Angels: In-Network Support for Minimum Distribution Time in P2P Overlays

Raymond Sweha remos@cs.bu.edu
Azer Bestavros best@cs.bu.edu
John Byers byers@cs.bu.edu

Abstract

This paper proposes the use of in-network caches (which we call *Angels*) to reduce the Minimum Distribution Time (MDT) of a file from a seeder – a node that possesses the file – to a set of leechers – nodes who are interested in downloading the file. An Angel is not a leecher in the sense that it is not interested in receiving the entire file, but rather it is interested in minimizing the MDT to all leechers, and as such uses its storage and up/down-link capacity to cache and forward parts of the file to other peers. We extend the analytical results by Kumar and Ross [1] to account for the presence of angels by deriving a new lower bound for the MDT. We show that this newly derived lower bound is tight by proposing a distribution strategy under assumptions of a fluid model. We present a *GroupTree* heuristic that addresses the impracticalities of the fluid model. We evaluate our designs through simulations that show that our GroupTree heuristic outperforms other heuristics, that it scales well with the increase of the number of leechers, and that it closely approaches the optimal theoretical bounds.

1 Introduction

1.1 Motivation

Motivated by P2P file swapping applications, the BitTorrent protocol [2] has established swarming, i.e., parallel download of a file from multiple peers with concurrent upload to other requesting peers, as one of the most efficient methods for multicasting bulk data.
In this paper, we consider a class of applications which require a set of nodes (leechers) to finish downloading a whole content from one or more nodes (seeders) *before* any of them can start using this content. Under the Bulk Synchronous Model (BSP) [3], such a download can be seen as a barrier synchronization for all nodes. Example applications that require this type of barrier synchronization include publish-subscribe networks (where content needs to be consistent across multiple mirror sites before it can be used), distributed backup/commit (where local processing must wait for global communication to be completed), real-time upgrade of software, and real-time distribution of network (re)configuration data (where switching to the new version cannot be done until all nodes are ready to do so).

1.2 Summary of Contributions

Due to the nature of the applications we consider, our overarching requirement is the minimization of the time it takes for the file to be distributed to all leechers, as opposed to other considerations such as average distribution time, or fairness, etc. [1] derived a theoretical lower-bound on the *Minimum Distribution Time* (MDT) of a file to a set of leechers and provided a fluid model that achieves this bound. Their result suggests that the MDT is governed by one of three potential bottlenecks in the P2P overlay: (1) the upload capacity of the seeder, (2) the download capacity of the slowest leecher, or (3) the aggregate network upload capacity of all the nodes in the swarm. We note that the third of these bottlenecks – namely the *network upload capacity* – is likely to be the most prevalent in many settings, due to the asymmetry between the upload and download capacities of most end hosts (e.g., under ADSL)[1]. Thus, in this paper we investigate the potential benefit of deploying resources to alleviate the network upload capacity bottleneck. To that end, we propose the use of helper nodes – which we call *angels*. An angel is not a leecher in the sense that it is not interested in receiving the entire file, but rather it is interested in minimizing the MDT

[1]We also note that nothing can be done "inside the network" to alleviate the first two potential bottlenecks.

to all leechers. As such an angel uses its storage and up/down-link capacity – to cache and forward parts of the file to other peers– in such a way that the network upload capacity ceases to be the MDT limiting bottleneck.

1.3 Paper Organization

This paper is organized as follows. In section 3, we extend the analytical results of Kumar and Ross [1] to account for the presence of angels by deriving a new lower bound on the MDT. Also, we show that this new lower bound is tight by proposing a distribution strategy under a fluid model assumption. In section 4, we present a *GroupTree* heuristic that addresses the impracticalities of the fluid model. Then, we evaluate our designs through simulations that show that our GroupTree heuristic outperforms other heuristics, that it scales well with the increase of the number of leechers, and that it operates near the optimal theoretical bounds. Our experimental findings suggest that, in contrast to focusing only on piece or peer selection, a heuristic that coordinates both criteria holds the potential for significant performance gains.

2 Related Work

The development of certain protocols, such as BitTorrent, transformed P2P from an interesting technology to a significant source of traffic on the Internet. According to [4], Bittorent traffic is 70% of all the Internet traffic. However, P2P file swapping, is not the "be all and end all" for swarming. Many other applications stand to benefit from the use of swarming as a basic building block for content delivery. For example, many research and technology products have adopted swarming for Content Distribution Networks (CDNs). Some products embraced the P2P paradigm in CDN such as BitTorrent DNA [5] which uses CDN to seed a P2P swarm, Pando [6] which moves from CDN to P2P after the number of subscribers exceeds a certain threshold, and Joltid [7] which uses caches to minimize inter-ISP traffic.

Leibowitz et al. [8] were the first to study the feasibility of caching P2P traffic. They contrasted the P2P traffic characteristics against HTTP traffic to determine if web-caches can work with P2P traffic. HPTP [9] used web-caches for P2P but that required changing the P2P clients to be able to contact the caches. Wierzbicki et al. [10] devised new cache replacement policies for web-caches that are optimized for P2P traffic.

Dan [11] built on Joltid to infer that cooperative caching between peering ISP's would save ISP's transit traffic. He formulates the problem as a constraint optimization problem for streams, with objective function that maximizes the number of streams that are relayed between peering ISPs, thus saving transit traffic. Saleh and Hafeeda [12] proposed using incremental caching instead of caching the whole object of streaming videos to minimize inter-ISP traffic. They found that sequential incremental caching achieves significant gains.

