MULTI+: A Robust and Topology-Aware Peer-to-Peer Multicast Service

Luis Garcés-Erice [a] and Ernst W. Biersack [b]

[a] *IBM Research, Zürich, Switzerland*

[b] *Institut EURECOM, Sophia-Antipolis, France*

To appear in Computer Communication, in late 2005

Abstract

TOPLUS is a lookup service for structured peer-to-peer networks that is based on the hierarchical grouping of peers according to network IP prefixes. In this paper we present MULTI+, an application-level Multicast protocol for content distribution over a TOPLUS-based peer-to-peer (P2P) network. We use the characteristics of TOPLUS to design an overlay Multicast protocol that allows every peer to connect to an available peer that is close. MULTI+ trees also reduce the amount of redundant flows leaving and entering each network, making efficient usage of the bandwidth. We also study the resiliency of MULTI+ Multicast trees when massive failure or disconnection of peers occur. While the tree reconstruction introduces additional hops, the end-to-end latency increases very little. At the end we compare the trees constructed by MULTI+ with the trees constructed by Scribe, a well known application-level Multicast system.

Key words: P2P, Multicast, Topology-aware, DHT, Content distribution
PACS:

1 Introduction

IP Multicast seems to be the ideal solution for content distribution over the Internet: it can serve content to an unlimited number of recipients, and it is bandwidth-efficient [1]. These two characteristics are strongly related. IP Multicast saves bandwidth because a single data flow can feed many recipients. The data flow is only split at those routers where destinations for the data are found via more than one outgoing port. Thus n clients do not need n independent data streams, which allows for IP Multicast's scalability. However, IP Multicast was never widely deployed in the Internet.

Lately, peer-to-peer systems have attracted lots of interest. Peer-to-peer systems allow to implement new services at the *application level* to enhance the capabilities of the network. In the last few years, various application-level Multicast protocols have been proposed [2–6], most of which are directly implemented on top of a P2P infrastructure (Chord [7], CAN [8] or Pastry [9]). The good scalability of the underlying P2P network allows to serve, as in the case of IP Multicast, content to a virtually unlimited number of clients (peers). However, P2P Multicast is implemented at the application layer as an overlay on top of and isolated from the underlying IP network layer. Thus, P2P Multicast will never be as bandwidth-efficient as IP Multicast: For example, a LAN hosting multiple peers that are all part of a P2P Multicast tree may find its outbound link saturated by *identical* data flowing to and from its local peers as these peers are typically not aware of the fact that they are sharing the same network. We have based our P2P Multicast protocol, called MULTI+, on TOPLUS because of its inherent topology-awareness. The algorithms in MULTI+ use the locality among peers provided by TOPLUS in order to build efficient Multicast trees, as we show later.

In the next section we provide a very brief overview of the otherwise vast literature on application-level Multicast. Then in Section 3 the main aspects of TOPLUS are presented. In Section 4 we describe MULTI+, which is built on top of TOPLUS. In Section 5 we study the properties (end-to-end delay, number of hops) of the trees constructed by MULTI+. We study the impact of peer failures on the MULTI+ trees in Section 6 and compare MULTI+ trees to the ones built by Scribe in Section 7. We conclude the paper in Section 8.

2 Related Work

There exist various proposals for application-level Multicast. Narada [2] organizes peers in a fashion similar to ours, using a hierarchical clustering of peers. There are also a number of P2P Multicast systems that use a DHT (distributed hash table), such as M-CAN using CAN [5] and Scribe [6] using Pastry, or Bayeux [4] based on Tapestry [10]. A comparison of these approaches can be found in [11]. Further examples of application-level Multicast are ALMI [12] or Yoid [13].

The authors in [14] build a topology-aware Multicast overlay. The connections among peers are optimized by detecting overlap in routes to a sender to limit the number of identical packets on the same link and to achieve better bandwidth utilization. This requires the use of `traceroute` and repeated probing of the network for constant adaption to changing conditions, especially upon arrival of new peers.

In Bullet [15], peers form a mesh instead of a tree for content distribution. The main idea is to increase the available bandwidth to each peer by downloading from mul-

tiple parents instead of just one, as is the case for trees. Basically, each peer benefits from parallel download [16] in a similar fashion as does SplitStream. However, the fact that each peer is able to download different chunks of data from different peers makes Bullet more suitable for large file distribution (like BitTorrent [17]) than for live content streaming.

ZIGZAG [18] offers low root-to-leaf delay by building trees with few levels. This approach organizes peers in clusters, and the arrival of new peers may cause a cluster to split, with the subsequent overhead to reorganize the P2P network. ZIGZAG is not topology-aware, and the stretch between direct IP routing paths from root to leafs and those of the application-level Multicast is high. As many identical copies flow through the same physical link, ZIGZAG can induce a relatively high link stress.

3 TOPLUS Overview

TOPLUS [19] is based on the DHT paradigm, in which a resource is uniquely identified by a key, and each key is associated with a single peer in the network. Keys and peers share a numeric identifier space, and the peer with the identifier closest to a key is responsible for that key. The principal goal of TOPLUS is simple: each routing step takes the message closer to the destination.

Let I be the set of all 32-bit IP addresses. Let $\mathcal{G}$ be a collection of sets such that $G \subseteq I$ for each $G \in \mathcal{G}$. Thus, each set $G \in \mathcal{G}$ is a set of IP addresses. We refer to each such set G as a *group*. Any group $G \in \mathcal{G}$ that does not contain another group in $\mathcal{G}$ is said to be an *inner group*. We say that the collection $\mathcal{G}$ is a *proper nesting* if it satisfies all the following properties:

(1) $I \in \mathcal{G}$.
(2) For any pair of groups in $\mathcal{G}$, the two groups are either disjoint, or one group is a proper subset of the other.
(3) Each $G \in \mathcal{G}$ consists of a set of contiguous IP addresses that can be represented by an IP prefix of the form $w.x.y.z/n$ (for example, $123.13.78.0/23$).

The collection of sets $\mathcal{G}$ can be created by collecting the IP network prefixes from BGP tables and/or other sources [20]. The resulting tree must be downloaded by peers, in the same manner as other P2P protocols download a list of servers to connect to initially. In this case, many of the sets $\mathcal{G}$ would correspond to ASes, other sets would be subnets in ASes, and yet other sets would be aggregations of ASes. This approach of defining $\mathcal{G}$ from BGP tables requires that a proper nesting is created. Note that the groups differ in size, and in number of subgroups (the fanout). If $\mathcal{G}$ is a proper nesting, then the relation $G \subset G'$ defines a partial ordering over the sets in $\mathcal{G}$, generating a partial-order tree with multiple tiers. The set I is

at tier-0, the highest tier. A group G belongs to tier 1 if there does not exist a G' (other than I) such that $G \subset G'$. We define the remaining tiers recursively in the same manner (see Figure 1).

