MOve: Design and Evaluation of a Malleable Overlay for Group-Based Applications

Ramsés Morales, *Student Member, IEEE,* Sébastien Monnet, Indranil Gupta, *Member, IEEE,* and Gabriel Antoniu, *Member, IEEE*

***Abstract*— While peer-to-peer overlays allow distributed applications to scale and tolerate failures, most structured and unstructured overlays in literature today are *inflexible* from the application viewpoint. The application thus has no first-class control on the overlay structure. This paper proposes the concept of an *application-malleable* overlay, and the design of the first malleable overlay which we call *MOve*. MOve is targeted at group-based applications, e.g., collaborative applications. In MOve, the communication characteristics of the distributed application using the overlay can *influence* the overlay's structure itself, with the twin goals of (1) optimizing the application performance by adapting the overlay, while also (2) retaining the large scale and fault tolerance of the overlay approach. Besides neighbor list membership management, MOve also contains algorithms for resource discovery, update propagation, and churn-resistance. The emergent behavior of the implicit mechanisms used in MOve manifests as follows: when application communication is low, most overlay links keep their default configuration; however, as application communication characteristics become more evident, the overlay *gracefully* adapts itself to the application. We validate MOve using simulations with group sizes that are fixed, uniform, exponential and PlanetLab-based (slices), as well as churn traces and two sample management-based applications.**

***Index Terms*— Peer-to-peer overlay, adaptability, malleable, group membership, volatility-resilience.**

I. INTRODUCTION

TODAY, peer-to-peer (P2P) overlays fall into two categories - (1) structured (i.e., *Distributed Hash Table*-based) overlays such as Pastry and Chord [2], [3], and (2) unstructured (i.e., gossip- or flooding-based) overlays such as Freenet, Gnutella, KaZaA [4]–[6]. These P2P overlays offer reliability in the face of massive failures and churn (node join and leave), as well as scalability to hundreds and thousands of nodes.

However, both these types of overlays have the common disadvantage that they are typically *inflexible* from the application viewpoint. The rules and invariants for selecting and maintaining neighbor nodes in the overlay, as well as for resource discovery, are all dictated in a rigid fashion (e.g., using the result of a hash function), without allowing the overlay to change according to the application's communication patterns. This usually means that the developer of a distributed application has a limited number of options – either go with the provided overlay, or design a new overlay from scratch. Furthermore, overlays are application-dependent and Internet-independent [7], so allowing an application to explicitly influence the overlay is a logical next-step.

In this paper, we propose the concept of an *application-malleable* overlay. An application-malleable overlay is defined as an overlay where the communication characteristics of the distributed application using the overlay can *influence* the overlay's structure itself. The twin goals of a malleable overlay are: (1) to optimize application performance from the overlay, while also (2) retaining the scale and fault tolerance of the overlay approach.

In order to realize and evaluate our design philosophy, we build a specific malleable overlay called *MOve* (for *Malleable OVErlay*) that combines elements of an *unstructured overlay* with application characteristics. In MOve, the structure and behavior of the overlay is influenced both by the underlying default unstructured overlay, and by application characteristics. The influence could either be (1) explicitly specified by the application or (2) implicitly gleaned by our algorithms.

In a P2P overlay, each node maintains a separate neighbor list - this is a membership list that specifies the *who knows whom* relationship. This neighbor list is partial in the sense that it contains only some of the nodes in the system [3], [8], [9]. MOve contains algorithms for neighbor list maintenance, efficient message propagation, and churn-resistance (i.e., resilience to nodes joining and leaving asynchronously). The most interesting feature of MOve is its emergent behavior. When application communication is low, MOve autonomically evolves to keep most of the overlay links in the default state so that most of the system looks like an unstructured overlay. However, as application communication characteristics become more and more evident, the overlay autonomically *gracefully* adapts itself to the application, but without forgetting its default structure.

To focus our approach on a particular class of applications, we choose *collaborative applications*, such as a distributed

Manuscript received September 1, 2006; revised March 1, 2007 and August 21, 2007; accepted August 21, 2007. The associate editors coordinating the review of this paper and approving it for publication were A. Keller and J. P. Martin-Flatin.

This material is based upon work supported by the National Science Foundation (NSF) under CAREER award CNS-0448246 and ITR award CMS-0427089. This work is also supported by a UIUC-INRIA collaboration grant. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF or other funding agencies. This journal paper is an advanced version of our SRDS [1] publication, and contains additional experimental (i.e., PlanetLab trace-based) and application-based data.

R. Morales and I. Gupta are with the University of Illinois at Urbana-Champaign (email: {rvmorale, indy}@cs.uiuc.edu).

S. Monnet and G. Antoniu are with IRISA/INRIA (email: {Sebastien.Monnet, Gabriel.Antoniu}@irisa.fr).

Digital Object Identifier 10.1109/TNSM.2007.070903.

whiteboard platform, an audio/video conferencing service, a replicated data-sharing service, or a distributed-gaming platform. All these applications rely on the notion of *application groups* - each process belongs to one or more groups, and interacts with other processes in common groups. For instance, the members of the same group may share a distributed state that needs to be updated (e.g., the whiteboard, the gameboard, the replicas of a mutable piece of data, etc.). Managing groups at the overlay level allows multiple applications to take advantage of this optimization without having to explicitly handle it.

MOve allows such *group*-based applications to influence the underlying overlay, that may be common to multiple, coexisting, collaborative applications. In the process of the neighbor list maintenance, MOve has the following two goals: (1) (*connectivity*) keep a low diameter for the overlay so that unstructured queries can be quickly propagated; and (2) (*volatility-resilience*) combat volatility arising from rapid node arrival and failure (i.e., "churn").

The basic idea in the MOve approach is to have each node maintain a neighbor list that, by default, consists of *non-application neighbors*, i.e., randomly selected neighbors. However, with the formation of more and more application groups, some of these non-application neighbors are automatically replaced by *application-aware* neighbors (shorthand: *application neighbors*). A non-application neighbor may either change status and turn into an application neighbor (if the neighbor belongs to a common group), or be replaced by a new application neighbor. We have implemented MOve, and our experiments show (1) that the system achieves logarithmic overlay path lengths for groups size distributions ranging from uniform to PlanetLab-slice-size-based ones; (2) that it gracefully manages transitions between application and non-application neighbors as the number of groups increases and decreases. In addition, MOve shows good volatility-resilience and scalability for two sample applications.

The next section presents research efforts related to the various aspects involved in this context. Then Section III describes a scenario motivating the need for efficient communication within groups of nodes. Section IV gives a general overview of our approach, and provides an analysis. Section V presents simulation results. Finally, Section VI concludes.

II. Related Work

In the past few years, many research efforts have focused on building overlays for peer-to-peer networks, essentially for large-scale immutable file-sharing. For this kind of application, predicting which node is going to communicate with which node is not trivial. Therefore, most algorithms for building overlays do not take communication patterns into account.

