Proximity-Aware Collaborative Multicast for Small P2P Communities

Francisco de Asís López-Fuentes*, Eckehard Steinbach

Technische Universität München
Institute of Communication Networks, Media Technology Group
80333 München, Germany
{fcoasis, Eckehard.Steinbach}@tum.de

Abstract

In this paper we describe a novel solution for delay sensitive one-to-many content distribution in P2P networks based on cooperative m-ary trees. Our scheme maximizes the overall throughput while minimizing end-to-end delay by exploiting the full upload capacities of the participating peers and their proximity relationship. Our delivery scheme is based on cooperation between the source, the content-requesting peers and the helper peers. In our solution, the source splits the content into several blocks and feeds them into multiple m-ary trees rooted at the source. Every peer contributes its upload capacity by being a forwarding peer in at least one of the m-ary trees. Our performance evaluation shows that our proposal achieves similar throughput as the best known solution in the literature (Mutualcast) while at the same time reducing content delivery delay.

1. Introduction

Many delay sensitive applications such as TV over IP or multi-party video conferencing require content distribution from one source to multiple receiver nodes. IP Multicast has been proposed as an efficient solution for one-to-many content dissemination. However, IP Multicast is not widely available in today's Internet. The construction of multicast trees at the application layer has been proposed as an alternative and has received considerable attention in the literature [1, 2, 3]. Application-level approaches provide more flexibility and are easier to deploy compared to network-based multicast support. In conventional tree-based distribution, the interior nodes redistribute the content, while the leaf peers receive the content only. Although multicast-tree based content distribution has shown to be highly scalable [3, 4, 5], it is not maximally efficient in collaborative environments, because the upload capacity of the leaf peers is not used during a multicast session. In order to achieve maximum throughput we need to exploit the full upload capacity of all participating peers. A solution is the construction of multiple concurrent trees where peers contribute with their upload capacity in at least one tree. The determination of the number of required trees to maximize the overall throughput is an open problem [6].

A scheme that fully exploits the upload capacity of all participating peers is Mutualcast [7]. This scheme has been shown to provably maximize the overall throughput during a multicast session. Mutualcast engages as many peers as possible and uses their full upload capacities in order to maximize the overall throughput. Although Mutualcast achieves the maximum possible multicast throughput in P2P networks with constrained upload capacities, it has some limitations for delay sensitive applications, because it only considers the peer upload capacity ignoring their proximity relationship. Additionally, due to its fixed and fully connected topology Mutualcast has limited scalability.

In this paper we present an algorithm to build multiple *m-ary* trees in which the upload capacities of the peers are fully exploited and proximity issues are explicitly considered. In our proximity-aware collaborative multicast scheme, the *m-ary* trees are rooted at the source and they always maintain a height of two levels from the root in order to avoid deep structures. The source splits the content into blocks and distributes every block separately by using a collection of *m-ary* trees. Every participating peer can receive one or more data blocks directly from the source. After this, every peer forwards the data block to its children in the corresponding *m-ary* tree. All peers contribute with their full upload capacity by being a forwarding peer in at least one of the *m-ary* trees. The goal of our scheme is to maximize the overall throughput, while minimizing end-to-end delay by considering peer proximity issues during a multicast

* This work has been supported by DAAD-CONACYT grant 68858

I-4244-0910-1/07/$20.00 © 2007

session. To our knowledge our approach is the first work that combines upload capacities and proximity information using multiple *m-ary* trees rooted at the source in an efficient manner for delay sensitive one-to-many content delivery. We compare the performance of our approach with Mutualcast [7], which is the best known scheme in terms of maximizing overall throughput. Our results show that our approach achieves an overall throughput similar to Mutualcast, while maintaining a smaller end-to-end delay.

Our contributions in this paper are the following:

1. A cooperative multicast scheme based on a collection of *m-ary* trees which achieves an overall throughput similar to fully connected schemes such as Mutualcast, while maintaining a reduced end-to-end delay.
2. An algorithm to obtain a near optimal *m-ary* tree collection rooted at the source which exploits the full upload capacity of all participating peers in combination with their proximity information during a multicast session. Our *m-ary* tree collection is different to the tree collection used by Mutualcast because our scheme avoids that a single peer has to forward its received blocks from the source to all peers in the fully connected topology. Hence, the worst case delay encountered in Mutualcast does not apply to our scheme.

The remainder of this paper is organized as follows. We discuss related work in Section 2. In Section 3 we declare our motivation and introduce our approach based on *m-ary* trees. Then, we explain how to build the collection of *m-ary* trees for a multicast session in Section 4. In Section 5 we evaluate the performance of our approach. Section 6 concludes the paper.

