Technical Report BUCS-TR-2010-024, Computer Science Department, Boston University

Angels In the Cloud
A Peer-Assisted Bulk-Synchronous Content Distribution Service[†]

RAYMOND SWEHA
remos@cs.bu.edu
Computer Science Dept
Boston University, USA

VATCHE ISHAKIAN
visahak@cs.bu.edu
Computer Science Dept
Boston University, USA

AZER BESTAVROS
best@cs.bu.edu
Computer Science Dept
Boston University, USA

***Abstract*—Leveraging client upload capacity through peer-assisted content distribution was shown to decrease the load on content providers, while also improving average distribution times. These benefits, however, are limited by the disparity between client upload and download speeds, especially in scenarios requiring a minimum distribution time (MDT) of a piece of content to a set of clients. Achieving MDT is crucial for *bulk-synchronous* applications, when every client in a set must wait for all other clients in the set to finish their downloads before being able to make use of the downloaded content. In this paper, we propose the use of dedicated servers, which we call *angels* to accelerate peer-assisted content distribution in general, and to minimize MDT in particular. An angel is not itself the content origin, nor is it interested in fully downloading the content; its only purpose is to enable a peer-assisted content distribution scheme to approach the theoretical lower-bound for MDT. To overcome scalability issues inherent in an optimal MDT construction, we propose and evaluate a content exchange strategy involving angels, which we call *Group Tree*. In addition to simulation results that demonstrate the near optimal performance of our proposed approach, we present the architecture and implementation of CLOUDANGELS – a service that allows the elastic, on-the-fly deployment of angels (in the cloud) to assist a content provider (off the cloud) in realizing its MDT objective.**

I. INTRODUCTION

Over the years, content distribution has evolved from the traditional client-server model (in which clients are passive recipients of content), to the end-system multicast model (in which clients relay content they receive to other clients in an orderly manner along a fixed, typically hierarchical topology) [1], [2], [3], to the peer-to-peer swarming model (in which clients self-organize in ad-hoc swarms to exchange fixed pieces of the content) [4], [5]. This evolution can be seen as increasingly tapping the uplink capacity of clients: in a client-server model, a client's uplink capacity is not used; in an end-system multicast model, a client's uplink capacity is used up to a limit typically dictated by the slowest link in the multicast topology; and in a peer-to-peer swarming model a client's uplink capacity could be fully utilized as long as other clients in the swarm can benefit from it. With the meteoric rise in demand for large volumes of media [6], content providers as well as content storage and distribution networks have realized the potential from leveraging client uplink capacities to significantly reduce their operating costs. Examples of *peer-assisted content distribution systems* include Akamai's Netsession [7], Octoshape Infinite Edge [8], Pando [9], and BitTorrent DNA [10]. This trend towards peer-assisted content distribution is embraced by clients as it yields faster download times and more resilient services, at no additional costs (at least to customers with "flat rate" Internet subscriptions).

Motivation and Setting: Existing peer-assisted content distribution systems (including all those mentioned above) can be seen as "ad hoc" or "best effort" in the sense that a client upload capacity is tapped only to the extent that other swarming clients are able to utilize it. Indeed, most mechanisms proposed and implemented for peer selection do not *primarily* aim to maximize the utility of a swarm's aggregate capacity, but rather focus on other goals, including dealing with freeloaders using rational tit-for-tat exchanges [4], or reducing inter-ISP traffic using geographic (or topological) locality [6]. These approaches are well justified when peer-assisted content distribution is not under the control of a single authority or does not cater to a common overarching "socially optimal" (not to mention legal) objective.[1] We contend that for many emerging applications and settings, either or both of these conditions may not hold. In particular, when content providers employ peer-assisted delivery mechanisms, they are in fact acting as a single authority that choreographs the operation of all clients involved, using specially-developed software clients. Moreover, for many emerging cooperative applications, optimizing the performance of individual clients may not be a rational objective as it may lead to suboptimal group performance.

Motivated by the above observations, in this paper we tackle the problem of peer-assisted content delivery in a setting where it is assumed that (1) the content provider choreographs the participation of all clients in a swarm, and/or (2) the objective of the content delivery system is to minimize the time it takes to distribute a common piece of content to all clients in the swarm.

Bulk-Synchronous Content Distribution: Our focus in this paper is on systems with a *Minimum Distribution Time* (MDT) objective. An MDT objective implies that the overarching common goal of the provider *and* clients is to minimize the time by which *all* clients finish their download. The need for such a bulk synchronous mode of operation (which is not new [11]) is paramount, with application to: enterprise-wide system administration requiring synchronous software patching or data replication, virtual community games and simulated reality environments requiring common content such as terrain information to be accessible to all players before

[†] This research was supported in part by NSF awards #0720604, #0735974, #0820138, and #0952145.

[1] A file-sharing application is a canonical example where both of these conditions hold. Indeed, early research on swarming protocols was singularly geared to issues stemming from file-sharing applications.

any progress could be made, publish-subscribe networks and distributed data stores requiring consistency across multiple sites, among many others.

To achieve an MDT objective for a group of client requires a judicious use of client uplink and downlink capacities, which are typically highly asymmetric as most ISPs offer their clients download speeds that are significantly larger than upload speeds. Using an ad-hoc, best-effort swarming mechanism, this disparity may result in an under-utilization of the downlink capacity to the clients, even if the uplink capacity of clients in the swarm is fully tapped. In a video dissemination system, this may not be a problem as long as the download rate to individual clients is larger than the playback rate [12]. But, for bulk content delivery that is not subject to a nominal playback rate, under-utilizing the swarm capacity is problematic.

