Self-organization in Cooperative Content Distribution Networks

Marc Schiely, Lars Renfer, Pascal Felber
Computer Science Department, University of Neuchâtel
CH-2007, Neuchâtel, Switzerland
{*marc.schiely, lars.renfer, pascal.felber*}*@unine.ch*

Abstract

Traditional client-server content distribution techniques usually suffer from scalability problems when dealing with large client population or sizable content. The advent of peer-to-peer (P2P) network offers the technical means to efficiently distribute data to millions of clients simultaneously with very low infrastructural cost. Previous studies of content distribution architectures have primarily focused on homogeneous systems, where the bandwidth capacities of all peers are similar. In this paper, we address the problem of heterogeneity and we propose mechanisms to improve content distribution efficiency by dynamically reorganizing the P2P network based on the effective bandwidth of the peers. Our techniques have been designed to be efficient in heterogeneous settings, adaptive so as to tolerate runtime changes like bandwidth fluctuations, and practical enough to be implementable in real systems. We analyze their effectiveness by the means of simulations and experimental evaluation.

1. Introduction

Peer-to-peer (P2P) systems, in which peer computers form a cooperative network and share their resources (storage, CPU, bandwidth), have attracted a lot of interest lately. After the apparition of the first truly successful P2P systems (e.g., Napster, Gnutella) and the significant amount of research conducted in Academia and in the Industry, most researchers now agree that P2P systems are more than just a fashion phenomenon. They offer great potential for building cooperative networks that are self-organizing, efficient, scalable, and reliable.

Research in P2P networks has so far mainly focused on content storage and lookup, but little work has been done about its actual distribution. By capitalizing the bandwidth of peer nodes, P2P architectures address some of the most challenging issue of todays Internet: the cost-effective distribution of bandwidth-intensive content to thousands of simultaneous users and the resilience to "flash crowds" (a huge and sudden surge of request traffic that usually leads to the collapse of the affected server). Indeed, as the client population and the size of the distributed content grow, the source quickly becomes a bottleneck.

Solutions based on content delivery networks (CDNs) are prohibitively expensive and rather static in nature, while protocols like IP multicast suffer from several flaws and are not widely deployed. In contrast, P2P networks have low cost and are inherently scalable; they can leverage the bandwidth of many peers, those receiving part of the content providing it to other peers, and thus reduce the load of the primary servers.

In this paper, we study the problem of the distribution of some (possible streaming) content from a single source to a large number of clients using P2P mechanisms. All interested clients start receiving the content at the same time and they cooperate with each other in order to maximize the distribution efficiency. The main metric we consider is the average time for each of the clients to receive the complete content. Earlier studies [1, 10] have developed analytical models and indicated theoretical limits for this problem, but they only considered homogeneous scenarios where all the peers have identical bandwidth. In particular, a comparison of several distribution architectures based on linear chains, trees, and parallel trees, has indicated that performance can be maximized if all the peers can use their upload capacity and the content is split in enough small blocks so that the peers are all active at the same time.

In contrast, in this research, we aim at providing techniques that are *efficient* in heterogeneous settings, *adaptive* so as to tolerate runtime changes like bandwidth fluctuations, and *practical* enough to be implementable in real systems. For the sake of simplicity, our study mostly focuses on architectures with binary trees; the principles and algorithms presented here do, however, also apply to other architectures, as will be discussed later in the paper.

Our contributions are as follows: We first analyze the problem of cooperative distribution of content from a single source to a large number of heterogeneous clients and

we identify the limitations of existing solutions. We propose techniques and algorithms that dynamically optimize the distribution network, based on the observed effective bandwidth capacities, in order to avoid bottlenecks and improve global throughput. These algorithms have several desirable features. Most notably, they are fully decentralized and work by only performing local reorganizations; as such, they might stop short of producing an optimal configuration, but perform extremely well under the aforementioned constraints. We analyze the properties of our algorithms and we evaluate them by the means of simulations, as well as experimentally in a LAN and in the Internet using the PlanetLab [8] testbed.

The rest of this paper is organized as follows: We first discuss related work in Section 2. We then present classical tree-based distribution architectures and analyze their shortcomings in Section 3. Section 4 introduces the principles, mechanisms, and algorithms proposed to dynamically improve the efficiency of tree-based content distribution. Section 5 presents results from simulations and experimental evaluation, and Section 6 concludes the paper.