In unstructured (i.e., gossip- or flooding-based) P2P overlays, such as [5] or [10], neighbor lists are usually built and maintained by randomly selecting a subset of the neighbors' neighbors.

In structured (i.e., DHT-based) overlays, the *who knows whom* relation is usually defined by means of a given topology (typically an extended ring); the position of each node in this given topology is determined by a hash function of its IP address [2], [3].

Although some of these previous proposals take into account certain criteria while building the overlay, (e.g., physical locality in the case of [2]), they do not take into account the application-related relations between nodes, which can express interaction patterns that may result from the way the overlay is solicited by the application.

Very few recent research efforts take into account these relations. Semantic overlay networks [11], [12] exploit the semantic relations between peers (based on the set of files they share). They propose solutions allowing to improve the efficiency of the search mechanisms for large-scale file-sharing applications, by creating shortcuts between peers which are semantically close. Efficient search is also the goal addressed by the *path-caching* technique, which consists in keeping data references along a given search path, in order to improve the efficiency of subsequent search operations.

However, for the group-based applications we target in this paper, such as distributed shared whiteboard (or gaming) platforms, or replicated data-sharing services, search efficiency is not the only property to optimize. In such applications, the members of a group share some data (e.g., a whiteboard, a replicated piece of data, a game state, etc.), which they all potentially read and write. When a peer writes this shared data, it is important for the updates to be efficiently propagated to the other members of the group. Consequently, the peers belonging to a same group have to be close to each other in the overlay (i.e., a few hops away), to enable the application to efficiently maintain the consistency of the members' views of the shared state.

The issue of update propagation in large-scale systems has been studied in [13]. This system proposes an efficient multicast scheme based on multicast trees built on top of the Pastry overlay. The problem we address in this paper is different, since any member of the group can be the source of multicast in our target applications. The approach we propose is also different, since it does not construct a membership mechanism based on an already *existing overlay*. Our goal is to build an *emergent and adaptive overlay* based on patterns derived from the application usage. To achieve this goal, our work is based on an unstructured overlay. This provides the ability to dynamically change links to adapt the overlay to the application needs without breaking the overlay structure.

The closest work related to ours is [14], which addresses the problem of building an adaptive overlay based on different criteria: topology, semantic proximity, bandwidth, etc. The problem is addressed in a generic way: the target scale and the target applications are not specified. The issue we address is more specific: it regards applications that need efficient updates within groups of nodes. Consequently, multiple criteria have to simultaneously be taken into account and controlled: application-dependent node relations, but also physical locality, as well as the connectivity of the resulting graph (expressed through the degree of clustering).

Finally, [15] is a work that has been started concurrently. The goal of [15] is offering an efficient overlay dedicated to publish/subscribe applications providing the ability to express range-based subscriptions. To achieve this goal, it focuses on clustering nodes with similar subscriptions.

III. Scenario

To motivate our work, we consider a large scale distributed gaming platform [16] application. This application may involve tens of thousands of nodes spread around the Internet. For efficiency reasons, the number of neighbors that a peer must know has to be bounded, since the related information requires monitoring and state updating. Therefore, each node only has a partial view of the system. However, this should not have a negative impact on the application's desired properties, such as *connectivity*, *message propagation efficiency* and *volatility resilience*, which are important for collaborative applications.

Connectivity: An overlay is said to be *connected* if there is a path (succession of edges or links) between every pair of nodes. This property is very important in an overlay as it provides the guarantee for a node to be able to communicate with all the other ones in the overlay.

Some particular node may have to lookup for a specific game instance in the platform. This may involve only a small subset of nodes (few tens). The neighbors contained in this particular node's neighbor list may not be involved in this particular game. The platform has to be connected to make it possible for a node to somehow reach all the other nodes. The lookup strategy of a node is application dependent: it could be done by visiting a website, or querying a custom search engine, or by flooding a search query on the overlay.

For efficiency purposes, the diameter of the overlay should be as small as possible. Note that in the gaming application, it is enough for a new player to find only one player for the wished game in order to be able to reach the other ones.

Efficient Message Propagation: While a game is running, the players store object replicas which represent the current state of the game (depending on the application, this can correspond to a shared white board, etc). Each time a player plays, his node updates the state of its local game board version (i.e., its replica). In order for the other players to be able to play, they have to be notified of the changes in the game board. Therefore, messages need to be propagated in an efficient manner within a group. To enable efficient message propagation, the overlay should minimize the number of hops between two peers belonging to the same group.

Volatility Resilience: Among several thousands of nodes spread over the Internet, it is likely that, from time to time, some nodes fail or get disconnected. At the global level (the entire platform), failures and disconnections may break down the whole graph's connectivity. At the level of a given game such events should not stop the game, which means that the remaining players have to remain connected together. Furthermore, the departure of one player may break some path in the group graph. The longest path between two nodes (the subgraph diameter) is thus likely to grow; however, our goal is to have the the update propagation mechanism stay efficient.

IV. Design

To address the issues described in the previous section, we propose the concept of a Malleable Overlay that combines elements of an unstructured overlay with application characteristics. In this section, we describe the design of MOve, a system which illustrates the proposed concept.

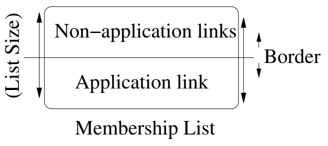


Fig. 1. The neighbor list on each node.

The first purpose of an overlay is to connect nodes together. Therefore, the first property to fulfill is the *connectivity* of the constructed graph. On the other hand, Section III highlighted the importance of providing the ability to perform efficient updates among groups of nodes within the overlay. This may be favored by introducing some *clustering*. Both *connectivity* and *clustering* have to be preserved while taking into account the dynamic nature of the environment, and adhering to the bounds on number of neighbors.

A. Random Graph Benefits

Graph theory shows that *random graphs* have good properties in terms of *connectivity* and *degree distribution*. For instance, in a random graph, if each node has at least $log(N)$ uniformly random neighbors (where N is the total number of nodes) the random graph will be connected with high probability [17] [18]. Estimating the size (i.e., N) of a large scale dynamic distributed system has also been addressed in previous studies [19]. In our case, the scale is not infinite (we target a few tens of thousands of nodes), therefore safe bounds can be assumed instead. For instance, 50 links per node will provide a large safety margin to theoretically connect 5×10^{21} nodes. On the other hand, random graphs also have the benefit of leading to a good degree distribution. An overlay based on a random graph may take advantage of this for *load distribution*. For these reasons, MOve's algorithms try to keep part of the overlay close to a random graph.

In our design, nodes maintain a neighbor list, containing links to the node's neighbors. For each node, an upper bound (l) is set on the size of the neighbor list. This bound is first set according to an initial approximation of the network size (while observing the condition $l > log(N)$). Then, during the execution, this value can be increased when necessary, if allowed by the available resources (see Section IV-B).