The aforementioned systems employ some sort of caching to enhance the performance of P2P content distribution. The following two systems address the incentive problem in providing help to P2P clients. The first one is Tribler [13]. Tribler is a social-based P2P system. It capitalizes on the trust between friends in social networks as an incentive for cooperation in content distribution. In Tribler's cooperative downloading, leechers (Collectors) recruit their idle friends (Helpers) who will help them find and download file pieces they cannot find themselves from their direct neighbors. Tribler solves the incentive problem of P2P systems, as friends take many actions based on trust instead of greedy rationalism. Over time, friends will exchange the role of Collector and Helper. However, the authors of Tribler did not consider the MDT of their system.
The second one is [14], where the authors used idle nodes as helper nodes in Bittorent. They measured their improvement based on steady-state performance and validated it with simulation. They neither discuss incentives for cooperation nor analyze the MDT of their system.

The dHCPS system [15] is similar to our GroupTree system as it builds a hierarchal P2P dissemination mechanism. The authors present a two-level hierarchy for streaming. Each cluster has a Head (the source for this cluster) which is a member of a superNode group. The source builds a P2P swarm of superNodes. First, the source swarms the file to the superNodes, then the superNodes swarm the file to nodes in its child cluster. dHCPS solves an optimization problem to decide how much upload bandwidth each head should dedicate to the superNode group versus its child group. However, this work did not specify how to cluster the nodes in the first place, which is a major difference from our proposed GroupTree system.

Our theoretical bound and fluid model construction are extensions to Kumar's work [1]. Kumar and Ross developed the lower bound of the Minimum Distribution Time of file dissemination from a seed to a set of leechers. They identified the three bottlenecks of the system. We observed that the most likely bottleneck is typically the network upload capacity. Towards this end, we introduced angels to alleviate this bottleneck, recomputed the lower bound and built a fluid model construction to achieve it. Independently, Kumar and Ross extended their work in a technical report [16], where they presented an idea similar to Angels. In this work, they proposed a two-level distribution system that cares only for the MDT of the higher-level. They developed independently a tight bound which is identical to the one presented in this paper and they developed a dissemination mechanism that achieves it. Our optimal construction is simpler than Kumar's construction. Beside finding the lower bound and achieving it, Kumar derived a bound on MDT to relax fluid model assumption when the data is fragmented. Our Simulation indicates that the cost of fragmentation is too high (Figure 8). Thus, we develop our GroupTree algorithm and show that it operates near optimal and scales well.

3 Optimal MDT Construction

3.1 Problem Statement

We aim to minimize the distribution time of a file of size F from a set of Seeders, S to a set of Leechers, L, with the help of a set of Angels, H. Following the model presented in [1], our goal is to minimize T, where $T = \max_{i \in L} T_i$ and T_i is the time it takes node i to finish the download. The only constraints in the system are the node upload and download capacities, i.e., $a(i)$ and $b(i)$, which under a fluid mode can be infinitesimally divided. We denote the data transfer rate from node i to node j as x_{ij}.
The five capacity constraints are:

$$a(S) \geq \sum_{i \in \{L,H\}} x_{si}, \text{ Seeder upload capacity} \quad (1)$$

$$a(i) \geq \sum_{j \in \{L,H\}} x_{ij}, \text{ Leecher upload capacity} \quad (2)$$

$$b(i) \geq x_{si} + x_{hi} + \sum_{j \in \{L\}} x_{ji}, \text{ Leecher download capacity} \quad (3)$$

$$a(h) \geq \sum_{i \in \{L\}} x_{hi}, \text{ Angel upload capacity} \quad (4)$$

$$b(h) \geq x_{sh} \text{ Angel download capacity} \quad (5)$$

We do not model network cross-traffic or losses. Thus, we aggregate the set of seeders and the set of angels into one seeder and one angel, with the upload/download capacity of the seeder and of the angel set to the aggregate capacities of all the seeders and of all the angels [17], respectively.

3.2 Our Fluid-Model Construction

Lemma 1. *Under the above network model, a distribution time T for a file of size F is achievable, where:*

$$T = \frac{F}{\min\{a(S), b_{min}, \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}\}} \quad (6)$$

Proof. We provide a construction that achieves the bound for T. Our construction follows three cases corresponding to the nature of the underlying bottleneck (the denominator of T).

Case A: The seeder's upload capacity is the bottleneck, i.e., $a(S) \leq \min\{b_{min}, \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}\}$. In this case, the seeder S, sends fresh data to each leecher, i, with rate x_{si}, and also sends fresh data to the angel with rate x_{sh}. Each leecher forwards data it receives from the seeder to the other $|L-1|$ leechers. Similarly, the angel forwards data it receives from the seeder to all $|L|$ leechers. Notice that once data is sent to a leecher, all the other leechers would successfully receive it as long as $x_{si} = x_{ij} \quad \forall j \in L$. Figure 1 illustrates the idea.

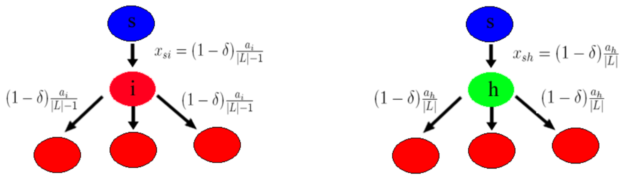

s
$x_{si} = (1-\delta)\frac{a_i}{|L|-1}$
i
$(1-\delta)\frac{a_i}{|L|-1}$
$(1-\delta)\frac{a_i}{|L|-1}$
s
$x_{sh} = (1-\delta)\frac{a_h}{|L|}$
h
$(1-\delta)\frac{a_h}{|L|}$
$(1-\delta)\frac{a_h}{|L|}$

Figure 1: Leechers and angel forward what they receive from the seeder to all the other leechers

To complete our construction, we show that there are values for x_{si} and x_{sh} that ensure that all leechers will successfully download the data with rates y_i equal to the seeder's capacity, i.e. $y_i = a(S)$, $\forall i \in L$, *without violating the upload/download capacity constraints of the various nodes.* To that end, consider the following rates:

$$
\begin{aligned}
x_{si} &= (1-\delta)\frac{a_i}{|L|-1} \\
x_{sh} &= (1-\delta)\frac{a_h}{|L|} \\
&\text{where } \delta = \frac{\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|} - a(S)}{\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}} \qquad (7)
\end{aligned}
$$

Notice that δ must be positive since we are dealing with the case where $a(s)$ is the bottleneck.

