Locate, Cluster and Conquer: A Scalable Topology-aware Overlay Multicast

Mohamed Ali Kaafar, Thierry Turletti, Walid Dabbous

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

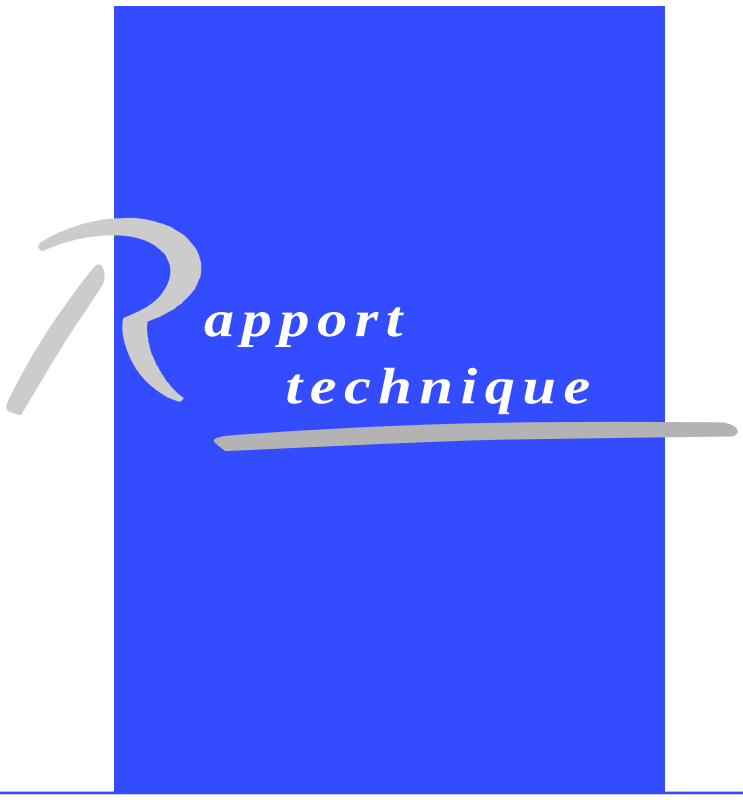
Locate, Cluster and Conquer: A Scalable Topology-aware Overlay Multicast

Mohamed Ali Kaafar — Thierry Turletti — Walid Dabbous

N° 0314 – version 2

version initiale November 2005 – version révisée December 2005

Thème COM



ISSN 0249-0803

Locate, Cluster and Conquer: A Scalable Topology-aware Overlay Multicast

Mohamed Ali Kaafar, Thierry Turletti , Walid Dabbous

Thème COM — Systèmes communicants
Projet PLANETE

Rapport technique n° 0314 – version 2*— version initiale November 2005 — version révisée December 2005 32 pages

Abstract: Recent proposals in multicast overlay networks have demonstrated the importance of exploiting underlying network topology data to construct efficient overlays. While they avoid virtual coordinates embedding and fixed landmarks measurements, these topology-aware proposals often rely on incremental and periodic refinements to improve each node's position in the delivery tree. However, these approaches are neither scalable, as they induce high communication cost due to refinement overhead, nor efficient because long convergence time is necessary to obtain a stabilized structure. In this paper, we propose a novel highly scalable locating algorithm to initially direct newcomers to the closest set of existing nodes. Each newcomer sends request to a few nodes to build its neighborhood information. On the basis of the locating process, we build a two-level topology-aware scheme, namely LCC. We evaluated the LCC scheme using real WAN deployment and by extensive simulation. WAN experimentation was carried over the PlanetLab wide area testbed using more than 200 machines, and simulations were performed using the BRITE topology generator with over 10000 nodes. We compare the scalability and efficiency of LCC with that of initially-randomly connected overlays. Results demonstrate promising performance of LCC, and show that locating-based overlays achieve 70% less link adjustments than initially randomly-connected structures, with three times faster convergence. Moreover, while being accurate, the locating process entails modest resources and incurs low overhead during new nodes arrivals.

Key-words: Distributed algorithms, P2P, Overlay multicast, Topology-aware, scalability, location, efficiency

* Details of the clustering mechanisms, and discussion of both PlanetLab and simulations results.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 —Télécopie : +33 4 92 38 77 65

Mécanismes scalables de transmission en multipoint pour réseaux superposés

Résumé : Nous nous intéressons au déploiement du multicast applicatif dans les réseaux de couverture à très large échelle, pour des applications temps réel. Nous proposons des algorithmes fortement "scalables" (passant à l'échelle) permettant de localiser et grouper les nouveaux venus dans des ensembles appropriés de nuds voisins, de manière à ce que les structures connectées aléatoirement, et reposant sur des raffinements périodiques, soient évitées. Ceci diminue de manière significative le coût en communication entre les noeuds du réseau, et assure un relai efficace des données avec un surcoût de contrôle minimal. De plus, le processus de localisation ne se base ni sur les techniques de coordonnées virtuelles, ni sur des points de mesures fixes (landmarks), le rendant fortement précis et robuste. En se basant sur cet algorithme de localisation, nous proposons un schéma de transmission multipoint applicatif hiérarchique prenant en compte la topologie du réseau sous jacent, et passant a l'échelle de milliers de membres. Nos simulations montrent que ce schéma a de meilleurs performances que d'autres structures hiérarchiques ou d autres mécanismes prenant en compte la topologie, construisant des arbres de transmission multicast plus efficaces.

Mots-clés : transmission multipoint pair à pair, réseaux superposés, Topologie, performances, passage a l'échelle.

1 Introduction

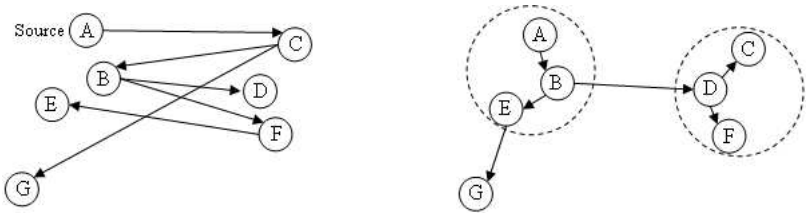
A key factor to "overlay networking" success is that an overlay service can be quickly constructed and easily deployed and upgraded, because it only requires engineering at the application level in end systems. In particular, several overlays support multicast-style data dissemination without requiring the widespread deployment of IP multicast. However, this application level multicast may suffer poor performance, scale and cost problems when the delivery tree construction process ignores the topology and link properties of the underlying network.

High speed Internet access is rapidly expanding and consumers may today go online from any location using wireless technologies, increasing the popularity of live streaming applications, such as real time video distribution. Supporting such one to many applications requires therefore highly scalable multicast protocols. In this paper, we focus on two main issues in overlay multicast: *scalability* and *efficiency*. We claim that there are barriers for the scalability of existing overlay multicast protocols. In fact these protocols rely either on global knowledge or periodical refinement and control processes, which induce additional overhead. On the other hand, users attending a video conferencing session or an event broadcast expect an acceptable quality as soon as they join the multicast session. It is then important to overcome an efficiency problem from which almost all current overlay multicast proposals suffer: the long convergence time to reach a stabilized quality state in the overlay delivery tree.

If the overlays are constructed randomly, nearby nodes in the overlay network may actually be far away from each other in the underlying network. This may waste too much network resources and degrade performance significantly. For example, in Fig. 1(a), overlay paths between nearby nodes in the underlying network such as nodes A and E and nodes C and F are very long. This is obviously inefficient. Therefore, it is important to construct topology-aware overlay networks, such as in Fig. 1(b) to achieve better performance with low overhead and low extra network traffic. In existing decentralized protocols, such as switch-trees [1], Hostcast [2], SpreadIt [3], MeshTree [4] and NICE [5], nodes maintain their relative positions to the root of the delivery tree. Periodically, each node tries to improve its on-tree position by finding a better parent, i.e. a non-descendant node that provides a lower delay to the root. In fact, these protocols initially construct randomly connected overlays and rely on incremental improvement to optimize the overlay. During overlay growth or frequent membership changes, additional overhead of link adjustments [1] is incurred for structure quality maintenance and partition repairs (see Fig. 2). These incremental improvements assume that nodes are stable during the period where refinements are processed. This is often not true, and may cause high overhead and control messages transmitted between nodes that are far away from each other.

Since our primary objective in this paper is to support live-streaming applications, and hence to minimize the delay between the source and the receivers, a multicast overlay delivery tree is typically studied to minimize the average incurred extra delay observed by

[1] Link adjustments include adding/deleting links operations



(a) Randomly connected overlay. (b) Locality-aware overlay.

Figure 1: Illustration of locality properties in overlay construction.

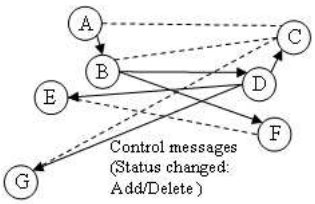


Figure 2: Intermediary state of a randomly connected overlay.

the receivers: the Average Relative Delay Penalty ($ARDP$), which is defined as the average ratio between the overlay delay (d') and the shortest path delay in the underlying network (d) from s to all other nodes: $\frac{1}{N-1}\sum_{i=1}^{N-1}\frac{d'(s,i)}{d(s,i)}$, where N is the number of nodes in the overlay.

