On Cooperative Content Distribution and the Price of Barter

Prasanna Ganesan
Stanford University
prasannag@cs.stanford.edu

Mukund Seshadri
U.C. Berkeley
mukunds@cs.berkeley.edu

Abstract

We study how a server may disseminate a large volume of data to a set of clients in the shortest possible time. We first consider a cooperative scenario where clients are willing to upload data to each other and, under a simple bandwidth model, derive an optimal solution involving communication on a hypercube-like overlay network. We also study different randomized algorithms, and show that their performance is surprisingly good. We then consider non-cooperative scenarios based on the principle of barter, in which one client does not upload to another unless it receives data in return. A strict barter requirement increases the optimal completion time considerably compared to the cooperative case. We consider relaxations of the barter model in which an efficient solution is theoretically feasible, and show that obtaining a high-performance practical solution may require careful choices of overlay networks and data-transfer algorithms.

1 Introduction

Consider a server S wishing to distribute a large file F to a set of clients C. This scenario arises in a variety of different contexts today. For example, the file F could be a software patch desired by all end hosts C that have the software installed. As another example, S could be a company transmitting a video file to a set of subscribers; *ESPNMotion* is a recent service transmitting sports highlights to end users in this fashion. Server S could be a node in a content-distribution network from which data needs to be propagated to all the other nodes C. Finally, S could simply be an end host, transmitting data to other end hosts like in the popular BitTorrent protocol [6].

In many of the above cases, S might not possess enough upload bandwidth to individually transfer the entire file F to each of the clients within a short period of time. The process could complete significantly faster if the clients helped by uploading partial fragments of F to each other. In this paper, we consider content-distribution algorithms that transfer F to all the clients in the shortest possible time.

The design of these algorithms are strongly influenced by assumptions about how cooperative clients are in offering their upload bandwidth to help other clients. If clients are perfectly cooperative, there is a great deal of freedom in devising the content-distribution algorithm. However, if clients are likely to be selfish, the distribution algorithm needs to build in mechanisms to force clients to upload data in order to improve their own download performance. The different mechanisms we study are all loosely based on the principle of barter – a client does not upload data to another client unless it receives data in return.

As one may imagine, the barter principle strongly constrains the design of a content-distribution algorithm. Our goal in this work is to devise optimal content-distribution algorithms under a variety of models – ranging from no barter requirement at all to a strict barter requirement – in order to understand the efficiency loss incurred due to barter. We may regard this loss as the price to be paid for dealing with selfish clients as opposed to cooperative ones. We now describe the different content-distribution models, and our contributions in each scenario.

The Cooperative Case Let us first consider a cooperative model where all clients are always willing to upload data at the maximum rate that they are capable of. In this case, one might imagine a variety of different algorithms for distributing content, ranging from a simple multicast tree rooted at the server S, to a more sophisticated multiple-tree structure like SplitStream [7], or even an unstructured solution like BitTorrent [6], in which nodes communicate in a random overlay, exchanging data whenever possible. This plethora of solutions leaves us with two questions:

- What is the optimal solution that minimizes the time taken for all clients to receive the file?
- How well do different natural strategies for content distribution perform compared to the optimal solution?

Section 2 addresses these two questions. Under a simple bandwidth model, we derive the optimal solution, analytically establish its completion time, and show that it can be implemented via communication on a hypercube-like overlay network. We also develop randomized algorithms implemented on unstructured overlays that perform surprisingly well and provide near-optimal performance.

What Price Barter? In Section 3, we consider different distribution models based on barter and study how to generalize our cooperative algorithms for this scenario. It turns out that the exact definition of the barter model has a big impact on the efficiency of content distribution. For example, in the *strict barter* model that we define, where one client transfers data to another only if it simultaneously receives an equal amount of data in return, the optimal solution is much worse than the optimal cooperative solution.

A key contribution of this paper is to develop different barter-like mechanisms and explore the three-way trade-off between the mechanisms' enforceability, their ability to incentivize uploads, and the efficiency of content distribution. To this end, we consider three different mechanisms based on barter, informally analyze their incentive structure, derive lower bounds and develop actual algorithms for content distribution under the mechanism.

From our analysis, we discover that there are indeed mechanisms that provide robust incentives for uploads while having theoretically feasible algorithms that are as efficient as the optimal cooperative algorithms. However, developing practical algorithms for content distribution introduces more challenges. We discover that randomized algorithms operating under the barter model are extremely sensitive to parameters such as the degree of the overlay network they operate on. Our simulations shed light on the critical value of the overlay-network degree, as well as the performance impact of different policies governing the actual data blocks that are exchanged.

We note that our focus is not on game-theoretic analysis of different mechanisms to identify the optimal strategy for selfish nodes. Rather, the question we consider is more basic: given the natural and intuitive fairness constraints imposed by a barter-based incentive mechanism, we study how to devise efficient content distribution under those constraints, assuming that any algorithm obeying the mechanism will be acceptable to nodes.

2 Cooperative Content Distribution

In this section, we study different algorithms for content distribution, assuming all clients are willing to upload data at their maximum upload bandwidth at all times. We first describe our bandwidth and data-transfer model, and then proceed to consider a variety of algorithms.

2.1 Model

As before, we have a server S and a set of clients C. For notational convenience, we assume that there are $n-1$ clients $C_1, C_2, \ldots C_{n-1}$, for a total of n nodes including the server.

Bandwidth Model We assume that all nodes (including the server) have the same upload bandwidth U and the same download bandwidth D (with $D \geq U$). Furthermore, we assume that all transmission bottlenecks are at the tail links, i.e., the effective transfer bandwidth from node X to node Y is equal to the minimum of X's available upload bandwidth and Y's available download bandwidth. While this bandwidth model is extremely simple, it will prove useful for us to reason about different algorithms.

Data-Transfer Model We assume that a data transfer from a node X to a node Y has to involve a minimum quantum of B bytes, that we call a block. The block size is assumed to be large enough to ensure that (a) the entire available bandwidth is saturated by the transmission, and (b) the propagation delay and the transmission start-up time, if any, are much smaller than the transmission time. (Of course, a node cannot begin transmitting a block until it has received that block in its entirety.)

Time Unit For notational convenience we will define B/U to be equal to 1 *tick*, so that each node can transmit at the rate of one block per tick.

Finally, we let the file F to be transmitted from the server consist of exactly k blocks $B_1, B_2, \ldots B_k$, of size B. (We ignore round-off issues that may make the last block smaller.)

Problem What is the best way to organize data transfers to ensure that all $n-1$ clients receive file F at the earliest possible time? In other words, if client C_i receives the complete file at time t_i, we want to minimize $\max_i(t_i)$.

2.2 Simple Examples and a Lower Bound

We illustrate our model by analyzing some simple algorithms for content distribution. We then compute a lower bound for the completion time achievable by any algorithm.