It is straightforward to verify that the choice of rates given in Eq.7 ensures that the leechers and the angel will not exceed their upload capacities (constraints 2 and 4), and that the total data sending rate of the Seeder x_s

$$
\begin{aligned}
x_s &= \sum_{i \in L,H} x_{si} = (1-\delta)(\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}) \\
&= \frac{a(S)}{\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}} * \frac{a(L)}{|L|-1} + \frac{a(H)}{|L|} = a(S) (8)
\end{aligned}
$$

satisfies the seeder's capacity constraint (constraint 1).

It is worth mentioning that since angels only need to download from the seeder, we only consider angels with download capacities greater than or equal to x_{sh} which is just a small fraction of the seeder's upload capacity, i.e. $b_h \geq x_{sh} = a(S)\frac{\frac{a(H)}{|L|}}{\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}}$. Similarly, the download rate of any leecher will be:

$$
\begin{aligned}
y_i &= x_{si} + x_{hi} + \sum_{j, j \neq i} x_{ji} \quad \forall j \in L \\
&= \sum_{j \in L,H} x_{sj} = a(S) \qquad (9)
\end{aligned}
$$

which satisfies both constraint 3 and the targeted download rate per leecher.

To summarize, when the seeder's upload capacity is the bottleneck, we showed that a rate $y_i = \min\{a(S), b_{\min}, \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}\} \quad \forall i \in L$ is achievable while satisfying the various imposed capacity constraints.

Case B: The network upload capacity (the leecher's share of the aggregate upload capacity) is the bottleneck, i.e., $\frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2} \leq \min\{a(S), b_{min}\}$. Our construction in this case requires a new dissemination strategy, which goes in two phases as outlined below.

Phase 1: The seeder divides the file into $L+1$ unequal size blocks proportional to the upload capacities of the L leechers and of the angel. Next, the seeder sends block i to the i-th leecher and block $i = L+1$ to the angel. Unlike in Case A, due to its upload capacity constraint, a leecher is not able to disseminate *all* the data it receives, from the seeder, to the other leechers. Hence leechers can only share parts of their blocks with each other, i.e. $x_{si} \geq x_{ij} \quad \forall j \in L$. Let W denote the duration of phase 1. The block sizes are chosen to guarantee that at time W each leecher would have completed the download of its corresponding block from the seeder and only parts of the other leechers' blocks.

Fig.2 illustrates this dissemination strategy, where it is clear that the leecher's download rate from the seeder differs from its upload rate to the other leechers.

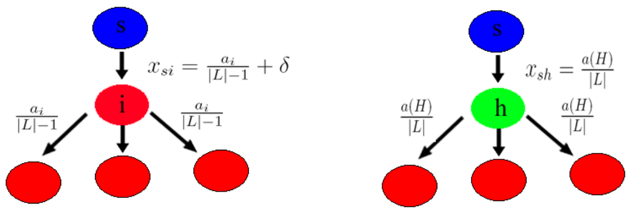


Figure 2: During phase one, the Seeder sends each leecher its perspective block which in its turn forwards part of it to the other leechers.

Phase 2: In this phase, i.e. for $t > W$, the leechers continue to send the data they got from the seeder in Phase 1 to all other leechers, while the seeder uses its upload capacity to help out with the dissemination of incomplete blocks. Figure 3 illustrates the state of file reception for leecher 1 during Phase 2.

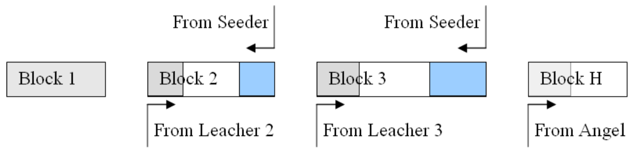


Figure 3: The state of the file reception after time W for leecher 1

To complete our construction, we propose the following rate allocation scheme, which we show to match the distribution time T in Eq.6, while satisfying imposed capacity constraints. To that end, consider the following assignments:

$$\begin{aligned} x_{si} &= \frac{a(i)}{|L|-1} + \delta \quad \forall i \in L \\ x_{sh} &= \frac{a(H)}{|L|} \\ x_{ij} &= \frac{a(i)}{|L|-1} \quad \forall i \in L, i \neq j \end{aligned} \tag{10}$$

Setting $x_{ij} = \frac{a(i)}{|L|-1}$ guarantees that leechers will not violate their upload capacity since $x_i = |L-1| * \sum_{j \in L} x_{ij} = a(i)$ (constraint 2). Likewise the angels (constraint 4). Assuming that the angel's download capacity is greater than or equal to $1/|L|$ of its upload capacity ensures the satisfaction of constraint 5. Choosing $\delta = \frac{a(S)}{|L|} - (\frac{a(L)}{|L|(|L|-1)} + \frac{a(H)}{|L|^2})$ guarantees that $\delta \geq 0$ as $a(S) \geq \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}$.

Substituting for δ, the aggregate rate by which the seeder sends data $x_s = \sum_{i \in L} x_{si} + x_{sh} = \frac{a(L)}{|L|-1} + |L|\delta + \frac{a(H)}{|L|}$ is given by: $x_s = \frac{a(L)}{|L|-1} + |L|(\frac{a(S)}{|L|} - (\frac{a(L)}{|L|(|L|-1)} + \frac{a(H)}{|L|^2})) + \frac{a(H)}{|L|} = a(S)$, thus satisfying constraint 1.