Kumar and Ross [13] derived a theoretical lower-bound on MDT and proposed a fluid model that achieves this bound. Their result suggests that MDT is limited by one of three potential bottlenecks in any P2P overlay: (1) the seeder upload capacity, (2) the slowest client download capacity, or (3) the aggregate upload capacity of all clients in the swarm. We note that the third of these bottlenecks, namely the *swarm upload capacity*, is likely to be the most prevalent in many settings, due to the aforementioned disparity between the upload and download capacities of most clients.[2]

Paper Contribution and Outline: In support of the MDT objective for bulk-synchronous content distribution, in this paper we investigate the potential benefit from the on-demand deployment of cloud resources to alleviate the swarm upload capacity bottleneck. To that end, we propose the use of helper nodes – which we call *angels*.[3] An angel is not a client in the sense that it is not interested in receiving the entire content (file) to be distributed, but rather it is interested in minimizing the MDT to all clients. 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 swarm upload capacity ceases to be the limiting factor for MDT. Doing so ensures that the content provider is able to fully leverage the uplink/downlink capacities of the clients.

In section II, we extend the analytical results in [13] to account for the presence of angels by deriving a new lower bound on MDT. Also, we show that this new lower bound is tight by constructing a distribution strategy under a fluid model assumption. Simulation results show that a direct implementation of our fluid construction is impractical. Therefore, in section III, we present a coordinated swarming strategy – called *Group Tree* (GT) – that addresses the impracticalities of the fluid model to fully utilize angels. Simulation results show that GT outperforms other strategies, scales well with the increase in the number of clients, and operates near the optimal theoretical bounds. These findings suggest that, in contrast to focusing only on piece or peer selection strategies, a strategy that incorporates both criteria holds the potential for significant performance gains. Armed with the promise from these simulation results, in section V, we present the blueprints of CloudAngels – an "angels on demand" cloud service that complements peer-assisted content delivery to achieve an MDT objective. In section VI, we report results from live Emulab experiments in which our CloudAngels service is used in a real content distribution sessions.

II. Optimal MDT Construction

In this section, we devise an optimal fluid dissemination strategy that achieves MDT. We begin with a mathematical formulation of the MDT problem. Next, we construct a swarming strategy (using a fluid model) and compute the download time for it. Next, we establish the lower bound for MDT and show that our choreographed swarming construction achieves that lower bound. Our fluid construction, while impractical given its underlying fluid model, provides us with the necessary insights for designing a near-optimal practical system, which we present in later sections of the paper.

A. MDT Problem Statement

We aim to minimize the distribution time of a file of size F from a set of Content Providers (providers), $p \in P$ to a set of Clients, $c \in C$, with the help of a set of angels, $a \in A$. Following the Uplink-Sharing fluid model presented in [15] and adopted by [13], our goal is to minimize T, where $T = \max_{i \in C}\{T_i\}$ and T_i is the time it takes client i to finish the download. The fluid model must make sure that each node's upload or download rate exceeds neither its upload capacity $u(i)$ nor its download capacity $d(i)$, which under a fluid model can be infinitesimally divided. Specifically:

$$u(p) \geq \sum_{i \in \{C,A\}} x_{pi}, \text{ Provider upload capacity} \qquad (1)$$

$$u(i) \geq \sum_{j \in \{C,A\}} x_{ij}, \text{ Client upload capacity} \qquad (2)$$

$$d(i) \geq x_{pi} + x_{ai} + \sum_{j \in \{C\}} x_{ji}, \text{ Client download capacity} \qquad (3)$$

$$u(a) \geq \sum_{i \in \{C\}} x_{ai}, \text{ Angel upload capacity} \qquad (4)$$

$$d(a) \geq x_{pa} \text{ Angel download capacity} \qquad (5)$$

The Uplink-Sharing model [15] does not account for network cross-traffic or losses. Thus, for notational convenience, we aggregate the set of providers and the set of angels into one provider and one angel, with the upload/download capacity of the provider and of the angel set to the aggregate capacities of all the providers and of all the angels, respectively. Chiu *et al.* proved the correctness of such an abstraction [16].

B. Main Results

In this section, we prove that under the aforementioned model, we can download the file in the minimum time possible while utilizing the full capacity of angels.

Lemma 1: In the presence of angels, under a fluid Uplink-Sharing model, a distribution time T for a file of size F is achievable, where:

$$T = \frac{F}{\min\{u(p), d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}} \qquad (6)$$

[2] We also note that nothing can be done "inside the network" (*e.g.*, by leveraging cloud resources) to alleviate the first two potential bottlenecks.

[3] The idea of introducing helper nodes was also considered in [14], where the focus was on using idle nodes to improve BitTorrent's steady-state performance as opposed to optimizing any specific objective.

Proof: We provide a fluid construction that achieves the bound for T. This construction follows three cases corresponding to the nature of the underlying bottleneck (the denominator of T). Below, is a proof sketch for the first of these three cases, in which the provider's upload capacity is the bottleneck. [4]

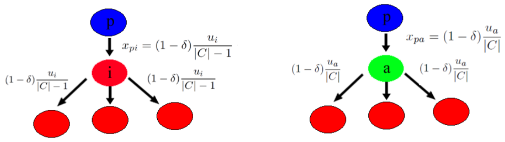
p
$x_{pi} = (1-\delta)\frac{u_i}{|C|-1}$
i
$(1-\delta)\frac{u_i}{|C|-1}$
$(1-\delta)\frac{u_i}{|C|-1}$
p
$x_{pa} = (1-\delta)\frac{u_a}{|C|}$
a
$(1-\delta)\frac{u_a}{|C|}$
$(1-\delta)\frac{u_a}{|C|}$

Fig. 1. Clients/angel forward content receives from provider to other clients.

