# MULTI+: Building Topology-Aware Overlay Multicast Trees

Luis Garcés-Erice, Ernst W. Biersack and Pascal A. Felber

Institut EURECOM
06904 Sophia Antipolis, France
{garces|erbi|felber}@eurecom.fr

**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 peer-to-peer (P2P) TOPLUS-based network. We use the characteristics of TOPLUS to design a protocol that allows for 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 more efficient bandwidth usage. We use different procedures to measure or calculate the distances among peers, in order to validate the algorithms in MULTI+.

*Keywords*: Overlay networks, Topology-aware, Multicast

## 1 Introduction

IP Multicast seems (or, at least, was designed) to be the ideal solution for content distribution over the Internet: (1) it can serve content to an unlimited number of destinations, and (2) it is bandwidth-wise economic. These two characteristics are strongly correlated. IP Multicast saves bandwidth because a single data flow can feed many recipients. The data flow is only split at routers where destinations for the data are found in more than one outgoing port. Thus $n$ clients do not need $n$ independent data flows, which allows for IP Multicast's scalability. However, IP Multicast was never widely deployed in the Internet: security reasons, its open-loop nature, made IP multicast remain as a limited use tool for other protocols in LANs. The core Internet lacks of an infrastructure with the characteristics of IP Multicast.

Lately, with the advent of broadband links like ADSL and the generalization of LANs at the workplace, the *edges* of the Internet started to increase their bandwidth. Together with the ever-cheaper and yet more powerful equipment (computational power, storage capacity), they give the millions of hosts connected to the Internet the possibility of implementing themselves services that augment at the application level the capabilities of the network: the Peer-to-Peer (P2P) systems. Various application-level multicast implementations have been proposed [2, 12, 26, 20, 6], most of which are directly implemented on top of P2P infrastructures (Chord [23], CAN [19] or Pastry [21]). The good scalability of the underlying P2P networks give these application-level multicast one of the properties of the original IP Multicast service, that of serving content to a virtually unlimited number of clients (peers). However, these P2P networks are generally conceived as an application layer system completely isolated from the underlying IP network.

Thus, the P2P multicast systems that we know may fail at the second goal of IP Multicast: a LAN hosting a number of peers in a P2P multicast tree may find its outbound link saturated by *identical* data flowing to and from its local peers, unless those peers are somehow *aware* of the fact that they are sharing the same network. This situation remains valid for higher-tier networks (e.g., ISPs): even if the capacity of links to the Internet is designed using conservative parameters, normally the resulting system does not allow for all or many hosts to sustain a continuous throughput from and to the Internet. This is a critical issue for ISPs due to P2P file-sharing applications and flat-rate commercial offers that allow a home computer to be downloading content 24 hours a day. This problem also affects application-level multicast sessions if peers do not take care of the network topology.

We have based our P2P multicast protocol on TOPLUS because of its inherent topology-awareness. We aim at building a P2P multicast network that avoids network link stress while remaining scalable. In TOPLUS, nodes that are topologically close are organized into groups. In turn, groups that are topologically close are organized into supergroups, and close supergroups into hypergroups, etc. The groups within each level of the hierarchy can be heterogeneous in size and in fan-out (i.e., number of subgroups). TOPLUS derives the groups from the network prefixes contained in BGP tables or from other sources.

Before we elaborate on a Multicast deployment in a TOPLUS network, we must state some assumptions about the system: First of all, there is a large population of peers participating in the network, which justifies the utilization of multicast in the first place. Second, the TOPLUS Multicast infrastructure may be used to transmit to many different multicast groups, and thus many multicast trees must live together without interfering each other. Third, a peer is only willing to transmit data from a multicast tree if the peer is in the multicast group (that is, a peer will not use its bandwidth just to forward data for others). However, when participating in the TOPLUS Multicast infrastructure, a peer must accept to cooperate with others in low-bandwidth maintenance tasks.

*Related Work.* Some examples of overlay networks which introduce topology-awareness in their design are SkipNet [17], Coral[8], Pastry [4], CAN [22]. Application-level Multicast has given some interesting results like the NICE project [2] or End System Multicast [12]. Among those using overlay networks, there are application-level multicast implementations like Bayeux [26], or using CAN [20] and Pastry (Scribe) [6]. There is an interesting comparative in [7]. Content distribution overlay examples are SplitStream [5] and [13]. Recently, the problem of data dissemination on adaptive overlays has been treated in [25]. Our approach differs mainly in the achievement of efficient topology-aware multicast trees with no or very little active measurement.

In the next section we present the main aspects of TOPLUS. Then we sketch MULTI+. Then we show and comment some results (more experiments are in a technical report [9]), before we conclude.

## 2 TOPLUS Overview

TOPLUS [10] 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.

