Fair File Swarming with FOX

Dave Levin Rob Sherwood Bobby Bhattacharjee
Department of Computer Science, University of Maryland, College Park, MD 20742
Emails: {dml, capveg, bobby}@cs.umd.edu

ABSTRACT

File swarming is a popular method of coordinated download by which peers obtain a file from an under-provisioned server. Critical problems arise within this domain when users act selfishly, yet most systems are built with altruism assumed. Working under the assumption that *all* peers are greedy, we introduce the Fair, Optimal eXchange (FOX) protocol. FOX, in addition to effective and robust application of the tit-for-tat incentive mechanism, provides theoretically optimal download times when everyone cooperates. Under our assumption of server and peer capabilities, we develop a strong threat model that provides peers with the incentives to not deviate from the protocol. From a theoretical perspective, we prove FOX's optimality and incentive properties, even when the network consists only of purely self-interested peers. We also discuss issues in implementing and deploying such a system, and address the cost of ensuring fairness in a domain where efficiency is so important.

1. INTRODUCTION

File swarming applications, such as BitTorrent [7], Bullet$'$ [9], and Slurpie [14], allow peers downloading the same file to trade parts of the file with one another instead of simply all going to the server. However, it is relatively trivial to construct free-riding strategies that can vastly degrade these systems' performance. For instance, peers could obtain a block from every other peer in the system without providing any in return.

Peers cannot be expected to "play fair" and share blocks on their own accord unless they have incentive to do so, and, as pointed out by Papadimitriou [12], such incentives should be built into the protocol itself. In this paper, we consider the specific problem of some large number (N) of peers attempting to download the same file from a server. Our solution, called FOX (Fair Optimal eXchange), incorporates an incentives framework built on a novel combination of overlay structure and threats, simultaneously providing both theoretically optimal download times and incentives to follow the protocol. The protocol we present for FOX requires no additional work at the server and, unlike previous systems, is distributed, difficult to exploit, and provably fair.

Rather than assume that some subset of users are altruistic, we expect each peer to act in a purely greedy way. Specifically, we assume that *any peer p will help or hurt other peers only if it improves p's download time*. Within this domain of self-interested peers, the overarching goal of our work is to investigate what primitives are necessary to obtain provable fairness. In many cases, such guarantees may be overkill; it has been observed that some BitTorrent "communities" consist of a group of altruistic users who share a niche interest [1]. Our work focuses on the more general problem of ensuring users of system-wide fairness without any prior knowledge of the other participants.

There are of course many definitions of fairness. Exact fairness, in which each peer uploads exactly as much as it downloads, requires a degree of bookkeeping that is unlikely to be implemented reliably and scalably in a large system without adversely affecting peers' download times. Instead, the fairness we would like to provide can be described informally as follows: *gross violations will be detected and, more importantly, no rational peer will grossly deviate from our protocol*. This is a property of FOX; in fact, in an idealized setting, FOX peers have no incentive to deviate at all. We present a formal, rigorous set of fairness properties in Section 5.

We consider the case where all peers are interested in downloading the file, and do not consider maliciousness. In most situations, this is a strong assumption, but we remind the reader that file swarming inherently has a single point of failure; a malicious node may simply attack the server. We provide incentives to non-malicious peers to cooperate by employing game theoretic incentive mechanisms.

1.1 What Is Wrong with Tit-for-Tat?

Tit-for-tat (TFT) is a simple, intuitive incentive mechanism. In its simplest application to file swarming, peer p sends peer q a block but does not send more until q sends a block in return. If q wants more blocks from p, it has the incentive to act fairly and return p's favor. There are many obvious variations on this theme; p could simply reduce the quality of service it offers q until q sends blocks to p, or p could continue sending until q develops too large of a deficit. The former (reduced service) is similar to what BitTorrent[1] currently uses in its choking algorithm. TFT has found its way to such widely used systems, so why propose a change now?

Current file swarming systems do not provide the proper basis for applying TFT. For two-player repeated games, TFT can prove to be an effective mechanism. In particular, TFT forms a *subgame perfect equilibrium* (SPE) for many repeated games, the repeated prisoners' dilemma perhaps being the most well known.[2] However, current file swarming systems allow peers to interact with any other peer in the system; greedy users may not be required to repeatedly interact with any other peer and therefore not give anything back to the system. It is therefore not clear what, if any, global properties can arise from applying TFT to systems such as BitTorrent, Bullet$'$, and Slurpie. Even less clear is whether a proof of the fairness properties TFT provides to these systems can be derived. Further, with unrestricted interaction, the amount of state peers must keep to maintain the history of their interactions can grow linearly with the number of peers.