2. Related Work

Many architectures for content distribution have been proposed. Most of these systems build an overlay network that is kept throughout the distribution process. Links are only changed if either a neighbor fails or the performance heavily degrades. Affected nodes then simply rejoin the tree starting at the root. Most architectures do not actively reconfigure links before a degradation occurs.

CollectCast [5] is an example of such a passive system. The authors propose an architecture that works on two different sets of nodes for media streaming. From a set of potential senders the best ones are taken and form the active set. The other potential senders are kept in a standby set. During the streaming process peers do passively measure bandwidth and latency. If the quality of the media streaming falls below a threshold, a peer from the active set is exchanged with one from the standby set. A similar exchange technique has been proposed in GnuStream [7] for use with the Gnutella system.

Other systems like Scattercast [3] try to construct near-optimal distribution trees in advance. A set of agents is deployed across the network. The agents together provide a multicast service. The number of clients that join an agent is limited by its bandwidth capacity. The goal of Scattercast is to construct a degree-constrained spanning tree across all agents and keeping the average delay between the source and all destinations at a minimum. This problem is known to be NP-hard.

One system which dynamically adapts to the network conditions was presented with TMesh [9]. The architecture aims at reducing latencies between nodes in a multicast group. Based on a set of heuristics, new links are added to the existing tree or mesh. If the new link reduces the overall latency then it is kept; otherwise, it is dropped.

We believe that the limiting factor in a content distribution system is not latency but bandwidth, more precisely upload bandwidth. Therefore our first goal is to optimize bandwidth usage rather than minimize the latency between the nodes. To the best of our knowledge, no other research has explicitly studied the problem of decentralized algorithms for dynamic reorganization of P2P content distribution networks with the goal of optimizing bandwidth usage in heterogeneous settings.

3. P2P Content Distribution

Tree-based Architectures. Different architectures have been developed for organizing clients in a P2P fashion for cooperatively distributing content, e.g., a large file. The key idea is to have clients that have already downloaded the file help redistribute it to other clients, instead of relying on a single source. The time necessary to send the file to all peers is not anymore proportional to the number of peers in the network as for classical client-server distribution, but proportional to the logarithm of the number of peers.

As an example, consider the situation where a server must replicate a critical file, e.g., an antivirus update, to all $100,000$ machines of a large company. Given a file size of 4 MB and a server (client) bandwidth capacity of 100 Mb/s (10 Mb/s) with 90% link utilization, a classical client/server distribution protocol would distribute the file by iteratively serving groups of 10 simultaneous clients in $u = \frac{32\,\text{Mb}}{9\,\text{Mb/s}} = 3.55$ seconds. Updating $100,000$ clients would thus necessitate $\frac{100,000}{10}u$, i.e., almost 10 hours.

In contrast, cooperative distribution leverages the bandwidth of the nodes that have already obtained the file, thus dynamically increasing the service capacity of the system as the file propagates to the clients. As each client that has already received the file can serve another client while the server updates 10 new clients, we can compute the number of clients updated at time t as $n(t) = 2n(t-u) + 10 = 2^{\lfloor t/u \rfloor}10 - 10$. Updating $100,000$ clients would thus necessitate less than 1 minute. The exponential increase of the number of served peers provides a sharp contrast with the linear progression of traditional client/server distribution (see [4] for a more detailed analysis).

The simplest architecture for cooperative content distribution consists in forming a chain (or pipeline) in which each client downloads the file from one peer and uploads it to another peer. The file is divided into small blocks of a given size that can be transmitted independently from each other: as soon as a block is received at one peer, it is forwarded to

the next peer. This architecture leads to impressively short distribution times in high speed networks with full duplex connectivity. The total distribution time is essentially the time to send the whole file to the first node plus the delay for the first block to reach the last node.

If each peer serves more than one other peer, we obtain trees instead of linear chains. As the bandwidths of upload connections have to be shared between several downloaders, such architectures are best adapted in settings where peers (especially those close to the source) have large upload capacities.