2. Related work

Because IP multicast has not been widely deployed, several application-level multicast schemes have been proposed for different applications [1, 2, 3, 7, 8, 9, 10]. Some of them are based on a single multicast tree [2, 3, 8] without considering the collaboration among peers to maximize the throughput, while others split the data over multiple trees to increase the overall throughput. Both Coopnet [9] and SplitStream [1] split the content and distribute the striped data using separate multicast trees. Coopnet proposes a centralized scheme to manage the multiple multicast trees from different sources, while SplitStream uses a decentralized scheme to construct a forest of multicast trees from a single source. Both systems fail to utilize the full upload capacity of all the participating nodes in the multicast group, limiting the maximum overall throughput. In Balanced Multicasting [10], the authors propose a balancing of the maximum amount of upload capacity and achievable capacity in all nodes and paths of a network. Although Balanced Multicasting increases the throughput for grid applications, it provides limited scalability and adaptability. The authors in [2] have proposed to construct an overlay mesh to maximize the throughput during a concurrent data distribution. While some systems [4, 5, 11] are designed for large-scale applications other systems such as Overcast [3], Scattercast [12] or Mutualcast [7] are designed for small-scale overlay multicast approaches. All these approaches exploit the peers' upload capacity only, while proximity issues (geographical position or connection quality) are ignored. In particular, Mutualcast achieves provably the maximum overall throughput during a multicast session. However, this approach does not address delivery delay or scalability when the multicast group size is increased.

Recently, proximity has been addressed as an important issue in service and application distribution in P2P networks. Some approaches [13, 14, 15] combine transmission capacities and proximity issues. In [13] Magellan is proposed, which is a multicast scheme based on cooperative applications. This approach reduces the total end-to-end hop distance in the distribution topology by using balanced trees, but without exploiting the full upload capacity of the participating peers. Also, the tree depth is increased as the group size increases. In [14] a distributed algorithm is presented to construct an overlay network based on the capacity of each peer and proximity information. The authors evaluate their proposed scheme using application layer multicast, however, they do not show the strategy to build the multicast tree. Zhu et al. propose in [15] to combine node capabilities and proximity relationship on load balancing schemes for DHT-based P2P systems. Our work has some similarity to [15] in the sense that we use node upload capacities and proximity relationship in combination with an *m-ary* tree for content delivery. However, several features make our approach different. First, our approach addresses multicast services and second, our approach uses a reduced collection of *m-ary* trees in order to maximize the overall throughput. Finally, our *m-ary* trees are built with a height identical to the Mutualcast scheme but combining proximity relationship with peer upload capacities.

3. Motivation

Our approach is inspired by [7], which presents a mechanism to obtain the maximum possible multicast throughput in P2P networks with constrained upload capacities. Peers can greatly benefit from the capacity of other requesting peers via collaboration, and hence the need to collaborate for multicast applications in large-scale and heterogeneous environments. However, the participating peers are typically in different geographical locations, such as is shown in Figure 1.

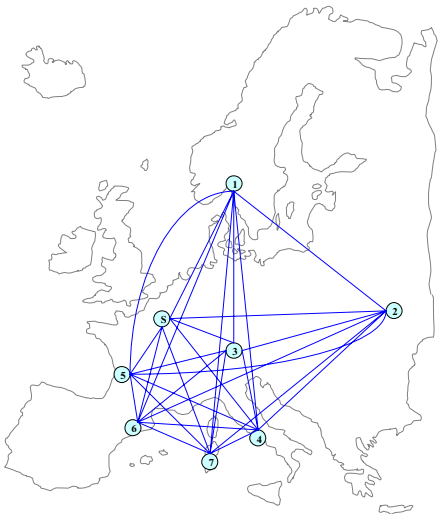


Figure 1. Peers are located in several geographical locations during a multicast session

We extend the Mutualcast scheme in [7] by adding proximity information. In this work we use as proximity information the Round-Trip-Time (RTT) between two peers. In Mutualcast, the source assigns each block of content to a single peer for redelivery. Each peer redelivers its assigned block to the rest of the requesting peers. In this case, the distribution tree has two levels from the source for each data block. Thus, when all requesting peers have the same proximity among them, all blocks are delivered within the same time to all peers. We denote the number of peers participating in the multicast group as *K*. This case is shown in Figure 2 where the source *S* distributes the blocks X_1 to X_4 to *K=4* requesting peers. The distance *d* (our proximity measure) among them is assumed to be identical.

