T-Chain: A General Incentive Scheme for Cooperative Computing

Kyuyong Shin*, Carlee Joe-Wong†, Sangtae Ha‡, Yung Yi§, Injong Rhee¶, Douglas Reeves¶
*Korea Military Academy, †Princeton University, ‡University of Colorado–Boulder, §KAIST, ¶NCSU
Emails: kyshin@kma.ac.kr, cjoe@princeton.edu, sangtae.ha@colorado.edu, yiyung@kaist.edu, {rhee,reeves}@ncsu.edu

Abstract—**In this paper, we propose a simple, distributed, but highly efficient fairness-enforcing incentive mechanism for cooperative computing. The proposed incentive scheme, called Triangle Chaining (T-Chain), enforces reciprocity to minimize the exploitable aspects of other schemes that allow free-riding. In T-Chain, symmetric key cryptography provides the basis for a lightweight, almost-fair exchange protocol, which is coupled with a pay-it-forward mechanism. This combination increases the opportunity for multi-lateral exchanges and further maximizes the resource utilization of participants, each of whom is assumed to operate solely for his or her own benefit. T-Chain also provides barrier-free entry to newcomers with flexible resource allocation, providing them with immediate benefits, and therefore is suitable for dynamic environments with high churn (i.e., turnover). T-Chain is distributed and simple to implement, as no trusted third party is required to monitor or enforce the scheme, nor is there any reliance on reputation information or tokens.**

I. Introduction

In many distributed systems, participants voluntarily pool or share their resources (e.g., computing power, storage space, and network bandwidth) in order to obtain mutual benefits. This general notion may be termed *cooperative computing*. Examples of cooperative computing include the Domain Name System, BGP-4 routing, grid and cloud computing, application-layer multi-cast, file sharing, streaming, etc.

The information and services provided by cooperative computing to its participants can be thought of as examples of a *public good*. In sharing the public good, cooperation among participants is important to ensure a satisfactory experience for all, but there may exist some *free-riders*, i.e., free-riding participants, who try to enjoy the benefits of the good without contributing to it. If free-riders are present, contributing participants can experience under-provisioning of the good, leading to inefficiency, unfairness, and even system collapse in some cases, i.e., the "tragedy of the commons" [1]. *Reciprocity* prevents such under-provisioning through the expectation that a participant who consumes resources will contribute equivalent resources for the benefit of others, while those unwilling to reciprocate will be excluded from the public good's benefits. Enforcing the principle of reciprocity, however, is surprisingly difficult in fully distributed systems [2]–[4].

A large range of incentive mechanisms to enforce reciprocity in cooperative computing have been proposed, which can be categorized into three groups: direct reciprocity [5]–[8], indirect reciprocity [9]–[13], and the use of coding or encryption [2], [7], [10], [14]. Currently, however, none of these techniques have been notably successful in preventing free-riding. Reasons may include their complexity and/or overhead, slow convergence times, the absence of trust among participants, and the ease of bypassing the proposed mechanisms [2].

Preventing free-riding may conflict with other important performance metrics in cooperative computing, such as efficiency (e.g., file downloading time), introducing a trade-off between the desired objectives (see [15] for details). This trade-off occurs because the goals of ensuring fairness and getting better efficiency (through donating bandwidth for fast newcomer bootstrapping) may conflict with each other. Bandwidth donated to newcomers, for example, is often exploited for the purposes of free-riding. Another challenge in designing incentive mechanisms comes from the diversity of cooperative computing applications. As a result, many existing incentive mechanisms emphasize one goal (either fairness or efficiency/bootstrapping speed) at the expense of the other. Thus, a good incentive mechanism in cooperative computing should satisfy two requirements simultaneously: (i) strict *fairness* to overcome free-riding; and (ii) *adaptive* (but not exploitable) newcomer bootstrapping for efficiency.