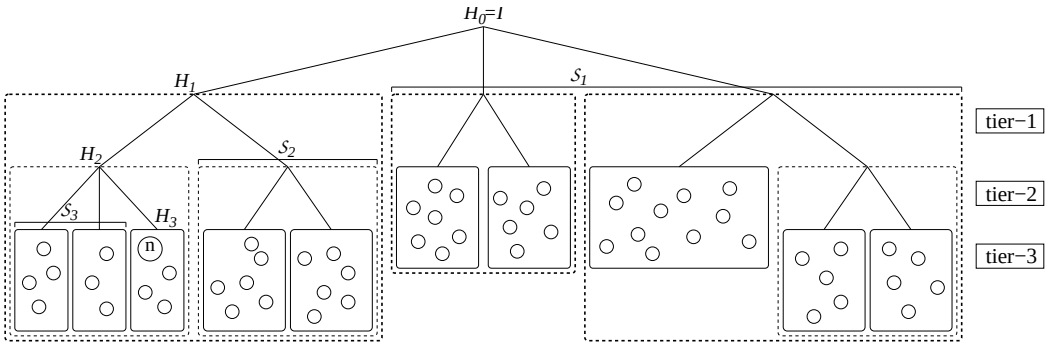


Fig. 1. A sample TOPLUS hierarchy (inner groups are represented by plain boxes)

3.1 Peer State

Let L denote the number of tiers in the tree, let U be the set of all currently active peers and consider a peer $p \in U$. Peer p is contained in a collection of telescoping sets in $\mathcal{G}$ denoted by $H_N(p), H_{N-1}(p), \cdots, H_0(p) = I$, where $H_N(p) \subset H_{N-1}(p) \subset \cdots \subset H_0(p)$ and $N \leq L$ is the tier depth of p's inner group. Except for $H_0(p)$, each of these telescoping sets has one or more siblings in the partial-order tree (see Figure 1). Let $\mathcal{S}_i(p)$ be the set of siblings groups of $H_i(p)$ at tier-i. Finally, let $\mathcal{S}(p)$ be the union of the sibling sets $\mathcal{S}_1(p), \cdots, \mathcal{S}_N(p)$.

Peer p should know the IP address of at least one peer in each group $G \in \mathcal{S}(p)$, as well as the IP addresses of all the other peers in p's inner group. We refer to the union of these two sets of IP addresses as peer p's *routing table*, which constitutes peer p's state. The total number of IP addresses in the peer's routing table in tier-L is $|H_L(p)| + |\mathcal{S}(p)|$. In [19] we describe how a new peer can join an existing TOPLUS network. Basically, a peer needs to know a *delegate* in each group in its routing table. An initial routing table is obtained from another peer in the same group, replacing subsequently the obtained delegates with equivalent ones (i.e., other peers from their respective groups).

3.2 XOR Metric

Let I' be the set of all binary strings of length s, with $s \geq 32$ and fixed. For a given key $k' \in I'$, let k be the 32-bit suffix of k', i.e. $k \in I$ and $k = k'_{31}k'_{30}\ldots k'_1k'_0$). Throughout the discussion below, we will refer to k instead to the original key k'.

The XOR metric defines the distance between two IDs j and k as

$$d(j,k) = \sum_{\nu=0}^{31} |j_\nu - k_\nu| \cdot 2^\nu \tag{1}$$

The metric $d(j,k)$ has the following properties, for IDs i,j and k:
If $d(i,k) = d(j,k)$ for any k, then $i = j$.
Let $p(j,k)$ be the number of bits in the common prefix of j and k.
If $p(j,k) = m$, then $d(j,k) \leq 2^{32-m} - 1$.
If $d(i,k) \leq d(j,k)$, then $p(i,k) \geq p(j,k)$.

The XOR metric $d(j,k)$ is a refinement of the longest-matching prefix scheme. If j is the unique longest-matching prefix for k, then j is the ID closest to k in terms of $d(j,k)$. Furthermore, if two peers have the same longest-matching prefix with respect to k, the XOR metric will break the tie and choose the peer that minimizes $d(j,k)$ as the "responsible peer" for key k.

4 MULTI+: Multicast on TOPLUS

4.1 A Multicast Tree

A simple Multicast tree is shown in Figure 2. Let S be the source of Multicast group m. Peer p is receiving the flow from peer q. We say that q is the parent of p in the Multicast tree. Conversely, we say that p is a child of q. Peer p is at level-2 of the Multicast tree and q at level-1. It is important to note that, in principle, the *level* of a peer in the Multicast tree has nothing to do with the *tier* the peer belongs to in the TOPLUS hierarchy.

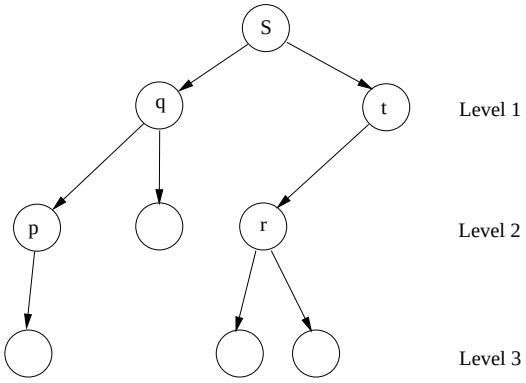


Fig. 2. A simple Multicast tree.

We want to construct Multicast trees where each peer (i) is close to its parent in terms of network delay and (ii) joins the Multicast tree at the lowest possible level

in the tree (close to the source) as possible. For this purpose, each peer attempts at join time to minimize the length (network delay) of the last hop and the number of hops from the source. In the example of Figure 2, the fact that p is a child of q and not of t, is because p is closer to q than to t. By trying to minimize the network delay between peers at join time, we also avoid rearranging peers of the Multicast tree, except when a peer fails or disconnects.

4.2 Building Low End-to-End Latency Multicast Trees

We use the TOPLUS network and its look-up algorithm to build the Multicast trees. Consider a Multicast IP address m and the corresponding key which, abusing the notation, we also denote by m. Each tier-i group G_i is defined by an IP network prefix a/b where a is an IP address and b is the length of the prefix in bits. Let m_i be the key resulting from the substitution of the leftmost b bits of m by those of a, i.e. $m_i = a_{31} \ldots a_{31-b+1} m_{31-b} \ldots m_0$ (see function `key_prefix` in Figure 6).