2.2.1 The Pipeline. One simple solution is to pipeline the download, i.e., S sends the file, block by block, to C_1, which pipelines it to C_2 and so on. The completion time for this strategy is $k+n-1$ ticks, since it takes k ticks to get all k blocks out of the server, and a further $n-1$ ticks for the last block to trickle down to the last client.

2.2.2 A Multicast Tree. Consider arranging all n nodes in a d-ary multicast tree ($d > 1$) with S at the root. Since each node can transmit data at the rate of 1 block per tick, it takes d ticks for a block to be transmitted from a node to all its children. Thus, the amount of time required for S to transmit all k blocks is kd. In addition, the last block transmitted by S needs to be propagated down the tree which requires time equal to d times the depth of the tree. Therefore,

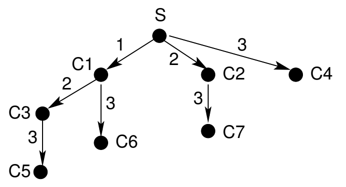

Figure 1. A binomial tree with $n = 8$. Edges labeled with the tick where they are used.

the total completion time using a $d-$ary multicast tree is equal to $d(k+\lceil\log_d(n(d-1)+1)\rceil-2) \simeq d(k+\lceil\log_d n\rceil)$.

2.2.3 The Binomial Tree. Consider the case when the file F consists of exactly one block, i.e., $k = 1$. We can then use the following strategy, depicted in Figure 1: During the first tick, S sends the block to C_1. In the second tick, S and C_1 transmit to C_2 and C_3 respectively. In the next tick, all four of these nodes transmit to four new nodes, and so on, thus doubling the number of completed nodes at each tick. The resultant pattern of data transmission forms a binomial tree, as seen in the figure. It is easy to show that the completion time for this strategy is $\lceil\log n\rceil$, and that this completion time is optimal for the case $k = 1$.

When $k > 1$, one simple way to extend the binomial tree strategy is to send the file one block at a time, waiting till a block finishes before initiating transfer of the next block. This strategy has a completion time of $k\lceil\log n\rceil$[1].

2.2.4 A Lower Bound. We now present the following theorem establishing a lower bound on the time required for cooperative content distribution.

Theorem 1. *Transmitting a file with k blocks to $n-1$ clients requires at least $k + \lceil\log n\rceil - 1$ ticks.*

Proof. Observe that after the first $k - 1$ ticks, there is still at least one block, say B_x, left in the server that is not possessed by any other node. The amount of time needed for block B_x to propagate to all nodes, starting from the server is at least $\lceil\log n\rceil$ ticks, since the number of nodes with B_x can at most double in each tick. Therefore, the overall content distribution requires at least $k + \lceil\log n\rceil - 1$ ticks. $\square$

The above lower bound is, in fact, tight, as will be demonstrated presently. Note that neither this lower bound, nor the completion times of the simple algorithms we have seen so far, depend on the *download bandwidth* of nodes. So long as the download bandwidth is larger than the upload bandwidth, the system is bottlenecked by the latter.

2.3 The Binomial Pipeline

We now present the Binomial Pipeline which achieves the optimal completion time for content distribution.

[1]All logarithms are to base 2 unless otherwise noted.

2.3.1 Two Simplifying Assumptions. We begin with two assumptions to explain the algorithm's intuition: (a) $n = 2^l$ for some integer $l > 0$, and (b) any pair of nodes can communicate with each other. Although a slightly suboptimal solution for this special case has been introduced in prior work [21], we describe it in detail to set the stage for the remainder of the paper. We partition the algorithm's operation into the following three stages.

The Opening The opening phase of the algorithm ensures that each node receives a block as quickly as possible, so that the entire upload capacity in the system can begin to be utilized. This phase lasts for l ticks (where $n = 2^l$), and is characterized by two simple rules:

- During tick i, the server S transmits block B_i to a client that possesses no data.
- Each client, if it has a block before tick i, transmits that block to some client that possesses no data.

Observe that since there are 2^l nodes in total, all clients have exactly one block at the end of l ticks. The communication pattern is, in fact, the same as in the binomial tree of Figure 1, with all nodes in the C_1-subtree having block B_1, all nodes in the C_2-subtree having B_2 and C_4 having block B_3. In general, we can partition the $n-1$ clients into l groups, $G_1, G_2, \ldots G_l$, of sizes $2^{l-1}, 2^{l-2}, \ldots 1$ respectively, with all nodes in G_i having B_i.

The Middlegame After the opening, all nodes have a block, and the algorithm enters the middlegame. During this phase, the objective is to ensure that every node transmits data during every tick, so that the entire system upload capacity is utilized. One may view the middlegame algorithm as a strategy that determines, at every tick, which node transmits which block to which client. The Binomial Pipeline ensures that $n-1$ nodes transmit during every tick t for all $t \geq l$, while maintaining the following invariants:

- Clients are partitioned into l groups $G_{t-l+1}, G_{t-l+2}, \ldots, G_t$ with sizes $2^{l-1}, 2^{l-2}, \ldots, 1$ respectively.
- For $t - l < i \leq t$, block B_i is possessed exactly by the clients in group G_i.
- All clients have blocks $B_1, B_2, \ldots B_{t-l}$. No client has blocks $B_{t+1}, B_{t+2}, \ldots B_k$.

Observe that all three invariants hold after l ticks, since the opening partitions nodes into l groups with each group possessing a unique block in $B_1, \ldots B_l$.

We illustrate the middlegame with the example of Figure 1 (with $n = 8$ and $l = 3$). After three ticks, clients are in three groups G_1, G_2 and G_3, as shown in Figure 2(a). The arrows in the figure show the transmissions that take place during the fourth tick: S hands off block B_4 to C_1, while nodes C_3, C_5 and C_6 are paired up with C_2, C_7 and C_4 respectively, exchanging the only blocks that they have. The consequence is the formation of a new set of three groups as

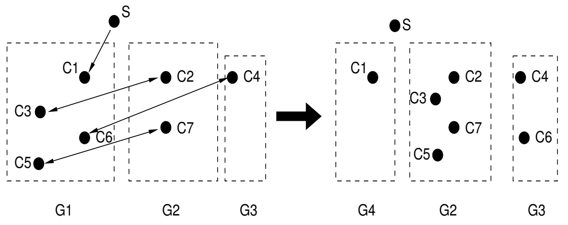

Figure 2. (a) Binomial pipeline transfers during the fourth tick (b) The new set of groups

shown in Figure 2(b), that restores the invariants; block B_1 is now held by all the nodes, four nodes have B_2, two have B_3 and one node alone has B_4. Observe that this situation is almost identical to that at the end of the third tick, with only the group names and members being different.