The neighbor list is composed of two kinds of links: *non-application links* and *application links*. Figure 1 illustrates a node's neighbor list.

B. Non-Application Links and Application Links

Non-application links are responsible for maintaining a global overlay, close to a random graph, with a low degree of clustering. If the application is in a state that does not need clustering (e.g., at initialization), the neighbor list will contain only non-application links. Recall that nodes do not need full knowledge about the network, and the number of non-application links may vary from node to node.

Application links are used to cluster together nodes that belong to a group i. Each member of group i creates k_i *application links* to other randomly-chosen members of the same group. This clustering will help fast propagation of state updates among the members of the group. It will also favor an efficient propagation of application-level multicast messages. Parameter k_i is determined by the application, and it must be at least $\lceil ln(|R_i|) \rceil$, where R_i is the number of members of group i. Essentially, the goal is to create a strongly connected graph for group i (i.e., there is a path that connects every pair of nodes).

Link Replacement Policy: When an application link needs to be created, it will be added to the neighbor list following one of four different ways. Assume that at node A we want to create an application link for group i, that points to node B. If the size of the neighbor list at A is smaller than l, and if there is no non-application link pointing to B, then (1) a new link will be added to the list. (2) If the size of the neighbor list has already reached l, but the node has enough resources available to maintain a larger neighbor list, then the node increases l to accommodate a new link. (3) If the node decides not to grow the list, then a non-application link will be dropped, and the application link will be added. (4) Finally, if there is a non-application link already pointing to B, then it will become an application link.

Link Sharing: The replacement policy helps keep the maintenance cost constant at the node. On the other hand, an application link can be *shared*. For instance, assume node A belongs to groups i and j. If node B joins groups i and j, it creates a single application link to A, knowing that this link is *shared*. A sharing count is maintained for such links.

C. Addressing the Connectivity Issue

The tradeoff between the good properties of random graphs and those enabled by favoring clustering between related nodes can be tuned by setting some bounds. The first bound is the size l of the neighbor list, which is managed as explained above. The second bound is k_i, which limits the number of links that are involved in the group i. If $(l - \sum_i k_i)$ is large enough (a few tens for the scale we are targeting), i.e., if the neighbor list contains enough non-application links, the good properties of random graphs are approximated in spite of the little clustering induced by taking the topology into account. On the other hand, it is important to notice, that $\sum_i k_i$ is usually greater than the number of *application links*. This is explained by the fact that some *application links* may be shared by multiple groups when group intersections are not empty.

A node joins the overlay by contacting any current overlay member. If the peer that receives the join request has space available in its neighbor list, it will reply with its current neighbor list, and will add a link to the joining node to its non-application links. If the neighbor list is full, the join request will be forwarded to a randomly chosen node. The forwarding of a join request is associated with a time to live (TTL). If all nodes that receive the forwarded request have full neighbor lists, the TTL will reach 0, and the last node to receive the forward will forcibly add a link to the new node to its non-application links. It will then reply with its current neighbor list (e.g., node C). We do this to ensure that the in-degree of a node is always above 0. The new node will use the received neighbor list to create its own list.

The *failure detection protocol* is based on the SWIM [9] protocol. Once during each protocol period, of length T (typically 60s)[1], each node sends a ping message to one of its neighbors. The target node is selected by sequentially traversing an array that represents the random permutation of the neighbor list. Once the array is completely traversed, a new permutation is computed. The node expects a reply to the ping message within a timeout of $t < T$. If the reply is not received on time, an indirect ping is sent to y nodes. These nodes will then send a ping to the intended target node, and, if they receive a reply, the reply will be sent back to the node that originated the ping. The intention of the indirect ping is to sidestep transient network problems. If no reply is received before the next protocol period, the ping target will be *suspected* of having failed. At the beginning of each protocol period, any node that has been suspect for one protocol period will be dropped from the neighbor list, i.e., declared dead.

To maintain randomness of the overlay, we are interested in keeping its *clustering coefficient* (CC) low. CC is defined as follows: given a random node A and two of its (random) neighbors B and C, CC is the probability that B and C are also neighbors of each other. CC of any graph thus lies in between 0 and 1. The lower the CC, the more random is the graph, while a higher CC implies more clustering. In order to keep CC low in MOve, each node periodically churns its membership list. Each node checks, once every U protocol periods (typical value =1), whether its non-application membership list has been modified. If not, it attempts to replace up to 50% of its membership list by sending a join message to a random neighbor and using the node identifiers in the reply message. The smaller the value of U, the more aggressive the replacement, but the higher the bandwidth cost.

D. Random Walk for Application Links

To cope with the dynamic nature of the infrastructure and avoid pathological topologies that may be induced by failures, it is important to periodically refresh the links. This is also useful in order to guarantee a small path between any two nodes in a given group. To this effect, we rely on another result from random graph theory [17]: adding $O(n)$ links to a graph with n vertices will reduce the diameter to $O(log(n))$ [2]. This result only applies to undirected graphs. Therefore we add the restriction that all application links are *bidirectional*. When an application link is created from A to B, B will also create a link to A. If node B deletes the link, so will node A. When an application link is shared, it will be maintained until the sharing count reaches 0. When an application link stops being used as such, it is changed to a non-application link. This simulates an undirected graph inside the application group. Note that this non-application link is then subject to being replaced later (see Section IV-C).

The graph is periodically refreshed, by having every node in an application group execute the following steps:

[1]Protocol periods are asynchronous at different process, although it is assumed that they have zero clock drift and hence the same T.

[2]Although this result was found for Erdös-Rényi random-graphs, and our application links are not trying to achieve a strict Erdös-Rényi random-graph, the overlay is random enough for the result to hold, as our experiments show.

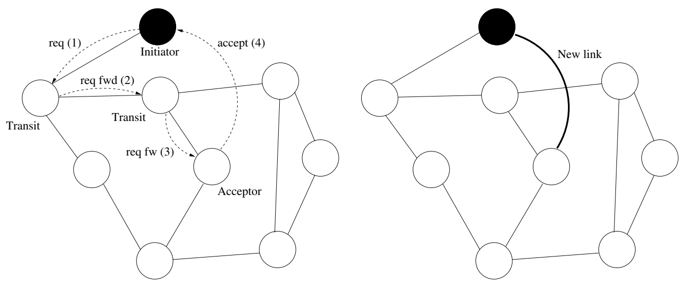


Fig. 2. The random-walk mechanism.

(1) Launch a random walk to get a new neighbor in the group. The random walk hops at most TTL times, using application links that belong to the group.

(2) Drop an old link when the new link is created.

Although it is assumed that the timeout to launch the random walk is an application parameter, note that nodes that belong to a group are not synchronized in initiating the random walks. Also note that the bidirectionality of the links is always enforced. Figure 2 illustrates the protocol.