Each leecher i will download with rate $y_i = x_{si} + \sum_{\forall j \neq i \in L} \frac{a_j}{|L|-1} + x_{sh} = \frac{a(L)}{|L|-1} + \frac{a(S)}{|L|} - (\frac{a(L)}{|L|(|L|-1)} + \frac{a(H)}{|L|^2}) + \frac{a(H)}{|L|} = \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}$, thus satisfying constraint 3.

To summarize, when the network upload capacity is the bottleneck, we showed that a rate $y_i = \min\{a(S), b_{min}, \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}\} \quad \forall i \in L$ is achievable while satisfying the various imposed capacity constraints.

Case C: The download capacity of the weakest leecher is the bottleneck,
i.e., $b_{min} \leq \min\{a(S), \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}\}$.
We consider two subcases, for which we provide a proof sketch below.

- Subcase C1: $b_{min} \leq \frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}$
- Subcase C2: $b_{min} > \frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}$

Proving achievability for Subcase C1 is similar to case A. We use the same fluid model of Fig. 1, except for a change in the value of $\delta = \frac{\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|} - b_{min}}{\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}}$. Similar to Case A, setting $x_{si} = (1-\delta)\frac{a_i}{|L|-1}$ and $x_{sh} = (1-\delta)\frac{a_h}{|L|}$ and making nodes forwarding received data to the other leechers satisfies constraints 2 and 4. The amount of data the seeder sends equals: $x_s = \sum_{\forall i \in L,H} x_{si} = \sum_{\forall i \in L}((1-\delta)\frac{a_i}{|L|-1}) + (1-\delta)\frac{a(H)}{|L|} = (1-\delta)(\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}) = b_{min} \leq a(S)$ satisfying constraint 1.
The download rate of any leecher will be:
$y_i = x_{si} + \sum_{\forall j \in L,H \& j \neq i} x_{ji} = \sum_{\forall j \in L,H} x_{sj} = b_{min}$ satisfying the distribution time T in Eq.6 and constraint 3.
To satisfy constraint 5, the angel download capacity most exceed a tiny fraction of the slowest peer download capacity $b_{min}\frac{\frac{a(H)}{L}}{\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}} \leq b_h$. Thus these rates satisfies the 5 constraints and the desired leecher download rate.

Similarly, proving achievability for Subcase C2 is similar to case B, except for a change in the value of $\delta = b_{min} - (\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|})$. Again, constraints 2 and 4 follow through from the assigned rates in equations 10 because $\delta \geq 0$. Also, we assume that $b_h \geq x_{sh} = \frac{a(H)}{L}$ which is a fair assumption to satisfy constraint 5.
The rate of data send from the seeder equals:

$$\begin{aligned} \sum_{\forall i \in L} x_{si} + x_{sh} &= \frac{a(L)}{|L|-1} + |L|\delta + \frac{a(H)}{|L|} \\ &= \frac{a(L)}{|L|-1} + |L|b_{min} - |L|(\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}) + \frac{a(H)}{|L|} \\ &= |L|b_{min} - (a(L) + a(H)) + \frac{a(H)}{|L|} \end{aligned}$$

But in this case b_{min} is the bottleneck, thus:

$$\begin{aligned} \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2} &\geq b_{min} \\ a(L) + a(S) + a(H) &\geq |L|b_{min} + \frac{a(H)}{|L|} \\ a(S) &\geq |L|b_{min} - (a(L) + a(H)) + \frac{a(H)}{|L|} \\ a(S) &\geq \sum_{\forall i \in L} x_{si} + x_{sh} \\ &= x_s \end{aligned}$$

Thus the constraint 1 is satisfied.
Now, lets check the rate of download of any leecher;

y_i:

$$\begin{aligned} y_i &= x_{si} + \sum_{\forall j \neq i \in L} \frac{a_j}{L-1} + x_{sh} \\ &= \frac{a(L)}{|L|-1} + b_{min} - (\frac{a(L)}{|L|-1} + \frac{a(H)}{|L|}) + \frac{a(H)}{L} \\ &= b_{min} \end{aligned}$$

Thus each leecher downloads with rate b_{min} satisfying constraint 3 and Equation 6.

Following the same steps in the proof for cases A and B, one can show that the distribution time T in Eq.6 is achievable for subcases C1 and C2, respectively. □

3.3 The Lower Bound of the MDT

Lemma 2. *Under the above network model, the minimum distribution time, T_{min}, has a lower bound given by:*

$$T_{min} \geq \frac{F}{min\{a(S), b_{min}, \frac{a(S)}{|L|} + \frac{a(L)}{|L|} + (\frac{a(H)}{|L|} - \frac{a(H)}{|L|^2})\}} \quad (11)$$

Proof. The bound given on MDT implies that the download rate of any leecher, y_{max} is bounded as follows:

$$y_{\max} \leq \min\{b_{min}, a(S), \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}\}$$

Implying that the bottleneck could be due one of three factors:

1. The minimum download capacity: $y_{\max} \leq b_{min}$. Clearly the minimum download rate can not exceed the minimum download capacity of any leecher.
2. The uplink of the seeders: $y_{\max} \leq a(S)$. In this case the seeder's upload capacity limits the minimum download rate of peers (e.g., with very powerful leechers and a week seeder).
3. The aggregate upload capacity of the network: $y_{\max} \leq \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}$. In this case the aggregate upload capacity of the network can not saturate the average download capacity of the leechers.