[1]http://www.bittorrent.com

[2]For a thorough treatment of these and other game theoretic concepts, we refer the reader to [11].

The goal of our work is to investigate how incentive mechanisms such as TFT can be effectively applied in a distributed manner. When successful, TFT can ensure fairness, but at what cost? In this preliminary work, we introduce our framework – the FOX overlay structure – in which a clearly defined structure is used to impose behavioral standards. We show that, in an idealized environment (Section 3), the FOX overlay structure (Section 4) is the first system of its kind to offer provable fairness (Section 5).

Unfortunately, the cost of this degree of fairness is high; in particular, when even one peer violates the protocol, all peers' downloads can be negatively affected. However, since it is not always possible to determine *who* is cheating, this is the only way to ensure that the defecting peer will be punished. As the need increases to provide incentives for cooperation, we believe the primitives made available in FOX will be broadly useful as a viable alternative to the many systems that require centralization or undue assumptions about the underlying platform.

2. PREVIOUS WORK

File swarming has received significant attention from researchers and users alike. Application-level multicast systems [2, 5, 6] can be considered some of the earliest such work, but they provide no incentives to their users to cooperate. Recent systems create a mesh in which, in the ideal case, all nodes send and receive portions of the file (henceforth *blocks*), thereby "filling the pipe" with data to provide much better download times [4, 7, 9, 14].

In mesh-based systems, since each node trades blocks for new blocks, designers have attempted to apply TFT as an incentive mechanism. This is most evident in BitTorrent [7], which uses a variant of TFT in which the number of blocks a peer p sends to peer q determines the quality of service q provides to p. As described above, however, TFT does not provide strong global properties when trivially applied to an unstructured system. Note also that, with TFT, blocks can be viewed as tradable currency. In general, without a means of verifying currency, forgery will occur; peers could simply forge garbage blocks and trade them for real ones. A means of verifying the currency in these systems is likely to involve additional work at the server (e.g., the server could sign each block). We show in Section 5 that FOX peers do not have incentive to forge garbage blocks.

Other mechanisms use currency that is more closely analogous to money [3, 8, 15]. In these systems, peers can obtain digital currency by paying for it with real money or by providing service to other peers. Currency exchange between peers is a difficult problem to implement; existing systems require tamper-proof hardware [3] or a trusted, centralized authority [15]. Designing a distributed, easily deployable, altruism-free system that does not require a central trusted node (such as the issuer of currency) is the focus of our work.

FOX provides incentives while only requiring nodes to monitor their immediate overlay neighbors. Scrivener [10] employs a similar approach, but assumes the different model of distributing potentially many different files. Conversely, with FOX, we are interested in optimizing download times for a given popular file in addition to maintaining strict fairness properties.

3. K-MELTDOWN MODEL

Consider many peers competing for throughput at a server. It is generally known that servers have some threshold for the amount of contention they can withstand, after which throughput tends toward zero for each connecting peer. We approximate this behavior in a simplistic way with the *k-meltdown model*, as follows:

> *As the number of connections to the server increases to some number, k, the throughput to each node stays constant. Once more than k clients connect, the server "melts down," that is, its total outgoing throughput drops immediately to zero.*[3]

We assume the clients also have a meltdown point and that it is the same k. This is the strongest assumption we make, as it implies homogeneous resources at the clients. However, we believe this assumption can be relaxed in practice.

Within the k-meltdown model, peers have the ability to halt the server and thus terminate every peer's outstanding download. The process of halting the system is an example of a *grim trigger strategy* [11]; once a player determines that it has effectively been denied sufficient service, it may invoke the trigger to cease all cooperation. This perpetual threat provides rational peers with the incentive to maintain a degree of system-wide fairness. Of course, it also provides a very easy means by which a malicious adversary can perform a denial of service (DoS) attack. However, as discussed above, a file swarming system is inherently "easy prey" for an attacker, as the server is a single point of failure.