Mutualcast does not control the delivery time. This is because each peer forwards its block to all other peers without considering that these peers may be distant from it. On the contrary, our approach distributes a block through two or more peers, but they evaluate the peer proximity before forwarding their blocks to the rest of the peers. The basic idea is illustrated in Figure 3 using one source and *K=5* requesting peers. The distance among peers is indicated by *d*, while X_i indicates the block management in each peer. Peers 1 and 2 have a distance of *d* from the source. Peers 3 and 4 have a distance of *2d* from the source. Peer 5 is *3d* away. The worst case distance occurs when block X_5 from the source is delivered through peer 5 to peer 4. In Figure 3a, we can see that block X_5 travels a maximum distance of *8d* when the Mutualcast approach is used. On the other hand, we can see in Figure 3b that when our approach is used block X_5 encounters at most a distance of *5d* during delivery from the source to all the requesting peers. How to build

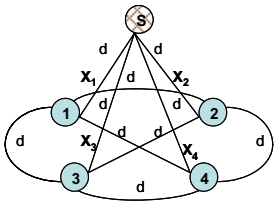
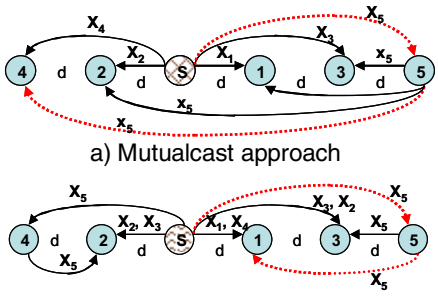


Figure 2. Mutualcast with same distance *d* (e.g., RTT) among peers



a) Mutualcast approach

b) *m-ary* trees rooted at the source

Figure 3. Illustration of the worst case distance for Mutualcast and our approach when block X_5 from the source is delivered

the collection of *m-ary* trees that achieves the previously described improved worst case distance is explained in the following section.

4. System Architecture

In our system, we assume that all participating peers collaborate by contributing their upload capacity. We assume asymmetric network access speeds (e.g. DSL) and hence the upload capacity is considered to be the limiting resource. Each peer can contribute to the data distribution in one or more *m-ary* trees. Similar to Mutualcast, the source splits the content into blocks. After this, our system builds a collection of *m-ary* trees over which the source delivers the content blocks. Every peer receives one or more data blocks from the source. Initially, peers with large upload capacity are used as forwarding peers, while peers with small upload capacity are placed as leaves in most multicast trees within the tree collection. When the remaining upload capacity of the best peers is small, they become leaves in the remaining trees.

We assume that the source knows the IP address and the upload capacity of all peers. Additionally, the distance among peers is assumed to be known. In this case our distance measure is the round-trip time (RTT). For each participating peer p_j the upload capacity C_j is stored in list $C = \{C_1,..,C_j,..,C_K\}$ while the distance $D_i(j)$ between peers p_i and p_j is stored in list $D_i = \{D_i(1),..,D_i(j),...,D_i(K)\}$.

For each peer p_j, every peer p_i calculates the normalized distance $D_i^n(j)$ as

$$D_i^n(j) = \frac{D_i(j)}{\max\limits_{D_i(j) \in D_i} \{D_i(j)\}} \quad (1)$$

where $0 < D_i^n(j) \leq 1$.

The normalized upload capacity C_j^n for every peer p_j is computed as

$$C_j^n = \frac{C_j}{\max\limits_{C_j \in C} \{C_i\}} \quad (2)$$

where $0 < C_j^n \leq 1$.

The normalized distance values $D_i^n(j)$ and the normalized upload capacity values C_j^n are stored in lists $D_i^n = \{D_i^n(1),..,D_i^n(j),..,D_i^n(K)\}$ and $C^n = \{C_1^n,.., C_i^n,.., C_K^n\}$, respectively. These lists will be used by the tree-construction algorithm to select peers. Peers with high C_j^n and low $D_i^n(j)$ have a high preference to be selected. We use a preliminary delivery rate (PDR) to approximately determine how many times a peer can be used as a forwarding peer in different distribution trees. The *PDR* is computed as

$$PDR = \frac{\sum_{j=1}^{K} C_j}{K(K - m)} \quad (3)$$

with $K > m$. K is the number of requesting peers in the multicast group and m is the number of peers directly connected to the root of an *m-ary* tree. The denominator of (3) represents the number of leaf peers being served by the forwarding peers in the set of distribution trees. In every tree there are $(K-m)$ leaf nodes. Hence, the number of leaf peers in a set of *m-ary* trees, where the number of trees is the same as the number of requesting peers, is equal to $K(K-m)$.