Chains and tree architectures have the disadvantage that the failure of a node adversely impacts the whole subtree rooted at that node. Indeed, once the only link to the subtree is broken, no data can flow to any of its peers. To address this problem, one can organize the peers into multiple spanning trees, with each peer belonging to all the trees and being interior node of at most one of them, and have the source send distinct blocks to each tree. Such architectures based on parallel trees have been used in SplitStream [2] to improve bandwidth efficiency and increase robustness. Obviously, the failure of a peer will affect at most one of the distribution trees and leave the rest operational. Analytical models and analysis of these architectures in homogeneous settings can be found in [1]. We shall primarily focus on architectures based on a single binary tree in the rest of the paper, although we shall briefly discuss extensions for n-ary and parallel trees.

Dealing with Heterogeneity. The performance of content distribution using a single tree composed of peers with heterogeneous bandwidth directly depends on the organization of the nodes in the tree. One slow peer p_s can increase the average reception time of all the peers in the subtree rooted at p_s, even if they have more bandwidth and computational power than p_s.

To show the effect of a single slow peer p_s in a balanced binary distribution tree of n nodes, we compute the average reception time depending on the height of p_s in the tree. We assume a symmetric bandwidth of B_f for the fast peers and the source S, and $B_s < \frac{B_f}{2}$ for the slow peer. To distribute a file of size F, we divide it into blocks and send them along the tree as a continuous stream of data. We shall neglect the delay of the first block to reach the bottom of the tree, as its impact on the average reception time of the file by the peers is negligible. We also assume that each peer stores the file locally and, hence, does not need to buffer communication flows. If we construct a tree composed only of fast nodes, each of them downloads the file in time:

$$T = \frac{2F}{B_f}$$

Distribution occurs at half the available bandwidth because each peer has to serve two other peers on a single link. T, in this case, also corresponds to the average download time $\overline{T}$ among all the peers.

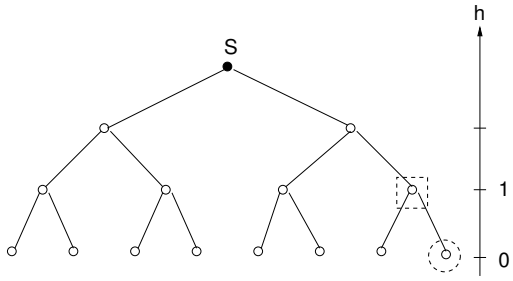

Figure 1. Positions of a slow node in the binary tree.

If we have now one slow peer at the bottom of the tree (node surrounded by a circle in Figure 1), then $\overline{T}$ becomes:

$$\overline{T}(n) = \frac{n-2}{n}\frac{2F}{B_f} + \frac{1}{n}\frac{F}{B_f - B_s} + \frac{1}{n}\frac{F}{B_s}$$

The first term in the equation refers to the peers which are not affected by the bandwidth limitation of the slow peer. The second and the third term correspond to the download times of the sibling of the slow peer and the slow peer, respectively.

If the slow peer is at the second level from the bottom of the tree (node surrounded by a square in Figure 1), $\overline{T}$ now becomes:

$$\overline{T}(n) = \frac{n-4}{n}\frac{2F}{B_f} + \frac{1}{n}\frac{F}{B_f - B_s} + \frac{1}{n}\frac{F}{B_s} + \frac{2}{n}\frac{2F}{B_s}$$

Again, we have the unaffected peers in the first term, and the second and third terms refer to the sibling of the slow peer and the slow peer itself. The last term corresponds to the download time for the children of the slow peer.

If we generalize the average download time per peer depending on the height h (from the bottom of the tree) of the slow peer, we get the following expression for $\overline{T}$:

$$\overline{T}(n,h) = \frac{F}{n}\left(\frac{(n-2^{h+1})2}{B_f} + \frac{1}{B_f - B_s} + \frac{1}{B_s} + \frac{(2^{h+1}-2)2}{B_s}\right)$$

As previously mentioned, the download time can be improved when using parallel trees, with each peer being interior node of at most one of the trees (only one peer can be a leaf of all trees). In such architectures, the position as interior node of the slow peer will also affect the average

performance of file distribution. With a parallel binary tree configuration, half of the file is sent in parallel to each tree and the download performance is obviously limited by the tree in which the slow peer is an interior node, i.e., has the highest position. In that case, we can compute the average download time as:

$$\overline{T}_{\parallel}(n,h) = \frac{F}{n}\left(\frac{n-2^{h+1}+1}{B_f} + \frac{2^{h+1}-1}{B_s}\right)$$