This paper proposes a general incentive mechanism for cooperative computing, enforcing direct and/or indirect reciprocity among participants, that is designed to meet the two afore-mentioned requirements. Under the proposed scheme, a participant $\mathbb{A}$ contributes an *almost complete* resource requested by another participant $\mathbb{B}$. $\mathbb{A}$ also informs $\mathbb{B}$ of the party to whom $\mathbb{B}$ must reciprocate. If $\mathbb{A}$ and $\mathbb{B}$ have symmetric interests, $\mathbb{A}$ can designate itself as the party to whom $\mathbb{B}$ must reciprocate. Otherwise, $\mathbb{A}$ designates another participant $\mathbb{C}$ as the participant to whom $\mathbb{B}$ must reciprocate. This represents a *pay-it-forward* policy [16]. It reduces the difficulty of finding a compatible participant with mutual interests by expanding the definition of reciprocation to include (almost) any participant. The (almost complete) resource contributed by $\mathbb{A}$ to $\mathbb{B}$ is completed if, and only if, the request for reciprocation is fulfilled. The steps just outlined constitute an almost-fair exchange protocol, in which neither party can gain advantage by terminating the protocol early.

In fulfilling its obligation to reciprocate, $\mathbb{B}$ likewise contributes an almost complete resource to the designated recipient, and requires that recipient to reciprocate exactly as $\mathbb{B}$ was required to do. The completion of one almost-fair exchange thus begins another almost-fair exchange, leading to a chain of reciprocal exchanges. In this method, resources for bootstrapping are dynamically adjusted, based on the arrival rate and demands of newcomers. The proposed solution is called Triangle Chaining, or T-Chain for short. T-Chain has several favorable points:

- *Incentive compatibility*: T-Chain triggers strong incentives for all participants to follow the given protocol in order to maximize their own benefits. This property is discussed in detail in Section II-B.
- *Fast but non-exploitable bootstrapping*: Newcomers can immediately participate in and fully contribute to the cooperative system. The way in which this is achieved, however, cannot be exploited for free-riding purposes, except under rare circumstances. In contrast to other incentive schemes, no fixed amount of system resources needs to be pre-allocated for newcomer bootstrapping. Rather, these resources are adjusted according to the system needs.
- *Fairness*: In order to complete the received resource, the recipient must reciprocate with virtually the same amount of work or resource contribution. This ensures excellent fairness among participants.
- *Robustness*: The creation of multiple identities, changing identities, or frequent changing of the neighbor set will not be helpful to a potential free-rider, since reputation is neither computed nor used in T-Chain. Opportunities for collusion by free-riders are extremely limited, because the party to whom reciprocation must occur is selected by the donor of the resource, not the recipient.

We illustrate the benefits and practicality of T-Chain by applying it to BitTorrent [5], currently the most popular file sharing application. BitTorrent suffers substantial loss of performance due to free-riding, despite repeated attempts to address the issue. Compared with current approaches (BitTorrent [5], PropShare [7], and FairTorrent [8]) under realistic conditions, T-Chain prevents, instead of merely penalizing, free-riding, and protects compliant peers from free-riding's detrimental effects. T-Chain also fully utilizes the upload capacity of compliant participants despite free-riding, in contrast to these other approaches. Finally, the additional overhead of T-Chain is about 1% of the normal bandwidth and storage requirements of BitTorrent.

The remainder of this paper is organized as follows. First, Section II details our method, T-Chain. Security considerations, newcomer bootstrapping performance, and an overhead analysis are discussed in Section III. Section IV presents our evaluation of T-Chain, comparing it with BitTorrent, PropShare, and FairTorrent. Section V briefly surveys related works and, finally, Section VI concludes the paper.

II. Design of T-Chain

In this section, we present a new method, T-Chain, for minimizing or preventing free-riding. We apply T-Chain to BitTorrent and assume that the resource being shared (and the limiting factor on the system performance) is upload bandwidth. We use some BitTorrent terminology and concepts (refer to Cohen's original paper [5] for details) for better understanding. Table I introduces our notation.