When the upload capacity of the content provider P is the bottleneck, *i.e.*, $u(P) \leq \min\{d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}$, P sends fresh data to each client i with rate x_{pi}, and also sends fresh data to the angel with rate x_{pa}. Each client forwards data it receives from the provider to the other $|C-1|$ clients. Similarly, the angel forwards data it receives from the provider to all $|C|$ clients. Notice that once data is sent to one client, all other clients would successfully receive it from that client as long as $x_{pi} = x_{ij} \quad \forall j \in C$. Figure 1 illustrates the idea. To complete our construction, we show that there are values for x_{pi} and x_{pa} that ensure that all clients will successfully download the data with rates y_i equal to the provider's capacity, *i.e.*, $y_i = u(P)$, $\forall i \in C$, *without violating the upload/download capacity constraints of the various nodes*. To that end, consider the following rates:

$$\begin{aligned} x_{pi} &= (1-\delta)\frac{u_i}{|C|-1} \\ x_{pa} &= (1-\delta)\frac{u_a}{|C|} \\ \text{where} \quad \delta &= \frac{\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|} - u(P)}{\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}} \end{aligned} \tag{7}$$

Those rates respect the five aforementioned constraints and achieve the desired MDT. It is straightforward to verify that the choice of rates given in Equation 7 ensures that the clients and the angel will not exceed their upload capacities (constraints 2 and 4), and that the total data sending rate of the provider x_p (given below) satisfies the seeder's capacity constraint (constraint 1).

$$\begin{aligned} x_p &= \sum_{i \in C,A} x_{pi} = (1-\delta)(\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}) \\ &= \frac{u(P)}{\frac{u(C)}{|C|-1} + \frac{u(A)}{|C|}} * \frac{u(C)}{|C|-1} + \frac{u(A)}{|C|} = u(P) \end{aligned} \tag{8}$$

■

Lemma 2: In the presence of angels, under a fluid Uplink-Sharing model, the minimum distribution time, T_{min}, has a lower bound given by:

$$T_{min} \geq \frac{F}{min\{u(P), d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}} \tag{9}$$

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

$$y_{\max} \leq \min\{u(P), d_{min}, \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}\}$$

This implies that the MDT bottleneck is due to one of three possibilities. The first is the uplink capacity of the provider: $y_{\max} \leq u(P)$. In this case the provider's upload capacity limits the minimum download rate of peers (*e.g.*, when clients are powerful and the provider is weak). The second possibility is the download capacity of the slowest client: $y_{\max} \leq d_{min}$. Clearly the minimum download rate cannot exceed the minimum download capacity of any client. The third possibility is the aggregate upload capacity of the swarm: $y_{\max} \leq \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}$. In this case the aggregate upload capacity of the swarm cannot saturate the download capacity of the clients.

Only the third of these cases is not obvious. First, we start with the case of no angels. We know that the aggregate download rate cannot exceed the aggregate upload capacity in the swarm: $\sum_{\forall i \in C} y_i \leq u(P) + u(C)$. To utilize a fraction z_a of the upload capacity of the angel, the angel must download fresh data with a rate of at least $\frac{z_a}{|C|}$. Thus, in case of using angels, we get:

$$\begin{aligned} \sum_{\forall i \in C} y_i + \frac{z_a}{|C|} &\leq u(P) + u(C) + z(A) \\ \frac{\sum_{\forall i \in C} y_i}{|C|} &\leq \frac{u(P)+u(C)+z(A)}{|C|} - \frac{z(A)}{|C|^2} \end{aligned}$$

Since the minimum is always less than the mean and the upper bound of $y_{\max}$ is achieved when $z(A) = U(A)$, we conclude that $y_{\max} \leq \frac{u(P)+u(C)+u(A)}{|C|} - \frac{u(A)}{|C|^2}$. ■

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

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

C. Implications on the Role of Angels

It follows directly from the aforementioned theoretical results that when the bottleneck lies in the aggregate upload capacity of the swarm and not in the source (provider's upload capacity) or the sink (slowest client download capacity), adding upload capacity to the swarm through the use of angels will necessarily improve MDT. But, can angels achieve these gains if they behave just like normal clients (*i.e.*, download the entire file)? The answer is *no*. If angels download more than $\frac{1}{|C|}$ of what they upload, they will not be able to upload the excess amount to clients. It would have been more beneficial to upload this excess data to the clients directly from the provider. For example, our optimal construction can speed up download rate to $(\frac{u(A)}{|C|} - \frac{u(A)}{|C|^2})$. If angels act like clients (and have capacities identical to those of clients) the download rate would be $\frac{u(P)+u(C)+u(A)}{|C|+|A|}$. This roughly equals $\frac{u(P)+u(C)}{|C|}$, which is identical to not having angels in the first place. Thus, adding angels that behave just like clients (by downloading the content in its entirety) only enlarges the swarm but does not optimize MDT. Even if the added angels' upload capacity is

[4] For the complete proof, we refer the reader to [17].

made (say) double that of the clients' upload capacity, the speed up from our construction would be double the speed up achieved by having angels act like clients.

To summarize, merely adding "participants" to a swarm will not result in lowering MDT, unless these participants behave in the specific manner prescribed in our construction. This is precisely the role we propose for angels: Angels are on-demand swarm participants that do not have and do not seek to obtain the content (they are not replicas or caches of the provider); they download just the right amount of the content (as prescribed in our construction) to minimize MDT.

III. Practical MDT Construction

The optimal fluid construction presented in the previous section makes two assumptions: (1) data dissemination is fluid instead of packetized, and (2) a client is able to open and use an arbitrary large number of concurrent connections, as large as the number of clients in the swarm. As we experimentally show later on (see section IV), a direct implementation of our fluid construction results in a very degraded performance due to the realities of OS and network stack implementations (*e.g.*, given the packet-based nature of transport protocols, and given the fact that setting up an unbounded number of connections leads to bandwidth fragmentation and TCP timeouts).

That said, our optimal MDT construction gives us two important insights that are crucial for building a near-optimal bulk-synchronous content distribution strategy: (1) the upload capacity of clients is the most scarce asset in the system, thus utilizing the clients upload capacity fully and as soon as possible is key, (2) angels can utilize their upload capacity fully to forward data even if they download data with rates significantly smaller than their download capacity.[5]

A. The Group Tree Coordinated Swarming Strategy

Taking into consideration the pros and cons of an optimal fluid construction, we build a coordinated swarming strategy that does not require any client to have more than k connections at any point in time. Our *Group Tree* (GT) strategy works in two phases as depicted in Figure 2. The first phase utilizes a tree-like structure with the aim of getting all nodes (angels and clients) to download, and hence upload, content as soon as possible. In the second phase, nodes swarm together to finish downloading the file. In the remainder of this section, we provide the details of these two phases of the GT strategy.

Initially, the provider (seeder) divides the file into k segments, where k is the upper-bound set on the out-degree of (number of connections allowed for) clients – a parameter of the GT strategy.