In general, the following transfers occur at any tick t:

- Server S selects any node in G_{t-l+1}, say C_x, and hands it B_{t+1}. C_x becomes the sole member of G_{t+1}.
- Each node in $G_{t-l+1} - \{C_x\}$ is paired with a unique node in the remaining groups, and transmits B_{t-l+1} to it, thus ensuring every node holds B_{t-l+1}.
- Similarly each node in group G_i, $t - l + 1 < i \leq t$, sends block B_i to a unique node, say C_r, in $G_1 - \{C_x\}$. Node C_r then migrates to group G_i.

The Endgame The middlegame proceeds as above until k ticks elapse, i.e., S transmits block B_k to create a new group G_k. After this tick, the middlegame algorithm runs out of blocks for S to transmit. To work around this, we define $B_{k+\alpha} = B_k$ for all $\alpha > 0$, and let the middlegame algorithm to continue running, i.e., the server keeps transmitting the last block when it runs out of new blocks to send.

After tick $k+l-1$, observe that all nodes have all blocks from B_1 to B_{k-1} (by the invariants) and the only groups left are $G_k, G_{k+1}, \ldots G_{k+l-1}$. But since $B_{k+\alpha} = B_k$ for all $\alpha > 0$, this implies that all nodes have B_k, thus indicating that the transfers are complete[2]. Since $l = \lceil \log n \rceil$, the completion time of the Binomial pipeline, when n is a power of two, is $k + \lceil \log n \rceil - 1$ which is optimal.

2.3.2 A Hypercube Embedding. Our algorithm description leaves open the question of exactly which nodes communicate with which others at what times; in fact, our algorithm is non-deterministic, and there are many options available at every tick. In practice, if a node has to maintain and manage connections with nearly all other nodes, the system is unlikely to scale well with n. Thus, we would ideally like our algorithm to require each node to interact only with its "neighbors" on a low-degree overlay network.

It is easy to show that no optimal algorithm can operate on an overlay network with degree less than $l = \log n$.

[2]Note that if $k < l$, we may once again set $B_\alpha = B_l$ for all $\alpha > l$ and proceed as usual.

Interestingly, the Binomial Pipeline can be executed on an overlay network with degree exactly equal to l – the hypercube. To explain, let us assign a unique l-bit ID to each of the n nodes, with the server node assigned the ID with all bits zero. Links in the overlay network are determined by the hypercube rules applied to node IDs, i.e., two nodes form a link if and only if their IDs differ in exactly one bit. Define the *dimension-i link* of a node to be its link to the node whose ID differs in the $(i+1)^{\text{st}}$ most significant bit. The Binomial Pipeline is then summarized as below.

During the t^{th} tick, for $1 \leq t \leq k + l - 1$, each node X uses the following rules to determine its actions:

- X transmits data on its dimension-($t \bmod l$) link.
- If $X = S$, it transmits block B_t. ($B_t = B_k$ if $t > k$.)
- Otherwise, X transmits the highest-index block that it has, i.e., the block B_i with the largest value of i. (If X has nothing, it transmits nothing.)

The first rule states that each node uses its l links in a round-robin order; at every tick, all data transfers occur across one dimension of the hypercube. The latter two rules dictate what data is transferred, and compactly characterizes the actions of the Binomial Pipeline.

2.3.3 Generalizing to Arbitrary Numbers of Nodes. What happens when the number of nodes n is not an exact power of two? We can carry over the intuition of the Binomial Pipeline to this general case too. Let $l = \lfloor \log n \rfloor$ and assign each client a non-zero l-bit ID. (The server receives the ID with all zeroes). We ensure IDs are assigned such that (a) each ID is assigned to at least one client, and (b) no ID is assigned to more than two clients. Observe that this is always feasible since the number of clients is at most twice the number of available IDs.

Consider the hypercube resulting from the ID assignment, in which each vertex corresponds to either a single node or to a pair of nodes. We may now run the same Hypercube algorithm as earlier, treating each vertex as a *single logical node*. All that remains is to describe how this "logical node" actually transmits and receives data.

If the logical node is a single physical node, it simply acts as in the Hypercube algorithm. So, consider a logical node (X, Y) consisting of two real clients X and Y. During every tick, the Hypercube algorithm requires (X, Y) to transmit some block, say B_i, and receive some block, say B_j. We use the following rules to determine the actions of X and Y during this tick:

- If X has B_i, it transmits B_i. Otherwise, Y is guaranteed to have B_i and Y transmits it.
- Whichever node is not transmitting B_i will receive B_j.
- Say X transmits B_i and Y receives B_j. If Y has a block that X does not, Y transmits this block to X. (And similarly if the roles of X and Y are reversed.)

It is easy to see that, at all times, X can have at most one block that Y does not, and vice versa. Therefore, once the Hypercube algorithm terminates after $k+l-1$ ticks, both X and Y might be missing at most one block. They may use an extra tick to exchange these missing blocks, ensuring that the overall algorithm terminates in $k + l = k + \lceil \log n \rceil - 1$ steps. Thus, this generalization of the Hypercube algorithm is optimal for all n.

Note that this algorithm uses an overlay network with the out-degree of each node being $\lceil \log n \rceil$, although the in-degree of some nodes may be as high as $2\lceil \log n \rceil$.

2.3.4 Other Observations. The Binomial Pipeline possesses many interesting properties and generalizations, besides the fact that it may be embedded in a hypercube-like overlay network. We list some of them here.

Individual Completion Times We have seen that the overall completion time of the algorithm is optimal. But when does each node complete receiving the file? It turns out that all nodes finish receiving the file at exactly the same tick, so long as $k > 1$.

Higher Server Bandwidths Another interesting question is to consider what happens if the server S had a higher upload bandwidth. If the server has an upload bandwidth of λU, where U is the upload bandwidth of the clients, it turns out that the natural strategy of breaking up the clients into λ equal groups and breaking up the server into λ virtual servers, one for each group, is optimal.

Optimizing for Physical Network In a situation where the available bandwidth between different pairs of nodes may be different, depending on their location in the physical network, we could "optimize" the hypercube structure using embedding techniques, such as those discussed in [12]. Such techniques help find the "best" hypercube that may be constructed with the given set of nodes, thus optimizing for the location of nodes in the physical network.

Dealing with asynchrony So far, we have assumed that the entire system operates in lock-step. In reality, different nodes may have slightly differing bandwidths. We may consider operating the hypercube algorithm even in these settings, with each node simply using its links in round-robin order at its own pace. This approach is closely related to the randomized algorithms that we discuss next.

2.4 Randomized Approaches

The optimal solution that we have seen required nodes to be interconnected in a rigid hypercube-like structure, and tightly controlled the inter-node communication pattern. In practice, such a rigid construction may not be particularly robust, leading us to investigate the performance of simpler, randomized algorithms for content distribution.