Considering that an overlay delivery tree is "efficient" if the $ARDP$ value is less than a threshold value (say 2), one could intuitively conclude that incremental refinements-based approaches incur a long delay before the overlay delivery tree converges to an optimized structure (denoted by convergence time in the rest of this paper).

Fig. 1 illustrates an example of the overlay state during the optimization process, before the delivery tree stabilizes. Since the initial tree configuration represented in Fig. 1(a), we observe that six add/delete links operations have been processed at the intermediate state represented by Fig. 2. The refinement-based approaches then depend on the choice of a refinement period, say RT. A small RT value reduces the convergence time, as more adjustment procedures are processed within a short time, but may introduce high overhead. A large RT may reduce overhead at the expense of increased convergence time.

In this paper, we propose a novel overlay multicast tree construction scheme, called LCC : Locate, Cluster and Conquer, designed to address the aforementioned scalability and efficiency issues. The scheme consists in two phases: a locating phase and an overlay construction phase. The locating phase is based on an accurate and scalable global positioning technique. Using partial knowledge of location-information for participating nodes, the algorithm consists in locating the closest existing set of nodes (cluster) in the overlay for

a newcomer. It allows then to avoid initially randomly-connected structures using neither virtual coordinates system embedding nor fixed landmarks measurements, hence performing fast convergence to an optimized overlay multicast data delivery. Furthermore, it reduces management overhead and delivery tree changes imposed due to periodical refinement process.

Then, on the basis of this locating process, the multicast overlay construction phase consists in building and managing a topology-aware clustered hierarchical overlay, in order to optimize the average end-to-end delay. In addition, this achieves scalability by drastically reducing the bandwidth requirements for overlay maintenance, and mitigating the effect of dynamic environment as most changes are not seen beyond clustered set of nodes.

Using two complementary evaluation methods: extensive simulations and a thorough PlanetLab testing over the Internet, our experimentations show that the LCC scheme outperforms other initially randomly connected and/or flat approaches resulting in lower convergence time and link adjustment rate. At the same time, LCC achieves better performance in terms of overlay latency and overhead.

The remainder of this paper is structured as follows. Section 2 presents the (abundant) related work in this domain. In Section 3, we detail the locating process and its different components. Section 4 presents the mechanisms of clustering and delivery tree construction. Through experiments on the PlanetLab testbed and simulations, in Section 5, we evaluate the LCC performance and efficiency, and compare it to initially randomly-connected structures. Finally, Section 6 concludes the paper.

2 Related Work

There has been tremendous interest in the construction of overlays to provide application-level multicast [1-10]. Basically, the contributions can be categorized in two classes: overlay-router approach and peer-to-peer (P2P) approach.

In the overlay-router approach [6] [7], reliable servers are installed across the network to act as application-aware routers. These routers are connected to the constructed overlay and the content is transmitted from the source to a set of receivers on a multicast tree consisting of the overlay servers. This approach is designed to be scalable since the receivers can get the content not only from the source, but also from the application-aware routers, thus alleviating bandwidth demand at the source. This approach needs dedicated infrastructure deployment and costly application-aware servers.

The P2P approach assumes no extra resources such as the dedicated servers mentioned above. Several earlier works are specifically designed to handle small groups, which is sufficient in many situations, such as collaborative work or enterprize video conferences. Narada [8], MeshTree [4], and Scattercast [9] are examples of distributed "mesh-first" algorithms where nodes arrange themselves into well-connected mesh on top of which a routing protocol is run to derive a low-cost delivery tree. These protocols rely on incremental improvements over time by adding and removing mesh links based on an utility function. Although these protocols offer robustness properties (thanks to the mesh structure), they do not scale to

large population in the overlay, due to excessive overhead resulting from the improvement process.

Other "tree-first" protocols like ZigZag [10] and NICE [5], rely on multi-layer clusters to construct a hierarchically-connected control topology which connects the group members. These protocols have been designed to support wide-area size multicast group with low overhead. However, they do not consider individual node fan-out capability. Rather, they bound the overlay fan-out using a (global) cluster-size parameter. In particular, since both protocols only considers latency for cluster leader selection, they may experience problems if the cluster leader has insufficient fan-out. Moreover, clustering techniques in these approaches rely on logical hierarchy clustering and not underlying network properties.

In [11], Wang et al. address NATs and firewalls issues in overlay multicast by clustering nodes into clusters leaded by a non "guarded" host (not behind a NAT). Their two-level hierarchy is similar to ours. However, a new member join, only obtains a list of already existing clusters and joins the closest. Further cluster improvements have to be processed. The overlay member has then to periodically search for a closer cluster and switch to it. This process may also induce several clusters' splits and merges.

Landmark clustering is also a general concept to construct topology-aware overlays. Ratnasamy et al. [12] use such an approach to build a topology-aware CAN [13] overlay network. Prior to joining the overlay network, a joining node has to measure its distance to each landmark. The node then orders the landmarks according to its distance measurements. The main intuition is that nodes with the same landmark ordering, i.e. nodes that have similar distances to all landmark nodes, are also quite likely to be close to each other topologically. An immediate issue with such a landmark-based approach is that it can be rather coarse-grained depending on the number of landmarks used and their distribution. Furthermore, a fixed set of landmarks (that all participating nodes have to know about) renders this approach unsuitable for dynamic networks. The most significant downside of this approach, however, is that it can lead to an extremely uneven overlay nodes distribution. This means that a small set of nodes could be responsible for a very large part of the overlay space, essentially turning them into hot spots. Other protocols are also based on structured overlays to exploit the "overlay topology" information in order to improve multicast efficiency. Based on distributed-hash table (DHT), they exploit either proximity routing as in Chord [14] or proximity neighbor selection as in Tapestry [15] and Pastry [16]. Typically, they randomly assign overlay identities (IDs) regardless of the underlying topology. Then, they try to make the physical proximity fit into the overlay routing state through the join process and table maintenance. These approaches although addressing scalability requirements, are neither accurate (introducing significant errors in topology positioning), nor complete (requiring additional mechanisms in tree optimization that introduce substantial complexity and bandwidth overhead). Other works such as MITHOS [17] and MULTI+ [18] propose a topologically-aware overlay construction, that typically rely on (non-accurate) virtual coordinates embedding or hierarchical grouping of nodes according to network IP prefixes.

3 The Locating Process

By adopting a network positioning strategy similar to the Meridian approach [19], we propose a new locating algorithm to direct newcomers to the "nearest" cluster. In this way, LCC does not construct an initially randomly-connected overlay structure, and avoids the use of virtual coordinates (embedding) and landmarks-based techniques. Basically, a newcomer initiates the locating process by sending a request to a boot node. The latter asks a set of selected representative nodes, considered to be close to the newcomer to probe it. The boot node informs the newcomer of the possible closest nodes. By iteratively contacting the closest nodes, the newcomer is able to locate itself either by joining an already existing cluster, or creating it own cluster. By using a selection criterion to limit the number of nodes probing the newcomer, requested nodes are able to minimize the measurement overhead.

3.1 Bootstrap and locating request

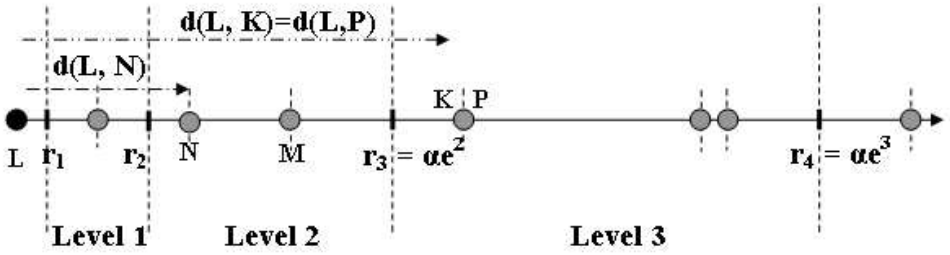


Figure 3: The locating system of node L.

Each LCC node keeps track of a fixed number of other nodes in the overlay, and organizes them into its locating system, which is a set of non overlapping levels. These levels are represented by intervals $[r_i, r_{i+1}]$, where r_i are exponentially increasing distances from the considered node taken as the origin. Each level is then bounded by a maximum distance, where the $r_i = \alpha e^{i-1}$ for $i \geq 1$ and $r_0 = 0$ (see Fig. 3). Nodes then measure the distances to the set of nodes they are aware of, and affect each node a position in the correspondent level in the locating system.

It is assumed that there is a global well-known host called Rendezvous Point (RP) in the overlay network. RP usually stands for a machine or a set of machines, used to bootstrap new members in the overlay. Initially, a newcomer, say node A, has to contact the RP, to obtain the identity of a randomly selected boot node, B. A then measures the distance (delay) from itself to B, $d(A, B)$ and affects B a level in its locating system, say level i. If $d(A, B) \leq R_{max}$ (defining the clustering criterion described in section 4-1), the locating process terminates, and A sends a request to join B's cluster. Otherwise, A

contacts B to inform it of its level, and to obtain the identity of known clusters leaders in A's neighborhood[2]. Node B, receiving A's request for locating closest clusters, simultaneously queries all cluster representative nodes, in the same level than A, as well as all representative nodes in the adjacent levels $i-1$ and $i+1$. However, if node B does not use any criterion to limit the number of queried nodes, it could ask distant nodes whose distance measurements to node A are useless and would introduce additional overhead. In fact, although being in the same or adjacent levels, nodes could be far from each other. Such nodes should not be taken into consideration, and should not probe A as they can not be closer to A than B.