In Phase 1, the swarm is organized into k binary trees. One of the trees is dedicated to angels, whereas the other trees are populated by clients. Clients are matched up and assigned to trees in such a way that all tree participants have similar upload capacities. Each tree is assigned a segment of the file that is proportional in its size to the nominal, individual upload capacity of nodes (clients or angels) in the tree. The seeder sends the segments in chunks to the root of each tree, noting that the tree nodes operate in a pipelined fashion: Once a client/angel receives a packet, it forwards it to its children in the tree. For example, as depicted in Figure 2, client (b, j) downloads the segment b of the file from client $(b, \lfloor \frac{j}{2} \rfloor)$. It only takes $\log(N)$ multiples of a packet transfer time for all clients to begin utilizing their upload capacity.

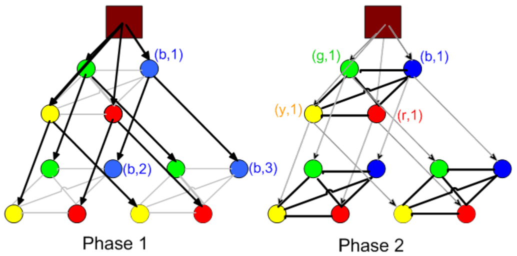

Fig. 2. Illustration of the GT strategy. In phase 1, $k = 4$ different segments are disseminated downward through $k = 4$ binary distribution trees. In phase 2, segments are disseminated laterally within cliques of size $k = 4$.

In Phase 2, sets of k nodes (from the k different trees) each having received one of the k different segments of the file, form a clique for swarming purposes. By construction, each clique would include one angel.[6] Each clique uses our optimal MDT construction to disseminate the file between its members. For example, as depicted in Figure 2, clients $(b, 1)$, $(r, 1)$, $(g, 1)$ and $(y, 1)$ form a clique. Notice that in this phase, the operation of angels and clients is different. In particular, each angel sends data to clients in its clique *without* receiving any data from these clients,[7] thus saving the precious upload capacity of clients.

In the first phase of our GT strategy, the seed (source of content) uses k binary subtrees (*i.e.*, fan-out = 2) to distribute a given segment to a particular set of nodes. Theoretically, the optimal fan-out for distributing segments in Phase 1 is the natural number, e. For a complete proof of the optimality of a fan-out of e, we refer the reader to [17]. Simulation results also reported in [17] show that a fan-out of 2 achieves better performance than a fan-out of 3, justifying our choice of a binary tree construction for the GT strategy presented in this paper.

B. Other Distribution Strategies

In addition to our GT strategy, which is rooted in our MDT construction, we have considered a host of other practical distribution heuristics that rely on peer and piece selection strategies to assess their appropriateness for achieving the MDT objective.

The most basic (baseline) strategy we considered is *Random*, whereby the sender chooses a receiver at random, and sends a random piece to it given that this receiver needs this piece. This strategy does not employ any intelligence in peer or piece selection. That said, in our implementation we ensured that no sender remains idle as long as a receiver needs a piece that the sender has. This strategy fully utilizes the resources in the system but without coordination.

[5] Actually, any excess download by an angel underscores an inefficiency, as a client could have used this wasted capacity.

[6] For implementation purposes, if the number of servers supporting angel functionality is small, the servers can time-multiplex their capacities between multiple cliques, effectively allowing on angel per clique.

[7] Recall that according to our construction, angels download only the prescribed amount of content.

To evaluate the benefit from smarter piece selection strategies, we implemented a *Local Rarest First* (LRF) piece selection strategy [18], which sends the piece with the least number of copies that 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) piece selection strategy. While impractical at scale, GRF allows us to evaluate the full benefit of optimizing piece selection depending on full piece distribution information.

To evaluate the benefit of peer selection, we implemented a *Fair Peer Selection* strategy (FPS), which tries to get each client a fair-share of the network's upload capacity. A sender chooses the least fortunate recipient to send data to, *e.g.*, the client with the least fan-in given that all the clients have the same upload capacity.

Finally, we implemented a strategy that combines piece and peer selection. The FPS+GRF strategy chooses the most needy recipient, and then sends the GRF piece to it. This strategy can be seen as independently supporting intelligent piece and peer selection strategies.

IV. Simulation Results

In this section we present simulation results comparing our GT strategy to the other MDT-agnostic strategies. We built a custom discrete-event simulator that simulates a provider, a set of angels, and a set of clients downloading a file consisting of a set of blocks. Each node builds a set of connections to some (or all) of 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. Our simulation is done at the session layer, thus ignoring transport layer effects, *e.g.*, due to packet loss or cross traffic, which we captured at the session level by introducing variability in the connection speed over time.

In our experiments, we use the following model parameters: the file size F is set to 24 MB (with a block size of 128KB); the provider upload capacity u_p is set to 512 Kbps; the client/angel upload capacity u_i is set to 128 Kbps and the download capacity d_i is set to 256 Kbps, $\forall i \in C, A$. These parameters ensure that the aggregate upload capacity of the swarm (without the angels) is the bottleneck, which is the scenario of interest as explained in section II-C.

In our first set of experiments, we evaluate the performance of a direct implementation of an optimal MDT construction as the number of clients scales up, and compare the achievable distribution time to the theoretical lower bound (predicted by our fluid model). Figure 3 shows that a direct implementation of the MDT construction does not scale. This is because the optimal construction assumes a fluid communication model, implying that a client can forward data the instant it receives it. In practice, clients need to transfer data in blocks. This means that as the number of clients in the swarm increases, the speed of each connection decreases correspondingly (since the client bandwidth is equally divided across all connections), causing the time to transfer a block to increase linearly. The figure also shows the results when the GT strategy is used under the same settings, showing near optimal MDT and scaling characteristics.