Given a message containing key $k$, the P2P lookup service routes the message to the current active peer that is responsible for $k$. The message travels from source peer $p_s$, through a series of intermediate peers $p_1, p_2, \ldots, p_v$, and finally to the destination peer, $p_d$. The principal goal of TOPLUS is simple: each routing step takes the message closer to the destination. We now formally describe TOPLUS in the context of IPv4.

Let $I$ be the set of all 32-bit IP addresses.[1] 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 prefix networks from BGP tables and/or other sources [14]. 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 require 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).

*Peer State.* Let $L$ denote the number of tiers in the tree, let $U$ be the set of all current active peers and consider a peer $p \in U$. Peer $p$ is contained in a collection of telescoping sets in $\mathcal{G}$; denote these sets 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 $\mathcal{S}_1(p), \cdots, \mathcal{S}_N(p)$.

[1] For simplicity, we assume that all IP addresses are permitted. Of course, some blocks of IP addressed are private and other blocks have not been defined. TOPLUS can be refined accordingly.

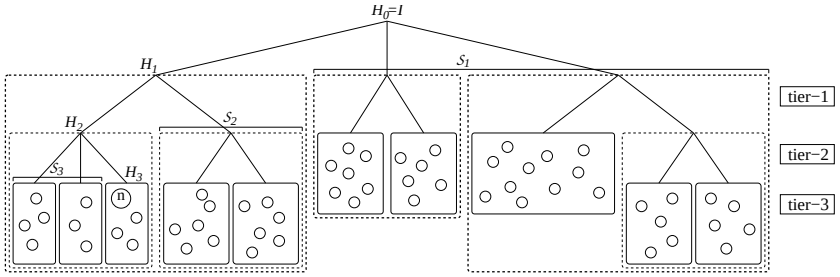



**Fig. 1.** A sample TOPLUS hierarchy (inner groups are represented by plain boxes)

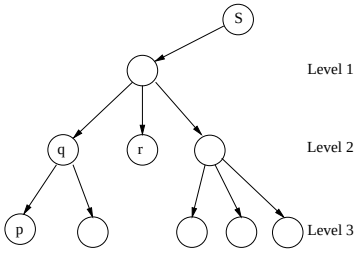



**Fig. 2.** A simple multicast tree.

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 collection 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 [10] we describe how a new peer can join an existing TOPLUS network. In the technical report [9] we propose algorithms to create a TOPLUS network using an initial permanent peer.

*XOR Metric.* Each key $k'$ is required to be an element of $I'$, where $I'$ is the set of all $s$-bit binary strings ($s \geq 32$ is fixed). A key can be drawn uniformly randomly from $I'$, or it can be biased as we shall describe later. For a given key $k' \in I'$, let $k$ be the 32-bit suffix of $k'$ (thus $k \in I$ and $k = k_{31}k_{30}\ldots k_1k_0$). Throughout the discussion below, we will refer to $k$ rather than to the original $k'$.

The XOR metric defines the distance between two ids $j$ and $k$ as $d(j,k) = \sum_{\nu=0}^{31} |j_\nu - k_\nu| \cdot 2^\nu$. The metric $d(j,k)$ has the following properties:

- 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 $p(j,k) = m$, $d(j,k) \leq 2^{32-m} - 1$.
- If $d(i,k) \leq d(j,k)$, then $p(i,k) \geq p(j,k)$.

$d(j,k)$ is a refinement of longest-prefix matching. If $j$ is the unique longest-prefix match with $k$, then $j$ is the closest to $k$ in terms of the metric. Further, if two peers share the longest matching prefix, the metric will break the tie. The peer $p^*$ that minimizes $d(k,n)$, $p \in U$ is "responsible" for key $k$.

## 3 MULTI+: Multicast on TOPLUS

*A Multicast Tree.* First we assume that all peers are connected through links providing enough bandwidth. A simple multicast tree is shown in Figure 2. Let $S$ be the source of the multicast group $m$. Of course, the source should be located in some group of a TOPLUS tree covering the whole Internet. 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 $q$, its parent in the multicast tree. Peer $p$ is in level-3 of the multicast tree and $q$ in level-2. It is important to note that, in principle, the *level* where a peer is in the multicast tree has nothing to do with the *tier* the peer belongs to in the TOPLUS tree.

In the kind of multicast trees we aim at building, each peer should be close to its parent in terms of network delay, while trying to join the multicast tree as high (close to the source) as possible. Each peer attempts at join time to minimize the number of hops from the source, and the length of the last hop. In the example of Figure 2, if $p$ is a child of $q$ and not of $r$, that is because $p$ is closer to $q$ than to $r$. Moreover, we want this condition to hold even if when $p$ joins, $q$ has not yet integrated the multicast tree: if $q$ joins later and is closer to $p$ than to $r$, then $q$ should become a child of $p$. By trying to minimize the network delay for data transmission between peers, we also avoid rearranging peers inside the multicast tree, except when a peer fails or disconnects. For the kind of applications we are targeting the content distribution network should be as stable as possible. For a discussion on this topic, see *Membership Management* below.