In Figure 2 we can see the effect in binary tree configurations of one or two slow peers depending on their height. Computations were performed with $B_f = 100$ Mb/s, $B_s = 10$ Mb/s and 1 Mb/s, $F = 650$ MB, and $n = 2^{17} - 1$ peers (including the source). In settings with two slow peers, each of them was in a different subtree from the source. Figure 2 shows a clear exponential increase in the average reception time $\overline{T}$, both with single and parallel trees, after the height of the slow nodes reaches approximately half the depth of the tree. This clearly demonstrates the necessity of dynamically reorganizing distribution trees to adapt to the effective bandwidth of the peers.

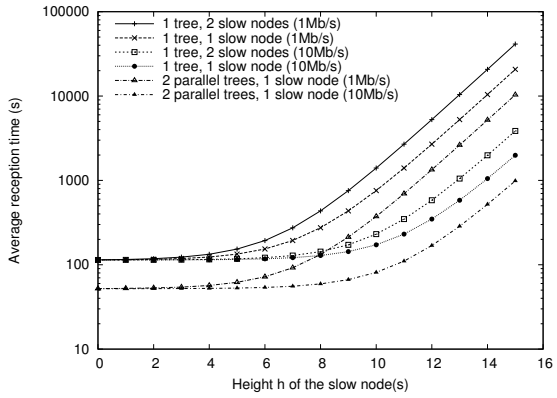

Figure 2. Average reception time depending on the height of slow nodes.

4. Dynamic Reorganization Algorithm

Motivations and Design Guidelines. In the previous section we have shown that, in heterogeneous settings, slow nodes should be located as deep as possible in tree-based content distribution architectures. Indeed, a slow node is a bottleneck for its whole subtree and the higher the slow node is, the more peers its subtree contains.

Therefore, our goal is to design an algorithm that dynamically optimizes the distribution tree by reorganizing peers according to their effective bandwidth. This directly raises the problem of estimating bandwidth capacities and moving peers at runtime in a practical and efficient manner.

Our algorithm was designed according to several guidelines: it should be fully distributed and symmetric, and not rely on a centralized entity (besides the data source that has a specific role); all operations and reorganizations should be performed locally or in the close neighborhood of a peer; decisions should be based on local information and no global knowledge should be necessary; the algorithm should be able to adapt dynamically to changes in the network; and the complexity and overhead should remain as low as possible.

These guidelines comply with the P2P design philosophy and are key to achieve high scalability. A consequence of the constraints they impose is that our algorithm may not yield an optimal configuration, which would necessitate non-local information and operations, as we shall discuss shortly.

Bandwidth Measurements. The limiting factor in most file distribution networks is the upload capacity of the nodes, which is typically lower than their download capacity (e.g., ADSL). Therefore we based our algorithm on the upload capacities of the nodes and we reorganize the peers when we detect nodes that have lower upload capacities than some of their children.

Each peer p must be able to estimate its upload capacity u. To that end, a node actively or passively measures the throughput u_i achieved when uploading data only to child i, and the throughput u_n obtained when uploading data simultaneously to all m children (see Figure 3). Further, let $d_i > 0$ be the download capacity of child i.

Based on these measurements, we can distinguish two cases:

1. $u_n < \sum_{i=1}^{m} u_i$ with $u_j = u_n$ for some nodes j and $u_k < u_n$ for some nodes k
2. $u_n = \sum_{i=1}^{m} u_i$

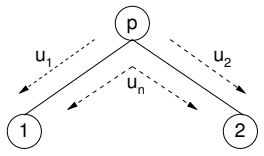

Figure 3. Throughput measured to estimate effective bandwidth.

In case 1 the transfer bandwidth is limited by the upload capacity of peer p. The upload capacity to all nodes u_n is not higher than the upload rate to a subset of its children.

We estimate the upload rate of p to be $u = u_n$. We also know that each child j has a download rate of $d_j \geq u_j$ and each child k has a download rate of $d_k = u_k$.

Case 2 occurs if the upload capacity u of p is not the limiting factor. The children are all downloading at their limits. We have $u \geq u_n$ and we know that each child i has a download rate of $d_i = u_i$.

Based on these estimations, a peer can easily compare its upload capacity with that of its direct neighbors to determine whether local reorganizations are necessary.