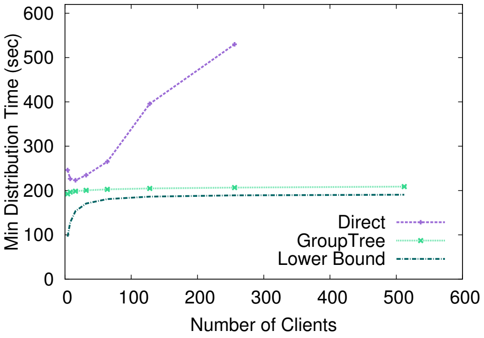

Fig. 3. Performance of a direct implementation of the MDT construction and of the GT strategy (also showing the analytical MDT lower bound).

In our second set of experiments we fixed the number of clients to 128 and varied the number of angels between 0 and 16, to examine the potential improvement in MDT due to angels. Figure 4 shows the performance of the GT strategy versus the theoretical lower bound, which makes it evident that the performance of the GT strategy tracks that of the MDT lower bound as the number of angels increases, subject to a relatively fixed overhead for carrying out Phase 1 of the GT strategy.

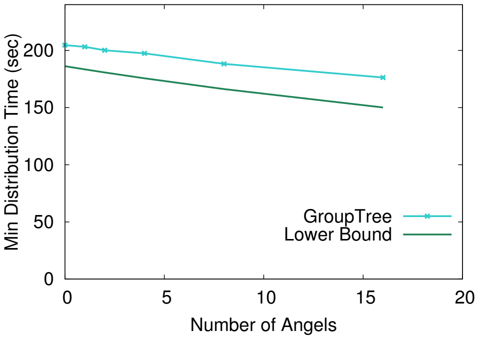

Fig. 4. Performance of GT as a function of the number of angels.

In our third set of experiments, we compared the performance of the GT strategy to those of the other distribution strategies presented in section III-B, as the number of clients varies from 4 to 512. Figure 5 shows the results (as well as the theoretical MDT lower bound). These results indicate that our GT strategy outperforms all others and that it operates within striking distance of the optimal MDT bound. Another observation from this set of experiments is that the performance of the GRF+FPS strategy (utilizing both piece and peer selection, albeit independently) is decidedly superior to the standalone GRF and FPS strategies.

Minimizing the worst-case distribution time may result in a degradation of the average distribution time, especially in settings wherein the client download capacities are highly diverse, *e.g.*, a setting where a few clients have much lower (outlier)

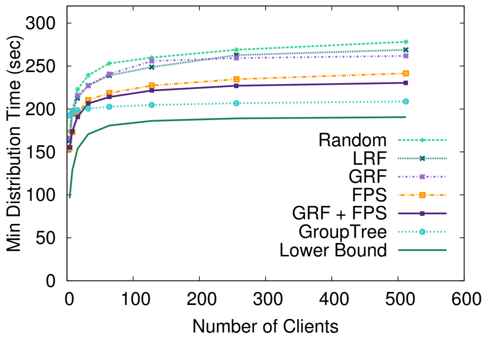

Fig. 5. MDT for various strategies as a function of the number of clients.

download speeds than most others. In such settings, aiming to reduce MDT will necessarily hurt the average distribution time. We argue that bulk-synchronous content distribution applications are unlikely to involve such disparate profiles for client uplink/downlink capacities. Rather, for our purposes, the set of clients involved in a bulk-synchronous download are likely to have comparable capabilities (*e.g.*, a set of broadband residential users in a virtual-reality gaming application, or a set of enterprise servers acting as consistent data repositories). In such settings, our experiments confirmed that optimizing MDT does indeed improve the average distribution time as well. Figure 6 shows the *average* distribution time for the swarm considered in the last experiment; it shows that our GT strategy, which while primarily aiming to optimize MDT, outperforms the other strategies with respect to the average distribution time as well.[8]

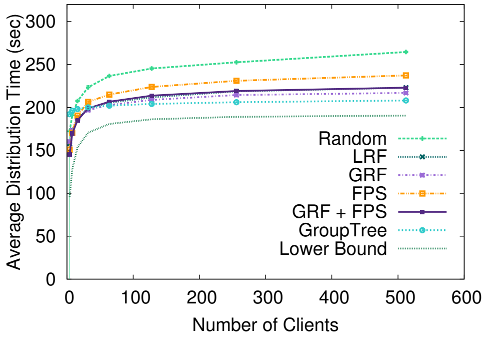

Fig. 6. Average distribution time as a function of the number of clients.

To summarize, our simulation experiments show that the GT strategy outperforms all other strategies, scales well, and operates near the optimal, theoretical MDT bound. Our findings also suggest that coordinated swarming strategies, which coordinate both piece and peer selection, such as GT result in much better performance than strategies that focus on either piece or peer selection (*e.g.*, the LRF, GRF, and FPS strategies) or those that consider both but in an uncoordinated fashion (*e.g.*, the FPS+GRF strategy).

V. CloudAngels: Blueprint for System Design

In this section we present the design and implementation of a cloud service that enables a content provider to utilize cloud resources to complement a peer-assisted content distribution system for the purpose of minimizing content distribution time. Our service, which we call CloudAngels, is based on the GT strategy presented in section III. We believe that a cloud offering is the most appropriate paradigm for an implementation of CloudAngels because the service needs to be able to instantiate angels on-the-fly: As content providers seek assistance with MDT content delivery, the system would need to instantiate a set of Virtual Machines (VMs) to act as angels. The aggregate capacity of the deployed angels would depend on the content provider's stated objectives – *e.g.*, how much uplink bandwidth to "add" to the swarm to achieve an optimal MDT, or to ensure that MDT is below a certain preset threshold.

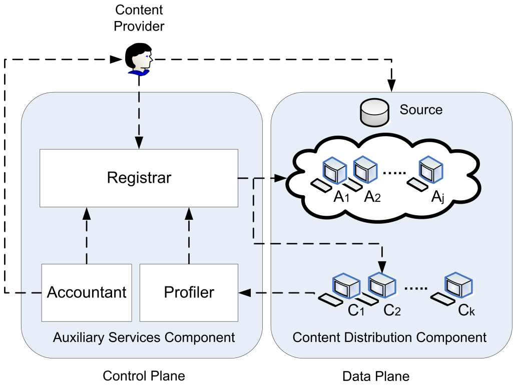

Fig. 7. CloudAngels: Architectural Elements.