The inner group that contains the peer responsible for m_i (obtained with a TOPLUS look-up) is referred to as the *responsible inner group*, or RIG, for m in G_i (note that this RIG is contained in G_i). Hereafter, we assume a single Multicast group m, and for that group and a given peer p we denote the RIG in $H_i(p) \in$ tier-i simply as RIG-i of p. This RIG – or, more precisely, the set of peers in this RIG – is the rendezvous point for all peers in $H_i(p)$. The deeper a RIG-i is in the TOPLUS hierarchy, the fewer peers can potentially use that RIG as a rendezvous point.

In the simple 3-tier example of Figure 3, we have labeled the RIGs for a given Multicast group (peers in gray are members of the Multicast group), where all inner groups are at tier-3. The RIG-i of a peer can be found following the arrows. The arrows represent the process of asking the RIGs for a parent in the Multicast tree. For example, p and q share the same RIG-1 because they are in the same tier-1 group. t's inner group is its RIG-1, but t would first contact a peer x (white) in its RIG-2 to ask for a parent. Note that this last peer is not in the Multicast tree (Figure 4). Also, note that all peers are in tier-3 groups, but in different levels of the Multicast tree (Figure 2).

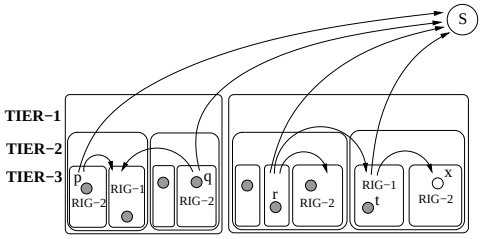


Fig. 3. The RIGs in a sample TOPLUS network.

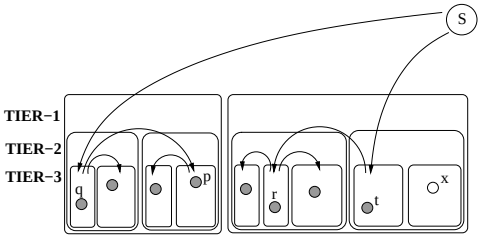


Fig. 4. Sample Multicast tree.

Assume a peer p in tier-$(i+1)$ (i.e., a peer whose inner group is at tier-$(i+1)$ of the TOPLUS tree) wants to join a Multicast tree with Multicast IP address m, which we call group m. The algorithm can be followed through the pseudo-code in Figures 5 and 6.

(1) The peer p broadcasts a query to join group m inside its inner group (recall that in TOPLUS a peer knows all the members of its inner group). If there is a peer q already part of group m, p can connect to q to receive the data (Figure 5, line 3).

(2) If there is no such peer q in the inner-most group, p must look for its RIG-i. A look-up of m_i inside p's tier-i group (thus among p's sibling groups at tier-$(i+1)$) locates the peer p_i in RIG-i responsible for m. p contacts peer p_i (Figure 5, line 9), and asks for a peer in Multicast group m (line 10). If peer p_i knows about a peer q that is part of m, it sends the ID of q to p, and p connects to q. Note that q is not necessarily a member of the RIG-i inner group. In any case p_i adds p to the list of peers listening to m, and shares this information with all peers in RIG-i. If p'' does not exist, p proceeds similarly for RIG-$(i-1)$: p looks up m_{i-1} inside p's tier-$(i-1)$ group (i.e., among p's sibling groups at tier i). This process is repeated until a peer receiving m is found, or RIG-1 is reached. In the latter case, if there is still no peer listening to m, peer p must connect directly to the source of the Multicast group.

```
function find_parent(m)
Require:
    m = Multicast address
    p = peer looking for parent
 1: i ← p.tier
 2: q ← NULL
    /* first try to find a connected peer q in inner group */
 3: q ← p.find_in_inner_group(m)
 4: while q = NULL do
 5:     if i = 1 then
 6:         q ← Multicast_source
 7:     else
 8:         m_i ← p.key_prefix(m,i, p)
 9:         p_i ← p.look_up(m_i)
            /* p_i is responsible for m_i and belongs to RIG-i */
10:         q ← p_i.ask_for_parent(m,p)
11:         i ← i − 1
12:     end if
13: end while
14: return q
```

Fig. 5. Find a parent in the Multicast tree.

One can see that the search for a peer to connect to is done bottom up.

function `key_prefix(m,i,p)`
Require:
m = Multicast address
i = tier of p's current look-up
p = peer looking for parent
1: $pref \leftarrow p$.prefix_of_tier(i)
2: $l \leftarrow pref$.length
3: **return** ((m AND $2^{32-l} - 1$) OR $pref$)

(a) Obtaining the modified key m_i that leads to RIG-i.

function `ask_for_parent(m,p)`
Require:
m = Multicast address
p = peer looking for parent
/* $peers$ contains a FIFO $peers\{m\}$ for each Multicast address m */
1: $q \leftarrow NULL$
2: **if** $peers\{m\} \neq \emptyset$ **then**
3: $q \leftarrow$ peers$\{m\}$.head
4: **end if**
5: peers$\{m\}$.tail $\leftarrow p$
6: **return** q

(b) Asking a peer in a RIG for a parent.

Fig. 6. Auxiliary functions used in `find_parent`.

Property 1 *When a peer p in tier $i+1$ joins the Multicast tree, by construction, from all the sets $H_{i+1}(p), H_i(p), \cdots, H_1(p)$ that contain p, p connects to a peer $q \in H_k(p)$ where $k = \max\{l = 1, \ldots, i+1\} : \exists r \in H_l(p)$ and r is a peer already connected to the Multicast tree. That is, p connects to a peer in the deepest tier group that contains both p and a peer already connected to the Multicast tree.*

This assures that a new peer connects to the closest available peer in the network. Notice that even in the case of failure of a peer in a RIG-i, the information is replicated in all other peers in the RIG-i. If a whole RIG-i group fails, while MULTI+ will be affected, the look-up process can continue in RIG-$(i-1)$. This property makes MULTI+ a resilient system.

Property 2 *Using Multicast over TOPLUS, the total number of flows in and out of a group defined by an IP network prefix is bounded by a constant.*

Due to lack of space, we do not further develop this important aspect of MULTI+. We refer the interested reader to the PhD thesis [21]. However, in the experiments below we will notice the low number of flows per network prefix.