3.2 The selection criterion

To reduce the number of useless probes, node B uses a *selection criterion*, consisting in asking only one representative node in a defined area to measure its distance to the newcomer A. In this way, B eliminates nodes that are close enough to the selected node from the candidates to probe A. Through different requests, each node maintains for each level i a matrix, M^i, representing learned distances of level i's nodes to each other, and to nodes in adjacent levels $i-1$ and $i+1$. Values in M^i are assigned as and when discovered through the other nodes' locating requests. If the distance is not known, it would be set to a large value in the matrix. This would result in selecting the concerned node even though it does not meet the selection criterion. The selection algorithm is described in the following:

B selects a random node, N^i_j, in level i or adjacent levels $i-1$ and $i+1$, and extracts from M^i its known distance vector, V^i_j, which is the j^{th} row in M_i. If $M^i_{jk} = d(N^i_j, N^i_k)$ is less than a threshold value, γ, then node N^i_k is represented by N^i_j. The threshold value is function of $d(A,B)$. More precisely, if the newcomer is close to the node B, the aggregation should be fine-grained and B should use a small γ value. But, if $d(A,B)$ is large, node B could use a greater γ value. In our algorithm, we choose:

$$\gamma_i = (d(A,B) - r_i)/r_{i+1} * d(A,B)$$

Selected nodes to probe the newcomer are represented by a matrix, say S^i. S^i is originally equal to M^i. At each iteration of the aggregation process run at each row of the matrix M^i, S^i is diminished by the columns of nodes that can be represented by the selected node N^i_j. The selection algorithm terminates when rows of S^i contains only distances of representative nodes.

Using this selection criterion, node B is able to reduce the number of representative nodes to measure their distances to A. These nodes have then to report the results back to B who could use an acceptance threshold β, where $0 < \beta \leq 1$ to determine the reduction in distance at each hop. The route acceptance threshold is met if one or more of the queried nodes are closer than β times B's distance to A. If no queried nodes meet the acceptance threshold, then the node with the smallest distance to A is selected. All selected nodes are then stored into a candidate list, that identifies a set of candidate cluster leaders list. The candidate list is sent to the requesting node A.

[2] We consider symmetric delays

3.3 Which cluster to join?

Node A selects cluster leaders sequentially from the candidate list. Among this list, A contacts cluster leaders whose distances are less or equal to R_{max} and initiates joining processes to their clusters respectively. If there are no such cluster leaders in the list, A re-initiates the locating process by contacting the cluster leaders sorted in increasing distances. The shortest distance and the candidate list are updated at each response from a requested cluster leader. This procedure is repeated until the clustering criterion is met. Finally, it is necessary to set a stop criterion so that the locating algorithm terminates within a given time period T. We can for example stop the algorithm after repeating the procedures C times as it is proposed in [20]. If the algorithm ends without satisfying the clustering criterion, A creates its own cluster.

4 Overlay construction

On the basis of the locating process, we build a two-level topology-aware overlay. Basically, the LCC overlay construction is divided into two phases: *Clustering* and *Delivery Tree construction and maintenance*. Fig. 4 illustrates the clustering approach and the two-level hierarchy of the LCC scheme once the two phases terminate.

During the clustering process, a node decides at which level it will join the overlay. If it creates its own cluster, it elects itself as the cluster leader and joins the overlay at the top-level topology. Cluster leaders inter connect among themselves to form an inter-cluster mesh. Otherwise, it is a cluster member and joins an intra-cluster mesh in order to derive its delivery tree within this cluster. Edge nodes are allowed to join both levels of the overlay in order to allow better inter-cluster connectivity. The second phase addresses the following questions: the connection between clusters and the clusters' members mesh and then delivery tree construction.

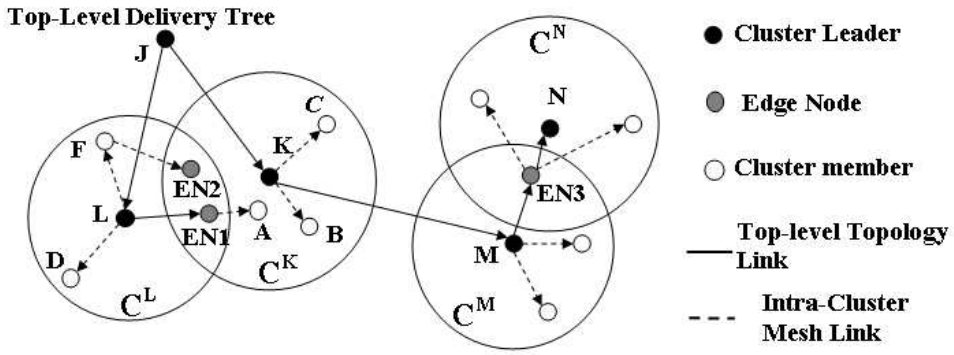


Figure 4: The two-level hierarchy of LCC.

4.1 The Clustering Process

The objective of this process is to maintain appropriate clusters, in terms of nodes proximity both inside the cluster, and between the clusters themselves. The clustering process is initiated by every node joining the overlay, once the locating process terminates. Overlay nodes are partitioned into clusters of nodes that can overlap, i.e. a set of nodes could be at the same time members of more than one cluster; these members are called *Edge nodes*. Every cluster has a unique representative node called *cluster leader*. We define the *Scope* of a node, represented by a maximum Radius, R_{max}, as the area in which other nodes are considered "nearby". A Cluster is then a set of nearby nodes, x_i that are in the Scope of a cluster leader, L, i.e. $d(L, x_i) \leq R_{max}$. By grouping together nodes that are close to a leader with a maximum threshold delay, members are expected to be close to each other, which leads to low overhead maintaining the cluster overlay topology. This also allows the multicast scheme to reduce the amount of redundant flows leaving and entering each network, making efficient usage of the bandwidth.

In Fig. 4, each circle represents a cluster, C^L, leaded by L, and containing the set of members $M(C^L)$. Solid links represent top-level topology links, including cluster leaders and Edge nodes. Dashed links are the intra-cluster links between cluster members, connecting either "simple" members among themselves, or with members and potential Edge nodes. Each cluster defines its clusters neighborhood. Typically, members maintain information about top-level nodes in their cluster and their links connected to foreign cluster nodes. In Fig. 4, nodes in $M(C^L)$ maintain:

$L \rightarrow C^J(J); C^K(EN1)$ $\qquad\qquad$ $EN1 \rightarrow C^K(A)$

Since, we focus on the construction of an overlay spanning tree rooted at the source node, s to distribute data to all the receivers of a multicast session, we need then to consider the degree constraints in the overlay multicast. Assuming that the media playback rate is B and the outgoing access link capacity of any particular node i, is b_i, the total number of streams the node can handle is $f_i = [b_i/B]$. f_i is defined as the fan-out or the out-degree bound of node i, representing the maximum number of connections that this node can establish with the outside world. Note that in this work, each node tries to estimate its connection type, relying on system and user specifications. Moreover, the protocol can use a history of maximum throughput of the most recent downloads, as an indication of its effective connection speed. These techniques are used in order to avoid each node to measure its bandwidth, which may involve high overhead [21] [22].

We also define cluster overall fan-out as the sum of available fan-out in a clustered set of nodes. If $i \in M(C^L)$ then the overall fan-out of cluster C^L, denoted by Of^L is $\sum_{i \in M(C^L)} f_i$.

In the following, we describe several procedures of the overlay hierarchy formation and maintenance. Although these mechanisms are similar to the ones in traditional unstructured overlay, we emphasize the different modifications we made to increase scalability and robustness.

4.1.1 Cluster Creation

In the initialization stage of overlay network formation, new clusters are more frequently generated since few nodes exist. A cluster formation is initiated by an overlay node which the locating process ends with no leaders found in its Scope. It creates then its proper cluster with a new cluster ID, informs the RP and contacts the closest known cluster leaders in order to bind top-level topology links. Contacted cluster leaders inform their members by flooding a "new cluster" message. Members should verify if they are in the Scope of the new created cluster, i.e. if they are potential Edge nodes.

4.1.2 Cluster Joining

A classical joining operation is initiated by a newcomer, A, joining the overlay and detecting a cluster leader in its Scope after the locating process terminates. In this case, it sends a "join request" message to the cluster leader, L, informing it of its current available out-degree, and of its potential other clusters.

Node fan-out allocation: In LCC, each top-level node (cluster leaders and potentially Edge nodes), has two types of neighbors: nodes in its own cluster and other top-level nodes. Since a cluster leader has limited available bandwidth in practice, it should carefully set its node degree to maintain a balance between connecting to other cluster centers for better performance and serving as many nodes as possible in its own cluster.

The leader verifies that its cluster overall fan-out Of^L is not reached (computing the overall cluster fan-out considers the new node as a cluster member). If not, it accepts the newcomer. Note that the overall fan-out is reached in the worst case where all nodes are Edge nodes, so that the cluster is composed of a cluster leader and Edge nodes attached to the top-level topology. So, if we consider the case where all of these nodes have a maximum fan-out of 2, this would lead to a saturated cluster. This situation is quickly recovered if we defend a node to become an Edge node to rescue one of its clusters for example, or by the joining of a non-edge node having a fan-out ≥ 1.