4.1. Building a collection of *m-ary* trees

Our approach is based on a heuristic construction of *m-ary* trees where all the participating peers collaborate with their upload capacity and their proximity information. The initial number of distribution trees is identical to the number of requesting peers. To avoid deep structures we fix the height H of the *m-ary* trees to two levels. Thus, the maximum number of requesting peers in each *m-ary* tree is $m(m+1)$. Assuming that the number of requesting peers in the multicast group K is known, the source determines the degree m to be used as

$$m = \left\lfloor \frac{1}{2}\sqrt{4K+1} - \frac{1}{2} \right\rfloor \quad (4)$$

After this, the source selects m children as the forwarding peers for each distribution tree. The algorithm to select forwarding peers works as follows. The source calculates $D_i^n(j)$ and C_j^n in (1) and (2) for each peer and the preliminary delivery rate PDR in (3). The source selects the m peers with the largest normalized upload capacities C_j^n to be the forwarding peers in the first *m-ary* tree. The $(K-m)$ leaf peers are assigned to the forwarding peers based on the proximity information. At most *m* leaf peers can be assigned to one forwarding peer. If the number of leaf peers is smaller than m^2 we additionally balance the assignment of leaf peers to avoid exhausting one peer in one tree. For every leaf node, he source then subtracts one time the *PDR* from the upload capacity of the forwarding peers. Selected peers are used as forwarding peers in several distribution trees until their remaining upload capacity is no longer sufficient for building the next tree. The source then selects the next peer with the largest C_j^n from C^n which has not yet been used. When the next best peer has an upload capacity less than its number of children times *PDR*, the source calculates a new *PDR* which is obtained by dividing the peer's upload capacity between the number of leaf peers that it must feed. After this, the source uses this peer and exhausts the peer's upload capacity. Finally, when all peers have been used, but the *m-ary* tree collection is still not completed, the source reconsiders the peers that have still not been exhausted but this time using the most recent *PDR* value. In every step, when there are more than *m* peers with the same upload capacity, the source begins by selecting the *m* closest peers to it.

Out of the *m* closest peers, each peer distributes the received block to those who still have not received the same block from another forwarding peer in the distribution tree. The selection of these children is based in their proximity in order to avoid adding long delay. This helps us to reduce the end-to-end delay. The collection is completed when the number of obtained *m-ary* trees is equal to the number of peers K. After this, we detect and delete repeated *m-ary* trees in order to obtain a reduced *m-ary* tree collection. A reduced *m-ary* tree collection allows us to reduce the number of blocks to be sent by the source and to increase their size. Once the reduced *m-ary* tree collection is obtained, linear programming (LP) can be used to compute the optimum block sizes that maximize the throughput f of the distribution tree collection. As explained later in this section, the set of multicast trees with their node upload capacities are translated to decision variables and constraints of the linear program. The delivery latency is the end-to-end delay from the source to the receivers. The

delay minimization has already been taken care of during tree construction.