*Building Multicast Trees.* We use the TOPLUS network and look-up algorithm in order to build the multicast trees. Consider a multicast IP address $m$, and the corresponding key that, abusing the notation, we also denote

$m$. Each tier-$i$ group $G_i$ is defined by an IP network prefix $a_i/b$ where $a_i$ 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 first $b$ bits of $m$ by those of $a_i$. The inner group that contains the peer responsible for $m_i$ (obtained with a TOPLUS look-up) is 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 $m$, and for that $m$ and a given peer $p$ we denote the RIG in $H_i(p) \in$ tier-$i$ simply as RIG-$i$ of $p$. This RIG is a rendezvous point for all peers in $H_i(p)$. The deeper that a tier-$i$ of a RIG-$i$ is in the TOPLUS tree, the narrower the scope of the RIG as a rendezvous point (fewer peers can potentially use it).

In the simple 3-tier example of Figure 3, we have labeled the RIGs for a given multicast group (peers in grey 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.) Some inner groups can be RIGs for more than one higher-tier group. If a peer is alone in its tier-1 group, it asks the source for a parent.

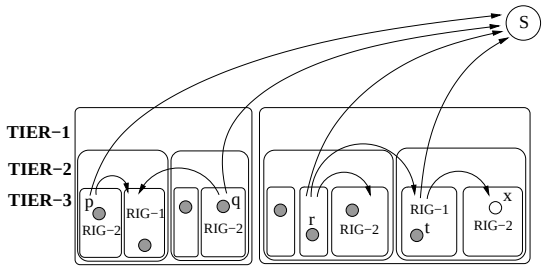



**Fig. 3.** The RIGs in a sample TOPLUS network.

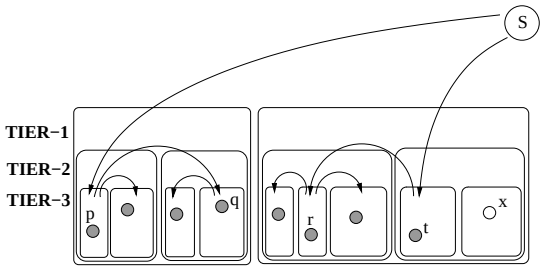



**Fig. 4.** Sample multicast tree.

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 $m$.

1. The peer $p$ broadcasts a query to join group $m$ inside its inner group. If there is a peer $p'$ already part of group $m$, $p$ connects to $p'$ to receive the data.
2. If there is not such peer $p'$, $p$ 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 RIG-$i$ responsible for $m$. $p$ contacts any peer $p_i$ in RIG-$i$, and asks for a peer in multicast group $m$. If peer $p_i$ knows about a peer $p''$ that is part of $m$, it sends the IP address of $p''$ to $p$, and $p$ connects to $p''$. Note that $p''$ is not necessarily a member of the RIG-$i$ inner group. In any case $p_i$ adds $p$ to the 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.

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

**Property 1** *When a peer $p$ in tier $i+1$ joins the multicast tree, by construction, from all the groups $H_{i+1}(p), H_i(p), \cdots, H_1(p)$ that contain $p$, $p$ connects to a peer $q \in H_k$ where $k = \max\{l = 1, \ldots, i+1\} : \exists r \in H_l$ and $r$ is a peer already connected to the multicast tree. That is, $p$ connects to a peer in the deepest tier group which 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, although MULTI+ is undeniably affected, the look up process can continue in RIG-$(i-1)$. We believe this property makes MULTI+ a resilient system. For a proof of this and other properties below, we refer to the Appendix.

*Multicast Tree Topology.* Any peer in any group can become a multicast tree node; only the RIGs are fixed, for a given multicast group and a given TOPLUS network structure. The topology of the tree is thus completely random, depending only on the peers that want to join a given multicast tree. However, although not deterministic, this tree is constrained to follow the TOPLUS structure. As it has been shown before, when a peer joins a multicast tree it will always connect to another peer that is close (according to TOPLUS proximity).

In general, we can assume that for any networked group in the Internet it is better to keep as much traffic as possible *inside* the group and avoid outbound traffic: Between ASs, routing at peering interfaces is done based mainly on economical considerations, whereas inside an AS normally other policies stand, more closely related to networking. In most LANs, the outbound link to the Internet presents an order of magnitude less bandwidth than the network itself. We show here that Multicast on TOPLUS minimizes internetwork traffic.