Node A contacts simultaneously all the cluster leaders in its Scope, and finally acknowledges them of its cluster membership state. It sends acknowledgement messages where it mentions clusters it has successfully joined. Neighboring types of clusters are changed accordingly.

Node acceptance: If node A is accepted as a cluster member in C^L, L updates the list of clusters neighbors that A potentially belongs to, and informs its cluster members of the new node connection. A is informed of the neighbors clusters. Since LCC does not specify the protocol to organize intra-cluster and inter-cluster delivery trees, any existing overlay construction protocol may be used. At the top-level topology, nodes consider their neighbors as the session already existing members, and depending on the deployed multicast delivery tree, would connect to the inter-clusters delivery tree. In the same manner, intra-cluster members contact their cluster members. In fact, after joining the new cluster, the cluster

leader randomly assigns the newcomer a cluster member to bootstrap the lower level mesh management. Meanwhile, from its cluster nodes, the newcomer can get the information for cluster maintenance.

4.1.3 Cluster's member state and information updating

Information updating is a distributed algorithm run by each node in the cluster. Specifically, each node is aware of any new member joining its proper cluster. This information is provided by the cluster leader, as described in the previous section. Moreover, each node maintains its cluster neighbors, and the distance to these clusters. These are the links established either by the cluster leader at the top-level topology or by the Edge nodes belonging to the cluster. Using "Keep Alive" messages exchanged by cluster members allows to share cluster state, and to update cluster information. We do not introduce a specific way for delivery tree maintenance, instead, the specific delivery tree construction built on both levels is used.

4.1.4 Information updating

Information about other overlay nodes and other clusters' members is obtained using a simple gossip style node discovery technique [9]. Basically, a node, i maintains a list of known nodes in the overlay, whatever they are in its cluster or in a foreign one. Periodically, i randomly picks a node from the list, say j and sends it a randomly constructed fixed-size set of other known members. When j receives this list, it updates its own known nodes list and replies using the same procedure. Using this simple informative exchange, nodes are able to maintain a minimal view of the overlay membership, which allows the locating phase to achieve better performance in terms of convergence time to the closest cluster. Next, we discuss how this knowledge influences the average locating phase hop counts, and then the convergence time of the algorithm.

4.1.5 Leaders election

A cluster leader election should define a way to maintain a balance between connecting to other cluster leaders for better overall performance, and serving as many nodes as possible in its own cluster. Hence, in LCC, the cluster leader election is based on priority vectors (PV), shared in the cluster. Each element in the PV has a priority weight that is used to sort appropriate eligible nodes. Basically, a PV is defined as:

$$PV =< f^{max}, 1/DL, T, 1/CL, Scope >$$

with f^{max}, DL, T and CL having decreasing priorities. f^{max} is the maximum fan-out value, DL is the perceived latency of the node in the intra-cluster delivery tree rooted by the current leader, CL denotes the minimum distance from the node to a known foreign cluster leader, T represents how long the node has stayed in the overlay, and finally *Scope* is a boolean indicating if the current cluster leader is in the node's Scope or not. The PV's elements are used to maintain the node's priority for leader election.

Nodes update periodically their PV. Each PV is distributed as part of "Keep Alive" messages transmitted to other cluster members. When receiving nodes' PVs, a node sorts cluster members with increasing PV order. In fact, cluster nodes construct a proactive rescue plan, where each node maintains a local cache storing shared information. This local cache consists in a sorted list of nodes that are eligible to become cluster leaders. In a dynamic network environment, a cluster leader may depart unexpectedly at any time. To indicate its existence, the leader periodically sends out "Keep Alive" messages to the whole cluster. If the leader fails, nodes will know after a period of time as they do not receive the "Keep Alive" message from the leader. At this moment, the second node in the list becomes automatically the leader and send out "Keep Alive" messages. If the second also fails, the third one will stand up. Therefore, if not all the nodes in the local cache fail, the cluster operates normally. It is important to notice that the cluster leader election obeys to the follwing major rule:

Flash insensitivity: For stability purpose, at the joining process, eligible nodes that potentially win the leader election, are maintained at an idle state, until their life time in the cluster reaches a greater value. Basically, the node's position in the local cache is maintained at a second position, and the current cluster leader is kept. This allows LCC to avoid multiple cluster leaders changes due to flash state. A flash state is caused by a potentially always election-winner node that would join and leave the overlay in a repetitive manner.

4.1.6 Dynamic Clusters topology

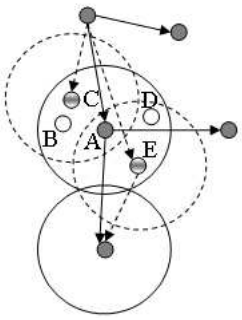


Figure 5: Dynamic clustering behavior: The solid lines show the clusters and the inter-cluster links before Leader A fails, the dashed lines afterwards. Two clusters have been created.

Since overlay nodes come and go, nodes Scope may contain a new cluster or cluster leaders may disappear from nodes' scopes. Clusters may be split to or merged. In fact, the split and merge procedures are also used to accommodate more members or to reduce the number of clusters. In this section, we discuss LCC behavior in case of cluster members

migration outside their cluster du to new cluster election. Fig. 5 shows an example of cluster split, after the cluster leader A fails. Here we suppose that the leader election rescue plan, has planished node C as the future cluster leader, and hence C will be promoted after A leaves the session. In this case, node D for example will not have the current cluster leader in its Scope. Node D has migrated. We distinguish between the three following clustering states:

Stabilized state where all cluster members have the cluster leader in their Scope.

Advanced state where, due to one or multiple leaders' failures, few nodes have migrated outside their original cluster, and the current leader is outside their Scope.

Temporary state where all migrated nodes detect their advanced state, construct a list of other migrated nodes, and start a re-stabilization process to evolute towards a stabilized state.

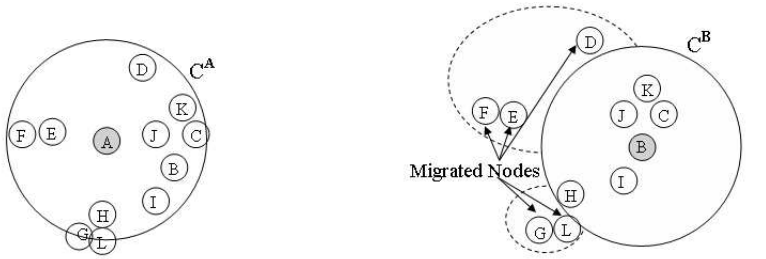
Nodes maintaining the local cache, shared between cluster members, verify at each local cache update operation either the first potential future leader is in their Scope or not. If not, they mark it as foreign cluster neighbor in their PV with $Scope$=0. At each received PV, a migrated node constructs a list of other migrated nodes. Once its list completed, the first ranked migrated node in the local cache initiates the re-stabilization algorithm. The algorithm is based on the local cache order, shared by the cluster members. It allows the migrated nodes to collaborate in order to verify their clustering state, and creates suitable new clusters after an advanced state. Fig. 6 illustrates a prototype of the algorithm. Fig. 8 describes the algorithm progress run by nodes after the Advanced state represented by the example shown in Fig. 7.

```
Algorithm 1 (Re-Cluster ( p )) First Triggered by the first in the eligible nodes cache

Global Data: Sorted Eligible nodes cache, Cache [ ]
        Potential Leaded Migrated Nodes, PLMN [ ]
        i  Myself.Order in (Cache)
        p Next requested node.Order in (PLMN)
1.      If p the last, consider p=0 unless p = i-1 //For the last node in the cache
2.      If (Myself.isLeaded) OR (potential leaded migrated nodes contacted)
3.              STATE = = STOP
4.      While (NOT STOP)
5.      If  Close(p)
6.              Leading (Cache(i+p)) //Cache (i+p) is leaded by myself
7.              Re-Cluster(i+p)
8.      Else
                Cache (i+p) removes me from  PLMN                                  ]
                Cache(i+p) verifies it has been contacted by every Cache (j); 1 < j < i ] Contacted node
                        If yes,  Re-Cluster triggered                              ] procedure.
                        Else, Idle ( )                                             ]

9.              If  (Leading < > NULL)
10.                     Idle ( ) //until node (k) contact; k > i
11.             Else
12.                     Re-Cluster(i+p)
13.             End If
14.     End If
End Re-Cluster ( p )
```

Figure 6: Re-stabilization algorithm running at the temporary clustering state



(a) Stabilized clustering state: Node A as cluster leader.