With grim triggers, we are able to provide a greater degree of fairness than existing systems, but it is clear that it is not the ideal way of doing so. This can be seen from a game theoretic perspective; a protocol for a repeated game should ideally consist of subgame perfect equilibria, but this is not possible with grim triggers since they are non-credible threats. Extending FOX to a subgame perfect equilibrium is a focus of ongoing work. One potential approach to this is for peers to melt down for only some period of time, thereby punishing, not canceling, others' downloads.

4. THE FOX OVERLAY STRUCTURE

Here, we introduce the FOX overlay structure. This structure forms the basis of our protocol that provides near-optimal download times and, with localized threats, provides incentive to peers to cooperate.

4.1 Notation and Assumptions

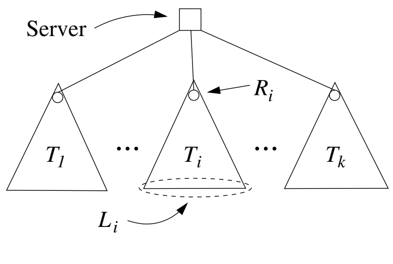

Figure 1: Notation pertaining to the tree structure.

[3]Even if a server had no such inherent meltdown point, it could willingly shut down upon the $(k+1)$st connection. Even with this seemingly self-destructive approach we ensure system-wide fairness.

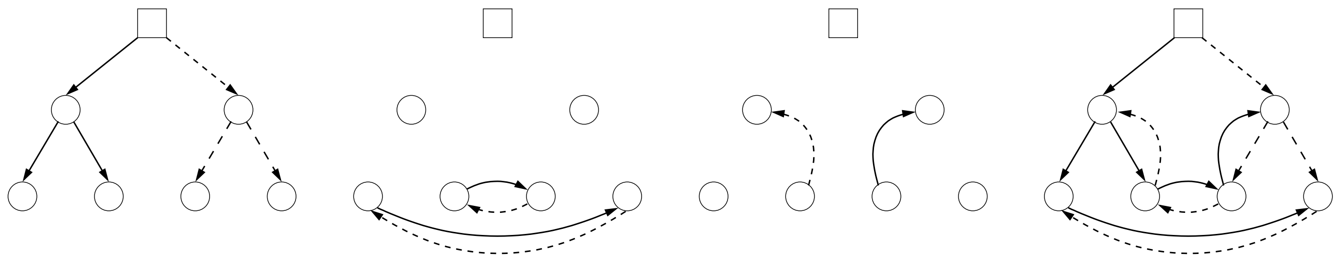

Figure 2: Example of the algorithm with $k = 2, N = 6$. Solid arrows denote the transfer of blocks b_{kt+1} and dashed arrows blocks b_{kt+2}, $0 \leq t \leq \left(\frac{B}{k} - k\right)$

There are N clients attempting to download a file of B blocks from one server. Per the k-meltdown model, each client is assumed to have a maximum outgoing bandwidth of k blocks per time step. For ease of exposition, assume that there are enough nodes to fill a k-ary tree of height H rooted at the server, that is

$$N \;=\; \frac{k^{H+1}-1}{k-1} - 1 \;=\; \left(k^H - 1\right)\frac{k}{k-1}.$$

Consider the tree in Figure 1. Let $R_i, 1 \leq i \leq k$, be the k peers connected to the server and let T_i be the subtree rooted at R_i. Lastly, let L_i denote the set of leaves of T_i. To arrive at our final structure, we will add loops to the tree in Figure 1, but will continue to refer to the members of the L_i's as leaves and $T_i \backslash L_i$ as internal nodes.

4.2 Filling the Pipe

To provide quick download times, our goal is to first fill the system's pipe, that is, to use each available outgoing link. We demonstrate how we do this in an iterative manner, beginning with the multicast tree in Figure 1. In a full, balanced, k-ary tree of height H, there are k^H leaves, thus with just the multicast tree, the fraction of nodes *not* sending data is

$$\frac{k^H}{N} \;=\; \frac{k^H}{k^H - 1} \cdot \frac{k-1}{k} \;>\; \frac{k-1}{k}.$$

Clearly, the multicast tree alone is not an adequate use of the available resources. We therefore make use of the leaf sets' links as follows.