TABLE I: Summary of Notations

$\mathbb{A}$, $\mathbb{B}$, ...	Participants (leechers or seeders) in a swarm
F	The file being shared by a swarm
p_{ik}	The i^{th} piece of F in k^{th} transaction in a chain
$K[p_{ik}]$	Encryption of p_{ik} with a symmetric key K
$F_{\mathbb{A}}$	The set of pieces completed (downloaded and decrypted) by $\mathbb{A}$ ($F_{\mathbb{A}} \subseteq \{p_1, p_2, \ldots, p_m\}$)
$K^{ik}_{\mathbb{A},\mathbb{B}}$	The symmetric encryption key used by $\mathbb{A}$ to encrypt p_{ik} when sent to $\mathbb{B}$
t_j	The j^{th} transaction of a chain
$\mathbb{D}_j$	Donor (i.e., uploader) in the j^{th} transaction
$\mathbb{R}_j$	Requestor (i.e., downloader) in the j^{th} transaction
$\mathbb{P}_j$	Payee in the j^{th} transaction

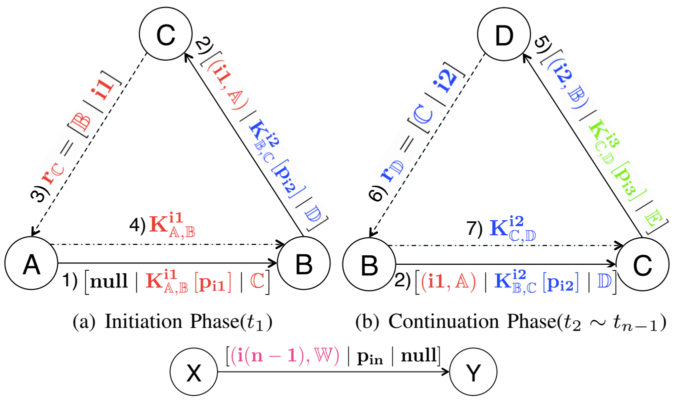
In BitTorrent, participants form a *swarm* sharing a single file divided into many fixed size pieces, each of which is further subdivided into blocks. Participants logically share or exchange file pieces, with blocks as the actual unit of transfer. It is assumed that participants are rational and selfish; they wish to maximize their benefits (i.e., minimize the time to download a file), while minimizing their contributions (i.e., file piece uploads to others).

A *seeder* creates and posts a *.torrent* file describing the file it wants to share. *Leechers* wishing to download the file retrieve the .torrent and contact a tracker identified there. The tracker forwards a list of up to 50 randomly selected members (i.e., seeders and other leechers) of the swarm. A newcomer attempts to establish a TCP connection to each member in that list; if the connection is accepted, they become neighbors. Leechers with fewer than 30 neighbors can ask the tracker for another list. A leecher can exchange file pieces with its neighbors and download pieces from the seeder(s); each leecher periodically sends its neighbors a list of its file pieces. Leechers download the file piece for which the fewest copies exist among their neighbors first, following the Local Rarest First (LRF) policy. They wish to download the file as quickly as possible, while seeders altruistically upload to others without expecting anything in return.

BitTorrent discourages free-riding using rate-based *tit-for-tat* (TFT) [5], in which a leecher initially *chokes* all connections to its neighbors and uploads no data. Roughly every 10 seconds, the leecher *unchokes* the k neighbors who have uploaded the most to it over the past 10 second interval, where k usually equals 4. Every 30 seconds, a leecher randomly selects and unchokes one additional neighbor, regardless of its past upload history. This *optimistic unchoking* allows newcomers to be easily bootstrapped into the system (i.e., some pieces are altruistically uploaded to newcomers) and helps leechers partner with new, potentially better, neighbors.

A. Basic T-Chain Protocol

T-Chain primarily changes BitTorrent's seeding and unchoking procedures. We modify what seeders upload to leechers, which neighbors are selected for unchoking, and how much is uploaded to each neighbor and when.