(b) Advanced clustering state: Node B elected as a new leader, nodes E, F, D, G and L migrated

Figure 7: Migration example after a new leader election

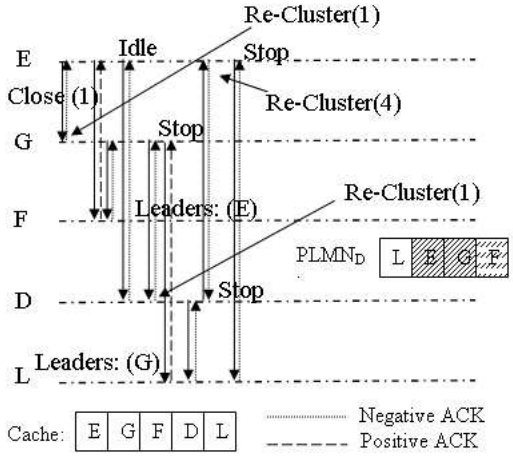


Figure 8: Example of the re-stabilization algorithm progress

The process is initiated by sending a re-stabilization request to the next migrated node. If the latter is in the requesting node Scope, it sends a positive acknowledgement and joins its cluster. Since it is led by a cluster leader, the node is not concerned by the re-stabilization algorithm. It only maintains a list of its leaders. Else, the node verifies that it has been contacted by all prior potential leaders in the cache. In fact, when sending a re-stabilization request, each node informs the requested node about other nodes that are in its Scope. Hence, each node is able through previous received requests to construct its prior potential leaders. In Fig. 8, node F does not send any request because it sends a positive acknowledgement to E, and node D does not trigger its Close () function until

it receives requests from both nodes E and G (F having been eliminated from the prior potential leaders, after E' request). Moreover, while receiving such requests, nodes update a set of "Potential Leaded Migrated Nodes" $PLMN$ that is used to not contact nodes that are leaders and are not in their Scope. Node D, when its Re-Cluster(1) function triggered, has only node L active in its $PLMN$. It then contacts node D to request re-stabilization. Finally, each node has to loop until all active nodes in its $PLMN$ are contacted, and it contacts a prior leader in the cache. After the re-stabilization process terminates, leaders creates a new cluster and informs their cluster neighborhood, in particular their previous cluster leader, of the cluster split. Finally, they inform their new members (if existing) and switch to a stabilized state.

4.2 Mesh Construction and Delivery Tree Management

The MeshTree protocol [4] proposes to embed the delivery tree in a degree-bounded mesh containing many low-cost links. The constructed mesh consists then of two main components: (i) a backbone structure, consisting in a low-cost tree and connecting nodes that are topologically close together, and (ii) additional links to form the mesh and improve the delay properties. Then, a delivery tree is derived from the mesh using a path-vector routing protocol.

In LCC, we adopt a similar approach for overlay construction. However, in order to ensure higher efficiency, we choose to run the mesh construction at both the top-level and the intra-cluster level. Furthermore, while MeshTree first constructs a randomly connected overlay and relies on incremental improvement, which involves adding/deleting links using a set of local rules run at each node, the LCC scheme, initially constructs location-aware overlay based of the locating and clustering processes.

The cluster leader is the primary responsible of connecting its cluster to the top-level overlay. Nevertheless, *Edge nodes* could also join the inter-cluster mesh. Top level nodes then act as particular nodes in the tree structure, where other clusters represent candidate neighbors in the backbone tree, or potential parents or children in the derived delivery tree (see Fig. 4). The set of neighbors for a node, say i, is represented by N_i, and is further classified into three groups:

N_i^b as the set of i's neighbors (parent and children) in the backbone tree,

N_i^d as the the set of i's neighbors in the delivery tree, and

N_i^o as the set of i's neighbors that are neither in N_i^b nor N_i^d

We then have $N_i = N_i^b \bigcup N_i^o \bigcup N_i^d$. The set of neighbors' links is defined by i that strictly enforces that $|N_i| \leq f_i^{max}$, guaranteeing in this way the degree constraint in the constructed mesh, and hereafter in the generated delivery tree.

4.2.1 The top level overlay construction

The cluster leader is the primary responsible of connecting its cluster to others. The cluster leaders act then as particular nodes in the MeshTree structure, where other clusters represent candidate neighbors in the Backbone tree, the constructed mesh and potential parents or children in the derived delivery tree. In the rest of this section, we will refer to the cluster leader as L_i and to potential edge node as EN. We also introduce the set of cluster neighbors similar to the set of node's neighbors defined in the previous section. Hence, $CN_{L_i} = CN^b_{L_i} \bigcup CN^o_{L_i} \bigcup CN^d_{L_i}$ where CN_{L_i} is the set of the cluster C^{L_i}'s neighbors.

Initially, a new cluster leader, L_i selects a fixed number of the closest cluster leaders (L_j), and sequentially sends to each of them a join request message. On receiving a join request message, the closest cluster leader, L_j accepts L_i if it has available fan-out or if it has a cluster neighbor in its extra-links neighbors set $CN^o_{L_i}$. In the second case, C^{L_i} will drop the cluster providing the worst extra link. Otherwise, L_i's request is rejected. If L_i is accepted, it is added as a child in the backbone and delivery tress. L_i is informed by an acknowledgement, which is forwarded immediately to the potential cluster members in $M(C^{L_i})$. L_i then acknowledges the parent. For other clusters that also accept L_i as a child, L_i includes them as potential extra links for the construction of the mesh, and informs them of this change. These clusters leaders will perform changes in their cluster neighbors set accordingly. At this stage, L_i and its potential cluster members have successfully joined the overlay and are connected to the backbone tree, and sharing extra links in the degree-bounded mesh.

4.2.2 The intra-cluster overlay

Two different cases are to be considered:

1. A newcomer, A having in its Scope a unique cluster leader, is accepted as a cluster member; it sends a join backbone request, and L would act as RP responding with random existing members in the cluster, so as to guide the new node to connect to the intra-cluster overlay. The connection process is established, and an intra-cluster refinement is periodically processed.

2. A newcomer, EN is accepted as an Edge node belonging to two or more clusters; Basically, EN after detecting its neighboring clusters, has to perform several joins to the clusters it belongs to. That is, since it joins multiple backbone trees of different clusters at the same time, it is involved in multiple delivery trees construction. Fig. 4 shows an example of two derived delivery trees in the same cluster, where node A connects to $EN1$ as a source in the top-level topology. $EN1$ is connected to the top-level delivery tree via the cluster leader L. $EN1$ offering better overlay latency, A is its child in the delivery tree. Note that node A is also connected to the backbone tree in cluster C^K and could be a child of K in the backbone tree. The extra-links mesh construction allows A to bind a link with $EN1$ and to derive a better delivery tree. EN has also the possibility to join the intra-cluster overlay as a child in the

delivery tree, by selecting the best overlay distance to the source, s via its potential parent in the delivery tree. Hence the Edge node would sort the clusters it belongs to in an increasing routing information order, and starts the join procedure as a "simple" member.

5 Performance Evaluation

To evaluate and test the LCC scheme, we both carried out simulation experiments, and we implement LCC and deploy it on the PlanetLab wide-area testbed [23]. The goal of simulation studies is to assess the effectiveness of proposed techniques for large scale overlays while Internet experiments illustrate the system performance under particular real-world environments.

5.1 Comparison

In order to compare LCC to initially randomly-connected overlays relying on periodic refinements, we experiment a variant of LCC, disabling the locating process and setting the Scope value to zero, thus emulating MeshTree behavior. We call this variant: Randomly connected Overlay or Flat MeshTree. We also introduce a random locating technique that does not maintain nodes within levels in the locating system. The basic idea of random locating, or *RLocating* is to request a randomly selected set of known nodes to measure their distances to the newcomer. The overhead of this approach can be controlled by choosing the stop criterion C that defines the procedure repetitions. However, although *RLocating* has the advantage of simplicity, this technique induces much more overhead, and long convergence time to a stabilized structure than the LCC locating technique. For reference purposes, we approximate the ideal multicast performance using the underlying direct unicast, measuring the latency along the direct Internet path from the source to each receiver. In our simulations, we also compare LCC to several proposed multicast overlay structures: OMNI [6], ZIGZAG [10], SpreadIt [3].

5.2 Metrics

We investigated the LCC scheme in terms of common performance metrics that characterize the multicast overlay. In particular we measure the Average Relative Delay Penalty ($ARDP$), the overlay latency and throughput distribution. Second, we observed the convergence time to an optimized structure ($ARDP < 2$) and the link adjustment rate of each node in the overlay. Moreover, our measurements concerned the locating process efficiency in terms of the protocol overhead and average number of clusters contacted by a newcomer in order to locate its nearest cluster. We also tested the LCC overlay robustness to nodes' failures, especially in case of catastrophic leaders failures.

5.3 Simulation Setup

We implemented our simulations on p2pns [24] and run experiments on large networks. Using the BRITE Internet topology generator [25], we were able to simulate 10^3 to 10^4 nodes networks. We used a node bandwidth heterogeneity reference model based on the Gnutella peer upstream bandwidth distribution observed by Saroiu et al. [26]. We picked one of the following distributed uniformly five bandwidths: 50 kbps, 400 kbps, 800 kbps, 3500 kbps, and 15000 kbps. Finally, we modeled the node join using Poisson distribution with an average rate of 30 and 60 node joins per simulation tick for network topologies of 1K and 10k nodes respectively. The duration distributions were modeled with an exponential distribution of 0.01 as parameter.

5.4 Experimentation on PlanetLab

We also build a prototype implementation of LCC, and tested it. We conducted our experiments over two different sets of wide spread PlanetLab nodes with 90 nodes in Set 1 and 212 nodes in Set 2. All the nodes in the first set are nodes in different sites in the United States. The second set consists of world wide nodes with 90 nodes in the United States, 20 nodes in Canada and south America, 80 nodes in Europe and 22 nodes in Asia. All experiments were conducted over several days in October 2005.