Instead of obtaining the same block, each R_i requests a different block from the server. At time step $t \geq 0$, R_i retrieves the $(kt+i)$th block of the file, b_{kt+i}. Note that, unlike most other file swarming systems, this requires no server involvement; the R_i's can choose their respective i's amongst one another. Let $\mathcal{B}_t$ be $\{b_{kt+1}, \ldots, b_{kt+k}\}$, the set of blocks uploaded by the server at time t. Once it receives b_{kt+i}, R_i multicasts it throughout T_i, as in Figure 2(a). When $v \in L_i$ receives block b_{kt+i}, it trades with some set of $(k-1)$ leaves, each from a different T_j. There are enough leaves to create pairings in which each leaf, in a single round, gets the $(k-1)$ blocks it lacks from the set $\mathcal{B}_t$. As shown in Figure 2(b), each leaf p only sends blocks to leaves which send blocks to p, the reasons for which are made clear in Section 4.3.

Note that, at this point, each leaf has each block in $\mathcal{B}_t$ and each internal node $v_i \in T_i \backslash L_i$ has block b_{kt+i}. Further, observe that there are more than $(k-1)$ times as many leaves as internal nodes. We therefore have the resources to deliver to each such v_i the $(k-1)$ blocks it is missing, $\mathcal{B}_t \backslash \{b_{kt+i}\}$, in one additional step; the leaves simply connect directly to their appropriate ancestor. We show this in Figure 2(c) and the resulting structure in Figure 2(d). In a full k-ary tree, all but k leaves have a total of k peers to send to: the $(k-1)$ other leaf nodes and the 1 internal node. The k other leaves will have $(k-1)$ connections to other leaves but no ancestor pointer, as exemplified by the leftmost and rightmost leaves in Figure 2(d). This accounts for the fact that one node in the system does not require any in-degree — the server itself — while all other nodes have in-degree k.

The connections between members of L_i and the internal nodes of T_i follow the same paradigm as the inter-leaf connections: send data to those who send data to you. Precisely, a node $u_i \in L_i$ sends one of the $(k-1)$ blocks it received from leaves in the other T_j's to one of the nodes on that path from u_i to R_i. This methodology forms the basis of an incentive mechanism similar to tit-for-tat, which we show below.

4.3 Threats

In addition to filling the pipe, the connections between peers described above have two important properties. One, they exhibit a give-and-take symmetry; each of peer p's outgoing edges has a corresponding incoming edge which "returns the favor." Two, the connections are stable; as opposed to systems such as BitTorrent or Slurpie, a node maintains its set of neighbors for the entire download.

Due to these two properties, when a peer misbehaves, there are immediate repercussions. Unlike systems where peers arbitrarily match up, peers in FOX expect a constant level of service from each other. When this service is not detected (i.e., some peer is not holding up its end of the pipe), punishment is swift. Specifically, each peer understands that, when another peer p detects a defection (i.e., it does not receive a block it should have), p will stop sending and go to the server. Since there are already k connections at the server, this would cause a meltdown for the whole system.

Obviously, in a practical system, a grim trigger as sensitive as this is neither viable nor desirable; node failures or trembles[4] would unnecessarily take down the system. Though clearly not a subgame perfect equilibrium, this threat does provide proper incentive to not defect. Designing more robust punishment mechanisms is a focus of our ongoing work.

[4]A player is said to *tremble* if the action it takes differs from its chosen strategy. Trembles are designed to capture involuntary faults or mistakes; e.g., in our setting, delay caused by network congestion.

4.4 Avoiding Last-Block Collapse

If self-interested peers can complete downloads at different times, a file swarming system runs the risk of *last-block collapse*, in which nodes begin dropping out of the system when they finish, leaving the remaining nodes with no help to finish their downloads. As presented thus far, FOX is vulnerable to last-block collapse; when each R_i gets its final block, it can disregard its respective T_i and instead trade the final block with the other R_j's. In doing so, each R_i saves the time it would take to propagate their final blocks down the tree, resulting in a download that finishes $O(\log_k N)$ steps faster. Note that they could not do this sooner than on the last block, else their successors could detect their defection in one time step and invoke the threat mechanism.

Last-block collapse arises in FOX because internal nodes do not have vested interest in their descendants near the end of the download. We enforce participation by augmenting our block exchange with a novel encryption algorithm. At the end of the algorithm, both internal nodes and leaves have information that the other needs to complete its download. The problem of vested interest is therefore reduced to the exchange of these final pieces of information.

4.4.1 Block Encryption Algorithm

The specific algorithm of encryption and dissemination is presented in Algorithm 1, where $[b]_K$ denotes block b encrypted with key K and bounds on t_0 are given in Section 4.4.2.