We illustrate our algorithm for a multicast group with seven requesting peers K and a sender. The upload capacity of the requesting peers and their distance from the source is given in Table 1. The distances (RTT) among the requesting peers and to the source are obtained from the network coordinates model in [16] with data acquired from CAIDA's Skitter project [17]. Initially, based on the number of requesting peers K, seven *m-ary* trees are created to distribute the seven blocks X_1 to X_7. Using equations (1) and (2), each peer and the source normalize the upload capacity and distance for every participating peer. Due to the limit of space, we show the normalized upload capacities and distances for the source only, and skip the details for the rest of the peers. The *PDR* and m values for the source are calculated from (3) and (4) to be 71.4 kbps and 3, respectively. Since our intention is to build *m-ary* trees which are balanced as much as possible, we then obtain in our example distribution trees with one forwarding peer feeding two leaf peers and two forwarding peers feeding one leaf peer each. In Figure 4, p_2, p_4 and p_6 are used as forwarding peers in the first *m-ary* tree to distribute block X_1. This is because peers p_2, p_4 and p_6 have the largest upload capacities in the multicast group. Although peers p_6 and p_3 have the same upload capacity, p_6 is preferred because it has a smaller normalized distance $D_i^n(j)$ to the source than peer p_3. The rest of the nodes receive block X_1 from the closest forwarding peer. Thus, peer p_2 selects peers p_3 and p_5 as its two closest leaf peers to forward block X_1, while p_6 and p_4 sends block X_1 to peers p_7 and p_1, respectively. The remaining upload capacity of p_2, p_4, and p_6, becomes 357.2 kbps, 328.6 kbps and 218.6 kbps, respectively. Comparing the remaining peer upload capacity to the *PDR,* the source determines that peers p_2, p_4 and p_6 can still be used as forwarding peers in three additional distribution trees. In the forth distribution tree, the capacity of p_2 is exhausted, while the available capacity of p_4 and p_6 is reduced to 43 kbps and 14.4 kbps respectively, which is smaller than *PDR*. Therefore, p_4 and p_6 cannot be used in another distribution tree for the time being. Now, the source determines that peers p_3, p_5 and p_7 can be used as the next forwarding peers in the fifth and sixth distribution trees. Afterwards, the remaining upload capacity of peers p_3, p_5 and p_7 becomes 14.4 kbps, 57.2 kbps and 57.2 kbps, respectively. Because the remaining upload capacity in these peers is smaller than *PDR,* they cannot be used in another distribution tree for now. The source then selects the next peer not yet used, which is peer p_1. However, this peer has a capacity less than *2*PDR* and thus in order to use this peer as forwarding peer, the source must adjust the *PDR* to half the capacity of the peer. After this, the source exhausts the capacity of the peer. Once all peers have been used as forwarding peers at least in one distribution tree, but the number of obtained *m-ary* trees is still different from the number of nodes, we need to exhaust the remaining upload capacity of peers p_5 and p_7 using the new *PDR*. The source then selects p_1, p_5 and p_7 as forwarding peers in this last distribution tree. Each forwarding peer sends their received block(s) to their closest leaf peers. Our obtained *m-ary* tree collection is shown in Figure 4. From Figure 4, we can see that the source uses the same *m-ary* tree structure for the delivery of blocks X_1, X_2 and X_3. The same situation holds for blocks X_5 and X_6. Here, we can eliminate duplicate trees in order to obtain a reduced *m-ary* tree collection, and the source now can distribute its content using four *m-ary* trees only. Each distribution tree delivers a specific block to all requesting peers.

Table 1. Upload capacity and distance from the source for every requesting peer

p_j	C_j (kbps)	$D_{source}(j)$ (ms)	C_j^n	$D_{source}^n(j)$
1	100	1495	0.2	0.633
2	500	2361	1.0	1.0
3	300	1716	0.6	0.726
4	400	1731	0.8	0.733
5	200	913	0.4	0.386
6	300	1200	0.6	0.508
7	200	390	0.4	0.165

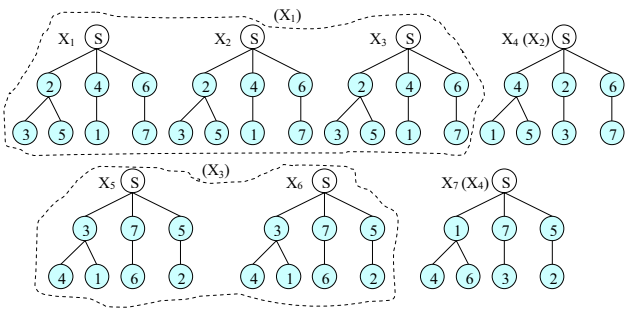


Figure 4. Preliminary *m-ary* tree collection. The data within parenthesis indicates the blocks which are reassigned for the reduced *m-ary* trees collection

The maximum throughput of our tree collection and the size of the blocks are determined using linear programming. Figure 5 shows the reduced *m-ary* tree collection from Figure 4 translated to a linear program. The source splits the content into five blocks. The blocks X_1 to X_4 are distributed from the source to the requesting peers through four *m-ary* trees, while the block X_5 is distributed from the source to each requesting peer directly. We assume in this example that the source has an upload capacity of 2000 kbps. The first constraint $3X_1 + 3X_2 + 3X_3 + 3X_4 + 7X_5 \leq 2000$ kbps considers the upload capacity of the source, which has to deliver three blocks

```
Maximize
  f = X1 + X2 + X3 + X4 + X5
subject to
c1: 3X1+3X2+3X3+3X4+7X5 <= 2000
c2: 2X3 <= 100.0
c3: 2X1 +2X3 <= 500.0
c4: 2X2 <= 300.0
c5: 2X1 <= 400.0
c6: 2X4 <= 200.0
c7: 2X2 <= 300
c8: 2X4 <= 200
```

Figure 5. A reduced *m-ary* tree collection is translated into a linear program for throughput maximization

to every *m-ary* tree and additionally sends the block X_5 to every peer directly. The rest of the constraints considers the upload capacity of the requesting peers p_1, p_2, p_3, p_4, p_5, p_6 and p_7. The solution gives a maximum throughput of 571.4 kbps, while the size of the blocks in kbits is X_1 = 200, X_2 = 100, X_3 = 150, X_4 = 50 and X_5 = 71.42, respectively.