4.3 Membership Management in MULTI+

Each peer p knows its parent q in the Multicast tree, because there is a direct connection between them. Because p knows the RIG where it got its parent's address, if p's parent q at level i of the Multicast tree fails or disconnects, p directly goes to the same RIG and asks for a new parent. If there is none, p becomes the new tree node at level i, replacing q and p must find a parent in level $i-1$ of the Multicast tree, through a join process starting at said RIG. If p had any siblings that were also children of its former parent q, those siblings will find p as the new parent when

they proceed like p. If more than one peer concurrently tries to become the new node at level i, peers in the RIG must consensually decide on one of them. It is not critical if a set of peers sharing a parent q are divided in two subsets with different parents upon q's depart.

Join and leave is a frequent process in a P2P network, but we expect the churn to be rather low due to the fact that in a Multicast tree, all peers seek the same content concurrently, throughout the duration of the session.

4.4 Parent Selection Algorithms

From the ideas exposed before, we retain two main parent selection algorithms for testing the construction of Multicast trees.

- FIFO, where a peer p joins the Multicast tree at the first parent found that has a free connection. When a peer gets to a RIG to find a parent, the RIG answers with a list of already connected peers. This list is ordered by arrival time of the connected peers to the RIG. Obviously, the first peer to arrive connects closer (in hops) to the source than the following ones. The arriving peer p tests each possible peer in the list starting with the first one until it finds one that accepts a connection.
- Proximity-aware, where, *given the first parent in the list has all connections occupied*, a peer connects to the closest parent in the list that still accepts one extra connection.

Note that we do not always check if we are connecting to the closest parent in the list. The idea behind this is that, while we implicitly trust MULTI+ to find a close parent, we prefer to connect to a peer higher in the Multicast tree (fewer hops from the source) than to optimize the last hop delay. If MULTI+ works as expected, the difference between these two policies should not be significant, because the topology-awareness is already embedded in the protocol through TOPLUS.

5 MULTI+ Performance: End-to-End Latency and Bandwidth Utilization

Obviously, the $O(n^2)$ cost of actively measuring the full inter-host distance matrix for n peers limits the size of the peer sets we can use [21]. P2P systems must be designed to be potentially very large, and experiments should reflect this property by using significant peer populations. Methods like [22] map hosts into a M-dimensional coordinate space. The main advantage is that given a list of n hosts, the coordinates for all of them can be actively measured in $O(Mn)$ time (the distances of the hosts to a set of M *landmark* hosts, with $M \ll n$).

5.1 Simulating Internet-Scale Deployments: Tang-Crovella Coordinates

CAIDA [23] offers to researchers a set of network distance measurements from so-called *Skitter* hosts to a large number of destinations. `Skitter` is a traffic measurement application developed by CAIDA. In a recent paper [22], the authors have used these and other data to obtain a M-dimensional coordinate space representing the Internet. A host location is denoted by a point in the coordinate space, and the latency between two hosts can be calculated as the distance between their corresponding points. The authors of [22] have kindly provided us with the coordinates of 196 297 IP addresses for our study. Hereafter we call this space the TC (from Tang and Crovella) coordinate space. We calculate distances using a Euclidean metric, defined $D(x_i, x_k) = \sqrt{\sum_{k=1,\ldots,M}(x_{ik} - x_{jk})^2}$, for any two hosts identified by their M-coordinate vectors x_i and x_j.

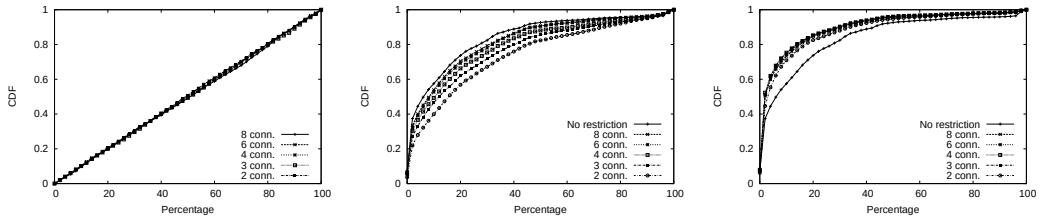
5.2 Studying a Multicast Tree with 5 000 Peers

In this experiment, we test the characteristics of Multicast trees built with MULTI+ for a set of 5 000 peers. We use the TC coordinate space to measure the distance between every pair of hosts. In order to make the experiment as realistic as possible, we use a TOPLUS tree with routing tables of reduced size, obtained from the grouping of small and medium-sized tier-1 groups into virtual groups, and this process introduces a distortion in the topological fidelity of the resulting tree [19]. The 5 000 peers are organized into a TOPLUS tree with 59 tier-1 groups, 2 562 inner-groups, and up to 4 tiers. We evaluate the two different parent selection policies described before: FIFO and proximity-aware. We also compare these two approaches with random parent selection. In all cases we test MULTI+ for different values for the maximum number of connections a peer can accept, which is also referred to as *outdegree*. We consider unlimited outdegree and outdegree between 2 and 8. We use the following performance metrics:

- The percentage of the peers in the total system, after the full Multicast tree is built, closer to a peer than this peer's parent. Those figures *exclude* the peers directly connected to the source (Figure 7).
- The level peers occupy in the Multicast tree. The more levels a Multicast tree has, the more delay we incur along the transmission path and the higher the probability that the transmission suffers from losses due to peer failure (Figure 8).
- The latency from the root of the Multicast tree to each peer (Figure 9).

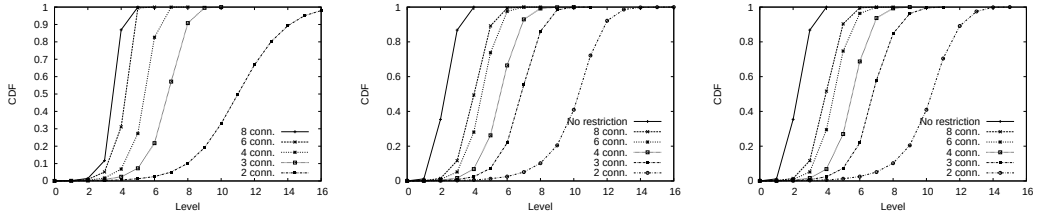
From the results in Figures 7 to 10 we can draw a number of conclusions:

Individual peers do not need to support a large outdegree to benefit from MULTI+ properties: an outdegree of 3 is sufficient, the improvement for higher outdegree is marginal.