The *HeapTop* Algorithm. *HeapTop* is remotely inspired from the well-known HeapSort algorithm, where the nodes of a tree are reorganized by exchanging selected father-child pairs. The goal is to move the nodes with highest bandwidth closest to the root of the tree. The property maintained by our algorithm is that, for every node p other than the root and every child c of p, we have $u_p \geq u_c$ (with u_p and u_c being the effective upload bandwidth of p and c, respectively).

As we only want to perform local operations, the only way we can reorganize the tree is by exchanging the position of a node with its parent. This operation can be easily implemented because both nodes are directly connected with each other and they essentially have to exchange their respective neighbors.

The algorithm starts with a random initial tree. We assume that all nodes in the tree can estimate their bandwidth capacity and that of their parent, as previously discussed.

Algorithm 1 *HeapTop* algorithm at peer p

```
1: loop
2:    q ← Parent(p)
3:    if q ≠ root and Bandwidth(q) < Bandwidth(p) then
4:       Exchange positions of p and q
5:    end if
6: end loop
```

Each node continuously executes the trivial operations shown in Algorithm 1. Peer p periodically compares its bandwidth capacity with that of its parent. If p's bandwidth is strictly bigger than its parent's bandwidth, then they switch positions, i.e., they exchange their neighbors. This operation can be performed efficiently as it is essentially local to p and its parent. The algorithm preserves the structure of the initial tree (even if it is not balanced), but the position of the nodes evolves over time.

For avoiding pairwise exchanges resulting from short bandwidth fluctuations, the estimations are based on a weighted moving average computed using the following formula:

$$u(t) = (1 - \alpha) \cdot u(t - 1) + \alpha \cdot u$$

The average bandwidth at time t is obtained by combining the latest sample u with the previous average value. The constant $\alpha \leq 1$ (typically $\frac{1}{8}$) is a smoothing factor that puts more weight on recent samples than on old samples and smooths out important variations.

In addition, in order to prevent unnecessary reorganizations of peers with similar bandwidth capacities, we shall only exchange the position of a peer p and its parent q if $u_q < \beta \cdot u_p$, with $\beta \leq 1$ (typically $\frac{9}{10}$).

Note there is no synchronization between the peers (except between pairs of neighbors when positions need to be exchanged). This implies that nodes can move upward or downward the tree at different speeds, and distinct configurations can be obtained from the same initial tree. Figure 5 shows a possible configuration obtained from the execution of the algorithm on the tree in Figure 4 (the numbers indicate the bandwidth capacities of the peers: large numbers correspond to high bandwidth).

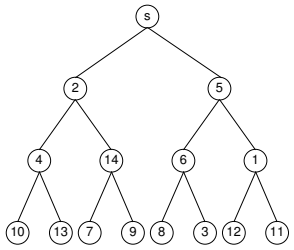

Figure 4. Original distribution tree (larger numbers mean more bandwidth).

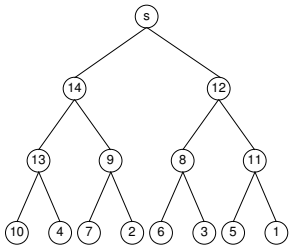

Figure 5. One possible configuration obtained from executing the algorithm.

Given the special role of the root node, it appears clearly that the peers cannot move from one 1^{st}-level subtree to another 1^{st}-level subtree. Further, within any subtree, a node in one branch may be further from the root than some other node with less bandwidth in another branch (see nodes 9 and 10 in Figure 5). As such, the resulting distribution tree may be slightly sub-optimal but performing further optimizations would necessitate non-local operations and higher complexity.

If there is no bandwidth fluctuation, the tree will quickly reach a stable configuration. In the worst case, a node located at depth $d \geq 1$ (the root is at depth 0) can initiate $d - 1$ exchanges. The actual number of exchanges depends on both the initial configuration of the tree and the order in which exchanges are performed.

Note that this algorithm can also be used with architectures based on parallel trees. Node exchanges are performed concurrently in each of the trees. If one wishes to meet the robustness property that a peer should be interior node of at most one tree, we lose some flexibility in the way the trees

can be organized: exchanges can only be performed if the robustness property still holds after the operation (only interior nodes can be freely exchanged). Although the resulting architecture provides better resilience to failures, it will be sub-optimal in terms of bandwidth efficiency.