Although we have conducted several experiments with different source nodes, we present in this paper a representative set of experimental results using "planetlab1.cs.cornell.edu" as the data source, generating a 560 kbits/s (70 kB/s) data stream sent to all the other group members over the overlay. Nodes join the overlay network at the average rate of one every 2 seconds. For network latency measurement between two nodes, we simply make a node ping the other x times and measure the round-trip times. We remove the top 20% and bottom 20% of the measurement results and take the average of the remaining values. In practice, pings have been conducted using ICMP ECHO messages, and we use 10 pings for latency measurements [3].

5.5 Performance Results

We first consider the latency property of the LCC overlay. Fig. 9 illustrates the latency from the source to all receivers in Set 2, ranked increasingly on the unicast latency. Results have been collected 2 minutes after the last node joined the overlay. We present here a snapshot of delay properties before convergence in order to show the importance of the locating process during the overlay construction. In this snapshot, with an $ARDP$ value of 1.25, LCC outperforms *RLocating* ($ARDP$= 1.72) and the randomly connected overlay ($ARDP$=3.97). The results demonstrate the efficiency of the locating process, while the random structure would require long convergence time and refinement operations to stabilize and improve the delivery tree quality.

[3] In [27] the author shows that latency measurement with 10 pings are already very accurate.

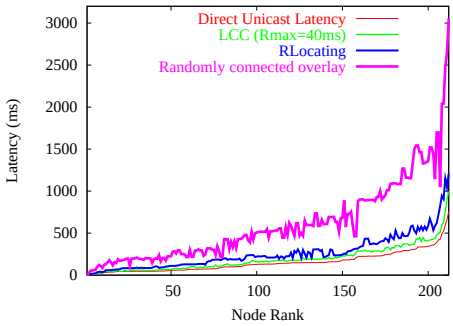


Figure 9: Real Snapshot of Nodes Latencies.

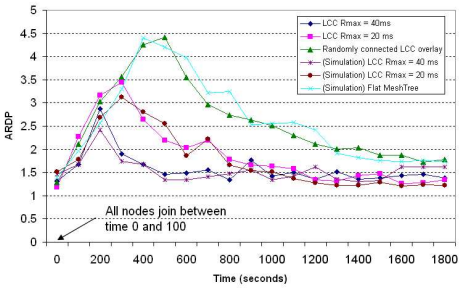


Figure 10: Convergence Time property.

Fig. 10 illustrates this convergence time, plotting $ARDP$ versus the multicast session time in both simulations (Overlay size = 2000 nodes) and PlanetLab testbed (Set 2). This metric describes the overlay structure evolution in time. In this experiment, all nodes join the overlay within the first 100 seconds. In the randomly connected overlay, $ARDP$ increases quickly and reaches higher values than LCC. The simulated Flat MeshTree evolution is slightly the same than the initially-randomly connected structure, confirming the long convergence time needed by those structures. In the experiments, we set the periodical improvement period RT to 30 seconds, for each of the receivers. We can see that in LCC overlays, $ARDP$ rapidly decreases to a value less than 2 after the first 200 seconds, i.e. less than 7 improvement rounds per node, in both simulations and real deployments. For the randomly connected overlays, it takes much more time, more than 1200 seconds, to $ARDP$ to stabilize. This indicates that LCC can converge very quickly, and induces less improvement rounds and link adjustments to build an efficient delivery tree during overlay growth or frequent membership changes.

Fig. 11 shows the LCC structure stability during membership changes. We start tracking the link adjustment counts right after the last node has joined. Results are collected at 30-

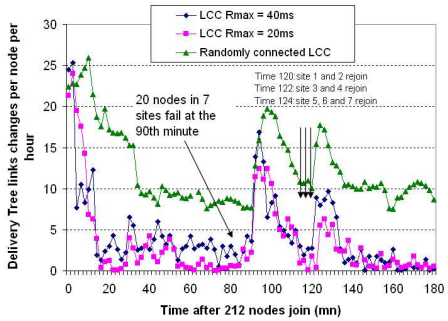


Figure 11: Link Adjustment rate.

second intervals. We observe that the link adjustment rate mostly falls below 5 per hour per node for the LCC overlay, whereas it stabilizes at roughly 10 per node per hour for the randomly connected overlay. We then inject a simultaneous 20-nodes failure in 7 different sites at the 90^{th} minute and we let them rejoin the overlay at the 120^{th} minute. We observe that the link adjustment activity is moderate (mostly under 5 per hour per node) during the membership changes. After the 140^{th} minute, the average link adjustment count falls around 2 per hour per node for LCC, which indicates around one link adjustment at a single node during each 30-second interval. Due to randomness in initially connecting newcomers to the clusters, the link adjustment rate of the randomly connected overlay is maintained at 10.

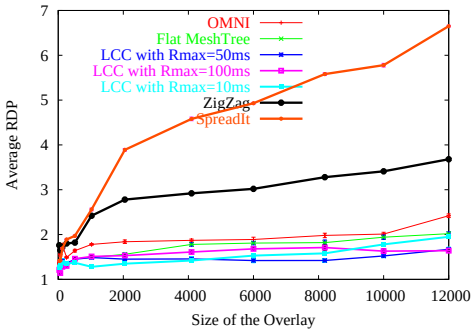


Figure 12: ARDP Comparison varying Overlay Size.

To be able to characterize the average incurred delay observed by the receivers in a large populated overlay, we expand the latency results by comparing LCC to different proposals simulations. We then plot the *ARDP* variation according to the overlay size for different overlays in Fig. 12. Without surprise, we observe high *ARDP* values for the SpreadIt overlay,

relying on random connections. ZigZag, although roughly maintaining a stable $ARDP$ value while the overlay size is increasing, suffers poor performance ($ARDP \geq 2.5$). This is mainly caused by the logical hierarchy clustering in ZigZag, not mapping the underlying network on the overlay.

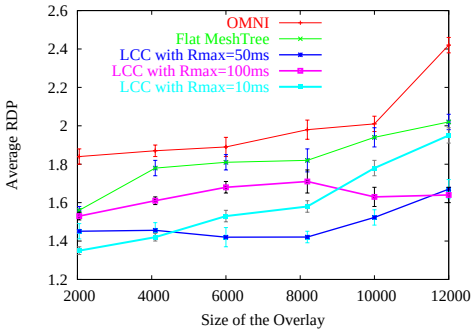


Figure 13: A zoomed in view of the portion [2000-12000] nodes of Fig. 12.

To make it easier to read, we zoom in on a portion of the graph in Fig. 13. We observe that the $ARDP$ values for different R_{max} in LCC are roughly maintained at values between 1.2 and 1.6. In OMNI and the Flat MeshTree structure, we observe that $ARDP$ values increase drastically. We also note that in larger overlays, for clusters defined with 10 ms as node's Scope, $ARDP$ increases slightly (still less than 2), as nodes are more widespread. Scopes of 50 ms and 100 ms are more efficient in the case of very large overlay. In LCC with a Scope value equals to 100 ms, $ARDP$ even decreases from 1.71 to 1.64 for overlay size greater than 8000 nodes. Larger Scopes are more convenient for large overlays where it is more likely that nodes are more widespread.

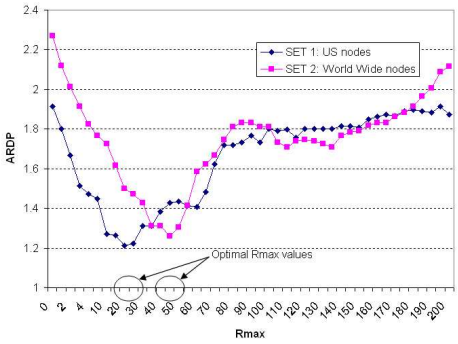


Figure 14: Clustering configurable View Scope versus application deployment.

We study this feature by deploying the LCC scheme on both Set 1 and Set 2 of the PlanetLab nodes, and comparing obtained performances. In fact, in our proposed scheme, R_{max} is a configurable value which could be set according to the network characteristics that the overlay would be deployed on. Deploying a thousands-members multicast application on a world-wide area implies setting the node's Scope to a large enough value to be able to cluster nodes, whereas for a local-area application, the Scope should be set to a smaller value. In Fig.14, we experiment two deployments of LCC on both set 1 and 2, while varying the clustering Scopes of the overlay nodes. We observe two different minimal values of $ARDP$ in the deployment of LCC. In set 1, where LCC deployment is restricted to nodes in the U.S, the "optimal" R_{max} value observed is 30 ms. While Scopes of nodes are set in the interval 25..35 ms, the average delay observed by the receivers is minimal. In set 2, the "optimal" R_{max} value is larger (between 45 ms and 50 ms).

Simulation results in Fig. 12 show the efficiency of the topology clustering, that absorbs the impact of overlay size by grouping nodes into clusters. It enables then the top-level overlay to construct efficient trees. Cluster leaders at that level are not aware of the number of nodes they are managing. By increasing R_{max} from 50 ms to 100 ms, we decrease the number of clusters on the top-level, while increasing the number of intra-cluster members, conserving then better performance on the cluster leaders levels. One can intuitively conclude that better delay performance is achieved at the expense of intra-cluster stress that increases drastically.

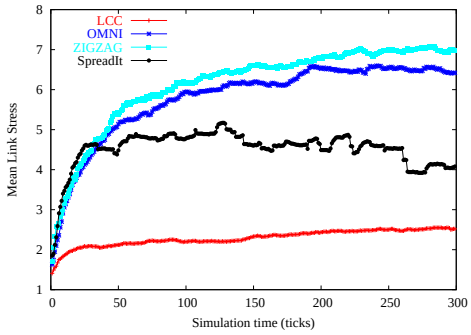


Figure 15: Stress vs simulation time.