Only the third of the above cases is not obvious. First, we start with the case of no angels, we know that the aggregate download rate can not exceed the aggregate upload capacity in the network:

$$\sum_{\forall i \in L} y_i \leq a(S) + a(L)$$

To utilize a fraction z_h of the upload capacity of the angel, the angel must download fresh data with a rate of at least $\frac{z_h}{|L|}$. Thus, in case of using angels, we get:

$$\sum_{\forall i \in L} y_i + \frac{z_h}{|L|} \leq a(S) + a(L) + z(H)$$

$$\frac{\sum_{\forall i \in L} y_i}{|L|} \leq \frac{a(S)+a(L)+z(H)}{|L|} - \frac{z(H)}{|L|^2}$$

Since the minimum is always less than the mean and the upper bound of $y_{\max}$ is achieved when $z(H) = a(H)$, we conclude that $y_{\max} \leq \frac{a(S)+a(L)+a(H)}{|L|} - \frac{a(H)}{|L|^2}$. □

3.4 The Optimality of our Construction

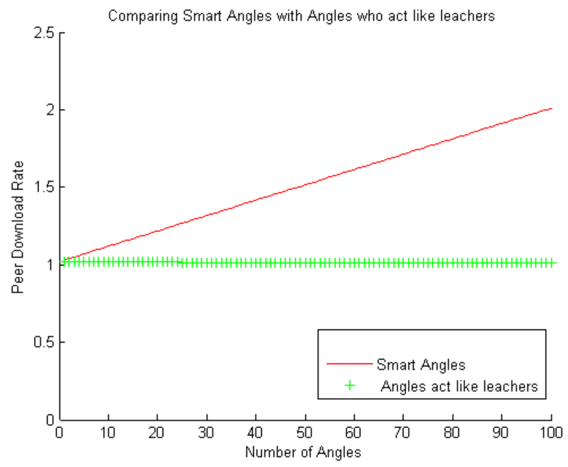
Theorem 1. *The minimum distribution time, T_{min}, given in Lemma 2 is tight.*

$$T_{min} = \frac{F}{min\{a(S), b_{min}, \frac{a(S)}{|L|} + \frac{a(L)}{|L|} + (\frac{a(H)}{|L|} - \frac{a(H)}{|L|^2})\}} \quad (12)$$

Proof. The proof follows directly from the fact that the construction given in Lemma 1 achieves the lower bound stated in Lemma 2. □

3.5 On the Usefulness of Angels

So far in this section, we showed that when the bottleneck is in the aggregate upload capacity of the network and not in the source (seeder upload capacity) or the sink (slowest leecher download capacity), adding more resources (namely, angels) will improve the MDT. But why can we not treat angels as leechers? Would making angels behave like leechers result in a similar reduction of the MDT? Figure 4(a) compares the maximum upload rate (the inverse of MDT) in two settings. In the first, angels behave as prescribed in the construction given in Lemma 1. In the second, angels behave as if they were additional leechers, interested in downloading the entire file. In both settings, the angels are as powerful as leechers. The results in Figure 4(a) suggest that making angels behave as additional leechers does not yield much gains, whereas making angels behave as prescribed in Lemma 1 yields a linear speedup (with the increase in the number of angels).



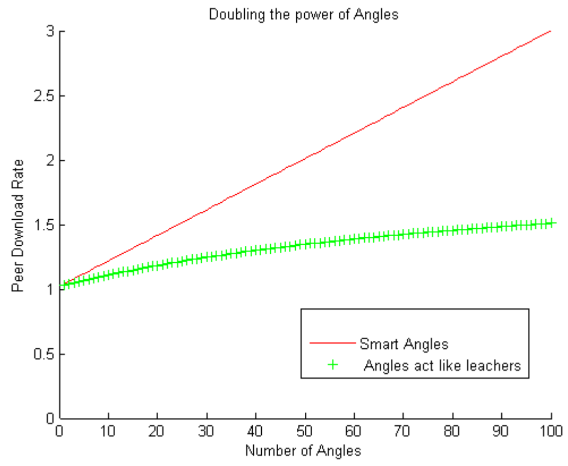


Figure 4: Comparing Angels who act like leechers to Angels who act optimally.

Figure 4(b) shows the result of the same experiment while doubling the angels' upload capacity. Under our construction, this increase in capacity translates to a linear speedup, suggesting that the angels' resources are effectively used to improve MDT. When angels behave as added leechers, we observe a speedup as well, but the speedup is sub-linear, suggesting an inefficient use of the deployed resources.

4 Performance Evaluation

4.1 Experimental Setup

We simulate the dissemination of one file, seeded by one seeder to a varying number of leechers, with help from a varying number of angels. Our simulation is done at the session layer, thus ignoring transport layer effects, e.g., due to packet loss or cross traffic. Each node in the swarm (i.e., leecher, seeder or angel) is characterized by two parameters: the node's upload capacity, a, and its download capacity, b. The file to be disseminated is of size F, divided into blocks of size B. Both F and B are parameters of the simulation. We assume that angels have no limitations on local storage (disk space). Furthermore, we assume that the network is of infinite capacity, i.e., any network bottlenecks are captured by the upload and download capacities of the various nodes in the swarm. We assume that there is an upper bound on the number of outgoing connections that a node may have (which could be set to infinity if no such constraint is required). Each node can change its neighbors periodically, or when it realizes that there is no need for maintaining such connection any longer.

We implement a discrete-event simulator that simulates a seeder, a set of angels, and a set of leechers downloading a file consisting of a set of blocks. Each node builds a set of *connections* to some or all the other nodes. Each connection has a queue of blocks to be transferred sequentially over that connection (the delivery of one block marks the beginning of the transfer of the next block). Upon receipt of a block, a node decides whether or not it should forward that block to other peers. A connection is terminated as a result of one of two events: either the expiration of a randomized timeout parameter, or when there are no more blocks to transmit over the connection. Upon the termination of a connection, a node establishes a new connection possibly to a new peer, if necessary.