Algorithm 1 Block encryption and dissemination.

```
 1: The $R_i$'s agree on a public-private key pair, $(K_{pub}, K_{priv})$.
 2: Simultaneously, $\forall i, \forall v \in L_i$, $v$ chooses its own (i.e., not
    shared with any other nodes) public-private key pair, $(K'_{pub}, K'_{priv})$.
 3: The $R_i$'s multicast $K_{priv}$ to their respective $T_i$'s, but
    only to the internal nodes and leaves with no assigned ancestor.
 4: for steps $t = t_0$ to $(\frac{B}{k} - k)$, in pipeline, do
 5:     $R_i$ computes $[b_{kt+i}]_{K_{pub}}$.
 6:     The internal nodes of $T_i$ multicast $[b_{kt+i}]_{K_{pub}}$ to all
        nodes in their tree.
 7:     The leaves perform their exchange with the encrypted
        blocks, that is, they trade their $[b_{kt+i}]_{K_{pub}}$ for other
        leaves' $[b_{kt+j}]_{K_{pub}}$.
 8:     In parallel, $\forall v \in \cup_i L_i$, $v$ uses its $K'_{pub}$ to calculate
        $[[b_{kt+j}]_{K_{pub}}]_{K'_{pub}}$ where $b_{kt+j}$ is the block $v$ is assigned
        to send to its ancestor.
 9:     The leaf $v$ sends this doubly-encrypted block up to its
        assigned ancestor.
10: end for
```

To summarize, the internal nodes (i.e., $\cup_i(T_i \backslash L_i)$) use a shared key to encrypt the final blocks of the file. The leaves, unable to decrypt them, trade these encrypted blocks during their leaf exchanges. When sending blocks to their ancestors, each leaf uses its own key to encrypt the already-encrypted blocks. Thus, upon completion, internal nodes need some leaves' K'_{priv} and each leaf needs K_{priv}. These are the final pieces of information mentioned above. Observe that the algorithm does not require a PKI; the public-private key pairs can be generated on a per-file basis by the peers themselves.

4.4.2 Key Exchange

After running the algorithm in Algorithm 1, each leaf has its own K'_{priv} but needs K_{priv} and that each internal node has K_{priv} but needs its appropriate set of $(k-1)$ different K'_{priv}'s. All that remains is the key exchanges. Each leaf is involved in one key exchange with its assigned ancestor, and each internal node is involved in $(k-1)$ exchanges. The simplest method for two peers to exchange keys is to alternate sending one bit of the key without letting the other get ahead by more than one bit. To ensure that no leaves get ahead of others, the internal nodes do not reveal a new bit until all $(k-1)$ of the leaves reveal their corresponding bit, resulting in $O(kM)$ total steps where M is the size of the keys in bits. In the worst case, one of the nodes in the exchange could get the final bit and leave the system (as they are, by that time, finished downloading). Thus, some nodes may finish their download at most one *bit* ahead of others, but the remaining node need only try both values. In essence, all participants finish their downloads simultaneously. A more advanced simultaneous key exchange protocol is presented in [13] which requires more communication than just the M bits of the keys but, during the exchange, verifies that the keys are correct.

The question remains: for how many time steps should the encryption process take place, i.e., what is the desired value of t_0 in Algorithm 1? If it were just for the final block ($t_0 = B/k$), for example, then the R_i's could simply exchange their final blocks instead of sending down the tree. If it were for less than $\log_k N$ steps ($t_0 > B/k - \log_k N$), the root nodes could exchange their final set of blocks with one another in less time than it would take for their blocks to reach the leaves. More generally, t_0 must be large enough so that, after receiving the last block from the server, the root nodes cannot trade with one another faster than it would take to complete the protocol. Thus, we require that blocks be encrypted for at least $\log_k N + kM$ steps ($t_0 \leq B/k - \log_k N - kM$), the number of steps from the time the R_i's get the block to the time the download is complete. Observe that, regardless of the value of t_0, only a single agreement at the R_i's and a single round of key exchanges are necessary. In the event that a node could fail, it may be helpful to use a weak encryption method, in which the time to crack the private key is less than the time to re-download the blocks but greater than the time to perform the key exchange itself.

5. ANALYSIS

In this section, we prove that FOX requires little state, provides near-optimal performance, and gives incentive for participants to behave in accordance with the protocol.