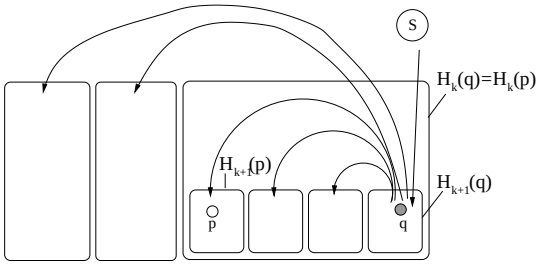



**Fig. 5.** Inbound and outbound flows at each network.

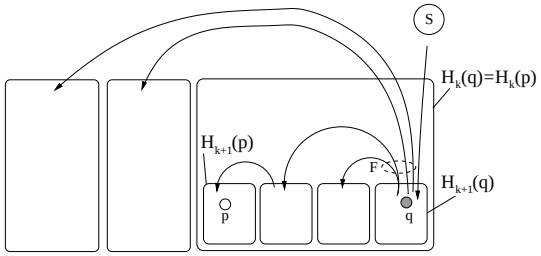



**Fig. 6.** Inbound and outbound flows at each network, with $F = 4$ connections per peer.

**Property 2** *For each group defined by an IP network prefix containing at least one peer connected to the multicast tree, there is only one inbound data flow.*

**Property 3** *For each group defined by an IP network prefix containing at least one peer connected to the multicast tree, the number of outbound data flows in the worst possible case is bounded by a constant.*

**Property 4** *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.*

Note that this is however an ideal scenario where each peer can transmit the received data flow to as many peers as necessary. In the real world, there is a bandwidth limitation. We study later a modification of the protocol that allows each peer in the multicast tree to set a maximum number of children, depending on its bandwidth. We assume that most peers have good-quality links to the Internet (at least a medium-range ADSL).

*Membership Management.* 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 $q$ in 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 $q$. Then $p$ 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 under its former parent, 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. Thus we do not consider "freeriding", i.e., peers that connect just to download some data and disconnect immediately afterwards, which introduces high dynamics in the overlay network.

*Multicast Flow Load Distribution.* Depending on the structure of the TOPLUS network, a node in a multicast tree may have many descendants. For example, in a tier-1 group with 100 tier-2 subgroups, the level 1 parent may have up to 100 descendants. The bandwidth of a single peer may not be sufficient. We now introduce two load balancing techniques to deal with that issue:

1. In the protocol we have just presented, all peers arriving to a RIG-$k$ looking for a parent in the multicast tree get assigned the same parent. This parent peer may become overloaded if a large number of children get the multicast flow from him. Assume that each peer can afford a maximum of $F$ outgoing flows. For each set of $F$ new peers arriving to a RIG-$k$, one of them should become the new parent for the next $F$ peers to come later. Notice that this policy alone would increment the number of levels in the multicast tree, and consequently the delay from root to leafs. To see this, in Figure 6 only the first $F$ groups can connect directly to $q$. The next $F$ peers must connect to one of the first $F$. The multicast tree is increased by 1 level for them (compare the hops from $p$ to $q$ in Figures 5 and 6). The problem is that the RIG-$k$ is only allowing $F$ connections to $q$, but there may be other peers in $H_{k+1}(q)$ that can accept connections. After all connections to $q$ have been allocated, $q$ can suggest as a response a connection to another peer in $H_{k+1}(q)$ receiving the data flow. The height of the tree does not increase. Then, only if there are no available peers in $H_{k+1}(q)$, $q$ can suggest a peer in one of the sibling groups of $H_{k+1}(q)$ in tier $k+1$. There is another constrain though: independently of the peers willing to serve content in $H_{k+1}(q)$, the network outgoing link may not have enough bandwidth, and another sibling group may need to be used.
2. At RIG-$k$, we assign to a new peer a parent that depends on the RIG-$(k+1)$ of origin of the new peer. Taking Figure 6 as an example, peer $r$ whose RIG-$(k+1)$ is not contained in $H_k(q)$ should have priority to connect directly to $q$. The reason of this is to allow peers from further groups to connect to a parent through fewer hops. Peers closer to $q$, that is, those contained in $H_k(q)$, can afford more hops when connecting to $q$ through other peers in $H_k(q)$, because the delay is smaller. Thus delays become more balanced.

The properties presented previously are still valid with these policies. To understand why, consider that the number of inbound flows does not change and that the property for the outbound flows is based on the worst possible case: these policies just move a part of the outbound flows from the most loaded group to its siblings. Moreover, note that this only happens in the case where the outbound link of the network of the worst case group cannot sustain the required flow of data. Thus we still have a constant bound in the number of flows coming in and out of a group.

*Limited Source Capabilities.* We have considered so far a source capable of serving content to at least one peer in each tier-1 group in the TOPLUS tree. For compact TOPLUS trees, this sums up to 256 potential flows. Although 256 flows is perfectly affordable to commercial or institutional multicast (think of Radio stations on the web today), it does not allow end users in the Internet to serve content which may potentially be of interest to many.