C
A
B
3) $r_{\mathbb{C}} = [\mathbb{B} \mid i1]$
2) $[(i1, \mathbb{A}) \mid K^{i2}_{\mathbb{B},\mathbb{C}} [p_{i2}] \mid \mathbb{D}]$
4) $K^{i1}_{\mathbb{A},\mathbb{B}}$
1) $[null \mid K^{i1}_{\mathbb{A},\mathbb{B}} [p_{i1}] \mid \mathbb{C}]$
D
B
C
6) $r_{\mathbb{D}} = [\mathbb{C} \mid i2]$
5) $[(i2, \mathbb{B}) \mid K^{i3}_{\mathbb{C},\mathbb{D}} [p_{i3}] \mid \mathbb{E}]$
7) $K^{i2}_{\mathbb{C},\mathbb{D}}$
2) $[(i1, \mathbb{A}) \mid K^{i2}_{\mathbb{B},\mathbb{C}} [p_{i2}] \mid \mathbb{D}]$
X
Y
$[(i(n-1), \mathbb{W}) \mid p_{in} \mid null]$

(a) Initiation Phase(t_1)

(b) Continuation Phase($t_2 \sim t_{n-1}$)

(c) Termination Phase(t_n)

Fig. 1: The initial, intermediate, and terminal transactions in each chain of T-Chain.

File piece exchanges in T-Chain are termed *transactions*. The j^{th} transaction t_j usually involves three parties: a *Requestor* ($\mathbb{R}_j$), a *Donor* ($\mathbb{D}_j$), and a *Payee* ($\mathbb{P}_j$). In transaction t_j, $\mathbb{D}_j$ will upload a file piece p_i to $\mathbb{R}_j$, who is one of its requesting neighbors. This file piece is first encrypted by $\mathbb{D}_j$ with key $K^i_{\mathbb{D}_j,\mathbb{R}_j}$ to ensure payment (i.e., reciprocation). To receive the decryption key $K^i_{\mathbb{D}_j,\mathbb{R}_j}$ from $\mathbb{D}_j$, $\mathbb{R}_j$ must reciprocate $\mathbb{D}_j$'s upload by uploading another encrypted file piece to payee $\mathbb{P}_j$ designated by $\mathbb{D}_j$. If $\mathbb{R}_j$ does not reciprocate, the encrypted file piece provided by $\mathbb{D}_j$ cannot be decrypted, preventing $\mathbb{R}_j$ from getting something (useful) for nothing.[1] A transaction completes as soon as $\mathbb{R}_j$ reciprocates the download and receives the decryption key.

In the next transaction, $\mathbb{R}_j$ and $\mathbb{P}_j$ of transaction t_j change roles to become the donor $\mathbb{D}_{j+1}$ and requestor $\mathbb{R}_{j+1}$ of transaction t_{j+1}, with a new payee $\mathbb{P}_{j+1}$; note that t_{j+1} is thus initiated by the requestor's reciprocation in transaction t_j. A sequence of transactions (t_1, t_2, $\cdots$), each completing the one before, can continue indefinitely, constituting a *chain*. As seen in Figure 1, each chain has three phases: (a) initiation, (b) continuation, and (c) termination.

1) Initiation Phase: Figure 1(a): As in BitTorrent, initially the seeder has all the pieces of a file F and will begin the process of distributing file pieces without expecting anything in return. The seeder ($\mathbb{A}$ in Figure 1(a)) begins a chain of transactions by uploading a file piece p_{i1}, selected (by $\mathbb{B}$) with the LRF policy, to a *randomly* selected requestor leecher $\mathbb{B}$. $\mathbb{A}$ is thus the donor of the first transaction.

Before uploading, $\mathbb{A}$ encrypts the file piece p_{i1} with a symmetric key $K^{i1}_{\mathbb{A},\mathbb{B}}$. The donor $\mathbb{A}$ also informs requestor $\mathbb{B}$ that it must reciprocate by uploading a file piece to payee (leecher) $\mathbb{C}$. The payee $\mathbb{C}$ may be *randomly* selected by $\mathbb{A}$ from among $\mathbb{A}$'s neighbors that desire at least one of $\mathbb{B}$'s file pieces.[2] If the selected payee $\mathbb{C}$ is not a neighbor of $\mathbb{B}$, then $\mathbb{B}$ should send a neighboring request before reciprocation; alternatively, $\mathbb{A}$ can provide a list of candidate payees to $\mathbb{B}$ for reciprocation.