4.2 Practical MDT-based Heuristics

The construction we provided in the previous section makes two strong assumptions: (1) nodes have no out-degree constraints – namely a node may disseminate content to all leechers in the system at the same time, and (2) network communication is fluid as opposed to packetized. To be practical, we implemented a number of heuristics that relax both of these assumptions.

Assuming that a node can connect to an unlimited number of nodes is impractical due to TCP and OS

Kernel constraints. Instead we assume that a node may have up to k concurrent connections at any point in time. A natural heuristic along these lines would be a *Recursive MDT* construction, where the seeder uses our optimal construction to transfer the whole file to k nodes, and recursively, each node which has the file connects to k new nodes and seed them, etc. Using *Recursive MDT* requires $(1+\log_{k+1}|L|)$ rounds, each of duration T_{MDT}. It is important to note that until the last round, i.e. until $t = T_{MDT} * \log_{k+1}|L|$ the number of inactive nodes is $\frac{k}{k+1} * |L|$, i.e. they do not send or receive data, which is such a loss in capacity. Our second heuristic, the *GroupTree* heuristic, addresses this problem by trying to minimize the activation time for all peers.

Our *GroupTree* heuristic works in two phases as depicted in figure 6. In Phase 1, the nodes form k binary trees. All the nodes in a tree have similar (matched up) upload capacities. The seeder divides the file into k blocks of sizes proportional to the upload capacities of the nodes in the tree, and seeds each tree with its block. Once a node receives a packet, it forwards it to its children in the tree. In Phase 2, each k nodes from different k trees, having different k blocks of the file, form a group. Each group uses our construction to disseminate the file between its members. Angels form one of the trees and are thus responsible for one of the blocks. In phase 2, each angel sends data to the leechers in its group without receiving any data. If the number of angels is small, angels can time-multiplex their capacities between multiple groups. Theoretically, the optimal fan-out of each tree should be the natural number, e. Experimentally, we found the optimal fan out to be 2.
Proof: The depth of an m-ary try, i.e. a group tree of fan-out m, which has N peers is $d = \lceil \log_m(\frac{N}{k}+1)\rceil$. The time needed to distribute a chunk of size F from a peer with upload capacity of c to m children will be $\frac{F}{c} * m$, which is linear in m. This happens because as we increase the fan-out each child's share of its parent's bandwidth will decrease.
$\therefore$ T, the time to disseminate $\frac{1}{k}$ of the file to each peer, will be:

$$\begin{aligned} T &= \frac{F}{c} * m * d \\ &= \frac{F}{c} * m * \lceil \log_m(\frac{N}{k}+1)\rceil \\ T &\geq \frac{F}{c} * m * \log_m(\frac{N}{k}+1) \\ &\geq \frac{F}{c} * \ln(\frac{N}{k}+1) * (\frac{m}{\ln(m)}) \end{aligned}$$

The optimal value of m is when $\frac{\partial T}{\partial m} = 0$

$$\frac{\partial T}{\partial m} = \frac{F}{c} * \ln(\frac{N}{k}+1) * [(\frac{1}{\ln(m)}) + (m * \frac{-1}{\ln^2(m)} * \frac{1}{m})]$$

$$\begin{aligned} At\ \frac{\partial T}{\partial m} &= 0 \\ (\frac{1}{\ln(m^*)}) + (\frac{-1}{\ln^2(m^*)}) &= 0 \\ \ln(m^*) &= 1 \\ m^* &= e \end{aligned}$$

Thus, the optimal fan-out of the group tree would be 2 or 3. We used our discrete time simulator to decide the optimal fan-out of the GroupTree. Figure 5 compares the distribution time of a file size 192 MB to an exponentially increasing number of peers. The upload and download capacities are similar the the ones used in section 4.4. The graph shows that a fan-out of 2 will be better than a fan-out of 3. From now on, we will use a fan-out of 2 whenever we use our GroupTree heuristic.

In contrast to the *Recursive MDT* heuristic, using *GroupTree* all nodes are activated after $t = T_{PKT} * \log_{k+1}|L|$ during phase 1, where T_{PKT} is the time to transfer one packet. The node activation time using the *GroupTree* heuristic is thus faster than that of the *Recursive MDT* heuristic by a factor equal to the ratio between the packet size and the file size, thus dominating in performance as the system will utilize its aggregate upload capacity much faster.

4.3 Other Heuristics

To provide a baseline against which one may evaluate the merit of our *GroupTree* heuristic, we implemented a host of other heuristics that focus on peer and piece selection strategies as opposed to relying on the MDT construction. One such heuristic is *Random*, where the sender chooses a receiver at

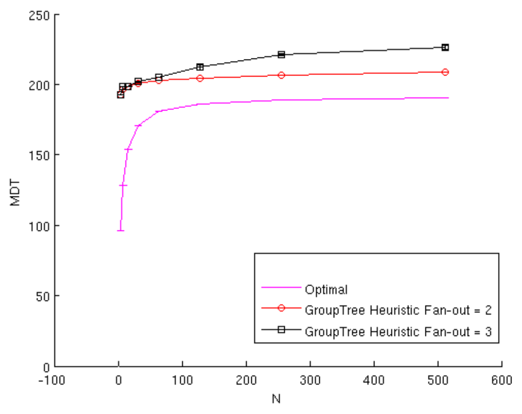


Figure 5: A comparison between our GroupTree heuristic with fan-out 2 and GroupTree heuristic with fan-out 3.