For this specific example, our solution reaches the same maximum throughput as Mutualcast. However, this is not always the case since we use heuristics to find a near-optimal solution. Calculating the optimal set of multicast trees on the fly is a hard task, because an exact solution requires to evaluate all possible combinations for all co-existing sessions in the overlay network and the number of combinations and constraints grows exponentially with the number of participating peers.

4.2. Throughput comparison with Mutualcast

The Mutualcast approach in [7] uses three distribution routes. These are (1) through the content-requesting peer nodes; (2) through the helper nodes and (3) directly from the source. The route 3 is chosen only when the source still has upload capacity after exhausting routes 1 and 2. Assuming that the source has an upload capacity C_S, the N requesting peers have an average capacity C_N and M helper peers have an average capacity C_M, the maximum throughput f is obtained as [7]

$$f = \begin{cases} C_S, & C_S \le C_1 + C_2, \\ (C_1 + C_2) + \dfrac{C_S - (C_1 + C_2)}{N_1}, & C_S \ge C_1 + C_2, \end{cases} \quad (5)$$

with $C_1 = (\dfrac{N}{N-1} C_N)$ and $C_2 = (\dfrac{M}{N} C_M)$.

Because our collection of *m-ary* trees is based on heuristics an exact solution is not always possible. However, we found through extensive simulations that in cases when balanced *m-ary* trees are used and the source capacity is abundant our approach can achieve an overall throughput identical to the maximum throughput given by Mutualcast.

5. Performance evaluation

We evaluate our approach in terms of overall throughput and delivery latency. We compute the maximum overall throughput using linear programming as explained in Section 4.1. The maximum required time so that all the nodes receive all data blocks is obtained too. The results obtained with our approach are compared with the results when using the Mutualcast approach.

We simulate different cases for multicast groups with 6, 7 and 10 requesting peers. In every case, we vary the source capacity from 1000 to 6000 kbps, while the upload capacity of each requesting peer is fixed. Heterogeneous upload capacities and proximity relationships are used for every peer in the different cases. The upload capacity of the requesting peers p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9 and p_{10} in kbps is 100, 500, 300, 400, 200, 300, 200, 400, 300 and 100, respectively. The round trip times between peers and the source are obtained from [16] and [17]. A comparison between *m-ary* trees rooted at the source and Mutualcast in terms of overall throughput for theses cases is shown in Table 2. The results show that for a heterogeneous multicast group, our proposed *m-ary* trees based approach achieves almost the same overall throughput as the Mutualcast approach. For both approaches we consider that the source has enough capacity and it can distribute extra content blocks directly to all requesting peers. We can see that both approaches lead to identical throughput if the source capacity has enough upload capacity.

We compare our approach and the Mutualcast approach in terms of delivery latency. Delivery latencies achieved by our approach and Mutualcast are compared in Figures 6 and 7. Figure 6 shows the maximum delivery delay by our approach and the Mutualcast approach for the delivery of all content blocks to all requesting peers in the multicast group. This delay is determined by the slowest distribution tree. The results show that when our

Table 2. Overall Throughput Comparison

Source Capacity (kbps)	Mutualcast Throughput (kbps)			Reduced *m-ary* trees Throughput (kbps)		
	Number of requesting peers			Number of requesting peers		
	6	7	10	6	7	10
1000	467	428	380	467	333	333
2000	633	571	480	633	571	480
3000	800	714	580	800	714	580
4000	967	857	680	967	857	680
5000	1133	1000	780	1133	1000	780
6000	1300	1142	880	1300	1142	880