E. Analysis

When the underlying overlay is *not malleable* to the application above, there is no link sharing. With our MOve approach, we show below that the number of links shared, and thus saved, is significant. This results in a reduction of the overhead induced by link maintenance. Concretely, when MOve does not limit the size of neighbor lists, our analysis shows that the link savings are positive, proportional to the number of non-application links, and under equi-sized groups, also proportional to the number of application links. Our experiments later in the paper show that the benefits are retained even with limited neighbor lists.

Formally, in an overlay of size N, at a node p, let $Nbrs_N(p)$ represent the set of non-application links at p. Assume that p belongs to k groups R_i ($i = 1$ to k). Let $Nbrs_{|R_i|}(p)$ represent the set of neighbors that p has in group R_i. Now, let $f_N(p) = |Nbrs_N(p)|$ and for each $i = 1$ to k, let $f_{R_i}(p) = |Nbrs_{|R_i|}(p)|$. Note that in our implementation, $f_x(p) = O(log(|x|))$, however our results are more general.

Theorem: Assume that: (1) each application group consists of members selected uniformly at random, (2) at node p, each non-application link for a group is selected uniformly at random among the group members and (3) at node p, each application link for a group is selected uniformly at random from among the group members. [3] Then, for the variant of MOve that does not limit neighbor list sizes: (a) the expected number of links saved by MOve is positive, and (b) it grows linearly as the number of non-application links is increased, and (c) it is proportional to the total number of application links if all groups at node p are equi-sized.

Proof: Without the MOve approach, the total expected number of neighbors maintained at node p is given by

$$Nbrs_{Worst-Case}(p) = f_N(p) + \Sigma_{i=1}^{k} f_{R_i}(p) \tag{1}$$

[3]The uniformly at random assumption (3) is reasonable since our neighbor list maintenance protocols achieve such random neighbor lists.

Now, with the MOve variant we are analyzing, the total expected number of neighbors for a node can be formally represented as union of $k+1$ sets as:
$Nbrs_{MOve}(p) = |Nbrs_N(p) \cup Nbrs_{R_1}(p) \cup Nbrs_{R_2}(p) \cup \ldots \cup Nbrs_{R_k}(p)|$.
This can be written as:

$$\begin{aligned} = \quad & |Nbrs_N(p)| + \Sigma_{i=1}^{k} |Nbrs_{R_i}(p)| \\ & -[(\Sigma_{i=1}^{k} |Nbrs_N(p) \cap Nbrs_{R_i}(p)|) \\ & +(\Sigma_{i \neq j, 1 \leq i,j, \leq k} |Nbrs_{R_i}(p) \cap Nbrs_{R_j}(p)|)] \\ & +[(\Sigma_{i \neq j, 1 \leq i,j, \leq k} |Nbrs_N(p) \cap Nbrs_{R_i}(p) \cap Nbrs_{R_j}(p)|) \\ & +(\Sigma_{i \neq j \neq l \neq i; 1 \leq i,j,l \leq k} |Nbrs_{R_i}(p) \cap Nbrs_{R_j}(p) \cap Nbrs_{R_l}(p)|)] \\ & - \ldots \pm |Nbrs_N(p) \cap_{i=1}^{k} Nbrs_{R_i}(p)| \end{aligned} \tag{2}$$

In order to simplify this, consider an individual term of the type $|Nbrs_{A_1} \cap \ldots \cap Nbrs_{A_m}|$, where each A_j is either a unique R_i or N. Now consider an arbitrary neighbor q of p that is in group A_1. Consider the event E that for a given $j (\neq 1)$, the same neighbor q (1) also belongs to group A_j and (2) is also a neighbor of p in group A_j (i.e., appears in $Nbrs_{A_j}(p)$).

Due to assumptions (2) and (3) in the above theorem, we have that the probability of the above event E is simply:
$Pr[E] = \frac{|A_j|}{N} \cdot \frac{f(A_j)}{|A_j|} = \frac{f(A_j)}{N}$

Thus, the individual term of the type $|Nbrs_{A_1} \cap \ldots \cap Nbrs_{A_m}|$ in fact has a value of:

$$|Nbrs_{A_1} \cap \ldots \cap Nbrs_{A_m}| = f_{A_1}(p).\Pi_{j=2}^{m}(\frac{f_{A_j}(p)}{N}) = \frac{\Pi_{j=1}^{m} f_{A_j}(p)}{N^{m-1}} \tag{3}$$

Substituting equation (3) into equation (2) and using equation (1) above, we get:

$$\begin{aligned} Nbrs_{MOve}(p) \quad = \quad & Nbrs_{Worst-Case}(p) \\ & -\frac{1}{N}.[(\Sigma_{i=1}^{k}(f_N(p).f_{R_i}(p)) \\ & +(\Sigma_{i \neq j, 1 \leq i,j, \leq k}(f_{R_i}(p).f_{R_j}(p))] \\ & +\frac{1}{N^2}.[(\Sigma_{i \neq j, 1 \leq i,j, \leq k}(f_N(p).f_{R_i}(p).f_{R_j}(p))) \\ & \quad + (\Sigma_{i \neq j \neq l \neq i; 1 \leq i,j,l \leq k}(f_{R_i}(p).f_{R_j}.f_{R_l}(p)))] \\ & - \ldots \pm [f_N(p).\Pi_{i=1}^{k}(f_{R_i}(p))] \end{aligned}$$

By exchanging the $Nbrs$ terms, and taking $f_N(p)$ common on the other side, we simplify to calculate the number of links saved by using MOve as:
$Nbrs_{Worst-Case}(p) - Nbrs_{MOve}(p)$, which is:

$$\begin{aligned} = \quad & f_N(p).[\frac{1}{N}.\Sigma_{i=1}^{k} f_{R_i}(p) - \frac{1}{N^2}.\Sigma_{i \neq j, 1 \leq i,j,k}(f_{R_i}(p).f_{R_j}(p)) + \ldots \\ & \quad \pm \frac{1}{N^{k-1}}.\Pi_{i=1}^{k}(f_{R_i}(p))] \\ & +[\frac{1}{N^2}.\Sigma_{i \neq j, 1 \leq i,j,k}(f_{R_i}(p).f_{R_j}(p)) - \ldots \mp \frac{1}{N^{k-1}}.\Pi_{i=1}^{k}(f_{R_i}(p))] \\ = \quad & f_N(p).[1 - \Pi_{i=1}^{k}(1 - \frac{f_{R_i}(p)}{N})] + [\Pi_{i=1}^{k}(1 - \frac{f_{R_i}(p)}{N}) - 1 \\ & +\Sigma_{i=1}^{k}(\frac{f_{R_i}(p)}{N})] \end{aligned} \tag{4}$$