Representing the number of copies of an identical packet sent over a single link, *Stress* higher than 1 is an indicator of a deviation from an "ideal" solution. Fig. 15 shows average physical network stress for each of the overlays OMNI, ZigZag, SpreadIt and LCC ($R_{max} = 50ms$). OMNI and ZIGZAG stress values stabilize between 6.5 and 7. The SpreadIt overlay instigates somewhat lower stress than OMNI and ZIGZAG with stress highly oscillating between 4 and 5 due to random connections established by newcomers. We note finally that LCC has an impressively low stress, significantly better than other overlays, with a steady stress value of 2.5 to 2.8. Topology information is of paramount importance in this

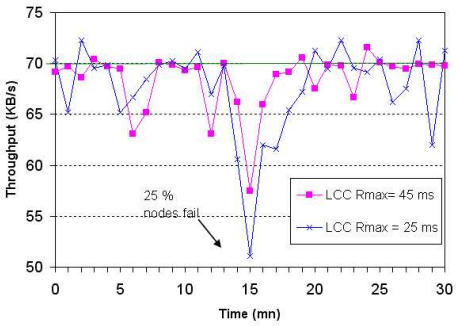


Figure 16: Observed throughput of a 30 minute data stream.

observation, as packets sent through the top-level hierarchy are sent to the cluster leader and in some cases to potential edge nodes. Clustering allows to reduce the amount of redundant flows, entering each network, considering clusters as local networks, and cluster leaders as multicast-enabled routers. Thus, we are able to make an efficient usage of the bandwidth.

Fig. 16 illustrates the throughput properties of the LCC overlay. We first observed the mean throughput of 212 nodes overlay of a 30-minutes data stream. 25% of nodes fail at the 13th minute. Nodes organized in a 45 ms Scope recover faster than LCC nodes with R_{max} = 25 ms. In fact, with a larger Scope, cluster size is more important and nodes that would reconnect first to nodes in their own cluster would be accepted with higher probability. Nevertheless, besides crash scenarios, nodes has mean throughput values not less than 65 KB/s.

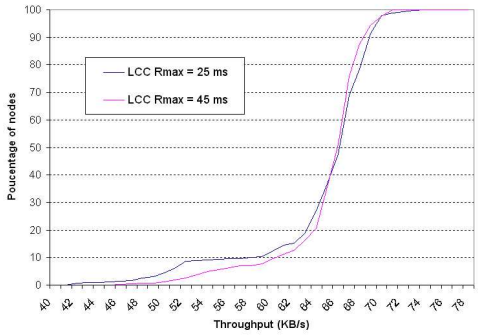


Figure 17: CDF of instantaneous achieved throughput at time 32 minutes.

In Fig. 17, we observe the distribution of instantaneous achieved throughput at time 32 minutes. Only 9.4% of nodes for LCC with R_{max} = 25 ms, and 5.6 % for LCC with R_{max}

= 45 ms, have a mean throughput less than 55 KB/s. More than 80% of nodes have a mean throughput superior to 60 KB/s.

Previous results prove that the locating process allows the LCC structure to considerably reduce the number of adjustments during the multicast session. The hierarchy established on the basis of this locating process achieves then good overlay performances. Nevertheless, this benefit is achieved at the expense of measurement overhead.

To evaluate the impact of maintaining an hierarchical overlay while locating the closest cluster to join, we measured the control traffic overhead in the overlay. We investigated this overhead metric in terms of average control traffic generated by overlay nodes, and quantified it in kbps.

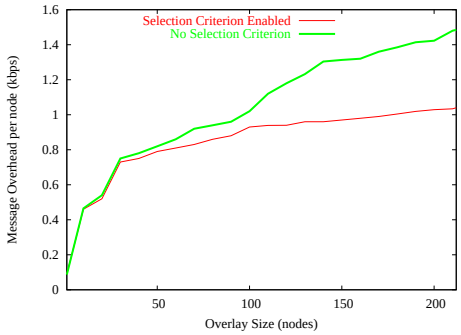


Figure 18: LCC overhead at the joining process.

We first studied the locating process overhead on the PlanetLab testbed, and compared between the both cases where the selection criterion is enabled in the locating process and when disabled. We can observe in Fig. 18 that the per node overhead in LCC (for R_{max} = 45 ms), enabling the selection, is reasonably small with less than 1 Kbps for a 212 nodes overlay. In addition, it increases very slowly when the number of members increases. In this experiment, locating procedure messages are roughly 50% less than a localization without a selection of nodes, when the overlay reaches its maximum size (212 nodes). When the selection criterion is enabled, the overhead is insensitive to the overlay size. Not selecting nodes during the locating process boosts the message overhead due to useless measurement operations. Boot nodes would contact all the nodes in the newcomer's level and the adjacent levels. These nodes will also measure their distance to the newcomer, which would also add network overhead.

We run simulations to evaluate the control traffic overhead in the overlay during multicast session and observe the protocol behaviors in larger network. Fig. 19 shows the average overhead per node when varying the overlay size. We assume a basic header size of 40 bytes per packet and we measure the total control message traffic sent and received by each node throughout a session. We note that the per node overhead in LCC, for different R_{max} values, is steady for small overlays. The maximum value reached for a 512 nodes overlay is 1.10

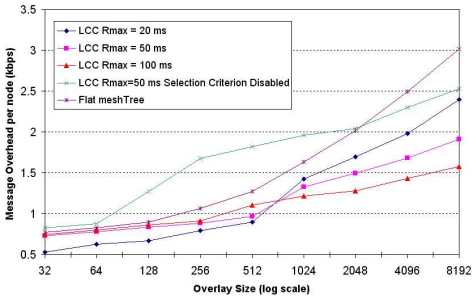


Figure 19: Simulation of protocol overhead.

kbps for LCC with R_{max} = 100ms. Variations in small overlays is quiet the same. Since control messages are not spread outside the clusters, top-level nodes perform "good" results and stabilize the overhead value while the overlay size is increasing. Hence, we observe that the LCC nodes, for R_{max} of 50 ms and 100 ms in the plot, incur in average less than 2 kbps message overhead, for more in an 8000-nodes overlay. We note that the above results include overheads due to network measurement, especially during the locating process, as we consider these results from the instant the node contacts RP.

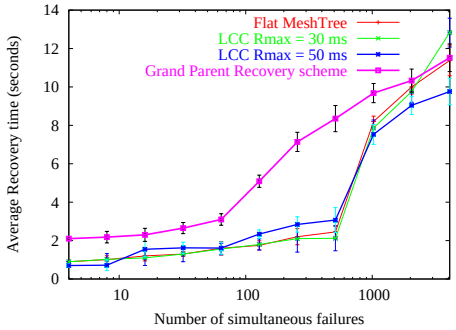


Figure 20: Failure Recovery time property.

An overlay tree needs to be reconstructed after a non-leaf node departure. It is important this node's children can quickly locate a new parent to resume the data flow. In addition, the recovery process should not result in a fan-out violation in any node. We believe that LCC is resilient to node failure because all a node has to do to recover from the failure of its neighbor is to redirect packet requests from that neighbor to a different one in its proper cluster, and hence a nearby node. We compare LCC to the grandparent recovery scheme studied in [28], in which the children of the node who departs first try to attach to their grandparent. The grandparent will try to accommodate them as long as it has spare

capacity. Otherwise, it will redirect them to its descendants. To evaluate the impact of clustering, we also compare LCC to the Flat MeshTree. We instruct a number of randomly selected nodes in a 5000 sized overlay, to leave the session. Then, we observe the recovery time of the overlay members, as the average time for an affected node to resume the data flow. Results in Fig. 20 show that LCC always yields a smaller recovery time than the tree-based grandparent recovery scheme. Moreover, LCC with $R_{max} = 50$ ms gets similar recovery time than a Flat MeshTree overlay.

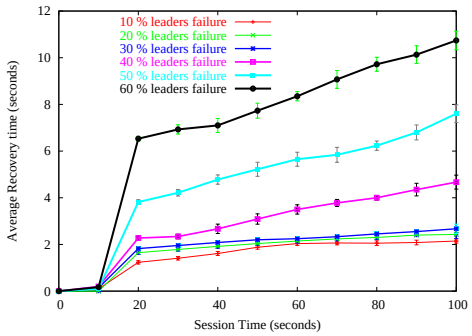


Figure 21: Impact of Leader failures: each 10 seconds a selected random set of cluster leaders fails.

We also studied the capacity of the LCC overlay to recover from cluster leaders failures. Each 10 seconds, a set of randomly selected cluster leaders are instructed to leave a 5000-sized LCC overlay with $R_{max} = 50$ ms at the same time. We observe the recovery time of other members in Fig. 21. The experiment allows us to observe the behavior of LCC in case of catastrophic failures. We observe that when less than 30 % of cluster leaders fail simultaneously and continuously each 10 seconds, the proactive rescue plan of leaders election achieves good performance since the recovery time is reasonable. Things are complicated when the percentage of cluster leaders failing simultaneously increases. The nodes' recovery time is high for both scenarios of 50% and 60% leaders failure. When cluster members elect a new cluster leader, the latter inheriting from the failed leader neighbors would first try to attach its cluster to the known cluster neighbors. As cluster neighbors are more likely to be affected by catastrophic failures, the recovery time increases.