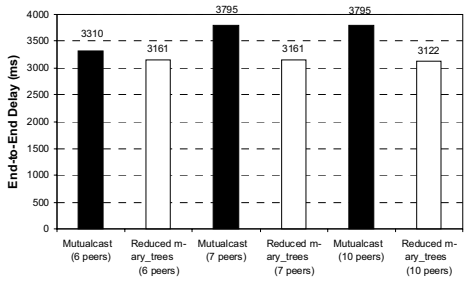


Figure 6. Maximum delay comparison

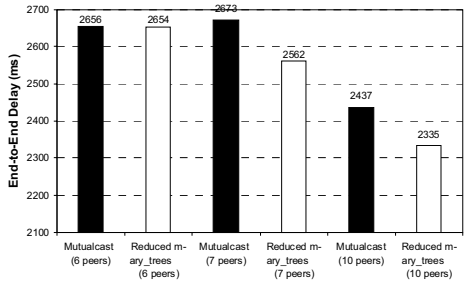


Figure 7. Average delay comparison

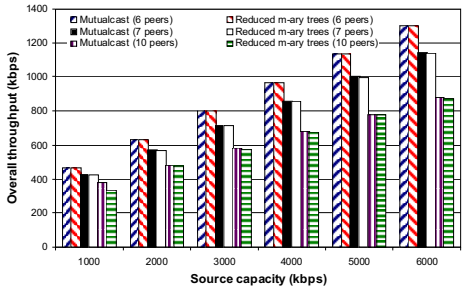


Figure 8. Overall throughput comparison

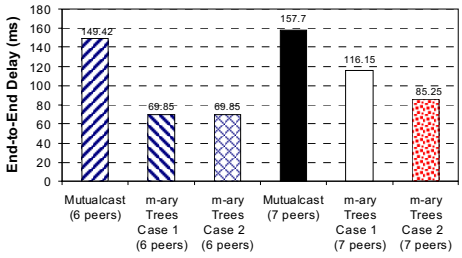


Figure 9. Maximum delay comparison

approach is used, the maximum end-to-end delay is smaller in comparison to Mutualcast. We attribute this improvement to the fact that our algorithm based on *m-ary* trees avoids deep structures and incorporates proximity information into the overlay topology. Figure 7 shows the average delivery time for all blocks for all peers. This delay is the sum of the maximum end-to-end delay in each distribution tree divided by the number of distribution trees. We can see that our approach based on *m-ary* tree rooted at the source shows a better average end-to-end delay than the Mutualcast approach.

To compare the simulated RTT values from the network coordinates model with real RTT values, we measure the RTT between different nodes in the Internet (e.g., Universities and Internet Providers) using SmokePing [18]. We evaluate our approach using nodes in the USA (San Diego, Berkeley, UC Davis and Boston) and Europe (Amsterdam, Zurich, Paris and Estonia) and assume upload capacities of 100, 500, 300, 1000, 400, 200, 300 and 200 kbps for the nodes respectively such as our evaluation based on the network coordinates model. We assume that the source is in Boston, while the requesting peers are allocated in the rest of the sites. The source capacity is varying from 1000 to 6000 kbps. Overall throughput and maximum delay for our approach and Mutualcast are compared in Figures 8 and 9, respectively. For our approach based on *m-ary* trees, we evaluate two cases. Case 1 is based on RTT only, while Case 2 considers RTT and geographical location. Here, the geographical location is used first to avoid that the source sends the same block X_i to two peers with similar RTT, that are close to each other when a third peer with similar upload capacity presents a better location for the rest of the requesting peers, and second to limit the connection of the children peers within a local region in order to reduce the end-to-end delay.

Figures 8 and 9 show that when there are six requesting peers (3 peers in USA and 3 peers in Europe) in the multicast group and the source has an upload capacity of 1000 kbps, the overall throughput using Case 1 and 2 is same as the Mutualcast throughput, while the maximum delay is reduced by 53% with respect to Mutualcast. Here, Case 1 and Case 2 lead to the same overall throughput and delay by using both an identical set of balanced and complete *m-ary* trees. When there are seven requesting peers in the group, Case 1 and Case 2 use different sets of *m-ary* trees. Case 1 achieves 98.3% of the Mutualcast throughput, while the overall throughput using Case 2 is 91.66% of the Mutualcast throughput. However, in both cases the maximum end-to-end delay is reduced by 26.5% and 46% compared to Mutualcast, respectively. Latencies shown in Figure 9 are smaller than latencies shown in Figure 6, because these are obtained from sites located in the Internet core.

Finally, we evaluate the delivery rate achieved by each distribution tree in our approach and the Mutualcast scheme during a multicast session. For this comparison we specifically consider the case of six peers when the source capacity is 1000 kbps. Here, both evaluated approaches achieve the same overall throughput. We assume that each block X_i is delivered by a distribution tree. The results are shown in Figure 10. For both

approaches, the blocks X_1 to X_5 within the square represent the blocks delivered by each distribution tree, while the block X_7 is delivered by the source to each requesting peer directly. While in our approach the source splits the total content in five blocks, Mutualcast requires to use an extra block X_6. This is because in Mutualcast each block X_i is assigned to one single node for redelivery. The results show that both models use different delivery rates in each distribution tree to maximize the overall throughput. Most distribution trees in our approach allow a bigger delivery rate than Mutualcast during a multicast session. This is because our approach uses fewer blocks but with a bigger size by distributing via five distribution *m-ary* trees only, while Mutualcast uses six distribution trees.