(a) Random parent selection. (b) FIFO parent selection. (c) Proximity-aware parent selection.

Fig. 7. Percentage of peers in the whole system closer than the peer actually used (for those not connected to the source.)

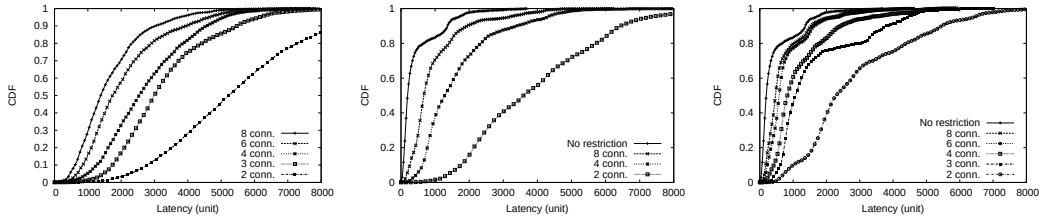


(a) Random parent selection. (b) FIFO parent selection. (c) Proximity-aware parent selection.

Fig. 8. Level of peers in the Multicast tree.

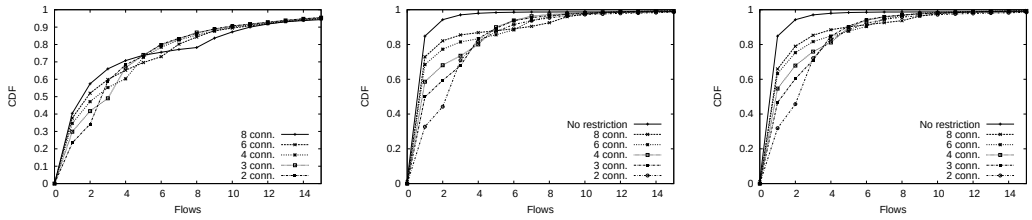
The proximity-aware policy performs better than FIFO in terms of end-to-end latency (Figure 9) and connecting to the closest parent (Figure 7). However, with respect to the number of flows per group (Figure 10) and level distribution in the Multicast tree (Figure 8), they are very similar. That is because both trees follow the TOPLUS structure, but the proximity-aware policy makes better decisions when the optimal parent peer can accept no more child peers.

The random parent selection policy organizes the tree in fewer levels than the other two policies (Figure 8(a)), because connections are not constrained to follow the TOPLUS structure. However those connections are not optimized, and the resulting end-to-end delay performance that is considerably worse (see Figure 9).



(a) Random parent selection. (b) FIFO parent selection. (c) Proximity-aware parent selection.

Fig. 9. Latency from root to leaf (in TC coordinate units) in the Multicast tree.



(a) Random parent selection. (b) FIFO parent selection. (c) Proximity-aware parent selection.

Fig. 10. Number of flows through group interface.

6 MULTI+ Resilience under Peer Failure

We have already seen that MULTI+ can build static Multicast trees with low end-to-end latency and low hop count. However, a P2P Multicast service is provided by the users of the service, and the effect of peers failures on the Multicast tree must be considered. We use the term "failure" as a generic term for a peer leaving the Multicast session. We do not distinguish between peers that leave gracefully by announcing their departure and peers leaving without announcement, since the impact on the resulting Multicast tree will be the same. We evaluate how the properties of the MULTI+ tree are affected by the failure of 5%, 10%, 20% and 40% of the peers. A 5-10% failure rate may correspond to failures due to a final user leaving the Multicast group or the end system (Home PC) crashing. A 40% failure rate may correspond to a massive attack against the P2P network (denial of service or virus).

When a peer in the Multicast tree fails, its children will notice the failure rapidly since the flow of data will be interrupted. The children then contact their RIG to get a new parent and also inform the RIG about the failed parent. We choose this approach since it avoids rebuilding the whole subtree rooted at the failed peer.

In general, a peer in MULTI+ seeks a parent that is (1) as close as possible to the source, in terms of overlay hops, and (2) as close as possible to that peer, in terms of latency. A peer p connected to a parent q that has failed will try to find a parent q' close to p that is at least as close to the source in terms of hops in the Multicast tree as q is, and can still accept one more child.

We need a metric to evaluate the "quality" of the choice of the new parent q' with respect to the best possible choice. For this purpose, we introduce the *proximity ratio* R. Let $J(q')$ denote the set of all peers with at least one available connection that are at the same level as q' or closer to the source (in hops) in the Multicast tree (with $q' \in J(q')$). Let $l(i,j)$ measure the network latency between peers i and j, the *proximity ratio* R is calculated as follows :

$$R = \frac{l(p, q')}{\min_{j \in J(q')}(l(p, j))} - 1$$

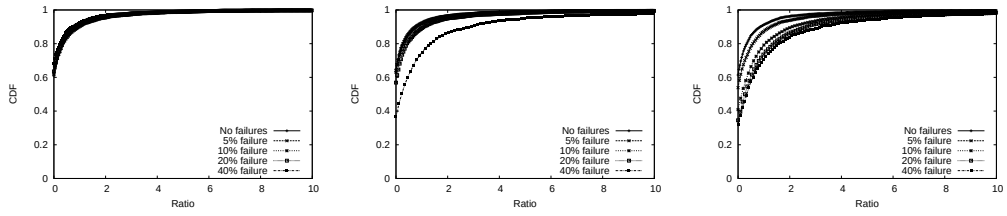
Thus, if we find no better parent for p than q' we have $R = 0$.

In Figures 11 to 13 we plot the Cumulative Distribution Function of:

- The proximity ratio R (Figure 11).
- Distribution of peers across the levels of the Multicast tree (Figure 12).
- Latency from root to leaf in the Multicast tree (Figure 13).

We plot these metrics for an unlimited number of connections per peer, and for 8 and 4 connections respectively. While not realistic, the unrestricted number of connections gives for most metrics of the MULTI+ tree a performance upper bound. The algorithm used for parent selection in all experiments is "proximity-aware", as it yielded the best results in the absence of peer failure.

We first observe that when there no failures, occur the outdegree of a peer has no impact in the proximity metric: More than 60% of the peers connect to the most optimal parent, that is, the closest peer higher in the Multicast tree (closer to the source). The proximity of child to parent connection deteriorates slightly in the presence of failures (Figure 11). This is good, but not surprising, since orphan children use the same TOPLUS algorithm to look for a new parent. The choice of a parent becomes less optimal, for lower values for the outdegree.