Recall that, in every tick, the optimal algorithm carefully finds a "maximal mapping" of uploading nodes to downloading clients to ensure that nearly all nodes upload data[3]. A natural simplification is to attempt a distributed, randomized mapping, instead of finding a maximal one.

2.4.1 Model. We will assume that nodes are interconnected in some overlay network G, and each node is aware of the exact set of blocks present at all its neighbors. Such knowledge is necessary for nodes to efficiently find clients to upload to in a distributed fashion. In a practical protocol, this knowledge could be maintained by letting every node inform all its neighbors every time it finishes receiving a block, like in BitTorrent. Note that we make no assumptions about the structure of the overlay network G.

2.4.2 Algorithm. During every tick, each node X uses the following two-step process to find a client to upload to: (We explain italicized segments subsequently.)
1. Let $\mathcal{N}$ be the set of neighbors which require a block that X has. Select a random node Y from $\mathcal{N}$ *with sufficient download capacity*.
2. Let R be the set of blocks available with X and desired by Y. Upload one of the blocks in R according to the *block-selection policy*.

Step 1 requires X to find a random "interested" neighbor Y. If many nodes simultaneously pick the same Y to upload to, there may be a downloading bottleneck at Y. To sidestep this problem, we exploit the fact that a real system is asynchronous; therefore, a handshake protocol between X and Y could be used to verify that Y has sufficient download capacity (and to resolve collisions), and X can avoid selecting Y otherwise. Note that if no node Y matches the requirements of Step 1, X does not transmit any data during that tick.

There are many possible block-selection policies that may be used in Step 2. The simplest is *Random*, in which a random block in R is uploaded to Y. (Again, a handshake protocol may be used to prevent Y from getting the same block from more than one sender). We may also use *Rarest First*, in which the least frequent block is uploaded. There are different ways to estimate block frequency and we omit details here.

A Note on Costs Observe that an implementation of this algorithm requires (a) a protocol to inform nodes of their neighbors' content and (b) a handshake protocol to decide on data transmission. The cost of implementing both these requirements grows as the degree of the overlay network G increases. (The cost of (a) grows linearly with degree.) It is therefore important to keep the degree of G small.

2.4.3 (Counter-)Intuition. A theoretical analysis of the above algorithm remains an open problem; instead, we try

[3]Note that such a maximal mapping is necessary but not sufficient for optimality. There are many maximal mappings for a particular tick which, if used, would rule out maximal mappings in subsequent ticks. In the special case of $n = 2^l$ however, all maximal mappings lead to optimality.

to gain some intuition about its performance. For simplicity, assume there are $n = 2^l$ nodes with infinite download capacity, interconnected in a complete graph. Observe that the algorithm's completion time is inversely proportional to the average fraction of nodes that upload data in each step.

Assume, optimistically, that the algorithm proceeds just like the optimal in the opening phase. (This is nearly true in reality.) Nodes are then in l groups, $G_1, G_2, \ldots G_l$ with group G_i owning block B_i. Consider what happens at the $(l+1)^{st}$ tick. Our algorithm maps each node X in group G_i to a random node Y that does not have B_i, i.e., a random node $Y \notin G_i$, while ensuring that multiple nodes in G_i do not map to the same Y.

Consider the nodes in G_1. Since $|G_1|$ is $n/2$, each of the G_1 nodes will map to a distinct node outside G_1 and all nodes end up owning B_1 at the end of this tick. For any node in group G_i, $i \neq 1$, all nodes outside G_i are interested in its content; since $|G_i| \leq n/4$, at least $3n/4$ nodes are interested, out of which $n/2$, i.e., two-thirds, belong to G_1. Therefore, at least one-third of the G_1-nodes, i.e., $n/6$ nodes, are not expected to receive any block in this tick. (Note that if a transfer occurs between a pair of nodes, neither of which is in G_1, then some node in G_1 will not be able to receive any block in that tick.)

Observe that these $n/6$ nodes will not upload any data at all in the next tick $(l+2)$, since the only block they will have is B_1, which is already owned by all the nodes. Therefore, at most five-sixth of the nodes will transmit data at tick $(l+2)$. Given the simple and weak inequalities used in our argument, it seems reasonable to conjecture that at most five-sixth of the nodes will transmit data at *every* tick. From this conjecture, we may guess that the randomized algorithm is at least 20% worse than the optimal algorithm. The experimental results, however, prove otherwise.

2.4.4 Results. We simulated the randomized algorithm using a synchronous simulation identifying the completion time T for different values of k and n, using a variety of different overlay networks G, and with different download bandwidths D and block-selection policies. We summarize the key results below. Block size B is assumed to be a constant across all simulations and completion time T is expressed in ticks. Since a tick is defined as B/U, T can be meaningfully compared across simulations.

Completion Time T vs. k and n First, we estimate T as a function of k and n to understand the quality of the algorithm. (We set G to be the complete graph, $D = U$ and use *Random* block selection, although the results are almost identical with other settings, as we discuss later.)

Figure 3 plots the mean value of T as a function of n with the x-axis on a log scale. (The error bars on each point represent the 95% confidence intervals on the mean, obtained through multiple algorithm runs.) We observe that

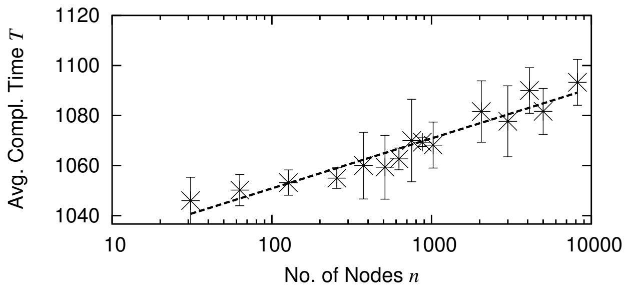

Figure 3. Completion Time T vs. n

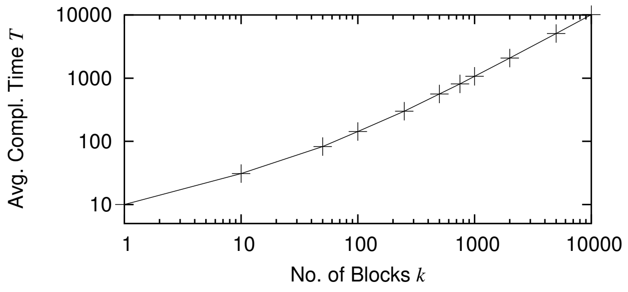

Figure 4. Completion Time T vs. k

T increases roughly linearly with $\log n$. Replicating the experiment with different values of k produces the same near-linear behaviour. Figure 4 plots T for different values of k on a log-log scale, keeping n fixed at 1024. We see that T increases linearly with k. (The same behaviour was observed with other values of n too.)