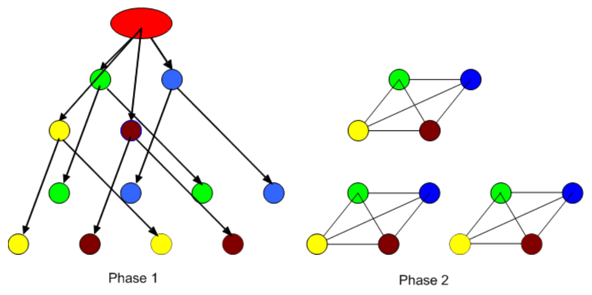


Figure 6: A binary group tree, where the nodes in each group form a complete graph.

random and sends a random piece to it given that this receiver needs this piece. This heuristic does not employ any intelligence in peer or piece selection, but in our implementation we made sure that no sender was idle while there was any receiver who could have received a piece that sender has. This heuristic fully utilizes the resources in the system but without coordination. To evaluate the benefit from smarter piece selection strategies, we implemented a Local Rarest First (*LRF*) heuristic [18], which sends the piece that the least number of the receiver's neighbors have. LRF tries to achieve balanced piece distribution depending on local information. In addition to *LRF*, we also implemented a Global Rarest First (*GRF*) strategy. While impractical, *GRF* allows us to evaluate the full benefit of optimizing piece selection depending on full piece distribution information.

To evaluate the benefit from peer selection, we implemented a *Smart-Topology* heuristic, which tries to get each leecher a fair-share of the network's upload capacity. A sender chooses the least fortunate recipient to send data to, e.g., the leecher with the least fan-in given that all the leechers have the same upload capacity. Finally, we implemented a heuristic that combines piece and peer selection. The *Smart-Topology+GRF* heuristic chooses the most needy recipient and then sends the GRF piece to it. This heuristic tries to implement intelligent piece and peer selection strategies independently.

4.4 Simulation Results

For our first experiment, we assign the following values to the parameters in our model: $F = 192kb$ with a block size of $1kb$. $a_S = 4kb/sec$, $a_i = 1kb/sec$ and $b_i = 2kb/sec$ $\forall i \in L, H$. $|L| = 128$ and the $|H|$ varies between 0 and 16 angels. Figure 7 shows the performance of our *GroupTree* heuristic versus the theoretical bound. We conclude that angels can decrease the MDT linearly with the increase in number of angels, and that our heuristic performs near optimal.

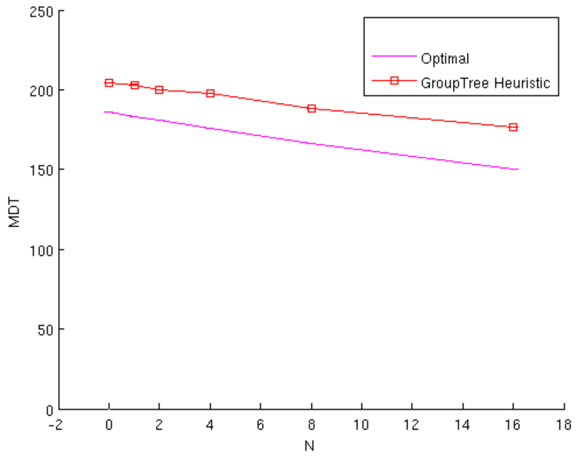


Figure 7: Comparing Optimal, VS our heuristic against an increasing number of Angels

Our second experiment compares the performance of different heuristics as the number of leechers scales up. In this experiment, we use the same set of parameters as before (ensuring that the aggregate system upload capacity is the bottleneck (case C in the MDT construction), which is the case where angels can be of help. We vary the number of leechers between 1 and 512. Figure 8 illustrates the differences in performance between the optimal solution (under the fluid model and unbounded degree assumptions), our *GroupTree* heuristic and the host of baseline heuristics mentioned above. These results show clearly that our heuristic outperforms all others and that it operates within striking distance of the optimal. In Figure 8, we also show the performance of a direct implementation of the optimal construction of section 3 (labeled *fragmentation*). Clearly, the direct implementation of the MDT construction does not scale. This is due to the fact that this construction assumes a fluid communication mode, implying that a node can forward data the instant it receives it. In practice, we need to transfer data in blocks. The linear increase in the number of leechers results in a linear decrease in each connection rate, causing the time to transfer a block to increase linearly. This causes huge losses in the beginning and at the end of downloads.

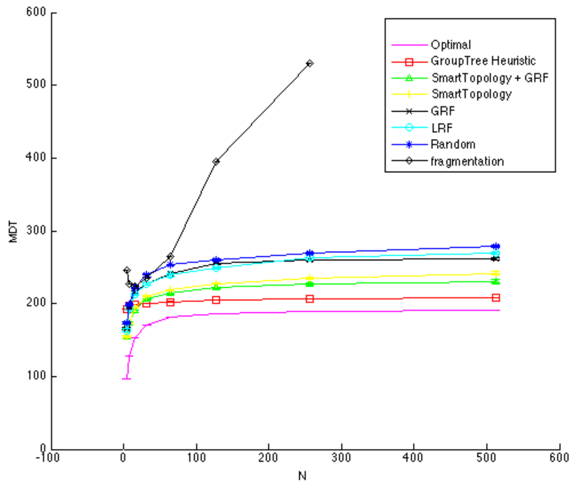


Figure 8: Comparing the MDT for a set of peers using multiple heuristics

Figure 9 shows the average finish time of peers using all the aforementioned strategies. Our GroupTree heuristic still outperform all of the other heuristics. The gains from using our GroupTree heuristic are more spelled in the worst case than the average case. This enforce the motivation to use our GroupTree heuristic when all the nodes have to wait for the last node to finish download.

To summarize, our simulations show that our *GroupTree* heuristic outperforms all other heuristics, that it scales well, and that it operates near the optimal theoretical bounds. Our experimental findings also suggest that heuristics (such as our *GroupTree* heuristic) which coordinate both piece and peer selection result in much better performance than heuristics that only focus on one or the other (e.g., the LRF, GRF, and Smart-Topology heuristics), or those that consider both but in an uncoordinated fashion (e.g., the Smart-Topology+GRF heuristic).