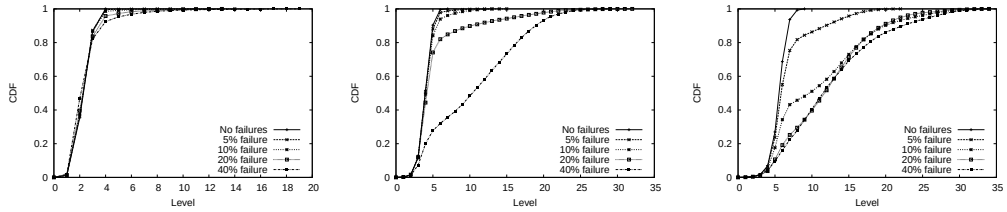
(a) Unlimited connections. (b) 8 connections. (c) 4 connections.

Fig. 11. Proximity ratio R between optimal parent and current one.

In Figure 12 we see that the low height of the Multicast tree is largely preserved for a low failure rates. However, large-scale failures will push peers down the tree, resulting in trees with up to 34 levels, as we can see for the case of maximum of 4 connections and 40% peer failure rate in Figure 12(c). The lower the outdegree per peer, the more important the effect of peer failures: Compare the 10% failure rate plot for an 8 connections limit in Figure 12(b) and the same failure rate for a limit of 4 connection per peer in Figure 12(c). The unrestricted MULTI+ tree is unaffected, because it is always possible for all the children of a peer to connect to their grandparent in the Multicast tree.

This phenomenon is easily explained: The higher the failure rate, the higher the probability of failure of a peer close to the Multicast source. The closer to the

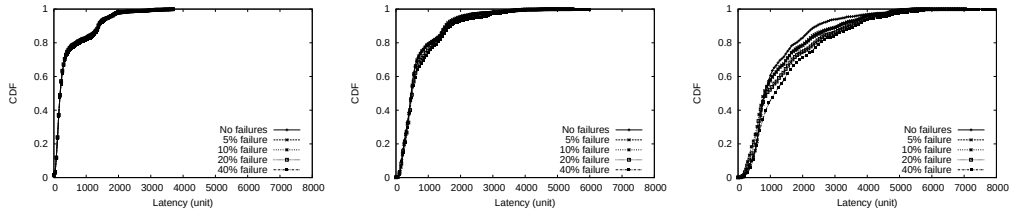
source a peer is in the tree, the more difficult it is to find an available connection at the same level. In the worst case, a peer that is directly connected to the source, when failing, will force its direct children (except the one that takes its place) to connect to *leaf* peers, low in the tree. Then the leaf peers under the direct children of the failed parent may find themselves twice as far from the source! This explains the degradation seen in Figure 12(b) and 12(c).



(a) Unlimited connections. (b) 8 connections. (c) 4 connections.

Fig. 12. Level of peers in the Multicast tree.

While the height of the tree increases quite a bit for high failure rates, the end-to-end latency is not much affected (Figure 13). However, the lower the outdegree per peer, the more difficult it will be to find a suitable new parent for the orphan children of the failed peer.



(a) Unlimited connections. (b) 8 connections. (c) 4 connections.

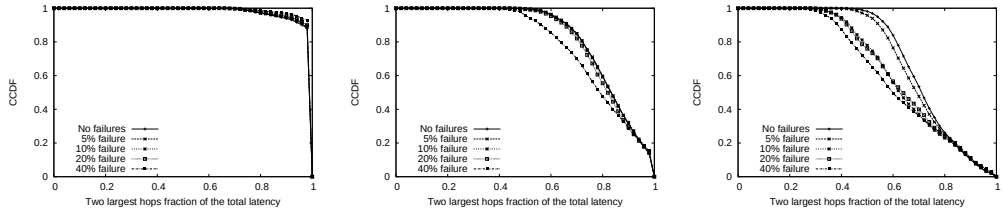
Fig. 13. Latency from root to leaf (in TC coordinate units) in the Multicast tree.

Most of the connections along the root to leaf path over the Multicast tree are done between close peers. It can be argued that even if the end-to-end latency is not affected a lot by peer failures, fewer peers in the system should lead to a shorter average latency from the root to each destination. However, as we have studied before [21], if we compare the results for sets of 1 000 and 5 000 peers, MULTI+ works asymptotically better with an increasing number of participants.Indeed, MULTI+ relies on the properties of TOPLUS in order to determine how close two peers are. The more peers in the system, the higher the probability that two or more peers are in the same lower-tier group of the TOPLUS tree, where the locality is very high [19]. When these peers connect to each other, the total end-to-end latency increases by very little. On the other hand, if a only a few peers are present, most connections are made between tier-1 groups of the TOPLUS tree, adding with each hop a considerable latency. The system is always in this last situation when it starts building up. But after that, when more and more peers join, neither increasing the number of peers increases a lot the average end-to-end latency, nor reducing the

number of peers reduces it significantly.

In Figure 14 we plot the CCDF (Complementary Cumulative Distribution Function) of the portion of the root-to-leaf latency represented by the *two* longest hops in the end-to-end path. Notice that our experiments do not consider the first hop coming directly from the source. We see that the more connections we allow, the larger fraction of the total latency is due to the largest two hops. This agrees with the fact that restricting the outdegree increases the number of levels in the tree (Figure 12). For an unrestricted outdegree, the two largest hops are responsible for at least 70% of the total end-to-end latency (Figure 14(a)), which is not very surprising since most peers are in level 2 or 3 of the Multicast tree (Figure 12(a)). However, the degeneration due to failed peers, increasing the number of levels, does not change the latency share due to the two largest hops. This means that the additional levels are not adding a lot of latency since the new connections are being made to close-by peers, thanks to the topology-awareness MULTI+ has inherited from TOPLUS. Similar phenomena can be observed when we restrict the outdegree per peer. As an extreme case when 40% of the peers fail, for a maximum outdegree of 4, the number of levels can get as high as 34 (Figure 12(c)). Yet (Figure 14(c)), the two largest hops for any peer make still for more than 30% of the end-to-end latency. Any additional hops to the 8 hops the MULTI+ tree features when no failures occur, must thus be between close peers. The same effect is observed in the topology-aware TOPLUS routing, where a first large hop leads to a destination group, and each successive hop takes a query closer and closer to the destination peer.