Given the above evidence, we hypothesized that, to a first order of approximation, T is a linear function of k and $\log n$. [4] Using least-square estimates over a matrix of 106 data points, we estimate that the expected completion time $T = 1.01k + 4.4 \log n + 3.2$, suggesting that the algorithm is less than 2% worse than the optimal for large values of k. This is a surprising result unexplained by our earlier intuition. A closer analysis of the algorithm's runs suggests that there is some "amortization" going on, by which a "bad" tick where few data transfers take place is compensated for by many succeeding "good" ticks where the transfer efficiency is 100%, belying the intuition developed earlier.

Effects of the Overlay Network The experiments above were performed using a complete graph as the overlay network. We now consider random regular graphs instead (in which each edge is equally likely to be chosen), and investigated the effect of graph degree on completion times. Figure 5 shows this effect with $n = 1000$, and two different settings of k: 1000 and 2000. We observe that the completion time drops steeply as the degree increases, and converges quickly to the final value when the degree is around

[4](We conjecture that the true equation is of the form $T = k + O(\textit{polylog}\ n) + o(k)$.)

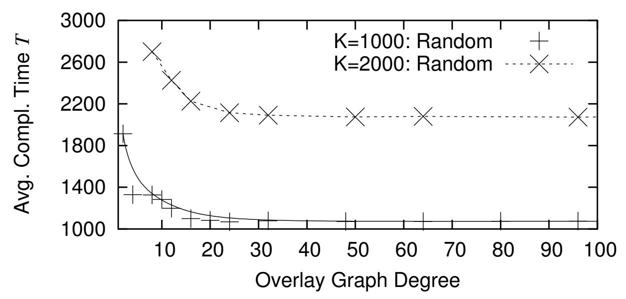

Figure 5. T **vs.** $Degree$ **with** $n = 1000$

25, irrespective of the value of k. This suggests that the phenomenon may be related to the mixing properties of G, with near-optimal performance kicking in when the graph degree is $O(\log n)$. Interestingly, we execute the same randomized algorithm using a hypercube-like overlay network (with average degree 10 for $n = 1000$) and find that the performance matches that of the randomized algorithm on the complete graph. Thus, *it is useful to replace a random graph by a hypercube-like structure if the objective is to minimize the graph degree.*

Block-Selection Policy and Incoming Bandwidth Constraints Finally, we also attempt using the Rarest-First block-selection policy instead of Random, and also vary the incoming bandwidth of nodes from U to infinity. We find that there are no significant differences in the overall results or trends as a consequence of these variations; we omit further details here.

3 Content Distribution in a Barter Economy

So far, we have assumed that clients freely offer their upload bandwidth in order to help other clients. In reality, clients are unlikely to behave in this fashion unless they have an incentive to do so. Therefore, we need mechanisms that ensure that it is in the clients' interest to upload data if they want good download performance.

A naive solution is to require the server to monitor the upload activity of all clients at all times, and to "cut off" non-uploading clients, by informing other nodes that they should stop providing these clients with data. This would incentivize clients to upload, since they'd realize that they would be cut off if they didn't upload. However, this mechanism is highly impractical. First, it is impossible for the server to verify whether one node uploads to another, since either of the two nodes might "lie". Second, there may be far too many clients for the server to monitor all of them all the time.

The natural solution to these problems is to make the mechanism decentralized, letting each node decide for itself whom to upload to and when. The decision to upload data or not is loosely guided by the principle of barter – X will not upload data to Y unless Y uploads to X in return. In this section, we consider different mechanisms guided by this principle, discussing three issues for each:

- How well are nodes incentivized to upload data?
- What is the fundamental efficiency limit imposed by the mechanism on content distribution?
- Are there simple content-distribution algorithms that are optimal, or near-optimal, for that mechanism?

Our answers to the first question will be based on intuitive arguments; we will *not* formally analyze the mechanism to identify the dominant strategy for selfish nodes. For the latter two questions, we are concerned with the fundamental obstacles imposed by the mechanism itself on content-distribution, and not with the inefficiency induced by selfish clients' strategic behavior. (Thus, we will consider any algorithm obeying the mechanism as an "acceptable" solution rather than requiring that the algorithm operate at a Nash equilibrium under the mechanism.)

3.1 Strict Barter

We begin by considering a mechanism where all data transfers between clients occur strictly by barter; client X will transfer a block to Y only if Y *simultaneously* transfers a block to X. Of course, the blocks received by X and Y must not be possessed by them already. The one exception to barter-based transfers is for the server itself, which uploads data without receiving anything in return.

3.1.1 Incentive Analysis. Observe that this mechanism creates a strong incentive for clients to upload data, since this is the only way to receive data from other clients. In particular, a client attempting to limit the *rate* at which it uploads data will experience a corresponding decay in its download rate, thus forcing the client to upload at the maximum rate to obtain best download performance. However, if nodes are capable of subverting the protocol itself, the mechanism can be "broken" by nodes uploading garbage data in order to increase their download rate. Thus, this mechanism may be inappropriate in such scenarios.

3.1.2 A Lower Bound. How does the barter constraint affect the completion time T for content distribution? We show the following lower bound on T .

Theorem 2. *Any content-distribution algorithm based on strict barter imposes a completion time of at least* $\min(k + n/2, \frac{n-1}{n}(k + n/2 + 1/2))$ *ticks. If the download capacity of nodes* D *is equal to the upload capacity* U*, any content distribution algorithm requires at least* $k + n - 2$ *ticks.*

Proof. Let us first consider the case $D = U = 1$ block/tick. Observe that the first block received by a client must be obtained from the server S, since the client cannot barter for a block without already owning a block. Since there are $n - 1$ clients, there is at least one client, say X, which possesses at most one block after the first $n - 1$ ticks. After this time, X needs to receive at least $k - 1$ more blocks; since its download capacity is only 1 block/tick, the total time taken for X to receive all its blocks is at least $n - 1 + k - 1 = k + n - 2$ ticks.

Now consider the case $D > U$. We use a different argument in this case. Again, since nodes cannot initiate barter before receiving their first block, and at most one client can receive a block from the server at each step, the number of clients capable of barter after t ticks, for $t \leq n-1$ is at most t. Since barter requires pairs of nodes to exchange data, the maximum number of uploads that can take place at time t is t if t is odd, and $t - 1$ otherwise. The maximum number of uploads when $t \geq n$ is n. Since the total number of uploads needed overall is $(n - 1)k$, it is easy to show that the total time required is as stated in the theorem. □

The analysis brings out the fact that barter is handicapped by a high start-up cost – it takes $n-1$ ticks before all nodes receive a block and are able to barter. In this period, only $n/2$ blocks are uploaded on average per tick, leading to a high overall cost linear in both k and n.

3.1.3 Algorithm. The lower bound presented above is fairly tight; we devise an algorithm called the Riffle Pipeline, which has a completion time close to the lower bound, as described by the following theorem.