The problem is that our method to find a nearby existing peer is not valid between tier-1 groups. In the absence of a peer to connect to, we connect by default to the source. We propose here another method: the source accepts a limited number of direct connections. Once the limit is reached, new incoming connections receive a list of peers currently receiving the flow. The peers can then connect to the closest non-overloaded in the list (according to some easily measurable metric like RTT, or calculating their distance in a pre-calculated coordinate space, as we shall see below.) This process can equally apply to normal peers, when the number of connections is limited: a peer refusing a new connection must give a list of peers already connected to him. To detect uncooperative or malicious behavior we can verify that those peers are indeed connected to the refusing peer. This process is repeated until a valid peer is found. Note that collusion is not possible without introducing loops in the tree.

*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 joins the multicast tree at the first parent found with 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 to the RIG. Obviously, the first to arrive connects closer (in hops) to the source. The arriving peer tests each possible peer in the list starting with the first one until it finds one that accepts a connection.
- Proximity-aware, where, *when the first parent in the list has all connections occupied,* a peer connects to the closest parent in the list still allowing one extra connection.

Note that we do not always verify 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 correctly,

the difference between these two policies should not be excessive, because the topology-awareness is already embedded in the protocol through TOPLUS. We do not constantly rearrange the multicast tree structure with each arrival, because the longer a peer has been in the system the more likely it is to stay connected [3], and short-term delay minimization optimizations may not pay in the long run in terms of network stability.

## 4 Testing MULTI+ with Skitter Coordinates

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 [9]. P2P systems must be designed to be potentially very large, and experiments should reflect this property by using significant peer populations. Some methods ([16, 18, 15, 24]) to map hosts into a $M$-dimensional coordinate space have been developed. 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 $M \ll n$). Then, the inter-host distance matrix can be calculated off-line, and the population of our experiment is only limited by CPU power and memory (a far less tight bound).

*Skitter Coordinates.* CAIDA [1] 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 [24], the authors have used these data to obtain a multi-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 [24] have kindly provided us with the coordinates of $196,297$ IP addresses for our study. Hereafter we call this space the Skitter coordinate space. To assert the accuracy of the Skitter data we have, we measured the degree of correlation between distances calculated from this coordinate space and the real measured distances for some random samples. 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 $x_j$. We have compared those distances with the distances given by measurements with the `ping` program and `King` [11] on different sets of hosts, obtaining very encouraging results [9]. From this we conclude that we can use the Skitter coordinate space to compute approximate distances between hosts.

*5000 Peers Multicast Tree.* In this experiment we test the characteristics of Multicast trees built with MULTI+ using the Skitter 12-dimensional coordinate space and a set of 5,000 peers. We use the 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 [10]. 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+ when we do not set a limit on the maximum number of connections a peer can accept, and for a limited number of connections, from 2 to 8 per peer. This limitation also applies to the source. However, in our test we give the source a non-existent IP address, so that our measurements do not get biased by the choice of a given peer. In the test we measure the following parameters, presented here using their CDF (Cumulative Distribution Function):

- The percentage of the peers in the total system, when the full multicast tree is built, closer to one peer that 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 in the multicast tree, the more delay we incur in along the transmission path and the more the transmission becomes subject to losses due to peer failure (Figure 8).
- The latency from the root of the multicast tree to each receiving peer (Figure 9).
- The number of multicast flows that go into and out of each TOPLUS group (network) (Figure 10).

There are two points that are capital in order to properly evaluate the results of these experiments:

- Many peers (31%) are *alone* in their respective inner groups. Moreover, 65% of the peers in are in inner groups with 3 or fewer peers. This reinforces the impact of the "topology-awareness" introduced by the TOPLUS layout, because connections inside an inner group are rare, and therefore mainly made among *different* inner groups.

– Peers join the system in a completely random order. This means that close peers do not arrive in sequence. In particular, when a peer appears in an inner group where a previous peer exists, the new peer may not find an available connection in the existing peer.

From our experiments we obtain very satisfactory results. From Figures 7 to 10 we draw a number of conclusions:

– Individual peers do not need to support a large number of outgoing connections to benefit from MULTI+ properties: three connections are feasible for broadband users, and the marginal improvement of 8 connections is not very significant.
– The proximity-aware policy performs better than FIFO in terms of end-to-end latency (Figure 9) and connection to 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 takes better decisions when the optimal parent peer has no available connections.
– In Figure 7(c) we can see that having no connection restrictions makes closeness to parent less optimal than having restrictions, for the proximity-aware policy. This is normal, since when we have available connections, a peer's main goal is to connect as high in the multicast tree as possible. See in Figure 8 how peers are organized in fewer levels, and in Figure 9 how the root-to-leaf latency is better for the unrestricted connection scheme. Still, we can assert that the multicast tree is following (when possible) a topology-aware structure, because most peers connect to nearby parents.
– 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 in any aspect is considerably poorer.

