(w)$ that its new successor is v and $\mathrm{succ}(w)$ that its new predecessor is v.
- S informs $\mathrm{pred}(v)$ that $\mathrm{succ}(v)$ is its new successor and $\mathrm{succ}(v)$ that $\mathrm{pred}(v)$ is its new predecessor.
- S asks $\mathrm{pred}(v)$ to send its predecessor information to the supervisor and to ask $\mathrm{pred}(\mathrm{pred}(v))$ to send its predecessor information to the supervisor, and
- S sets $n = n - 1$.

Thus, the supervisor only needs to handle a constant number of messages, at most 8, for each join/leave of a peer.

2.6. Examples

For a supervised hypercubic network, simply select F as the family of functions on $[0,1)$ with $f_i(x) = x + 1/2^i \pmod 1$ for every $i \geq 1$. Using our framework, this gives an overlay network with degree $O(\log n)$, diameter $O(\log n)$, and expansion $O(1/\sqrt{\log n})$, which is better than what pure hypercubic peer-to-peer systems like Chord [21] can achieve.

For a supervised de Bruijn network, simply select F as the family of functions on $[0,1)$ with $f_0(x) = x/2$ and $f_1(x) = (1+x)/2$. Using our framework, this gives an overlay network with degree $O(1)$, diameter $O(\log n)$, and expansion $O(1/\log n)$, which is also better than the previous pure de Bruijn peer-to-peer systems [11, 12].

3. Concurrency

In this section we extend our approach to concurrent join and leave operations and also provide a way to allow multiple supervisors.

3.1. Concurrent Join/Leave Operations

In order to be able to handle d join or leave requests in parallel, we extend Invariant 2.5 with the following rule:

4. Every peer v in the system is connected to d predecessors and $\mathrm{pred}_i(v)$ for $i = 1, 2, \cdots, d$ and d successors $\mathrm{succ}_i(v)$ for $i = 1, 2, \cdots, d$, where $\mathrm{pred}_i(v)$ (resp. $\mathrm{succ}_i(v)$) is the ith predecessor (resp. successor) of v on the cycle.

In addition to this, given that v is the node with label $\ell(n-1)$, Invariant 2.7 needs to be extended to:

Invariant 3.1 *At any time, the supervisor stores the contact information of v, the $2d$ successors of v, and the $3d$ predecessors of v.*

These invariants can be preserved during join/leave operations and the following claim holds:

Claim 3.2 *The supervisor needs at most $O(d)$ work and $O(1)$ time (given that the work can be done in parallel) to process d join or leave operations.*

3.2. Multiple Supervisors

We now show how multiple supervisors can work together in maintaining a single supervised peer-to-peer system. In a network with k supervisors $S_0, S_1, \cdots S_{k-1}$, the $[0,1)$-ring is split into the k regions $R_i = [(i-1)/k, i/k)$, $1 \leq i \leq k$, and supervisor S_i is responsible for region R_i. Every supervisor manages its region as described for a single supervisor above, and the borders are maintained by communicating with the neighboring supervisors on the ring. Each time a new node v wants to join the system via some supervisor S_i, S_i forwards it to a random supervisor to integrate v into the system. Each time a node v under some supervisor S_i wants to leave the system, S_i contacts a random supervisor (which may also be itself) to provide a replacement node. Using Chernoff bounds we get:

Claim 3.3 *Let n be the total number of nodes in the system. Then it holds for every $i \in \{1, \ldots, k\}$ that the number nodes currently placed in R_i is in the range $n/k \pm O(\sqrt{(n/k)\log k} + \log k)$, with high probability.*

4. Robustness

In this section we show how to extend the basic scheme to provide robustness guarantees against random node failures and also adaptive adversarial join/leave attacks [18].