Theorem 3. *If the client download capacity D is at least twice the upload capacity U, it is possible to complete content distribution under barter within $k + n - 1$ ticks.*

Note that the above theorem assumes $D \geq 2U$; otherwise, there is an additional $(1 + 1/n)$-factor overhead involved. We first describe the algorithm for the special case $k = n - 1$, i.e. the number of clients is exactly equal to the number of blocks. The Riffle Pipeline is described by the data-transfer schedule below.

During each tick t, $1 \leq t < 2n - 2$,

- If $t < n$, server S sends block B_t to client C_t.
- For each $1 \leq i \leq n/2 - 1$, if $i \leq (t - 1)/2$ and $t-i \leq n-1$, client C_i barters with client C_{t-i}, giving up block B_i and obtaining block B_{t-i} in return.

Observe that S talks to clients in the sequence $C_1, C_2, \ldots C_n$. Client C_1 also talks to clients in the same sequence (excluding itself), but trailing S by a tick. C_2 also uses the same sequence (excluding C_1 and itself), but trailing C_1 by a tick, and so on. The algorithm completes in $2n - 3 = k + n - 2$ ticks.

To illustrate the above schedule, consider client C_1. In the first tick, it receives block B_1 from S. It does nothing in the second tick, while C_2 receives block B_2 from S. Client C_1 then obtains B_2 from C_2 in tick 3 by bartering block B_1; gets B_3 from C_3 in tick 4, again bartering B_1; and so on. Thus, after n ticks, client C_1 obtains all the blocks; all the other nodes also obtain block B_1 by this time. Client C_2 barters with nodes in the same sequence as C_1, but one tick behind; consequently, C_2 obtains all its blocks after $n + 1$ ticks, while all the nodes also obtain block B_2 by this time. The overall process completes in $2n - 3 =$ ticks. (Note that at tick $2n - 3 = k + n - 2$ both clients C_{n-2} and C_{n-1} complete, thus making the completion time $2n - 3$ instead of $2n - 2$.)

What happens when the number of blocks is not exactly equal to $n - 1$? If $k = \lambda(n - 1)$ for some integer λ, we may simply break up the blocks into λ groups, and distribute them one group at a time. After the first n ticks, client C_1 finishes receiving all blocks in the first group, and may start receiving a block from the tick group from server S. With every additional time tick, one more node finishes receiving blocks from the first group and may join the riffle pipeline for the second group. Similarly, after $2n$ ticks, client C_1 can start off the pipeline for the third group of blocks, and so on. The overall completion time for this algorithm is $(\lambda - 1)n + 2n - 3 = k + n - 3 + k/(n - 1)$.

If $D \geq 2U$, we can shave $\lambda - 1$ off the completion time by starting the riffle pipeline every $n - 1$ ticks, instead of n ticks. Thus, at time n, node C_1 would be performing barter with client C_{n-1} while simultaneously downloading the first block of the next group from the server. This reduces the completion time to $k + n - 2$.

Finally, we arrive at the case where k is an arbitrary number, not necessarily a multiple of n. Let $k = \lambda n + c$. We may then run the same algorithm as earlier for the first λ cycles. After these cycles, there are only c blocks left to be delivered and $n > c$ nodes left. We may break up these n nodes into $\lceil n/c \rceil$ groups, with c nodes in each group, except possibly the last. The original algorithm is then applied to the first of these groups, i.e., the server sends the c blocks to the c distinct nodes in the first group, and then lets the nodes exchange data among themselves. Once the server is done with the first group, it moves on to the second group, and so on, until it reaches the last group. If there are exactly c nodes in the last group, the server can again apply the original algorithm and we are done. However, if the number of nodes is less than c, we apply the entire new algorithm *recursively* to solve the problem of distributing c blocks to a number of nodes less than c.

If the download bandwidth $D \geq 2U$, it can be shown that the new algorithm has a completion time of at most $k+n-1$ to distribute the k blocks over n nodes.

3.2 Credit-Limited Barter

Strict barter suffered from two issues. First, nodes could cheat the mechanism by uploading junk and receiving legitimate data in return. Second, there is a high start-up cost involved, which increases the completion time to $k + \Theta(n)$ instead of $k + \Theta(\log n)$.

Both these problems can be solved by modifying the barter mechanism to allow some *slack*. In the credit-limited barter model, any node X is willing to upload a block to a node Y so long as the net data transferred from X to Y so far is at most s. Thus, a node can get s blocks "for free" from another, but has to upload blocks in return if it wants any more. We refer to s as the credit limit. Observe that the credit limit allows us to eliminate the start-up problem; since nodes can get their first block for free, all nodes can receive a block within a logarithmic amount of time.

3.2.1 Incentive Analysis. Credit-limited barter is a robust way to incentivize nodes to upload data. Let us assume that each block has a cryptographic signature that nodes are aware of. (For example, the server could provide this data to each node.) If nodes receive credit for uploading data only after the uploaded data has been verified by the receiver, there is no longer an incentive to upload junk. However, this mechanism does have one loophole: since a node has a credit limit of s with every other node, it could obtain s blocks from each of them without ever uploading data. If k is less than $s(n-1)$, the node may be able to get away without uploading anything at all!

This problem appears fundamental to any *distributed* incentive scheme. One work-around is to impose a *total* credit limit on borrowings of each node but that may be hard to enforce. A different solution is to let the server S "dictate" the overlay network, i.e., a client receives credit only at a small number of "designated neighbors" (while also ensuring s is small, say 1.) Simple cryptographic schemes can be used to implement this restriction, and the enforcement cost is low since establishing the overlay is a one-time operation.

3.2.2 A Lower Bound. What is the impact of credit-limited barter on the completion time? The best lower bound we can show is simply the same as the lower bound for the cooperative case, $k + \lceil \log n \rceil - 1$.

When $s = 2$ and $n = 2^l$, this bound is, in fact, tight. To see this, consider Section 2's Hypercube algorithm with $n = 2^l$; each client gets one free block during the first l ticks, which is within the credit limit of $s = 2$. Subsequently, all inter-client transfers are symmetric, ensuring that the maximum credit limit of any client at the end of any tick is only 1. Since credit for uploads is only granted at the end of the upload, we require $s = 2$ to ensure that content distribution obeys the mechanism. Note, however, that the Hypercube algorithm for arbitrary n does not satisfy the credit-limited barter constraints unless s is very large.

We also note that the Riffle Pipeline satisfies the credit-limited barter constraint with $s = 1$, thus showing that $k + n - 2$ is an upper bound on the optimal completion time.

3.2.3 A Randomized Algorithm. We have seen that it may be feasible to have efficient algorithms under credit-limited barter, at least for special values of n. This raises the question of whether we can devise practical algorithms that operate well under this mechanism.