We already knew that MULTI+ is able to make topology-aware tree constructions *inside* TOPLUS groups,since the network latency among peers inside a group is typically smaller than among peers in different groups. However, TOPLUS alone cannot assume anything about peers in different sibling groups. In that case, MULTI+ proceeds connecting to the closest *available* parent using the proximity-aware parent selection.



(a) Unlimited connections. (b) 8 connections. (c) 4 connections.

Fig. 14. Fraction of the end-to-end latency due to the two largest hops.

7 End-to-end latency: A Comparison of Scribe and MULTI+

Finally, we perform a comparative study between MULTI+ and one of the best known P2P application-level Multicast systems, Scribe [6]. Scribe is a P2P application that runs on top of a structured P2P network called Pastry [9]. Like TOPLUS, Pastry aims to create a topology-aware overlay. Each Pastry peer maintains a number of links to other peers called neighbors, which require permanent checking to assure that they are indeed the closest peers in the network.

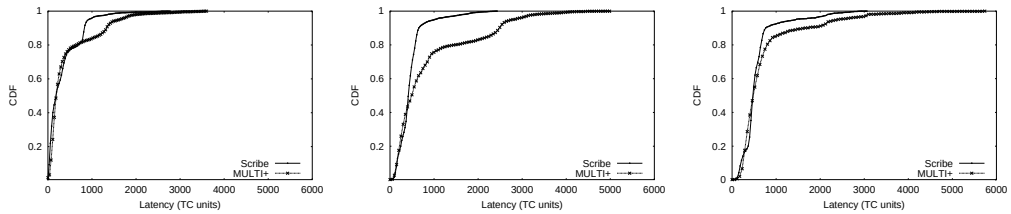
In Scribe, a peer p that wants to join a Multicast group issues a query to look up the key that identifies the Multicast group. When the query reaches a peer q that is already part of that Multicast group, the new peer p connects as a child of q in the Multicast tree. If the outdegree is not restricted, some peers will end up having a disproportionate number of children. This is normal, because the routing paths of the different join requests have little chance of converging, except at peers whose ID is close to the key identifying the Multicast group. In MULTI+, the number of connections to each peer is always bounded by a constant: In the worst case, a peer parent to the peers in all its sibling groups at each tier of the TOPLUS tree inside the peer's tier-1 group (see Figure 1). The number of these groups is limited by the size of the TOPLUS routing table. By Property 1 (see above), it can be shown [21] that only one inbound connection is needed in each group in order to build the Multicast tree, and thus the number of connections to a peer in the worst case is bounded by a constant.

On the other hand, peers in Scribe must limit the outdegree of the Multicast tree, which is done through a process called *bottleneck remover* [6]. When a parent q can only accept a limited number of connections and a new peer p finds all of them occupied, q suggest as a new parent for p its child c, whose added latency toward q and p is minimal. In this sense, Scribe tries to build minimal end-to-end delay Multicast trees.

The peers in the Pastry overlay network are organized in a ring. As a result, loops can be formed when building the Multicast trees with Scribe. In this case, two peers attempting to connect to the same Multicast group can find themselves connected to each other in a loop (through other intermediate peers). In order to avoid the formation of those loops, Scribe introduces a per-group organizer mechanism that checks for loops. Notice that loops cannot happen in MULTI+ by construction, because of the nature of TOPLUS (which is not a cyclic overlay).

In order to evaluate Scribe, we downloaded the FreePastry [24] source code. FreePastry (as of October 2004) contains, among other things, the Java code for a full implementation of Pastry and a basic implementation of Scribe in FreePastry. We have modified the source code so that Pastry can read a list of nodes with their TC coordinates and use these coordinates to calculate latencies among peers. This way,

both MULTI+ and Scribe will work with the same data and the results obtained will be comparable. We simulate a single Multicast tree on a 5 000 peer population, randomly selected from the TC address space. However, the current implementation of Scribe lacks the mechanism that avoids loops in the Multicast trees. As a result, the trees built by Scribe do not contain all the 5 000 peers. To compare Scribe and MULTI+, we first build the tree under Scribe with as many of the 5 000 peers as possible. We then take the peers included in the Multicast tree formed by Scribe and use them as the peer population for the MULTI+.



(a) Unlimited connections (5 000 peers). (b) 8 connections (3 651 peers). (c) 4 connections (2 261 peers).

Fig. 15. Root-to-leaf latency in Scribe and MULTI+.

The metric that interests us the most is the distribution of the end-to-end latency obtained with both approaches. The results are shown in Figure 15, for trees with an unlimited outdegree, or an outdegree of 8 and 4 respectively.

- With an unlimited outdegree (Figure 15(a)), both approaches perform in a similar fashion for most of the peers except for about 20% of the peers experience lower end-to-end latency using Scribe than MULTI+. Please remember that there is no limit in Scribe for the number of connections a peer can have. Thus, shorter delays can be obtained by connecting many children to the same parent, even if this parent may not be able to sustain the resulting load. For MULTI+, however, there is an implicit bound on the outdegree of a peer.
- If we limit the outdegree to 8 children per parent (Figure 15(b)), Scribe succeeds building a tree with 3 651 peers out of the total 5 000. The resulting tree of Scribe offers superior performance for about half of the peers.
- Finally, if the outdegree is limited to 4, Scribe can build a tree with only 2 261 peers. MULTI+ (Figure 15(c)) achieves a performance similar to that of Scribe for most of the peers. However, the worst case end-to-end latency perceived by any peer is much higher in MULTI+ than in Scribe.

For the experiments we conducted, Scribe provides for a subset of all peers a lower end-to-end latency than MULTI+. However, for most peers MULTI+ comes close, in particular in case of low outdegrees. The improved performance of Scribe on top of Pastry has an additional cost. Pastry obtains topology information through constant measurement and search for close-by neighbors, while TOPLUS does not need any measurement in order to provide a sense of topology-awareness to the applications on top. Also, MULTI+ trees are created as the peers arrive and no

additional optimizations are made later to adapt the trees. We have considered in our experiments only the peers that Scribe was able to manage without creating loops; introducing the organizer in Scribe to undo loops will introduce extra complexity. MULTI+ on the other hand could have built trees containing all 5 000 peer without any problem. In conclusion, we can say that MULTI+ builds Multicast trees with slightly higher end-to-end delay for a subset of the peers than Scribe, but introduces no traffic for measurements and does not need to check for loops as does Scribe.