5. Evaluation

Simulation Setup. For evaluating the behavior of *HeapTop* in different environments, we implemented a Java simulator that faithfully reproduces the operations of the algorithm and evaluates its efficiency. The main criterion considered is the average upload bandwidth capacity using the tree generated by *HeapTop*, as compared with that of the initial randomly generated tree and of an optimal tree.

We have simulated three main classes of peers, chosen to match the observations we have made of real-world populations in an earlier study of the BitTorrent protocol [6]. These classes represent effective connection throughputs frequently encountered in the Internet:

- F: fast nodes with 1024 Kbit/s upload bandwidth.
- M: medium-speed nodes with 512 Kbit/s upload bandwidth.
- S: slow nodes with 128 Kbit/s upload bandwidth.

As previously mentioned, the upload bandwidth is the limiting factor and we do not explicitly take into account download capacities (peers of classes M and S typically have asymmetric bandwidth).

Each peer has a given probability to fall in one of the considered classes. Binary trees are constructed by iteratively adding each node at a valid position, chosen by traversing the tree from the root until a leaf or a node with a single child is encountered. We experimented with both unbalanced and balanced trees. As the differences in the measurements were negligible, we only show results for balanced trees and note that they are also valid for unbalanced trees.

For comparison with an optimal configuration, a tree was constructed by organizing the nodes from root to leaf in decreasing order of upload capacity. Each result is the average of 50 executions.

Simulation Results. We have first evaluated the improvement factor of *HeapTop* with different population sizes and various proportions of nodes in each class. To that end, we have used the class distributions shown in Table 1.

The improvement factor f is defined as the ratio of the average bandwidth $\overline{B}_{HT}$ of the tree generated by *HeapTop* to the average bandwidth $\overline{B}_R$ of the random initial tree: $f = \overline{B}_{HT}/\overline{B}_R$.

	Class F	Class M	Class S
$D1$	90%	5%	5%
$D2$	60%	30%	10%
$D3$	50%	25%	25%
$D4$	30%	60%	10%
$D5$	25%	25%	50%
$D6$	5%	90%	5%
$D7$	5%	5%	90%

Table 1. Distributions of peer classes for the simulations.

Figure 6 shows that the improvement factor is significant, with *HeapTop* being as much as 6 times more efficient than the initial tree. Further, it increases with a logarithmic behavior as the number of nodes grows. This can be explained by the analysis of Section 3, which showed that with a single slow node located at height h, the performance of the whole network degrades as a function of 2^h. As the height of a binary tree composed of n peers is proportional to $log(n)$, the logarithmic shape of the improvement factor is not surprising.

One can also observe that the difference between *HeapTop* and the initial tree decreases when there are many slow peers, as there is less room for optimization (some slow peers must end up as interior nodes).

Another desirable property of *HeapTop* is to maintain the average number of exchanges per node as small as possible. As one can see in Figure 7, this value mostly depends on the class distribution and is not higher than 1.2 exchanges per node. The size of the peer population, i.e., the tree depth, has little impact on that metric.

As discussed in Section 4, *HeapTop* does not generate optimal trees because it only performs local reorganizations. Figure 8 shows that the constructed trees are extremely close to the optimum (more that 0.95 for most configurations) and do not depend much on the size of the peer population. Taking into account the simplicity and efficiency of the algorithm, this is clearly an acceptable approximation of the optimal tree.

We also simulated *HeapTop* with an architecture based on parallel binary trees. As in SplitStream, we enforced each peer to be inner node of at most one of the trees. After generating both trees, *HeapTop* was run on the inner nodes of each tree. Figure 9 shows the improvement factor for different population sizes and various class distributions. One can observe that the gain is still significant (up to almost 400%). Further, the relative performance of the class distributions is different than for a single tree because only interior nodes can be reorganized. Figure 10 shows the best improvement factor observed during the simulations (up to 750%) and gives a measure of the potential benefits of *HeapTop* for parallel trees.

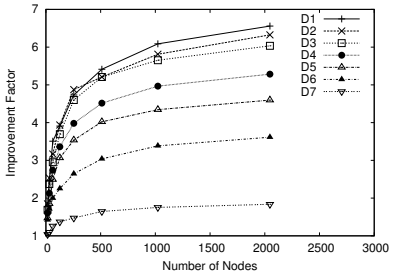

Figure 6. Average improvement factor for different population sizes and various class distributions.