A natural approach is to modify the randomized cooperative algorithm, from Section 3.2.3, to obey the credit-limit constraint. Once again, we consider nodes connected in an overlay network G. At the beginning of each tick t, each node X attempts to find a neighbor to upload to, just as in our cooperative algorithm. The node picks a random neighbor Y which is interested in X's content, has sufficient download capacity *and is below the credit limit.* Node Y is then given a block chosen according to the block-selection policy – Random or Rarest-First.

3.2.4 Results. We used synchronous simulations to investigate the performance of this randomized algorithm for a wide range of values of k, n and s, as well as with different overlay networks G, and block-selection policies. We summarize the main observations here. (As in Section 2.4.4, B is constant and T is expressed in ticks.)

Impact of Graph Degree Once again, we consider our overlay networks to be random regular graphs G, of different degrees, and study the effect of the graph degree d on the completion time T, for different values of k and n. Figure 6 plots the mean completion time T (with 95% confidence intervals) against d for experiments using $k = n = 1000$, and using Random block selection. Consider the solid line marked *s=1* representing the case $s = 1$. We observe that the graph degree d plays a dramatic role in determining completion times. For $d < 80$, the algorithm performs very poorly, with its completion time being off the charts for $d < 50$. However, there is a sharp transition at $d = 80$, after which its performance is nearly optimal, *and identical to the performance in the cooperative case*. (The plots for other values of s, k and n are similar.)

One might imagine that, since each node is entitled to $d * s$ free blocks in total, the improved performance with increasing d is due to the increased total credit per node. However, this is not the whole story, as evidenced by the dotted line marked $s * d = 100$. In this case, we consider various values of d, and set the credit limit s such that $s * d = 100$, i.e., each node is always entitled only to a total of 100 free blocks. We observe that there is still a dramatic difference in the observed performance with different values of d. Thus, *the graph degree plays a fundamental role in determining the performance of the randomized algorithm; increasing the credit limit with lower graph degrees is nowhere near as powerful as increasing the graph degree itself.*

In practice, a degree-80 random graph may be hard to build; worse still, using even a slightly lower degree may have a serious impact on completion times. (We did, how-

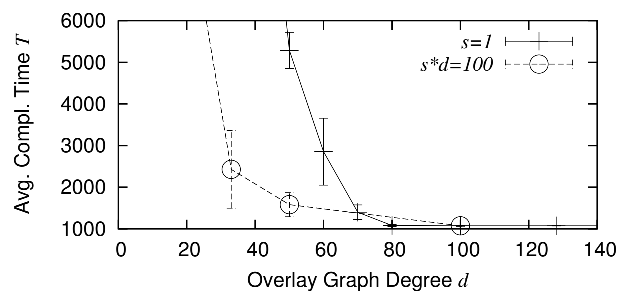

Figure 6. T vs. d with $s * d$ kept constant, for Random block selection

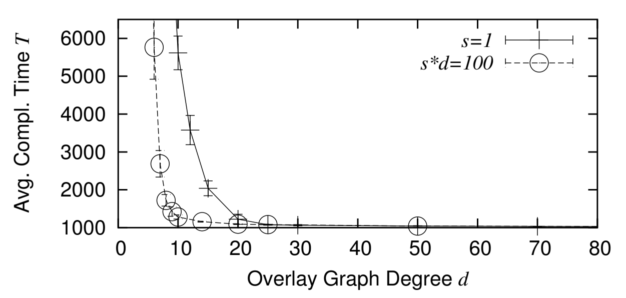

Figure 7. T vs. d with $s * d$ kept constant, for Rarest-first block selection

ever, notice that the effect on the *average* time for nodes to finish is less dramatic than on the completion time.)

The Impact of Rarest-First To study the impact of the block-selection policy, we repeat the earlier experiments, this time using Rarest-First block selection instead of Random. The results, as shown in Figure 7, mimic the results for the Random policy in terms of general behavior, but with a crucial difference: the degree threshold at which the algorithm approaches optimal behavior is now around 20, a fourfold improvement over the degree 80 that was required with the Random policy. (In comparison, using a degree-20 network with Random block selection results in a completion time more than 20 times worse.)

Thus, *the block-selection policy plays a critical role in determining the completion time.* The results depicted above were obtained assuming that nodes have access to perfect statistics about block frequencies. However, the results are almost identical even using simple schemes for estimating frequencies based on the content of nodes' neighbors.

Finally, we experiment with a variation of the algorithm where nodes are constrained in a low-degree overlay network, but allowed to *change* their neighbors periodically. Initial results from this approach appear promising, and we intend to explore it further in future work.

3.3 Triangular Barter

A different way to relax the barter requirement is to allow "transitive" use of credit – A will upload to B if B is simultaneously uploading to C and C is simultaneously uploading to A. We call this triangular barter. This is more flexible than simple barter, since B can receive data from A even if B does not have data that is useful to A. Of course, we could generalize this idea to allow "cyclic barter", involving cycles of any length – but cheat-proof implementation of this generalization is likely to be complex. (Note that this nearly converts barter into a cash-based economy.)

The combination of triangular barter with a credit limit is rather intriguing because it enables provably optimal content distribution with a deterministic algorithm! Observe that Section 2.3.3's generalization of the Hypercube algorithm obeys triangular barter with a credit limit $s = 2$. We do not discuss triangular barter any further in this work, but intend to investigate randomized algorithms for triangular barter, and their potential use in low-degree overlay networks in future work.

4 Related Work

Cooperative Content Distribution SplitStream [7] performs cooperative distribution of streaming data. It uses a clever arrangement of parallel multicast trees to ensure that all nodes upload data at full capacity. It is optimized to deal with dynamic nodes and heterogeneous bandwidths. If bandwidths are homogeneous, SplitStream is near-optimal with a completion time of roughly $k + c \log n$, where c is the number of multicast trees used (a similar analytical result is obtained in [5]). Our results suggest that simple, randomized solutions are good enough in the static cooperative case, avoiding the need for more complex designs. Our randomized algorithms could probably be fine-tuned by using better peer-selection strategies, as suggested by [11].

Prior studies have described the theoretical result of Section 2.2.4, and cooperative content distribution algorithms which are optimal for special values of n [21, 19]. Interestingly, we have recently discovered a *near*-optimal hypercube-based approach [2], and a different optimal algorithm [3] in the discrete-mathematics literature; these algorithms were studied in the context of message broadcast among an arbitrary number of processors.

A recent technical report [15] solves the same cooperative distribution problem that we address in Section 2, using a different algorithm, that is optimal for arbitrary n. In contrast, our approach has a simpler hypercube embedding, leading to a provably low degree bound. The authors of [15] also simulated a randomized algorithm, and found, like us, that the finish times depend linearly on k and $\log n$. However they did not investigate values of k greater than

50; this precludes a more detailed comparison of their results to ours.