8 Conclusion

We have presented MULTI+, an application-level Multicast service for P2P systems. MULTI+ relies on TOPLUS in order to determine the parent in the Multicast tree. MULTI+ is able to create topology-aware Multicast trees without introducing any extra traffic for active measurement. Admittedly, out-of-band information regarding the TOPLUS routing tables must be calculated offline (a simple process) and downloaded. The proximity-aware parent selection in MULTI+ further improves the end-to-end latency. MULTI+ also decreases the number of redundant flows that must traverse a given network (even when only few connections per peer are possible), which allows for better bandwidth utilization. MULTI+ remains robust under massive peer failure, and it is able to maintain low end-to-end delay even in case of a limited outdegree per peer.

Luis Garcés-Erice (lge@ieee.org) obtained his MSc in Telecommunication Engineering (EE+CS) in the Public University of Navarra (Spain) in 2001. He obtained his PhD in 2004 from Télécom Paris University (France) working at Institut EURECOM. Currently he is at IBM Research Zürich (Switzerland) as a Post-Doc. His research interests are mainly in P2P and distributed systems, network protocols and more recently middleware and sensor networks.

Ernst Biersack (erbi@eurecom.fr) received his M.S. and Ph.D. degrees in Computer Science from the Technische Universität München, Munich, Germany and his Habilitation à diriger la Recherche from the University of Nice Sophia Antipolis, France.
Since March 1992 he has been a Professor in Telecommunications at Institut Eurecom, in Sophia Antipolis, France. For his work on synchronization in video servers he received in 1996 (together with W. Geyer) the Outstanding Paper Award of the IEEE Conference on Multimedia Computing & Systems. For his work on reliable Multicast he received (together with J. Nonnenmacher and D. Towsley) the 1999 W. R. Bennet Price of the IEEE for the best original paper published 1998 in the ACM/IEEE Transactions on Networking.

References

[1] S. E. Deering, D. R. Cheriton, Multicast routing in datagram internetworks and extended LANs, ACM Transactions on Computer Systems 8 (2) (1990) 85–110

[2] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable application layer Multicast, in: Proceedings of SIGCOMM, ACM Press, Pittsburgh, PA, USA, 2002, pp. 205–217.

[3] Y. hua Chu, S. G. Rao, S. Seshan, H. Zhang, A case for end system Multicast, IEEE Journal on Selected Areas in Communication, Special Issue on Networking Support for Multicast 20 (8) (2002) 1456–1471.

[4] S. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, J. Kubiatowicz, Bayeux: An architecture for scalable and fault-tolerant wide area data dissemination, in: Proceedings of the 11^{th} International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Vol. 2429, ACM, Port Jefferson, NY, USA, 2001, pp. 11–20.

[5] S. Ratnasamy, M. Handley, R. M. Karp, S. Shenker, Application-level Multicast using content-addressable networks, in: Proceedings of the 3^{rd} Third International COST264 Workshop on Networked Group Communication, Vol. 2233 of LNCS, Springer, 2001, pp. 14–29.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, A. Rowstron, Scribe: a large-scale and decentralized application-level Multicast infrastructure, IEEE Journal on Selected Areas in Communications 20 (8) (2003) 1489–1499.

[7] I. Stoica, R. Morris, D. Karger, M. Kaashoek, H. Balakrishnan, Chord: A scalable Peer-to-peer lookup service for Internet applications, in: Proceedings of SIGCOMM, ACM Press, San Diego, CA, USA, 2001, pp. 149–160.

[8] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of SIGCOMM, ACM, San Diego, CA, USA, 2001, pp. 161–172.

[9] A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale Peer-to-peer systems, in: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Vol. 2218 of LNCS, Springer, Heidelberg, Germany, 2001, pp. 329–350.

[10] B. Y. Zhao, J. Kubiatowicz, A. D. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing, Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley (April 2001).

[11] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, A. Wolman, An evaluation of scalable application-level Multicast built using Peer-to-peer overlays, in: Proceedings of INFOCOM, IEEE, San Francisco, USA, 2003, pp. 1510–1520.

[12] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, ALMI: An application level Multicast infrastructure, in: Proceedings of the 3^{rd} USENIX Symposium on Internet Technologies and Systems (USITS '01), San Francisco, CA, USA, 2001, pp. 49–60.

[13] P. Francis, Yoid: Extending the Multicast Internet architecture, white paper, http://www.icir.org/yoid/ (1999).

[14] M. Kwon, S. Fahmy, Topology-aware overlay networks for group communication, in: Proceedings of NOSSDAV, ACM, Miami Beach, Florida, USA, 2002, pp. 127–136.

[15] D. Kostic, A. Rodriguez, J. Albrecht, A. Vahdat, Bullet: High bandwidth data dissemination using an overlay mesh, in: Proceedings of the 19^{th} ACM Symposium on Operating Systems Principles (SOSP-19), Bolton Landing, NY, USA, 2003, pp. 282–297.

[16] P. Rodriguez, E. W. Biersack, Dynamic parallel access to replicated content in the Internet, IEEE/ACM Transactions on Networking 10 (4) (2002) 455–465.

[17] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, L. Garcés-Erice, Dissecting BitTorrent: Five months in a torrent's lifetime, in: Proceedings of Passive and Active Measurements, Vol. 3015 of LNCS, Springer, Juan-les-Pins, France, 2004, pp. 1–11.

[18] D. Tran, K. Hua, T. Do, ZIGZAG: An efficient peer-to-peer scheme for media streaming, in: Proceedings of INFOCOM, IEEE, San Francisco, USA, 2003, pp. 1283–1292.

[19] L. Garcés-Erice, K. W. Ross, E. W. Biersack, P. A. Felber, G. Urvoy-Keller, TOpology-centric Look-Up Service, in: Proceedings of COST264/ACM 5^{th} International Workshop on Networked Group Communications (NGC), Vol. 2816 of LNCS, Springer, Munich, Germany, 2003, pp. 58–69.

[20] B. Krisnamurthy, J. Wang, On network-aware clustering of Web sites, in: Proceedings of SIGCOMM, ACM, Stockholm, Sweden, 2000, pp. 97–110.

[21] L. Garcés-Erice, A hierarchical P2P network: Design and applications, Ph.D. thesis, Télécom Paris, Institut EURECOM, Sophia-Antipolis, France (December 2004).

[22] L. Tang, M. Crovella, Virtual landmarks for the Internet, in: Proceedings of the Internet Measurement Conference (IMC), ACM, Miami Beach, Florida, USA, 2003, pp. 143–152.

[23] CAIDA, http://www.caida.org/

[24] Freepastry, http://www.cs.rice.edu/CS/Systems/Pastry/FreePastry/ (October 2004).