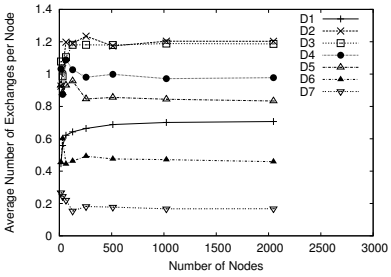

Figure 7. Average number of exchanges per node for different population sizes and various class distributions.

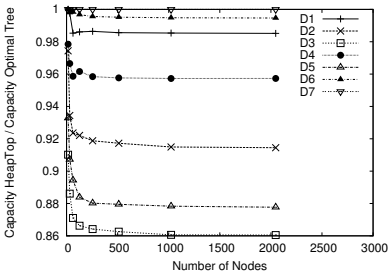

Figure 8. Bandwidth capacity of the *HeapTop* tree vs. an optimal binary tree for different population sizes and various class distributions.

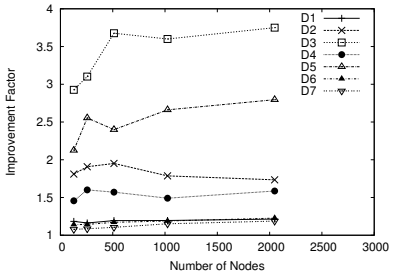

Figure 9. Average improvement factor with two parallel trees for different population sizes and various class distributions.

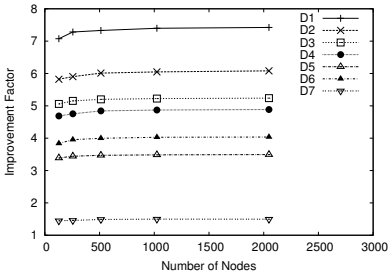

Figure 10. Best case improvement factor for two parallel trees for different population sizes and various class distributions.

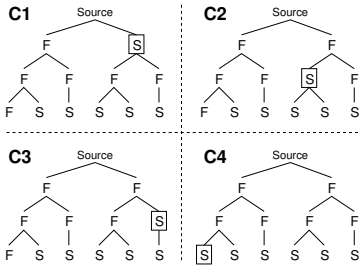

Figure 11. Configurations for average reception time tests with `crcp`.

Experimental Setup. To evaluate experimentally the effect of the *HeapTop* algorithm, we have developed a content distribution tool called `crcp` (cooperative remote copy) that implements P2P replication of files to large populations of hosts. Each file is split in blocks that are sent independently on encrypted connections. The current version of `crcp` supports linear chain and tree architectures, which are dynamically constructed by the source when initiating file replication.

Experimental Results. We have first evaluated our mechanisms in a local area network (LAN), with 13 Linux computers connected to a switch, one of them acting as the source and the rest as clients. Six of the client peers had network cards configured at 10 Mb/s, the other 6 and the source at 100 Mb/s. The file to distribute had a size of 564 MB. To demonstrate the efficiency of the *HeapTop* algorithm the file was distributed on trees according to the four configurations on Figure 11, where a slow node moves down the tree to improve distribution efficiency.

The average reception times are shown in Figure 12. As expected, file distribution becomes more efficient when the slow node is deep in the tree. This confirms that the *HeapTop* algorithm achieves better performance for file distribution than a fixed tree construction by moving slow nodes far from the source.

We have then performed large-scale experiments with `crcp` on 25 hosts of the PlanetLab infrastructure and compared the performance of initial random trees and the trees obtained using the *HeapTop* algorithm. Although we observed some variance in the experiments, due to load fluctuations in the network and at the nodes, *HeapTop* produced trees that were systematically faster than the initial configurations, with an average improvement factor of 1.55 and peaks of over 1.70.

A careful look at the reception times of each of the nodes helps us to understand the reason for this improvement. Figures 13 and 14 show the performance of individual peers,

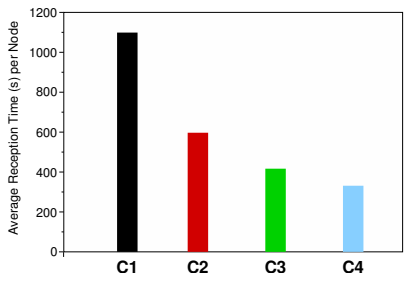

Figure 12. Average reception times with a slow node at different positions.