6. Conclusion

Finding a good tree topology that maximizes the overall throughput and limits delivery delay is critical in delay sensitive multicast applications. We have proposed and evaluated a content distribution approach based on *m-ary* trees. We compare our proposed approach with the Mutualcast scheme which achieves the maximum possible throughput, but potentially leads to large delivery delays. Our heuristic tree construction approach generates a collection of *m-ary* trees rooted at the source by combining the full upload capacities of all participating nodes and their proximity relationship. Our results demonstrate that our proposed multicast scheme provides a good balance between reduced end-to-end delay and maximum overall throughput while maintaining a scalable structure by avoiding a fully-connected topology.

7. References

[1] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments", *In Proc. of ACM SOSP*, pp. 298-313, October 2003.

[2] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", *In Proc. of ACM SOSP*, pp. 282-297, October 2003.

[3] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr., "Overcast: reliable multicasting with an overlay network", *In Proc. of the OSDI'00*, pp. 197-212, October 2000.

[4] B. Banerjee, B. Bhattacharjee and C. Kommareddy, "Scalable Application Layer Multicast", *In Proc. of ACM SIGCOMM,* pp. 205-217, August 2002.

[5] M. Castro, P. Druschel, A. M. Kermarrec and A. Rowstron, "SCRIBE: a large-scale and decentralized application-level multicast infrastructure", *In IEEE JSAC*, pp. 100-110, Vol.20, No.8, 2002.

[6] Y. Cui, B. Li, K. Nahrstedt, "On Achieving Optimized Capacity Utilization in Application Overlay Networks with Multiple Competing Sessions", *In Proc. of ACM SPAA 2004,* pp. 160 – 169, June 2004.

[7] J. Li, P. A. Chou, C. Zhang, "Mutualcast: An Efficient Mechanism for One-To-Many Content Distribution", *In Proc. of ACM SIGCOMM ASIA Workshop*, April 2005.

[8] R. Cohen and G. Kempfer, "An Unicast-based Approach for Streaming Multicast", *In Proc. of IEEE INFOCOM*, pp. 160-169, April 2001.

[9] V. N. Padmanabhan, H. Wang, P. Chou, K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking", *In Proc. of ACM NOSSDAV*, pp. 177-186, May 2002.

[10] M. Burger, T. Kielmann, H. E. Bal, "Balanced Multicasting: High-throughput Communication for Grid Applications", *In Proc. of ACM/IEEE SC'05*, pp. 46-54, November 2005.

[11] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz and J. Kubiatowicz", Bayeux: An Architecture for Scalable and Fault Tolerant Wide-area Data Dissemination," *In Proc. of ACM NOSSDAV*, pp. 11-20, June 2001.

[12] Y. Chawathe, "Scattercast: an Architecture for Internet broadcast distribution as an infrastructure service", PhD thesis, University of California, Berkeley, August 2000.

[13] S. Birrer, F. E. Bustamante, "Magellan: Performance-based, Cooperative Multicast", *In Proc. of IEEE WCW*, pp. 133-143, 2005.

[14] J. Zhang, L. Liu, C. Pu, "Constructing a Proximity-aware Power Law Overlay Network", *In Proc. of IEEE GLOBECOM*, pp. 636-640, 28 Nov.-2 Dec., 2005.

[15] Y. Zhu, Y.Hu, "Efficient Proximity-Aware Load Balancing for DHT-Based P2P Systems", *In IEEE TPDS* Volume 16, Issue 4, pp. 349 – 361, April 2005.

[16] T. S. Eugene Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-based Approaches", *In Proc. of IEEE INFOCOM*, pp. 170-179, June 2002.

[17] Cooperative association for internet data analysis (CAIDA) http://www.caida.org.

[18] SmokePing. http://oss.oetiker.ch/smokeping/

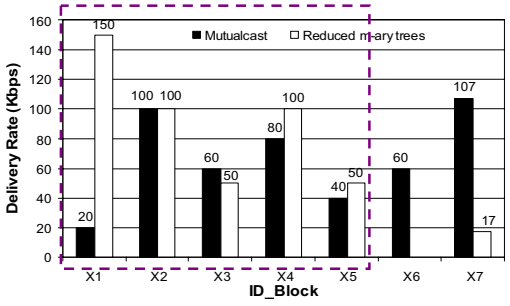


Figure 10. Delivery rate on each distribution tree