Finally to evaluate the behavior of newcomers during the locating process, we observe the average number of cluster leaders contacted by newcomers. We consider as a hop each contacted cluster leader during the locating process. Fig. 22 depicts the average number of hops versus the total number of known neighbors for each cluster leader, in case of the *RLocating* technique and the LCC locating process. We differentiate between enabling and disabling the selection criterion in the LCC locating process. We also plot the average number of hops resulting from simulations of 200 newcomers running the locating process (with the selection criterion enabled), after the overlay size has reached respectively 2000,

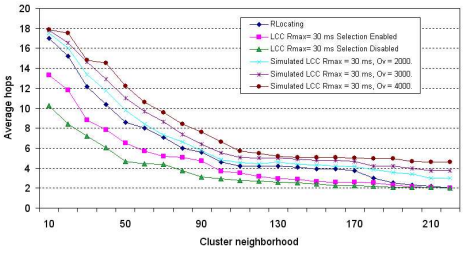


Figure 22: Average hops vs Known cluster neighbors.

3000 and 4000 nodes (respectively denoted Ov = 2000, Ov = 3000 and Ov = 4000 in Fig. 22). Hence, we are able to compare the number of contacted cluster leaders in both small and large overlays. The figure confirms that the more the knowledge of nodes the contacted cluster leader has, the less the number of hops will be. However, the number of nodes, k maintained per level in each node's locating system, represents an inherent tradeoff between accuracy and overhead. A large value of k increases leader's information about other overlay members and helps making better choices when guiding newcomers. On the other hand, a large value of k also entails more state, more memory and more bandwidth at each node. Enabling the selection criterion has little impact on the efficiency, and performs almost as well as the locating process contacting all nodes in the newcomer's level and adjacent levels, i.e. where the selection criterion is disabled. When the selection criterion is enabled, the experiment shows that the LCC locating process guides newcomers to the closest cluster in at average 70% less hops than the *RLocating* technique. This proves the efficiency of the selection-enabled locating system.

Through the simulations, we observe that even when joining large overlays (4000 nodes), newcomers have slightly the same behavior. In particular, using the simulated overlay network consisting of 4000 nodes when the newcomers initiates the locating process, the average number of hops is less than 10 when managing 60 nodes as cluster neighborhood. Our results also show that average locating time when contacting 10 hops is 860 ms.

We also observe the distribution of nodes in the locating process through simulation. The experiment has been set up for a maximum neighborhood knowledge of 20 nodes per cluster leader's locating system and with the stop criterion $C = 15$ hops. The maximum overlay size reached is 2000 nodes. We observe in fig. 23 that more than 80% of all nodes are able to terminate the locating process in less than 15 hops.

Finally, we evaluate the locating process accuracy. Table 1 represents the percentage of nodes connected to the closest cluster after the locating process terminates (upon nodes' arrival) and 300 seconds after the last node joins (after the overlay converges). The locating process is strengthened by the clustering process, where if a cluster member has no spare capacity, the newcomer is redirected to the closest member in the cluster.

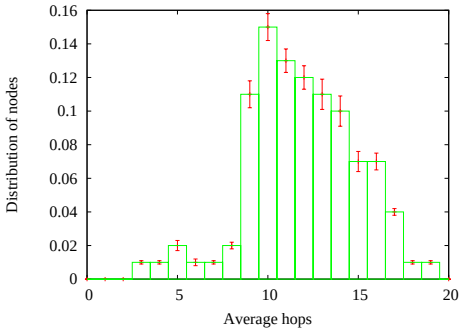


Figure 23: Distribution of nodes during the locating process.

Table 1: Percentage of nodes connected to the closest cluster and to the closest node, depending on the maximum fan-out allowed.

Maximum fan-out allowed	2	4	6	8	10	213
% of nodes connected to the closest cluster upon arrival	76	77	77	80	84.2	91
% of nodes connected to the closest cluster after convergence	89	92	92	91	93	94
% of nodes connected to the closest node upon arrival	72	73.5	74	75	82	89

6 Conclusion

In this paper, we proposed a locating process run by overlay nodes at bootstrap. This "Locate-first" approach allows overlays to achieve scalability, by reducing the amount of overhead due to refinement process and link adjustment during the multicast session. Each newcomer, after contacting a few nodes, is able to locate the closest set of nodes in the overlay. The locating phase includes a selection algorithm to minimize the number of probes during the process, and hence to minimize measurement overhead. On the basis of this locating process, we described an hierarchical topology-aware overlay where each located newcomer decides to join an already existing cluster or creates its own cluster. Thus, LCC is able to exploit the location information to build efficient, scalable and optimized resource-usage multicast overlay. We implemented LCC and conducted real-world experiments proving that LCC performs better than initially randomly-connected proposals, in terms of delay, overhead and convergence time to a stabilized overlay. We also verified scalability of our scheme through different simulations. Results also showed that the locating process is accurate and entails modest resources. In future works, we will focus on further experiments in order to tune different parameters, such as the stop and selection criteria parameters in the locating

process. We will also extend the scheme to multi-layer hierarchy for scalability purposes, in particular to manage cluster size expansion. Finally, we will investigate techniques to secure the overlay and possibly detect/prevent users from cheating.

References

[1] D. A. Helder and S. Jamin. *End-host multicast communication using switch-trees protocols.* In GP2PC, 2002.

[2] Z. Li and P. Mohapatra. *Hostcast: A new overlay multicast protocol.* In IEEE ICC 2003.

[3] H. Deshpande et al. *Streaming live media over a peer-to-peer network.* Technical Report 2001-31, Standford University, 2001.

[4] S. W. Tan, A. G.Waters, and J. Crawford. *Meshtree: A Delay optimised Overlay Multicast Tree Building Protocol.* Technical Report 5-05, University of Kent, University of Kent, April 2005.

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. *Scalable Application Layer Multicast.* In ACM SIGCOMM, 2002.

[6] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. *Construction of an efficient overlay multicast infrastructure for real-time applications.* In IEEE INFOCOM 2003.

[7] K. A. Hua, D. A. Tran, and R. Villafane, *Overlay multicast for video on demand on the internet*, in ACM Symposium on Applied Computing 2003.

[8] Y. H. Chu, S. G. Rao, and H. Zhang. *A case for end system multicast.* In ACM SIGMETRICS 2000.

[9] Y. Chawathe, S. McCanne, and E. Brewer. *RMX: Reliable Multicast for Heterogeneous Networks.* In IEEE Infocom 2000.

[10] D. Tran, K. Hua, and T. Do. *Zigzag: An Efficient Peer-to-Peer Scheme for Media Streaming.* In IEEE Infocom 2003.

[11] W. Wang, C. Jin and S. Jamin. *Network Overlay Construction under Limited End-to-End Reachability.* In IEEE Infocom 2005.

[12] S. Ratnasamy, et al. *Topologically-Aware Overlay Construction and Server Selection.* In IEEE Infocom 2002.

[13] S. Ratnasamy, et al. *A Scalable Content-Addressable Network.* In ACM SIGCOMM 2001.

[14] F. Dabek et al., *Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service.* In Hot Topics in Operating Systems 2001.

[15] B. Y. Zhao et al., *Tapestry: A Resilient Global-scale Overlay for Service Deployment.* In IEEE JSAC, January 2004

[16] M. Castro, et al., *Proximity neighbor selection in tree-based structured peer-to-peer overlays.* Tech. Report MSR-TR-2003-52, 2003.

[17] M. Waldvogel, R. Rinaldi. *Efficient Topology Aware Overlay Network.* In ACM SIGCOMM Com. Comm., January 2003.

[18] L. G. Erice, E. W. Biersack, and P. A. Felber. *MULTI+: A Robust and Topology-Aware Peer-to-Peer Multicast Service.* To appear in Comp. Comm., 2005.

[19] B. Wong, A. Slivkins, and E. G. Sirer. *A Lightweight Approach to Network Positioning without Virtual Coordinates.* In ACM SIGCOMM 2005.

[20] X. Y. Zhang, et al., *A Construction of Locality-Aware Overlay Network: mOverlay and Its Performance.* In IEEE JSAC, JANUARY 2004.

[21] K. Lai and M. Baker. *Measuring Link Bandwidths using a Deterministic Model of Packet Delay.* pages 283.294, 2000.

[22] M. Jain and C. Dovrolis. *End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput.* In Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, ACM Press, 2002.

[23] *An open, distributed platform for developing, deploying and accessing planetary-scale network services.* See http://www.planetLab.org

[24] *A Simulator for Peer-to-Peer Overlay Algorithms.* See http://p2pns.sourceforge.net

[25] A. Medina, A. Lakhina, I. Matta, and J. Byers. *BRITE: Universal topology generation from a user's perspective.* Technical Report BUCS-TR-2001-003, Boston University, January 2001.

[26] S. Saroiu, P. K. Gummadi, and S. D. Gribble. *A measurement study of peer-to-peer file sharing systems.* In MMCN 2002.

[27] K. Shen. *Structure Management for Scalable Overlay Service Construction.* Im USENIX NSDI 2004.

[28] M. Yang and Z. Fei. *proactive approach to reconstructing overlay multicast trees.* In INFOCOM 2004.

Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803