We show in Table 1 the maximum number of flows going through a TOPLUS group interface. MULTI+ reduces the average and maximum number of flows *per network,* when compared to an arbitrary connection scheme and assuming each TOPLUS group (IP network prefix) represents a physical network. On average, each group counts 3 or 4 flows under MULTI+ (independently of the parent choice policy), while 7 are used with the random algorithm. The proximity-aware parent selection clearly improves both the number of flows per group (Table 1) and the end-to-end latency (Table 2). In this 5,000 peer set, even limited to two connections per peer, MULTI+ builds multicast trees with significantly better properties than a random one. Indeed, MULTI+ is conceived for large scale deployments counting many peers, and its properties become asymptotically better the more peers we introduce in the system.

**Table 1.** Maximum number of flows leaving/entering a group.

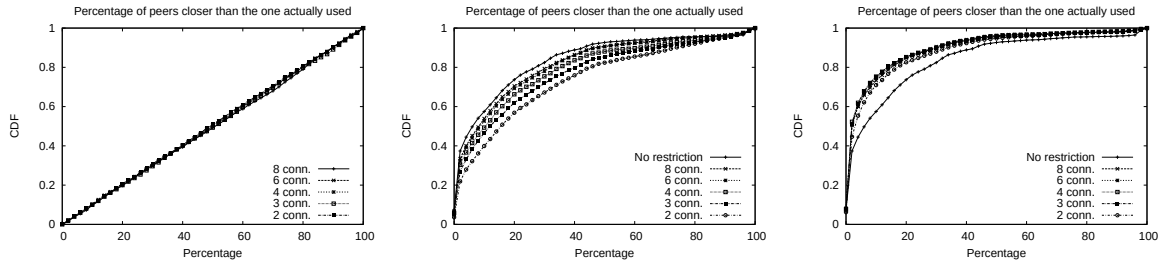
| Max. connections | 2 | 3 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| FIFO | 260 | 198 | 145 | 107 | 109 |
| Proximity-aware | 165 | 109 | 101 | 39 | 55 |
| Random | 885 | 919 | 907 | 931 | 897 |

**Table 2.** Average latency from root to leaf.

| Max. connections | 2 | 3 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| FIFO | $3,905(\pm53)$ | $2,527(\pm45)$ | $1,641(\pm32)$ | $1,135(\pm28)$ | $965(\pm24)$ |
| Proximity-aware | $2,871(\pm47)$ | $1,630(\pm35)$ | $1,205(\pm26)$ | $797(\pm19)$ | $698(\pm18)$ |
| Random | $5,339(\pm60)$ | $3,291(\pm40)$ | $2,742(\pm35)$ | $2,072(\pm33)$ | $1,646(\pm28)$ |

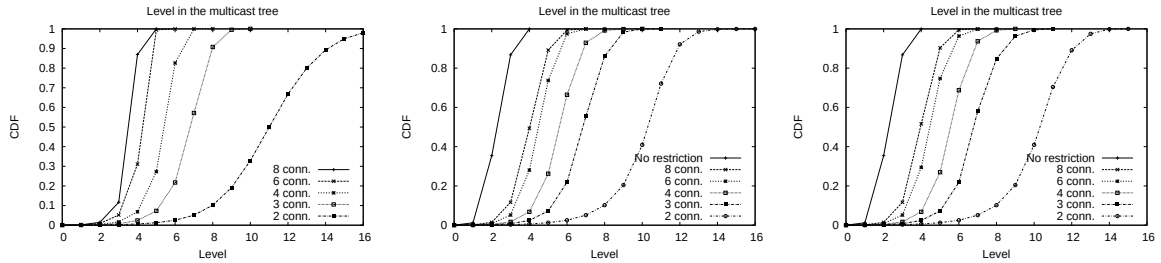
## 5 Conclusion and Future Work

We have presented MULTI+, a method to build application-level multicast trees on P2P systems. MULTI+ relies on TOPLUS in order to find a proper parent for a peer in the multicast tree. MULTI+ exhibits the advantage of being able to create topology-aware content distribution infrastructures without introducing extra traffic for active measurement. Admittedly, out-of-band information regarding the TOPLUS routing tables must be calculated offline (a simple process) and downloaded (like many P2P systems today require to download a list of peers for the join process). The proximity-aware scheme improves the end-to-end latency, and using host coordinates calculated offline and obtained at join time (as is done for TOPLUS) avoids the need for any active measurement. 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. As future work, we plan to evaluate the impact of leaving and failing peers on the multicast tree performance.



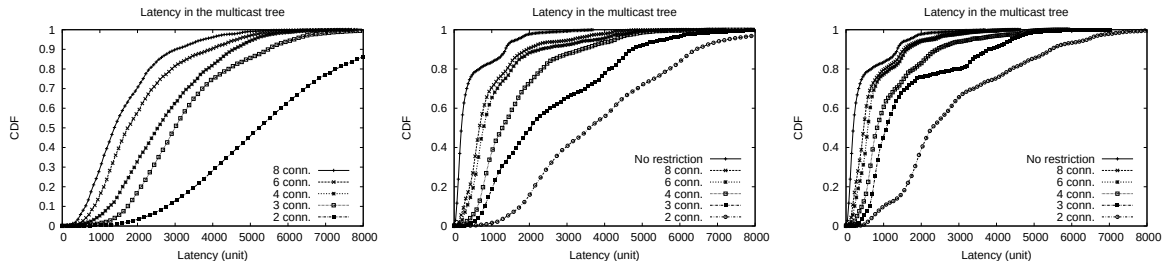

(a) Random parent selection. (b) FIFO parent selection. (c) Proximity-aware parent selection.