The above result consists of two terms (each within square braces). The second of these two terms can be shown to be ≥ 0 (by using telescoping), and the first term is clearly positive. Finally, the first term is linear in $f_N(p)$, as desired. This proves (a) and (b). To prove (c) for equi-sized groups at node p, substitute $f_{R_i}(p) = f_R(p)$ for all i in equation (4) above. Then, we get that $(Nbrs_{Worst-Case}(p) - Nbrs_{MOve}(p))$ is:

$$\begin{aligned} = \quad & f_N(p).(1 - (1 - \frac{f_R(p)}{N})^k) + (1 - \frac{f_R(p)}{N})^k - 1 + \frac{k.f_R(p)}{N} \\ \simeq \quad & f_N(p).\frac{k.f_R(p)}{N} - \frac{k.f_R(p)}{N} + \frac{k.f_R(p)}{N} \\ = \quad & f_N(p).\frac{k.f_R(p)}{N} \end{aligned}$$

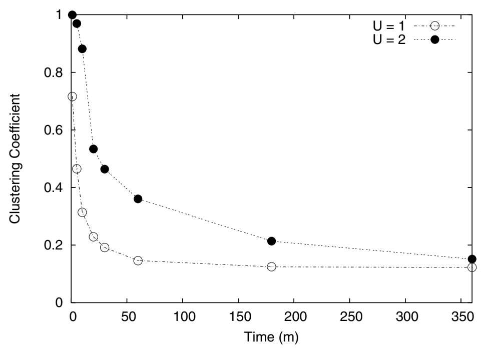


Fig. 3. Clustering coefficient vs time.

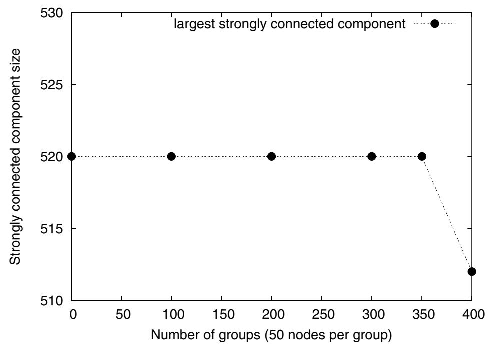


Fig. 4. The largest connected component is the overlay itself, until it reaches an overutilization with 400 groups composed of 50 nodes each.

This proves (c). $\ll$

V. Experimental Evaluation

Our algorithms are implemented in Java, and evaluated in a discrete event simulation environment. The GT-ITM [20] random topology generator, following the transit-stub model, is used to provide an underlying internetwork to our simulations. Ten transit nodes are used and each stub node joins the overlay. The end-to-end latency of a message corresponds to the shortest path between the sender and receiver nodes. The non-application link protocol period is $T = 60s$.

A. Non-Application Link Clustering

For this experiment we used a topology with 520 nodes, a protocol period for the failure detection mechanism of 1 minute. Parameter U, which determines the number of protocol periods that a node will allow an unchanged list before randomly refreshing it, is tested with values 1 and 2. Each node stores a strict maximum of 50 links. Figure 3 shows how the clustering coefficient changes with time. After only 50 minutes, the links among the nodes show a very low degree of clustering for $U = 1$. The same happens with $U = 2$ after one hour.

B. Connectivity

As the number of groups increases and becomes large, there is a possibility of overlay partitioning. In this experiment, we try to break the connectivity of the overlay by taking it to an extreme scenario. The basic parameters are the same as in the previous experiment, and again we use 520 nodes. Figure 4 measures the size of the largest strongly connected component in the overlay. This size is equal to the total number of nodes when the overlay is not partitioned. We vary the number of groups from 0 to 400 (each group is composed of 50 nodes, the k parameter is set to $\lceil ln(50) \rceil$). As the plot shows, the overlay maintains strong connectivity until the number of groups exceeds 350. In this case, we notice a small decrease in size of the largest component, which is due to the overlay partition. In a real system with such a demand for groups, instead of the strict maximum of 50 links used in this experiment, the bound on the total number of links can be increased on a node with enough resources when the number of non-applications links reaches a threshold. Note that 50 links can take less than 2KB of memory.

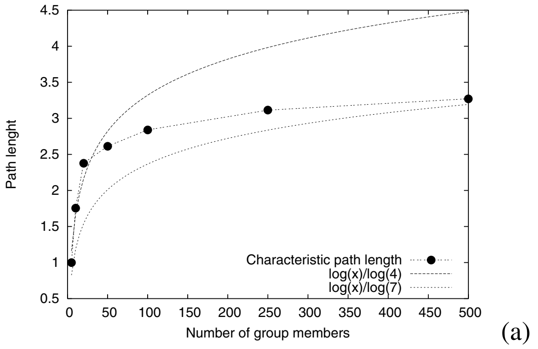
C. Application Link Clustering

For the first part of this experiment, we use a 1000 node network and an application running for 2 hours with one group. In order to evaluate the quality of the graph constructed by the application links in terms of distance (in hops) between group members, we measure the characteristic path length [21] of this group. The characteristic path length is the average of the shortest path over all node pairs. The experiment is run several times varying the group size from 5 to 500 members. The k parameter (i.e., the number of application links for this group on each node) is set to $\lceil ln(groupsize) \rceil$. Figure 5(a) shows that the characteristic path length grows slowly with the group size. Even for 500 nodes it is only 3.27. This shows that the creation of one application link at each node of the group, using random walks, achieves its objective of providing a small number of expected hops between any pair of nodes of the group. Note also that characteristic path length follows closely the logarithm with base k. Figure 5(b) shows the same experiment, without network topology, using five thousand nodes and varying the group size up to two thousand nodes.

The second part of the experiment evaluates a 2000 node network for three hours. Instead of having just one group, we have 50 groups with varying size. This way we can determine if there is any negative protocol interaction among concurrent groups on a single node. All nodes belong to at least one group. No underlying transit-stub topology was used. In Figure 6(a) we use the number of nodes from 50 PlanetLab slices[4] to create the groups. The mean group size is 74. Figure 6(b) shows the result from 50 groups, with exponentially distributed sizes, using an expected group size of 74. Figure 6(c) does the same but with uniformly distributed groups sizes. It is clear from these plots that the characteristic path length is $log_k(groupsize)$.

One small caveat found during the experiments is that in rare occasions, small groups (e.g., with as few as seven nodes), may become partitioned. To avoid such partitions, the length of the random walk should be decreased, or the frequency of the random walks should be decreased.

[4]Live data taken from http://summer.cs.princeton.edu/status/, 2006-08-21.