We identify the following agents as the essential players in our system:

Provider: (P) This is the entity in need of the CloudAngels service's help with the distribution of content (namely, a file within its possession) to a group of clients engaged in a bulk-synchronous application.

Clients: ($C_1, C_2, \cdots, C_k$) These are the agents constituting the swarm interested in the bulk-synchronous delivery of content from the provider. We assume that the provider relays to the CloudAngels service the list of clients authorized to access the content.

Angels: ($A_1, A_2, \cdots, A_j$) These are the agents (VMs) that are created on-the-fly to help the content provider distribute the content subject to MDT objectives.

Figure 7 illustrates the design of our CloudAngels system, which consists of two major components: (1) the Auxiliary Services Component (ASC) which acts within the control plane to choreograph the GT strategy, and (2) the Content Distribution Component (CDC) which acts within the data plane to distribute content to clients using the GT strategy.

In support of CDC, the ASC provides two sets of services denoted in Figure 7 by the *"Profiler"* and the *"Accountant"*. The CloudAngels profiler is responsible for the collection of information about each client's upload and download capacity. Clients may either self-report their capacity[9] or else, the profiler may use other methods to estimate their upload/download capacities. The CloudAngels accountant is responsible for

[8] This hints to the utility of coordinated swarming beyond MDT, but is not a focus of this work.

[9] We note that for bulk-synchronous applications, the MDT objective is both a rational/selfish objective for the individual clients, as well as a socially-optimal objective for the swarm, and thus clients are incentivized to truthfully report their upload/download capacities.

identifying the level of assistance (in terms of the number and capacity of angels, if any) that the bulk-synchronous content distribution group would need to meet its MDT objectives. This objective may simply be to minimize MDT, or alternately, it may be to ensure that MDT remains below a preset threshold (for a minimal quality of service dictated by the bulk-synchronous application). To do so, the accountant estimates the distribution time achievable without the help of angels, and based on the desired MDT objective, calculates the characteristics (number and capacities) of the angels that need to be deployed to meet this objective. The accountant keeps track of these aspects for purposes of charging the provider for the service.

The CDC supports the main CLOUDANGELS content distribution functionality as follows. Upon receipt of a request from the content provider to distribute a file, the CDC deploys a *"Registrar"* process to manage the request. The registrar does not get involved in the data transfer but is responsible for managing the bulk-synchronous swarm. According to the GT swarming strategy, the registrar assigns identifiers to clients to choreograph their connection to one another. The registrar issues each client a signed token allowing it to introduce itself as an authorized client in the swarm. This token includes the file name and a computed unique identifier (ID) for the client in the swarm. The client ID is in the form of a tuple (i, j) where i is the segment that the client needs to subscribe to in Phase 1 of the GT, and j is the location of the node in the distribution tree. The registrar determines this ID based on the upload bandwidth of the client (among other possible information in support of additional objectives, *e.g.*, topological location for efficient intra-ISP swarming).

To distribute a new file x, the content provider divides the file into k segments $(x_1, ..., x_k)$ and starts k server processes to handle requests for each one of these segments – requests from the roots of each one of the k (binary) trees constituting the first phase of the GT strategy. When the root of tree i sends its signed token to the content provider, the appropriate process serves segment x_i in 64KB-block increments, eventually resulting in the k different segments being transmitted through the roots of the k distribution trees.

In the first phase of the GT strategy, a client with ID (i, j) needs to download segment x_i. To do so, the registrar sends the client the IP address of its parent in the binary tree – namely, $(i, \lfloor \frac{j}{2} \rfloor)$. The client then begins its download, and starts a server process that listens for upload requests from the client's children in the tree (during the first GT phase) and from the client's siblings in the clique (during the second GT phase). Every time a child or a sibling contacts the server, it forks a thread and starts uploading segment x_i, giving priority to requests for x_i from its children over its siblings.

In the second phase of the GT strategy, the client with ID (i, j) needs to download all the other segments x_l $l \neq i$. For segment l, the registrar sends the client the IP address of its sibling in the l tree. Client (i, j) contacts its siblings to download the other segments as soon as these segments are available at the siblings.

To meet the MDT objective requested by the provider, the registrar instantiates (on the fly) the necessary number of appropriately provisioned angels (as prescribed by our MDT construction). Each angel is assigned an ID (a, j), where a corresponds to segment x_a of the file that the angels are responsible to disseminate (recall, the size of x_a is a function of the MDT construction). In the first GT phase, angels act like clients, disseminating segment x_a among themselves using the binary tree. In the second GT phase, angels run a server listener and serve their clique siblings segment x_a, but unlike clients they do not request or download any of the other segments of the file.

VI. CLOUDANGELS: EXPERIMENTAL EVALUATION

In this section we present results from extensive experimental evaluations of our CLOUDANGELS service. Our main motivation in performing these experiments was to (1) establish confidence in the performance of our implementation by comparing its average performance to that of widely used swarming solutions, (2) establish the effectiveness of deploying angel in the cloud for the purpose of meeting MDT objectives, and (3) provide evidence of the scalability characteristics of our system.

Experimental Setup: We used Emulab [19] to acquire virtual nodes for our experimental needs. To marginalize hardware and operating system interference, we requested configurations in which all nodes were identical. Subject to this requirement, we were able to get a total of 36 Emulab nodes (all running on 64-bit Xeon processors with 2GB of RAM, with a 64-bit Fedora 8 Linux Distribution operating system). This allowed us to evaluate GT swarms organized in binary trees of depths 1, 2, 3, and 4.[10] To introduce variance among nodes in terms of their upload and download capacities, we used Linux packet filters *tc*.

In our experiments, we divided the set of emulab nodes under our control into three groups (for purposes of the first GT phase) with the nodes in each group assigned an upload capacity of 5mbps, 6mbps, and 7mbps, and a download capacity of 15mbps, 18mbps, and 21mbps, respectively. This assignment is reasonable given the typical (factor of two to three) mismatch between broadband upload and download speeds. In addition to these three groups of clients, a fourth group of nodes was set aside to serve as angels, with each angel node provisioned with a download capacity of 24mbps and an upload capacity that we set in our experiments to one of the following values: 0, 6, 12, 18, and 24mbps (based on the MDT needs of the swarm).