5.1 Per-Peer State

Recall that each node stores k incoming connections, at most k outgoing connections, and one public-private key pair: (K_{pub}, K_{priv}) for all $v \in T_i \backslash L_i$ and (K'_{pub}, K'_{priv}) for all $v \in L_i$. Thus, the amount of state per peer is $O(k)$.

5.2 Download Time

There are two main properties of FOX to note: that each node receives the file in the same amount of time and that this time is nearly optimal. In our analysis, we will assume that each node has the same incoming and outgoing bandwidth resources. Without further loss of generality, we assume that each node can send out 1 block to each of its

k links per time step. Thus, the server can send out only k different blocks each round, giving us the trivial lower bound of $\Omega(B/k)$ rounds until all nodes finish downloading. This lower bound does not take into account the time to propagate data throughout the system.

THEOREM 1. *The number of steps until all FOX peers have completed the download is* $O(\log_k N + \frac{B}{k})$.

PROOF. Initially filling the pipe takes $O(\log_k N)$ steps: $\log_k N$ to get down the multicast tree, 1 to swap blocks amongst leaves, and 1 to send back up to ancestors. Since no links are used more than once in these, we can pipeline the operation, requiring precisely $\left(\frac{B}{k} - 1\right)$ additional rounds. The key exchange at the end of the download consists of $O(kM)$ steps, each requiring 1 bit to be sent. If β is the block size sent each round during the file transfer then key exchange takes $O(\frac{kM}{k\beta}) = O(\frac{M}{\beta})$ time. It is reasonable to assume that the key size, M, is significantly smaller than the file's block size, β, hence $O(\frac{M}{\beta}) = O(1)$. Thus, the time for all FOX peers to complete downloading is $O(\log_k N + \frac{B}{k})$. $\square$

5.3 Incentives

Here, we discuss some of the incentives that FOX provides to its participants. Combined, these incentives indicate that the protocol in Section 4 is a stable equilibrium.

5.3.1 Sending Correct Data

As discussed in Section 2, if there is no means of verifying an individual block's integrity, then using tit-for-tat in a file swarming system may give peers the incentive to generate garbage blocks, effectively forging the system's currency. In short, peers have the incentive to send data as soon as possible, garbage or not. For the remainder of this discussion, we assume that each peer perceives the task of sending data independent of the data itself; if it has a correct block to send, it will not go through the trouble of generating a garbage block to send in its stead.

Contrary to other tit-for-tat file swarming systems, we structure the flow of data in such a way that peers do not benefit from sending blocks before they have the correct data to send.

CLAIM 1. *Even without a means of verifying a block's integrity, no peer has the incentive to send forged blocks.*

PROOF. Once peer p has begun receiving blocks, it has no shortage of correct data to send, so consider the case when p has yet to receive data. Suppose $p \in L_i$ for some i. Sending a garbage block to some $q \in L_j, j \neq i$, yields no reward, as q will not have a real block to trade until the same time p does. Similarly, sending forged data to an ancestor has no positive effect; receiving blocks from leaves does not make internal nodes change their behavior; it simply keeps them from invoking the grim trigger. Now consider $p \in T_i \backslash L_i$ for some i. Sending a garbage block to its children will give the leaves of the subtree rooted at p a block they can trade with leaves from other T_j's. However, the leaves in L_j will not have data to send in return until the first round of multicast is complete. Thus, peer p will not receive data in response to its garbage block any sooner than if it had followed the protocol exactly, and hence yields no benefit from such a deviation. $\square$

5.3.2 Maintaining the Structure

Unlike current file swarming systems, we provide a specific topology instead of allowing peers to arbitrarily match up. Here, we consider the desirability of our structure, in particular the underlying multicast tree. One could envision that a peer, attempting to minimize the amount of data it has to send, would maintain a single child. With the following, we show that internal nodes have the incentive to give as much to the system as possible.

CLAIM 2. *Every peer has the incentive to ensure that the underlying multicast tree is a balanced, full, k-ary tree.*

PROOF. Observe that, for a tree of height H, the runtime in Theorem 1 is $O(H + \frac{B}{k})$, thus it is within every peer's best interest to minimize H. H is minimized when the tree is balanced and, in the k-meltdown model, when each internal node maintains as close to k children as possible.