Several other mechanisms([4, 13]) for cooperative content distribution have been proposed which have sub-optimal completion times in the scenarios considered here, but are specifically tailored toward goals like locality, robustness, and ability to handle rapid peer arrivals/departures.

Incentive Structures Reference [19] analyzes the incentive structure of BitTorrent as a "bandwidth game" and shows that all nodes upload at peak capacity in a homogeneous system. Their model assumes that (a) nodes can only make a one-time choice determining the upload capacity to use, (b) nodes cannot allocate bandwidth differently to different peers, (c) nodes are aware of the exact bandwidth they would receive from every other node, and (d) every pair of nodes always has useful data to exchange. We do not use these assumptions in our analysis of any mechanism, since our focus is on understanding the complexities arising from individual nodes' content. (A price we pay is the inability to make formal statements about Nash equilibria.) Furthermore, as discussed earlier, the mechanisms we consider are more robust and natural than that of BitTorrent.

A qualitative discussion of the taxonomy of incentive patterns can be found in [17]. The barter model has been discussed in the context of other resources like computation [8] and storage [9]. Note, however, that these resources are not amenable to instant exchange (unlike actual content) and therefore pose a different set of problems. The authors of [16] sketch an incentive mechanism which is similar to ours in the use of "credit thresholds"; their goal is to enforce fair sharing of bandwidth, whereas ours is to minimize completion time. An alternative to the barter system is the use of a currency system; such mechanisms (e.g. [20], [18]) typically involve a greater degree of complexity and centralization than the pairwise barter system discussed here.

A mechanism for "cyclic" barter (which we discussed in Section 3.3) has been evaluated via simulations in [1]: indivisible objects are bartered in a complete graph overlay, and the authors' metric of interest is the mean download time per object. In contrast, our focus is on the *total* time taken to disseminate a single (large) file, and comparing this time in the barter and the cooperative scenarios.

BitTorrent Our randomized algorithms bear a strong resemblance to, and were inspired by BitTorrent [6], which is a widely deployed peer-to-peer content distribution mechanism. (An idea of the scale and operating parameters of this protocol deployment may be obtained from the measurement study in [14].)

In [10, 21, 19], models of BitTorrent are developed in order to analyze the evolution of the system upload bandwidth as nodes join and leave, as well as the completion time. These analyses use simplifying assumptions about the content at different nodes, e.g., by parameterizing the efficiency of content distribution, or assuming it is optimal [19]. This approach is orthogonal to ours, where we focus on a static set of nodes, and pay attention to the complications arising from the *content* at the different nodes in order to understand the efficiency of content distribution.

As ongoing work, we are studying the performance of BitTorrent in greater detail, through asynchronous simulations. Our preliminary results suggest that, even with perfect tuning of protocol parameters, the completion time with BitTorrent is more than 30% worse than the optimal time (from Section 2.2.4). We note that this performance is under the assumption that selfish clients will actually follow the specified BitTorrent behavior. However, since a typical BitTorrent client almost always uploads to a certain minimum number of neighbors irrespective of the reciprocal download rate, selfish clients have little incentive to conform to the specification, and can exploit the system if they are sophisticated enough.

5 Conclusions and Open Questions

We have described optimal and near-optimal algorithms for content distribution under both cooperative and barter-based models. Randomized algorithms work surprisingly well in cooperative settings, providing near-optimal performance, and their theoretical analysis poses interesting open problems. We considered different barter-based mechanisms and found that devising efficient, practical content-distribution algorithms on a low-degree network is non-trivial, raising questions about the best barter mechanism to implement, and the algorithms and parameters to use. Finally, it would be interesting to design mechanisms that provably ensure that rational selfish behavior of clients leads to optimal content distribution.

References

[1] K. G. Anagnostakis and M. B. Greenwald. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *Proc. 24th International Conference on Distributed Computing Systems*, 2004.

[2] A. Bar-Noy and S. Kipnis. Broadcasting multiple messages in simultaneous send/receive systems. *Discrete Applied Mathematics*, 55(2):95–105, 1994.

[3] A. Bar-Noy, S. Kipnis, and B. Schieber. Optimal multiple message broadcasting in telephone-like communication systems. *Discrete Applied Mathematics*, 100(1-2):1–15, 2000.

[4] D. Bickson, D. Malkhi, and D.Rabinowitz. Efficient large scale content distribution. In *Proc. 6th Workshop on Distributed Data and Structures*, 2004.

[5] E. Biersack, P. Rodriguez, and P. Felber. Performance analysis of peer-to-peer networks for file distribution. In *Proc. 5th International Workshop on Quality of Future Internet Services (QOFIS)*, 2004.

[6] BitTorrent protocol specification. http://wiki.theory.org/BitTorrentSpecification.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proc. SOSP*, 2003.

[8] B. Chun, Y. Fu, and A. Vahdat. Bootstrapping a distributed computational economy with peer-to-peer bartering. In *Proc. First Workshop on Economics of Peer-to-Peer Systems*, 2003.

[9] L. Cox and B. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *Proc. 19th ACM Symposium on Operating Systems Principles*, 2003.

[10] G. de Veciana and X. Yang. Fairness, incentives and performance in peer-to-peer networks. In *Proc. Allerton Conference on Communication, Control and Computing*, 2003.

[11] P. Felber and E. Biersack. Self-scaling networks for content distribution. In *Proc. SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, 2004.

[12] P. Ganesan, Q. Sun, and H. Garcia-Molina. Apocrypha: Making p2p overlays network-aware. Technical report, Stanford University, 2003. http://dbpubs.stanford.edu/pub/2003-70.

[13] C. Gkantsidis and P. Rodriguez-Rodriguez. Network coding for large scale content distribution. In *Proc. IEEE INFOCOM*, 2005.

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

[15] J. Mundinger and R. Weber. Efficient file dissemination using peer-to-peer technology. University Of Cambridge, Statistical Laboratory Research Report 2004-01, 2004.

[16] T. Ngan, A. Nandi, A. Singh, D. S. Wallach, and P. Druschel. Designing incentives-compatible peer-to-peer systems. In *Proc. 7th International Conference on Electronic Commerce Research*, 2004.

[17] P. Obreiter and J. Nimis. A taxonomy of incentive patterns - the design space of incentives for cooperation. In *Proc. Second International Workshop on Agents and Peer-to-Peer Computing*, 2003.

[18] Paypal. Website http://www.paypal.com.

[19] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Proc. SIGCOMM*, 2004.

[20] D. A. Turner and K. W. Ross. A lightweight currency paradigm for the P2P resource market. In *Proc. 7th International Conference on Electronic Commerce Research*, 2004.

[21] X. Yang and G. de Veciana. Service capacity of peer to peer networks. In *Proc. IEEE INFOCOM*, 2004.