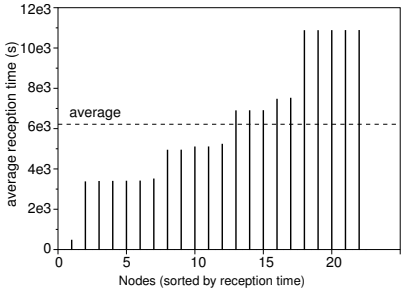

Figure 13. Reception times of the peers for the initial random binary tree.

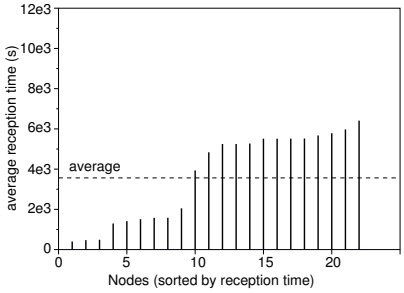

Figure 14. Reception times of the peers for the *HeapTop* binary tree.

sorted by reception times, when sending a 29 MB file to 22 hosts, for the initial and *HeapTop* trees respectively. We can observe that, in the former case, low-bandwidth peers slow down their descendant, which produces clear steps in the figure. Such bottlenecks do not appear in the latter case, as many of the peers can download the file with no speed limitations besides their own bandwidth. Further study would be necessary to observe how *HeapTop* dynamically adapts to the bandwidth fluctuations and unpredictability of the Internet, and how it could be extended to explicitly deal with failures.

6. Conclusion

P2P content distribution architectures are expected to play a big role in future distributed systems because of their impressive scalability, remarkable performance, and low cost. In this paper, we have studied the limitation of classical tree-based architectures when peers have different bandwidth capacities. We have proposed simple and efficient algorithms to dynamically reorganize the peers so as to optimize distribution efficiency. These mechanisms are adaptive, decentralized, and only perform local reorganizations; as such, they follow the P2P design philosophy and are extremely scalable. We have extensively studied their effectiveness by the means of simulations and experimentations and we have observed significant efficiency gains (up to more than 600%) depending on the number of peers and their respective bandwidth. These results demonstrate the importance of explicitly taking into account bandwidth limitations and fluctuations in P2P content distribution architectures, in order to avoid wasting the most essential resources of the network—the service capacity of the peers.

Acknowledgements. This work is supported in part by the Swiss National Foundation Grant 102819.

References

[1] E. Biersack, P. Rodriguez, and P. Felber. Performance analysis of peer-to-peer networks for file distribution. In *Proceedings of the 5th International Workshop on Quality of future Internet Services (QofIS'04)*, pages 1–10, Sept. 2004.

[2] 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 the ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2003.

[3] Y. Chawathe. Scattercast: An adaptable broadcast distribution framework. *Multimedia Systems*, 9(1):104–118, 2003.

[4] P. Felber and E. Biersack. *O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, M. van Steen (Eds.): Self-Star Properties in Complex Information Systems*, chapter Cooperative Content Distribution: Scalability through Self-Organization, pages 343–357. Springer-Verlag, 2005.

[5] M. Hefeeda, A. Habib, B. Boyan, D. Xu, and B. Bhargava. PROMISE: peer-to-peer media streaming using CollectCast. In *Proceedings of ACM Multimedia 2003*, Nov. 2003.

[6] M. Izal, E. Biersack, P. Felber, G. Urvoy-Keller, A. A. Hamra, and L. Garces-Erice. Dissecting BitTorrent: Four months in a torrent's lifetime. In *Proceedings of the 5th Passive and Active Measurement Workshop*, Apr. 2004.

[7] X. Jiang, Y. Dong, D. Xu, and B. Bhargava. Gnustream: A P2P media streaming system prototype. In *Proceedings of the International Conference on Multimedia and Expo (ICME),*, July 2003.

[8] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. In *Proceedings of HotNets-I*, October 2002.

[9] W. Wang, D. A. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *Proceedings of the International Workshop on Networked Group Communications (NGC)*, Oct. 2002.

[10] X. Yang and G. de Veciana. Service capacity of peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, Mar. 2004.