For comparative purposes, we have also configured our emulab nodes (subject to the same upload and download provisions) to run an unmodified BitTorrent (BT) client to allow us to establish a baseline model for MDT performance (against which the advantage of CLOUDANGELS could be evaluated). The specific client we used is the Instrumented BitTorrent mainline client (version 4.0.2) [20].

In our experiment, the content provider is represented using a single seeding node, whose upload capacity was set to 27mbps, and the content provided by the seed had a size of 30MB.

[10] A GT swarm with an outdegree bound of k organized in *full* binary trees of depth d could be as large as $k * (2^d - 1)$. With 36 nodes, we are able to consider *full* GT topologies of depth three, and only partially-full GT topologies beyond that.

Experimental Results: To establish a performance baseline, Figure 8 shows the MDT achievable when content is distributed (1) using a client-server model, (2) using the unmodified BT client, and (3) using our CLOUDANGELS service without deploying *any* angels (*i.e.*, simply coordinating the content distribution according to the GT strategy). As expected, increasing the number of clients participating in the swarm results in a linear increase of the MDT for a client-server distribution strategy, but results in a sublinear increase of the MDT for swarming strategies (namely our CLOUDANGELS service and BT), underscoring the benefit from tapping the uplink capacities of the clients.

Figure 8 also shows that the scaling characteristics of CLOUDANGELS (without angels) and of BT are fairly similar in terms of their growth pattern, with the MDTs achievable by CLOUDANGELS consistently below those of BT. These results give us confidence that the implementation of our GT coordinated swarming strategy is efficient in the sense that it is competitive with (and indeed noticeably better than) vanilla BT swarming implementations. Needless to say, since in these experiments no angels were deployed, the lower MDT of CLOUDANGELS may be due to the superiority of the GT swarming strategy or it may be simply the result of a leaner implementation, which does not include all the bells and whistles (and hence overheads) associated with a BT client.

An important observation from the results in Figure 8 relate to the "step-wise" nature of the growth in CLOUDANGELS MDT as the number of clients is increased. This step-wise behavior is a direct result of the increase in the depth of the binary tree used in the first GT phase.

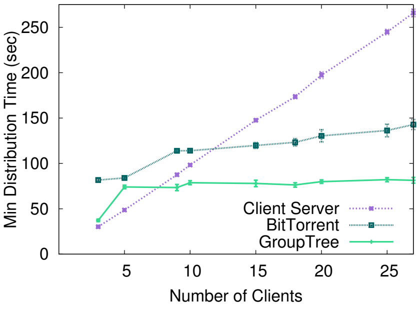

Fig. 8. Achievable MDT as a function of the number of clients for swarming-based approaches (CLOUDANGELS and BT) versus non-swarming-based approaches (client-server).

Figure 9 shows the average (as opposed to minimum) distribution time for the same experimental setup. It confirms the findings from our simulation experiments (see Figure 6) – namely, that the reduction in MDT does not come at the expense of average distribution time.

We now proceed to presenting results in which angels were deployed by the CLOUDANGELS service. Figure 10 shows that as we increase the upload capacity of angels the MDT decreases. Figure 11 shows the improvement in MDT as a result of deploying a fixed amount of angel capacity, as the number of clients increases. Naturally, as the number of clients increases the effectiveness of CLOUDANGELS (with a fixed set of angels) in reducing MDT becomes increasingly limited.

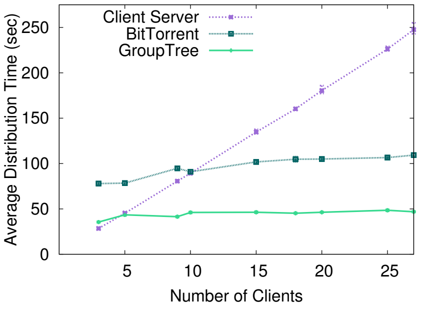

Fig. 9. Average distribution time as a function of the number of clients for swarming-based approaches (CLOUDANGELS and BT) versus non-swarming-based approaches (client-server).

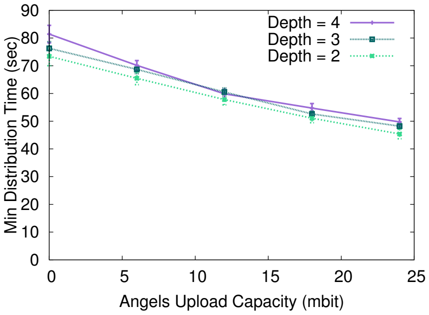

Fig. 10. CLOUDANGELS: Impact of angels' upload capacity on MDT.

VII. RELATED WORK

While not studied before, the bulk-synchronous content distribution problem and solutions we discuss in this paper build on and leverage a vast body of prior work in a number of areas. We examine the most relevant of these next,[11] noting that we have referred to other related works throughout the paper.

MDT bounds and implications: Kumar and Ross [13] established MDT bounds for file dissemination from a seed to a set of clients, and identified three limiting conditions for MDT – namely the seeder, client, and network (swarm) bottlenecks. Based on the observation that the best operating regime for a swarm is when the seeder is not the bottleneck, the Antfarm system [21] directs its resources in support of swarms for which the seeder is the bottleneck. For swarm-assisted content distribution systems (such as those offered by major providers [7], [8]), we argue that the seeder (provider) is likely to be well provisioned, making the network the likely MDT limiter, and providing the motivation for all our constructions and for the CLOUDANGELS prototype we presented in this paper. While our theoretical MDT bounds (in the presence of angels) are similar to those independently derived by Kumar and Ross [22], our fluid construction (to achieve MDT) is different. The fluid construction in [22] separates clients into two tiers, with the MDT bound achieved only for the top tier. In our fluid construction, which is simpler, the MDT bound is achieved for *all* clients, making our construction more appropriate for bulk-synchronous content delivery.

[11] We note that we do not consider content delivery acceleration schemes – such as caching and replication – that may indirectly improve MDT to be relevant related work.