Next, consider the leaves' outgoing edges. If not maintained precisely as described in Section 4, then there will be a peer that does not have k incoming edges and will therefore invoke the grim trigger. Thus, every peer has the incentive to maintain the structure. $\square$

5.3.3 Maintaining Position

The structure itself is desired by the participants, but that does not immediately imply that they will be willing to do their part. Instead, we must ask: are there certain positions within the structure that are preferable? If this were the case, the stability of the system would be in question as nodes would be vying for these better positions.

Observe, however, that since all nodes finish downloading the file at the same time, they have no incentive to change their position. This follows from our assumption that utility equals the time to download the entire file. If, however, a node perceived benefit in minimizing the time to download intermediate blocks, it may have the incentive to become a leaf (since they receive $(k-1)$ of the k blocks per time step earlier than the internal nodes). In this case, we could extend the encryption procedure to cover every step of the download, removing whatever use the nodes could have had for receiving the intermediate blocks sooner. This would not increase the number of messages; in Algorithm 1, only t_0 would change. It would, however, increase the amount of work performed at each R_i and each leaf, which is why we minimize the number of encryptions to $(\log_k N + kM)$ in Section 4.

5.3.4 Additional Threats

A final concern is whether participants in FOX can use the k-meltdown model as a means of threatening other nodes into providing more resources than is strictly required by FOX. Note that, in any such threat, the threatening node must offer the nodes a means of completing the download. If instead they were offered nothing, then the meltdown would be an equivalent outcome and the threat would be baseless. A simple solution to this is for the threatened nodes to counter with another threat; any selfish activity will result in a meltdown.

6. DISCUSSION AND OPEN QUESTIONS

In this paper, we have addressed the problem of file swarming with greedy participants in a way that is distributed and does not require explicit server involvement. We have presented the k-meltdown model of node throughput capabilities. Working within this model, we developed a novel data transfer topology that, with threats of grim trigger, provides near-optimal file download times with incentives for the peers to cooperate. These incentives indicate that our system is a stable equilibrium. Each participant in our system requires only $O(k)$ state where k is the peer's out-degree. Also, by using a novel cryptographic method, we ensure that all nodes finish their download at the same time, thereby avoiding system collapse on the final block.

FOX answers the question that provable fairness can be obtained, but introduces a large cost: the potential for system collapse. This is a byproduct of our k-meltdown model and of our assumption that nodes are purely self-interested. We raise the question: can one obtain provable fairness in a completely decentralized system without having such undesirable, potential outcomes? Put another way, what new operating points are possible with provable fairness? For example, can one address heterogeneity and exploit network locality in a file swarming system consisting of purely greedy nodes?

We are currently addressing such questions and we intend FOX to be a complete, practical protocol. Though these initial results appear promising, it remains to be seen whether such nice properties can be maintained in a real environment. Given the considerations of node failure, heterogeneous network conditions, and the difficulty of node synchronization, it is a seemingly difficult problem to solve in a fully decentralized manner. Yet, we believe that, since the FOX structure mainly specifies a data path, it can used as a component of a more robust system.

REFERENCES

[1] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on Cooperation in BitTorrent Communities. In *P2PECON '05*, 2005.

[2] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient Multicast Using Overlays. In *ACM SIGMETRICS*, 2003.

[3] L. Buttyan and J.-P. Hubaux. Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks. *ACM/Kluwer Mobile Networks and Applications (MONET)*, 8(5), 2003.

[4] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. In *ACM SIGCOMM*, 2002.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Content Distribution in a Cooperative Environment. In *ACM SOSP*, 2003.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), 2002.

[7] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[8] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives for Sharing in Peer-to-Peer Networks. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, 2001.

[9] D. Kostić, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat. Maintaining High Bandwidth Under Dynamic Network Conditions. In *Proceedings of USENIX*, 2005.

[10] A. Nandi, T.-W. Ngan, A. Singh, P. Druschel, and D. S. Wallach. Scrivener: Providing Incentives in Cooperative Content Distribution Systems. In *Middleware*, 2005.

[11] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[12] C. Papadimitriou. Algorithms, Games, and the Internet. In *STOC*, 2001.

[13] B. Schneier. *Applied Cryptography*. John Wiley & Sons, 2nd edition, 1996.

[14] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie: A Cooperative Bulk Data Transfer Protocol. In *IEEE INFOCOM*, 2004.

[15] S. Zhong, J. Chen, and Y. R. Yang. Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks. In *IEEE INFOCOM*, 2003.