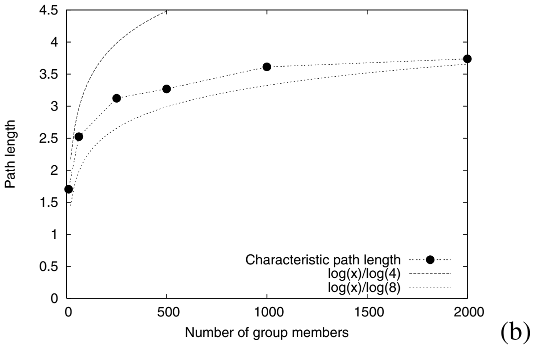
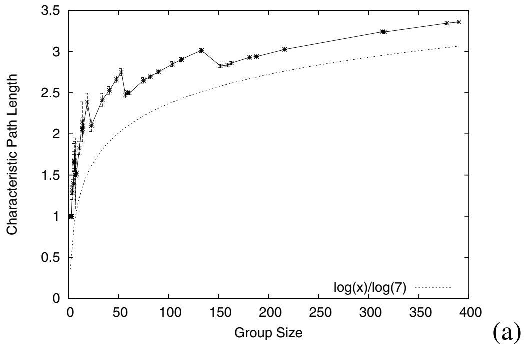
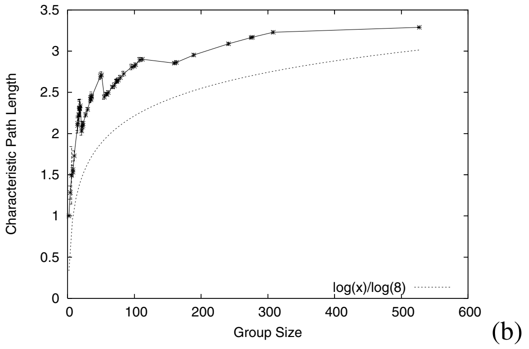


Fig. 5. The characteristic path length of a group graph is $log(groupsize)$ as expected. Plot (a) plot shows a one thousand node network, and plot (b) a five thousand node network without underlying topology.





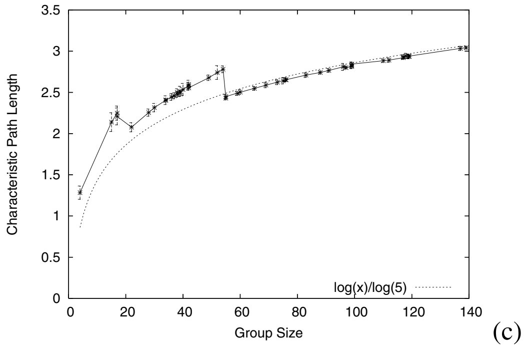


Fig. 6. Three 2000 node networks, with 50 groups. Groups sizes are taken from (a) PlanetLab, from (b) an exponential distribution, and (c) a uniform distribution, respectively. The expected group size is 74. The protocol can properly manage more than one group per node, as the Characteristic Path Length remains $log_k(groupsize)$, where k is the number of application links for the group.

D. Benefits of Link Sharing

When the platform contains many groups, the probability of non-empty group membership intersections grows. In this case, application links at each node can be shared by multiple groups, e.g., node A belongs to groups i and j, and its peer, node B, belongs to groups i and j, allowing them to use only one application link between them for communications related to i and j. However, different distributions of nodes across groups may give different link sharing benefits. Figure 7(a) shows the results of simulations upon system of 520 nodes with 60 groups of 100 nodes each. Nodes are uniformly distributed across available groups. Parameter k is set to 5. For readability, only 100 nodes are shown in the figure. The figure shows that the real number of existing application links per node is much lower than the worst case (which is k times the number of groups to which a node belongs). This is evident since the worst-case envelope is well above the real-case envelope in the figure. This result is due to link-sharing across group intersections, which allows the overlay to use fewer application links when it is solicited by the application. Figure 7(b) repeats the experiment with a different distribution: the nodes are distributed among the groups following a normal distribution with mean 260 and a standard deviation of 104. This case shows that the number of application links at each node grows at a lower rate than the worst case, due to effective link sharing.

E. Twisting the Overlay

We have also analyzed how the overlay reacts to the application needs (which may differ in the number and size of groups to be created). Simulations were run on a 520-node network, with a varying number of groups having a fixed group size set to 100 (and 5 as k parameter). The results (Figure 8) show that the total number of links is almost constant, while the border between application links and non-application links moves. The creation of groups leads to an increase of the number of application links, which progressively replace the non-application links.

On a topology-less 2000 node network (i.e., without underlying network transits or stubs), using 60 groups, with exponentially distributed group sizes and uniformly distributed group sizes, and an expected group size of 100 we can see the same behavior as before. Notice in Figure 9 that there is a small difference in the behavior when group sizes are exponentially distributed –the line plotting non-data links with exponentially distributed group size is not as straight. This is explained by the fact that in this experiment some nodes will tend to be members of more groups than other nodes. Figure 8's fixed group size and Figure 9's uniformly

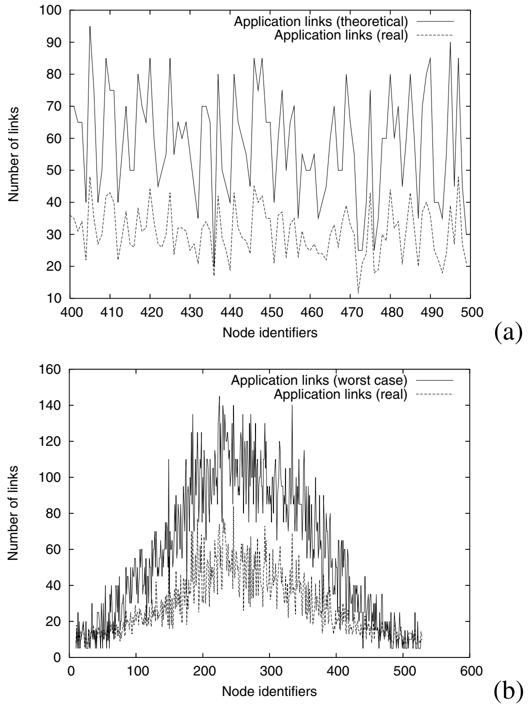


Fig. 7. Link sharing among groups. Plot (a) shows how link sharing is below the worst case when nodes are uniformly distributed across groups. Plot (b) shows the case when nodes are distributed across groups following a normal distribution. In both plots the lower line shows the real number of application links.

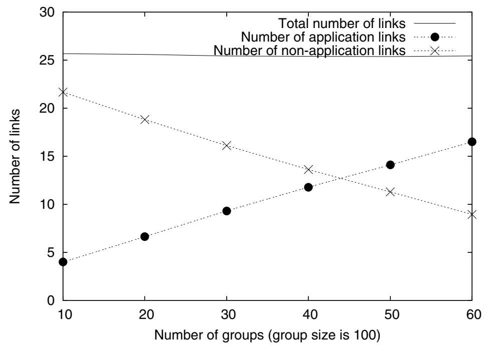


Fig. 8. Controlling the clustering.