4.1. Robustness against random faults

We call a supervised network robust if as long as at most a constant fraction of the nodes fail, the supervisor can still recover the rest of the network. In order to be robust against random faults, the supervisor uses the scheme of Section 3.1 with $d = c \log n$ for some constant $c > 1$. Thus, each node is connected to $c \log n$ predecessors and successors. The supervisor stores the contact information along the lines of Invariant 3.1. The following theorem can be shown.

Theorem 4.1 *Even if every node in the network fails independently with a constant probability, the scheme above in which the supervisor only maintains a logarithmic amount of information at any time suffices to fully restore the network and each leave operation executed during the recovery needs $O(\log n)$ work, with high probability.*

4.2. Robustness against adversarial attacks

When considering adaptive adversarial attacks (e.g., [18]) 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 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 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 4.2 *At all times,*

1. *S_1 has $\overline{n}/8$ 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 invariant describes 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 region that x belongs to and $S_i(R)$ is the set of S_i-nodes belonging to R. For every 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 $R(r(\ell(n)))$ 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 4.3 *For all 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 Invariant 2.5(3), all S_R-nodes are connected to all $S_{R'}$-nodes.*

The supervisor is connected to all the nodes in S_R in the regions $R(r(\ell(n)))$, $\mathrm{pred}(R(r(\ell(n))))$ and $\mathrm{succ}(R(r(\ell(n))))$.

The set S_1 is also 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 $\ell(n)$.

The join and leave operations are extended as follows.

Join: The supervisor assigns to the new node the label $\ell(n)$ and integrates it so that the Invariants 4.2 and 4.3 are satisfied. Each time a new node u is added where $r(\ell(n))$ is the successor of $r(\ell_v)$ for a node v at a position in S_3 that has not performed a random transposition yet, a random node $w \in S_3$ is picked and the positions of v and w are switched. (This is realized by the supervisor informing all nodes in $S_1(R(\ell(n)))$ which positions are to be switched so that this is reliably done without involving the supervisor.) Each time a new node causes the supervisor to switch from a region R to $\mathrm{succ}(R)$, the nodes in $S_2(R)$ are merged into $S_1(R)$ as prescribed by Invariant 4.3.

Leave: 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 and fills the position of that random node with the last node in S_5. (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 has the necessary information to correctly initiate the 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 4.3.

These operations yield the following result.

Theorem 4.4 *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.*

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. Technical Report UMIACS-TR 2002-53/CS-TR 4373, U. Maryland, College Park, 2002.

[2] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. In *ACM SIGMETRICS*, number 1, pages 102–113, 2003.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *ACM SIGMETRICS*, 2003.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20:1489–1499, 2002.

[5] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over a peer-to-peer network. Technical report, 2001-31, Stanford University, 2001.

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

[7] D. Jefferson et al. Distributed simulation and the time warp operating system. In *ACM SOSP*, 1987.

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

[9] 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.

[10] J. Jannotti, D.K. Gifford, K.L. Johnson, F. Kaashoek, and J.W. OToole. Overcast: Reliable multicasting with an overlay network. In *Proc. of OSDI*, pages 197–212, 2000.

[11] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *IPTPS*, 2003.

[12] M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *ACM SPAA*, pages 50–59, 2003.

[13] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter P2P networks. In *IEEE FOCS*, 2001.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, 2001.

[15] S. Ratnasamy, M . Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of 3rd International Workshop on Networked Group Communication*, 2001.

[16] C. Riley and C. Scheideler. A distributed hash table for computational grids. In *IEEE IPDPS*, 2004.

[17] C. Riley and C. Scheideler. Guaranteed broadcasting using SPON: A supervised peer overlay network. In *3rd International Zürich Seminar on Communications (IZS)*, 2004.

[18] C. Scheideler. How to spread adversarial nodes? rotate! In *ACM Symp. on Theory of Computing (STOC)*, 2005.

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

[20] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM*, 2002.

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

[22] R. Zhang and Y. C. Hu. Borg: a hybrid protocol for scalable application level multicast in peer-to-peer networks. In *IEEE Infocom*, 2003.

[23] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of NOSS-DAV*, 2001.