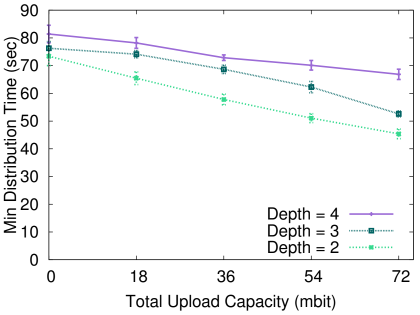

Fig. 11. CloudAngels: MDT as a function of number of clients for fixed angel capacity.

Overlays for optimized bulk-synchronous delivery: Smaragdakis *et al.* [23] considered an alternative bulk-synchronous delivery framework, in which the content does not originate from a single source (the seed) but rather from the clients themselves. Thus, each bulk-synchronous delivery involves an exchange of content from every client to all other clients in the swarm. In that work, the focus is not on a coordinated swarming construction, but rather on the construction of an optimal topology for minimizing MDT in that setting.

Cooperative swarming: An inherent feature of bulk-synchronous swarms is that all clients share a common objective – that of minimizing MDT. This implies that, unlike the "selfish" behaviors expected in filesharing swarms (for example), clients participating in a bulk-synchronous content delivery swarm must be cooperative. There has been a relatively small number of studies that considered cooperative swarming [23], the most prominent of which is perhaps BitTyrant [5], [24], which can be seen as a variant of BT that abandons the more rational tit-for-tat mechanism in favor of a mechanism that pushes the swarm to a more "socially optimal" operation, thus indirectly assuming that clients are cooperative.

End-system multicast: In essence, our GT strategy can be seen as an (asynchronous) end-system multicast [1], [2], [3] strategy. In that respect, the dHCPS system [25] resembles GT in that it relies on a two-level hierarchical P2P dissemination mechanism for streaming, whereby each cluster has a Head (the source for this cluster) which is a member of a superNode swarm fed by the source. dHCPS solves the optimization problem for deciding how much upload bandwidth each head should dedicate to the superNode swarm versus its own group, whereas the main aspect of the GT strategy is the assignment of clients to trees and cliques (supernodes and clusters) and the determination of the segment sizes for each. Another end-system multicast system that resembles GT's distribution strategy is ChunkySpread [26], which divides a file into appropriately-sized chunks which are disseminated through different trees to avoid the overhead due to routing/ordering of individual chunks.

VIII. Conclusion

In settings where bulk-synchronous content distribution is necessary, minimizing the maximum time it takes any client in a set to download content becomes the overarching goal. In this paper, we have developed a provably optimal fluid construction that enables a content source (provider) to leverage the resources of helper nodes (angels) to meet specific minimum distribution time (MDT) objectives. Armed with insights gained from this fluid model, we proposed the *Group Tree* coordinated swarming strategy that forms the basis of a cloud service (CloudAngels), enabling content providers to leverage elastic cloud resources, on the fly, for the purpose of meeting their MDT objectives. We presented performance evaluation results that confirm the utility and potential of our paradigm and system.

Our current work is focused on making the GT strategy more agile, by allowing dynamic adjustments to the assignment of segments to clients as a result of changes in network conditions, and on extending the CloudAngels service to enable pipelined bulk-synchronous distribution of content from a single provider to a set of clients as well as to enable concurrent bulk-synchronous distribution of content from multiple providers.

Acknowledgement: We would like to thank John Byers, Jonathan Appavoo, and George Atia for their invaluable feedback at various stages of this work.

References

[1] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1456–1471, 2002.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *SIGCOMM 2002*, Pennsylvania, USA.

[3] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *SOSP*, 2003.

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

[5] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent," in *NSDI*, 2007.

[6] OpenP4P, http://www.openp4p.net.

[7] Akamai Netsession, http://www.akamai.com/client/.

[8] Octoshape, https://www.octoshape.com/.

[9] Pando Networks, http://www.pandonetworks.com/cdn-peering.

[10] Bittorrent DNA, http://www.bittorrent.com/dna/technology.

[11] BSP, http://www.bsp-worldwide.org.

[12] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *ACM SIGCOMM '07*, Kyoto, Japan.

[13] R. Kumar and K. W. Ross, "Peer-Assisted File Distribution: The Minimum Distribution Time," in *HOTWEB 2006*, Boston, USA, 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] J. Mundinger, R. Weber, and G. Weiss, "Optimal scheduling of peer-to-peer file dissemination," *Journal of Scheduling*, 2008.

[16] D. M. Chiu, R. W. Yeung, J. Huang, and B. Fan, "Can Network Coding Help in P2P Networks?" in *WiOpt 2006*, Boston, USA.

[17] R. Sweha, A. Bestavros, and J. Byers, "Angels: In-network support for minimum distribution time in p2p overlays," in *BUCS-TR-2009-02*.

[18] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest First and Choke Algorithms are Enough," in *IMC 2006*.

[19] Emulab, https://www.emulab.net/.

[20] Instrumented BitTorrent, https://gforge.inria.fr/projects/bt-instru/.

[21] R. S. Peterson and E. G. Sirer, "Antfarm: efficient content distribution with managed swarms," in *NSDI 2009*, Boston, USA.

[22] R. Kumar and K. W. Ross, "Optimal peer-assisted file distribution: Single and multi-class problems," Tech. Rep., 2007, http://cis.poly.edu/~ross/papers/MinimumDistributionTime.pdf.

[23] G. Smaragdakis, A. Bestavros, N. Laoutaris, J. W. Byers, P. Michiardi, and M. Roussopoulos, "Swarming on optimized graphs for n-way broadcast," in *INFOCOM*, 2008, pp. 141–145.

[24] D. Carra, G. Neglia, and P. Michiardi, "On the impact of greedy strategies in bittorrent networks: The case of bittyrant," in *P2P '08*.

[25] Y. Guo, C. Liang, and Y. Liu, "dhcps: decentralized hierarchically clustered p2p video streaming," in *CIVR '08: Proceedings of the 2008 international conference on Content-based image and video retrieval*. New York, NY, USA: ACM, 2008, pp. 655–662.

[26] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast," in *ICNP 2006*.