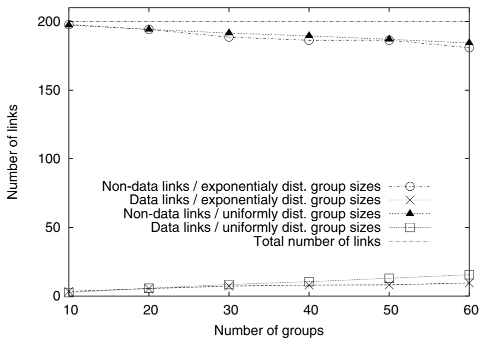


Fig. 9. Controlling the clustering with variable group sizes on a 2000 node network. Group sizes are taken from an exponential and a uniform distribution. The expected group size is 100.

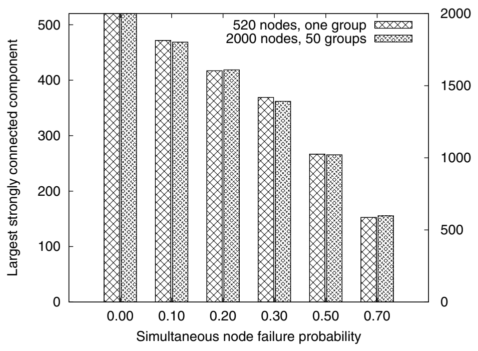


Fig. 10. This plot shows simultaneous failures on a 520 node network with a single group (left bar) and a 2000 node network with 50 groups and exponentially distributed group sizes (right bar).

distributed groups sizes give more uniform group membership across overlay nodes.

F. Resilience to Node Failure

In this experiment, after 2 simulated hours, each node was subjected to a crash with some probability (all crashes occur simultaneously). The probabilities used were 0.10, 0.20, 0.30, 0.50 and 0.70. We measured the size of the largest strongly connected component immediately after killing the nodes. The points on Figure 10 are each the average over 5 simulations. Below a crash probability of 0.70, the largest strongly connected component is always the size of the remaining overlay (i.e., the overlay remains strongly connected). With 0.70 death probability, we experienced a small degree of partitioning: 2 or 3 nodes were disconnected on some of the runs.

The left bar represents a 520 transit-stub network with one group with 300 nodes. The right bar represents a 2000 node network with 50 groups, where group sizes are exponentially distributed. The purpose was to measure how the clustering of a large number of groups would affect the resilience to simultaneous node crashes. As the plot shows, non-application links help the system to be resilient to simultaneous node failures, even if the application forces MOve into heavy clustering.

G. Heterogeneous Network

In order to consider heterogeneous group sizes, we use traces from PlanetLab. Specifically, the PlanetLab cluster is organized into a set of “slices” (each slice is basically a group) that overlap with each other. Feeding our experiment with the number of nodes from 50 real PlanetLab slices, we simulate 2000 nodes for two hours. The initial distribution of soft link capacity at the nodes follows a power law. Nodes are configured with at least 10 initial links. Our purpose is to measure what happens when there are nodes with different capacities. Figure 11 shows that only a small number of nodes are forced to increase their number of links after joining their groups. The increase results from the link demand needed to maintain membership in multiple groups. Furthermore, the increase in links is small. Recall that if a node is at the end of a random walk for an application link, it is forced to accept the link. We conclude that guaranteeing a small characteristic path length has very small impact on the resources of a node.

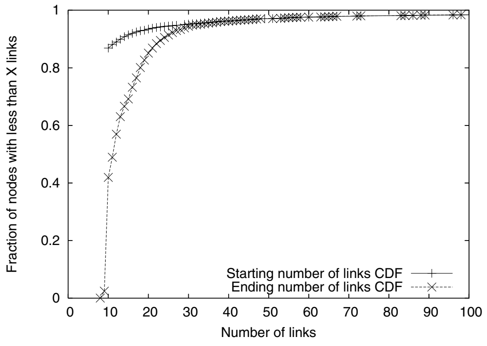


Fig. 11. In this experiment we measure the CDF for the number of links at a node starting the simulation and after two hours. Maintaining groups has a small impact on nodes with little resources.

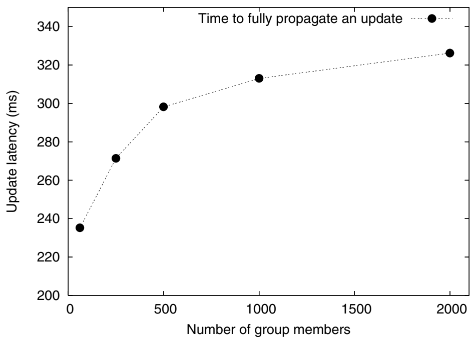


Fig. 12. Time taken to propagate an update on all replicas. Small latency is achieved due to small characteristic path length.

H. Sample Application 1 - Update Propagation

This experiment evaluates a causally-consistent [22] update application we implemented on top of MOve. Such a management facility would be useful in Enterprise and PlanetLab-style clusters. Update messages are propagated using the *application-links* to all nodes in the application group. We consider a topology-less system of 5000 nodes with inter-node latency selected randomly in the interval $[10ms, 50ms]$. Figure 12 shows the scalability of the update propagation (notice the sublinear plateauing of the curve) for up to 2000 members in the update group. The latency increase is slow because the characteristic path length of the group always stays below 4. Further details of this application can be found in our extended techreport version referenced in [1].

I. Sample Application 2 - MON over MOve

We briefly touch on the implications of using MOve underneath a management-based application called MON [23], which is already running on PlanetLab. MON allows the user of any slice in a PlanetLab-like cluster to execute monitoring commands within the slice (e.g. count, min, max, etc.), by basically creating on-demand overlays (usually trees or DAGs). We envision that a system like MOve could be run across all PlanetLab nodes, with one group per slice, and each MOve group (i.e., application links) would be used to build on-demand overlays for that relevant slice. We implemented the on-demand DAG building operation of MON on top of MOve. Figure 13 shows that using MOve underneath MON greatly

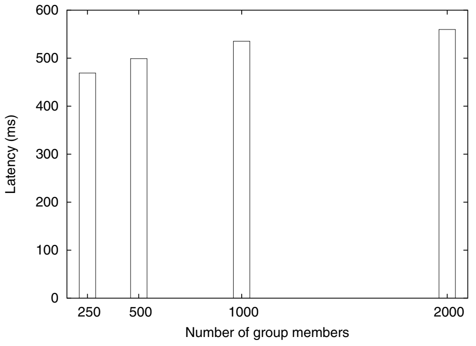


Fig. 13. On-demand DAG build operation latency.