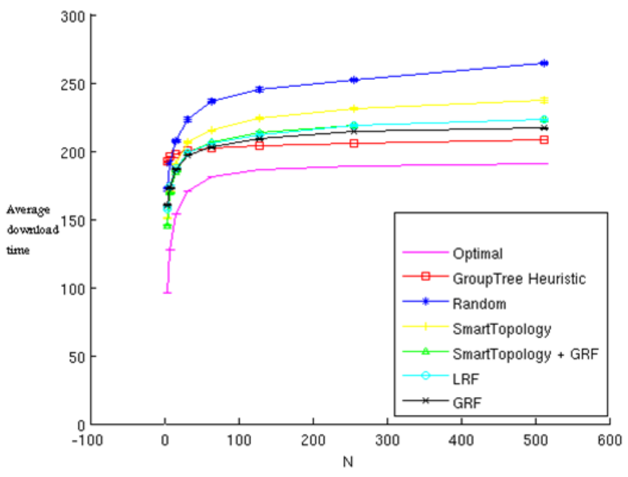


Figure 9: Comparing the average download time of peers using multiple heuristics

5 Conclusion

In this paper, we introduced the idea of *Angels*, special nodes that are not interested in downloading a file in a P2P environment but in helping leechers download faster. We used angels to minimize the distribution time of a file among a set of leechers. We derived a lower-bound on the Minimum Distribution Time (MDT), and constructed a fluid-model that achieves this bound. We observed that the bottleneck of the network is the leechers upload capacity, which proves that a system can fully utilize and added resources to the network. Our Initial experiments proved that implementing the fluid-model construction, without modifications, is impractical. Hence, we developed the GroupTree heuristic to address some of the impracticalities in our construction, such as bounded node-degree and fragmentation. Our GroupTree Algorithms coordinates between piece and peer selection. We studied a host of alternative heuristics that only focus on piece or peer selection. We build a discrete-time simulator to compare our heuristic against a host of those heuristics. The experimental results suggest that a system that coordinates be-

tween piece and peer selection outperforms heuristics that focus on only one of them or even applies both of them in an uncoordinated fashion.
The conducted experiments assume that all the leechers have similar upload and download capacities. We conjuncture that with non-homogeneous leechers, the performance our GroupTree heuristic will be further improved. In the future, we intend to extend this work to utilize angels to optimize the MDT of a set of files simultaneously. Furthermore, we plan to investigate the use of angels to optimize other QoS metrics, such as achieving a certain bound on the download time.

References

[1] R. Kumar and K. W. Ross. Peer-Assisted File Distribution: The Minimum Distribution Time. In *Hot Topics in Web Systems and Technologies, 2006. HOTWEB '06. 1st IEEE Workshop on*, pages 1–11, 2006.

[2] B. Cohen. Incentives Build Robustness in BitTorrent, 2003.

[3] BSP Website. www.bsp-worldwide.org, 2007.

[4] OpenP4P Website. http://www.openp4p.net, 2008.

[5] Bittorrent DNA Website. http://www.bittorrent.com/dna/technology, 2001-2009.

[6] Pando Networks Website. http://www.pandonetworks.com/cdn-peering, 2004-2008.

[7] Joltid Website. http://joltid.com/peercache, 2002-2009.

[8] Nathaniel Leibowitz, Aviv Bergman, Roy Ben-Shaul, and Aviv Shavit. Are File Swapping Networks Cacheable? Characterizing P2P Traffic. In *In Proc. of the 7th Int. WWW Caching Workshop*, 2002.

[9] G. Shen, Y. Wang, Y. Xiong, B. Zhao, and Z. Zhang. HPTP: Relieving the Tension between ISPs and P2P. *IPTPS 07: Proc. of International Workshop on Peer-to-Peer Systems*, 2007.

[10] A. Wierzbicki, N. Leibowitz, M. Ripeanu, and R. Wozniak. Cache Replacement Policies Revisited: the Case of P2P Traffic. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 182–189, 2004.

[11] Gyorgy Dan. Cooperative Caching and Relaying Strategies for Peer-to-peer Content Delivery. In *IPTPS 08: Proc. of International Workshop on Peer-to-Peer Systems*, 2008.

[12] Osama Saleh and Mohamed Hefeeda. Modeling and Caching of Peer-to-Peer Traffic. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society.

[13] J. A. Pouwelse, P. Garbacki, J. Yang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. Van Steen, and H. J. Sips. Tribler: A social-based peer-to-peer system. In *In The 5th International Workshop on Peer-to-Peer Systems (IPTPS'06*, 2006.

[14] J. Wang, C. Yeo, V. Prabhakaran, and K. Ramchandran. On the Role of Helpers in Peer-to-Peer File Download Systems: Design, Analysis and Simulation. *IPTPS*, 2007.

[15] Yang Guo, Chao Liang, and Yong Liu. dhcps: decentralized hierarchically clustered p2p video streaming. In *CIVR '08: Proceedings of the 2008 international conference on Content-based image and video retrieval*, pages 655–662, New York, NY, USA, 2008. ACM.

[16] R. Kumar and K. W. Ross. Peer-assisted file distribution: The minimum distribution time. In *A Technical Report*, 2007.

[17] Dah M. Chiu, R. W. Yeung, Jiaqing Huang, and Bin Fan. Can Network Coding Help in P2P Networks? In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006 4th International Symposium on*, pages 1–5, 2006.

[18] Arnaud Legout, G. Urvoy-Keller, and P. Michiardi. Rarest First and Choke Algorithms are Enough. In *IMC'06*, 2006.