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



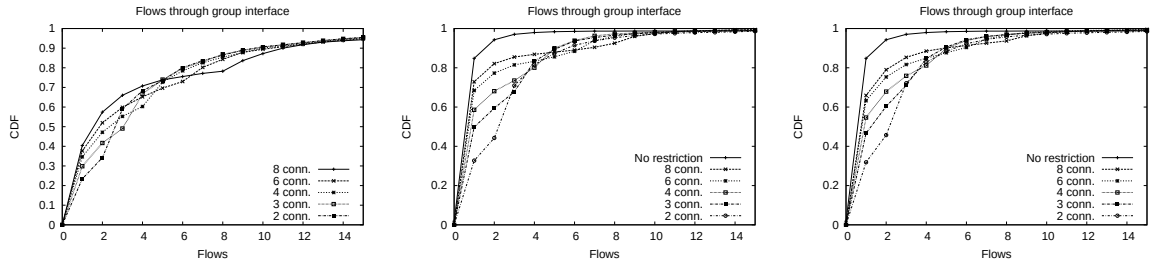

(a) Random parent selection. (b) FIFO parent selection. (c) Proximity-aware parent selection.

**Fig. 8.** Level of peers in the multicast tree.




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

**Fig. 9.** Latency from root to leaf (in Skitter 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.

*Acknowledgments.* This work could not have been done without the kind help of Prof. Mark Crovella, who provided us with the Skitter coordinate space data of more than $190,000$ IP addresses for our experiments.

## References

1. "CAIDA", http://www.caida.org/
2. S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast", In *Proceedings of ACM/SIGCOMM*, August 2002.
3. F. E. Bustamente and Y. Qiao, "Friendships that last: Peer lifetime and its Role in P2P Protocols" In *Proceedings of 8th Int. Workshop on Web Content Caching and Distribution*, Hawthorne, NY USA, September 2003.
4. M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Topology-aware routing in structure peer-to-peer overlay network", In *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, June 2002.
5. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment", In *Proceedings of SOSP'03*, New York, USA, October 2003.
6. M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure", *IEEE Journal on Selected Areas in Communications*, 20(8):1489 –1499, October 2003.
7. M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays", In *Proceedings of Infocom*, San Francisco, USA, April 2003.
8. M. Freedman and D. Mazieres, "Sloppy hashing and self-organizing clusters", In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, March 2003.
9. L. Garces-Erice, E. W. Biersack, and P. A. Felber, "MULTI+: Building Topology-Aware Peer-to-Peer Multicast Trees", RR-04-107, Institut Eurecom, Sophia-Antipolis, France, April 2004.
10. L. Garces-Erice, K. W. Ross, E. W. Biersack, P. A. Felber, and G. Urvoy-Keller, "Topology-Centric Look-Up Service", In *Proceedings of COST264/ACM Fifth International Workshop on Networked Group Communications (NGC)*, pp. 58–69, Munich, Germany, September 2003.
11. K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating Latency between Arbitrary Internet End Hosts.", In *Proceedings of 2nd Internet Measurement Workshop 2002*, Marseille, France, November 2002.
12. Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast", *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast*, 20, 2003.
13. M. M. John W. Byers, Jeffrey Considine and S. Rost, "Informed Content Delivery Across Adaptive Overlay Networks", In *Proceedings of ACM SIGCOMM 2002*, pp. 47–60, August 2002.
14. B. Krisnamurthy and J. Wang, "On Network-Aware Clustering of Web Sites", In *Proceedings of ACM SIGCOMM 2000*, August 2000.
15. H. Lim, J. C. Hou, and C.-H. Choi, "Constructing internet coordinate system based on delay measurement", In *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement (IMC-03)*, pp. 129 – 142, Miami Beach, FL, USA, October 2003.
16. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches", In *Proceedings of Infocom 2002*, June 2002.
17. N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer and A. Wolman, "SkipNet: A Scalable Overlay Network with Practical Locality Properties", In *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03)*, March 2003.
18. M. Pias, J. Crowcroft, S. Wilbur, and T. H. S. Bhatti, "Lighthouses for Scalable Distributed Location", In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, March 2003.
19. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", In *Proceedings of ACM/SIGCOMM*, August 2001.
20. S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Application-Level Multicast Using Content-Addressable Networks", In *Proceedings of NGC 2001*, pp. 14–29, November 2001.
21. A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp. 329–350, Heidelberg, Germany, November 2001.
22. S. Shenker, S. Ratnasamy, M. Handley, and R. Karp, "Topologically-Aware Overlay Construction and Server Selection", In *Proceedings of Infocom 2002*, New York City, NY, June 2002.
23. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications", In *Proceedings of ACM/SIGCOMM*, August 2001.
24. L. Tang and M. Crovella, "Virtual Landmarks for the Internet", In *Proceedings of the ACM/SIGCOMM Internet Measurement Conference (IMC-03)*, Miami Beach, Florida, USA, October 2003.
25. Y. Zhu, J. Guo, and B. Li, "oEvolve: Towards Evolutionary Overlay Topologies for High Bandwidth Data Dissemination", *IEEE Journal on Selected Areas in Communications, Special Issue on Quality of Service Delivery in Variable Topology Networks*, Third Quarter 2004.
26. S. Q. Zhuang et al., "Bayeux: An Architecture for Scalable and Fault-tolerant Wide Area Data Dissemination", In *Proceedings of NOSSDAV'01*, pp. 124–133, June 2001.