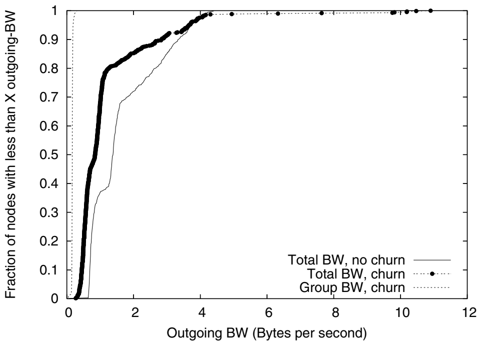


Fig. 14. Outgoing bandwidth CDF for nodes in a 620 node network, and a 300 node group. Results with churn and no churn are shown.

improves the latencies of overlay building - our latencies are around 500 ms for a group of up to 2000 nodes, while the original MON system had latencies of 1.3s for a slice of 330 nodes [23]. At the same time, the MON trees built on top of MOve reached 100% nodes in the group most of the time.

J. Message Overhead

In order to measure both the message overhead as well as churn-resilience of MOve, we inject churn traces collected from PlanetLab [24] into a MOve system of 620 nodes, with one application group of 300 nodes running within it. Parameter $U = 1$ in this experiment. Figure 14 shows that (1) bandwidth consumption under churn (max. 11 Bps) and no churn (max. 4.14 Bps) are reasonably low and have almost comparable distributions to each other, and (2) the message overhead of maintaining application links for a large group of size 300 is also low (max. 0.3 Bps).

VI. Conclusion

This paper described *MOve*, the first malleable overlay that allows group-based applications (e.g., collaborative applications, PlanetLab-slice-based management applications, etc.) to "twist" (i.e., influence) the underlying overlay, while retaining the scale and fault tolerance of the P2P approach. When the application influence disappears, MOve reverts back gracefully to its original default (i.e., random) structure. By marking and managing, at each node, *non-application links* (i.e., default) and *application links* (i.e., per group), in such a way as to re-use common neighbors across groups, MOve is able to scale to systems and groups with 1000's of nodes, while maintaining low characteristic path length, adapting to churn with low per-node bandwidth, effectively reusing neighbors where possible, and showing resilience to massive failure. Basic experiments with two sample management applications - update propagation and MON – show scalability and latency advantages of using MOve underneath management applications.

REFERENCES

[1] S. Monnet, R. Morales, G. Antoniu, and I. Gupta, "MOve: design of an application-malleable overlay," in *Proc. SRDS: Symposium on Reliable Distributed Systems*, Oct. 2006.

[2] A. I. T. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. Middleware*, Nov. 2001.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for Internet applications," in *Proc. SIGCOMM*, Aug. 2001, pp. 149–160.

[4] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: a distributed anonymous information storage and retrieval system," in *Proc. International Workshop on Design Issues in Anonymity and Unobservability*, no. 2009, July 2000, pp. 46–66.

[5] A. Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, May 2001, ch. Gnutella, pp. 94–122.

[6] "KaZaA," http://www.kazaa.com/.

[7] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing J.*, vol. 6, no. 1, 2002.

[8] G. D. Caro and M. Dorigo, "Antnet: distributed stigmergetic control for communications networks," *J. Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.

[9] A. Das, I. Gupta, and A. Motivala, "SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol," in *Proc. DSN'02*, June 2002.

[10] S. Voulgaris and M. V. S. Daniela Gavida, "Cyclon: inexpensive membership management for unstructured p2p overlays," *J. Network and Systems Management*, vol. 13, no. 2, June 2005.

[11] S. Handurukande, A.-M. Kermarrec, F. Le Fessant, and L. Massoulié, "Exploiting semantic clustering in the eDonkey p2p network," in *Proc. SIGOPS European Workshop*, Sept. 2004, pp. 109–114.

[12] S. Voulgaris and M. van Steen, "Epidemic-style management of semantic overlays for content-based searching." in *Proc. 11th Euro-Par*, ser. LNCS, Aug. 2005, pp. 1143–1152.

[13] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: the design of a large-scale event notification infrastructure," in *Proc. Networked Group Communication 2001*, pp. 30–43.

[14] M. Jelasity and O. Babaoglu, "T-man: fast gossip-based contruction of large-scale overlay topologies," University of Bologna, Mura Anteo Zamboni 7 40127 Bologna (Italy), Tech. Rep. UBLCS-2004-7, May 2004.

[15] S. Voulgaris, E. Rivière, A.-M. Kermarrec, and M. van Steen, "Sub-2-sub: self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks," in *Proc. IPTPS'06*, Feb. 2006.

[16] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *Proc. NSDI'06*, May 2006.

[17] B. Bollobas, *Random Graphs, Second Edition*. Cambridge University Press, 2001.

[18] A. J. Ganesh, A.-M. Kermarrec, and L. Massouli, "Scamp: peer-to-peer lightweight membership service for large-scale group communication," in *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication 2001*, pp. 44–55.

[19] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Decentralized schemes for size estimation in large and dynamic groups," in *Proc. IEEE NCA*, July 2005.

[20] K. L. Calvert, M. B. Doar, and E. W. Zegura, "Modeling Internet topology," *IEEE Commun. Mag.*, vol. 35, no. 6, pp. 160–163, June 1997.

[21] D. J. Watts and S. H. Strogatz, "Collective dynamics of "small-world" networks," *Nature*, no. 393, pp. 440–442, June 1998.

[22] P. Hutto and M. Ahamad, "Slow memory: weakening consistency to enhance concurrency in distributed shared memories," in *Proc. ICDCS*, 1990, pp. 302–311.

[23] J. Liang, S. Ko, I. Gupta, and K. Nahrstedt, "MON: on-demand overlays for distributed system management," in *Proc. Usenix Worlds'05*, Dec. 2005.

[24] P. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems," in *Proc. ACM SIGCOMM*, 2006.

Ramsés Morales is currently a PhD student in the Computer Science department of the University of Illinois at Urbana-Champaign. He received his MS in Computer Science from the same university in 2005 supported by a Fulbright Fellowship. Research interests include P2P systems, Distributed Protocols with Self-* Behavior, and Grid Computing.

Sébastien Monnet received a PhD degree in Computer Science in 2006 from the University of Rennes I (France). Since September 2007, he is an associate professor at University of Pierre and Marie Curie (Paris 6, France). His research interests include: distributed algorithms, fault tolerance, consistency models and protocols, P2P systems and Distributed Shared Memory (DSM).

Indranil Gupta completed his PhD in Computer Science from Cornell University in 2004. Indranil received the NSF CAREER award in 2005. He has previously worked in IBM Research and Microsoft Research. He obtained his BTech (Computer Science) from the IIT-Madras, in 1998. He is a member of ACM and IEEE.

Gabriel Antoniu is a research scientist in the PARIS research group at IRISA/INRIA Rennes, France. He received his Bachelor of Engineering degree from INSA Lyon, France in 1997; his MS degree in Computer Science from ENS Lyon in 1998; his PhD degree in Computer Science in 2001 from ENS Lyon. His research interests include large scale data management for grids, P2P systems, DSM, distributed multithreaded systems.