# A Appendix: Proof of the Properties

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

**Proof:** For this peer $p$, $\forall i$ RIG-$i \subset H_i(p)$. To join the multicast tree, $p$ first looks up RIG-$i$. RIG-$i$ memorizes $p$. If RIG-$i$ does not know about another peer in the multicast tree, there is no such peer in $H_i(p)$. $p$ continues looking up RIG-$i-1$, and so on, until $p$ reaches a RIG-$k$ who knows about a peer $q$ in the multicast tree. Because RIG-$k$ remembers $q$, RIG-$k \subset H_k(q)$. Thus RIG-$k \subset H_k(q)$ and RIG-$k \subset H_k(p)$. Assuming a proper nesting in TOPLUS, RIG-$k$ can not be in two different groups of the same tier. Thus $H_k(q) = H_k(p)$ and both $p$ and $q$ are contained in the same group. And $H_k(p)$ is the lowest-tier group containing both because RIG-$k$ is the first to know about $q$. □

**Property 2** *F*or each group defined by an IP network prefix containing at least one peer connected to the multicast tree, there is only one inbound data flow.

**Proof:** We proof by induction for a peer $p$ in tier-$i+1$ contained in a set of groups (or networks defined by IP network prefixes) $H_{i+1}(p), H_i(p), \cdots, H_1(p)$:

- If there is one peer $q$ already connected to the multicast tree in $p$'s inner group $H_{i+1}$, $p$ connects to $q$. Else, $p$ must connect to another peer outside $H_{i+1}$. Thus only one flow may go into the network $H_{i+1}$.
- In the look-up of a peer already connected to the multicast tree, assume $p$ gets to $H_k(p)$ in tier-$k$. Thus, by property 1 no peer exists connected to the multicast tree in $H_{i+1}(p), H_i(p), \cdots, H_{k+1}(p)$. Only the data flow to $p$ will go into $H_{i+1}(p), H_i(p), \cdots, H_{k+1}(p)$. If there is one peer $q \in H_k(p)$ receiving the multicast flow, $p$ connects to $q$, and $p$ adds no data flow into $H_k(p)$. Else $p$ proceeds the look-up in $H_{k-1}(p)$. In any case only one data flow can go into the network defined by $H_k(p)$. □

**Property 3** *F*or each group defined by an IP network prefix containing at least one peer connected to the multicast tree, the number of outbound data flows in the worst possible case is bounded by a constant.

**Proof:** Assume as before a peer $p$ in tier-$i+1$ contained in a set of groups (i.e., networks defined by IP network prefixes) $H_{i+1}(p), H_i(p), \cdots, H_1(p)$. Assume peer $p$ is looking for a peer to connect to. Without loss of generality, let $H_k(p)$ be the tier-$k$ group where the RIG-$k$ knows about a peer $q$ connected to the multicast tree. $p$ connects to $q$ and adds an outbound data flow (from $q$ to $p$) to $H_{k+1}(q)$. From the previous property, there can only be at most one inbound data flow in $H_{k+1}(p)$. The same is valid for every sibling group of $H_{k+1}(q)$. The number of siblings of $H_{k+1}(q)$ is a constant. Thus the number of outbound data flows from $H_{k+1}(q)$ towards tier-$(k+1)$ groups is a constant. For a worst case scenario, assume now that $q$ is also the peer known by RIG-$(k-1)$ in $H_{k-1}(q)$. Applying the same reasoning to the siblings of $H_k(q)$, we have another constant number of outbound flows from $H_{k+1}(q)$ towards tier-$k$ groups. A constant number of outbound flows at every tier, and the number of tiers being bounded by a constant, the number of outbound data flows is bounded by a constant for group $H_{k+1}(q)$. □

**Property 4** *U*sing 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. The proof is trivial from the two previous properties. □