[1]We assume that each key is used to encrypt only one file piece and never used thereafter, which need not be the case in practice).

[2]A different method is used when $\mathbb{B}$ is a newcomer (Section II-C1).

The requestor $\mathbb{B}$ satisfies the reciprocity requirement by uploading another file piece p_{i2} to $\mathbb{C}$, encrypted with its own key $K^{i2}_{\mathbb{B},\mathbb{C}}$; the piece is chosen by $\mathbb{C}$ using the LRF policy. While uploading, $\mathbb{B}$ also informs $\mathbb{C}$ that this upload is reciprocation for $\mathbb{A}$'s upload of p_{i1} to $\mathbb{B}$. If (and only if) $\mathbb{C}$ notifies $\mathbb{A}$ that $\mathbb{B}$ has reciprocated,[3] $\mathbb{A}$ will release the key $K^{i1}_{\mathbb{A},\mathbb{B}}$ to $\mathbb{B}$. This key allows $\mathbb{B}$ to decrypt the file piece p_{i1}, completing the first transaction of the chain. Note that $\mathbb{B}$'s upload reciprocation to the payee $\mathbb{C}$ starts a second transaction in the chain; this is indicated with blue in Figure 1(a).

2) Continuation Phase: Figure 1(b): As described above, the fulfillment of one transaction requires another to be started. In the second transaction, $\mathbb{B}$ acts as $\mathbb{A}$ did in the first transaction. Along with uploading an encrypted file piece p_{i2} to $\mathbb{C}$, $\mathbb{B}$ selects a qualified participant $\mathbb{D}$ (among its neighbors) and designates it to $\mathbb{C}$ as the payee to whom $\mathbb{C}$ must reciprocate. The transaction then completes as in the initiation phase.

The donor $\mathbb{B}$ cannot choose the requestor $\mathbb{C}$ in this phase of the chain, since $\mathbb{C}$ is selected by $\mathbb{A}$ in the previous transaction. However, $\mathbb{B}$ can freely choose the payee $\mathbb{D}$ to whom $\mathbb{C}$ must reciprocate. This choice follows one of two strategies:

- *Direct reciprocity.* If the requestor $\mathbb{C}$ possesses at least one file piece that $\mathbb{B}$ needs, $\mathbb{B}$ designates itself as the payee $\mathbb{D}$ to whom $\mathbb{C}$ must reciprocate. This designation represents simple bi-lateral reciprocation between $\mathbb{B}$ and $\mathbb{C}$, as in BitTorrent's TFT policy; note that $\mathbb{C}$ is still free to choose the payee for the next transaction in the chain, which can continue as usual. Similarly, there is no harm or confusion if one leecher takes part in multiple chains simultaneously. In general, this is desirable to fully utilize the available upload bandwidth of all the leechers (see Section II-C3 for details).
- *Indirect reciprocity.* If direct reciprocity is not possible, the donor $\mathbb{B}$ randomly chooses a payee among its neighbors who need at least one of $\mathbb{C}$'s file pieces (including the file piece p_{i2} about to be uploaded). If no such neighbor exists in the donor's (not requestor's) neighbor set, then the chain terminates, as discussed in the next section.

Note that as the fraction of newcomers increases, the probability that indirect reciprocity will select a newcomer as the payee will also increase. T-Chain thus automatically adjusts the resources allocated to newcomers depending on their prevalence in the system.

3) Termination Phase: Figure 1(c): A leecher $\mathbb{X}$ who is required (by $\mathbb{W}$) to upload a file piece to $\mathbb{Y}$ will normally designate another leecher, $\mathbb{Z}$, to whom $\mathbb{Y}$ must upload. However, if $\mathbb{X}$ has no neighbor (including itself) who needs to download at least one piece from $\mathbb{Y}$, $\mathbb{X}$ will upload an *un-encrypted* file piece to $\mathbb{Y}$, releasing $\mathbb{Y}$ from the responsibility to reciprocate and terminating the chain.

Termination of a chain should occur only when newcomers stop joining a swarm. Under these conditions, as all leechers

[3]Receiver reports or notifications are assumed to be communicated directly by the payee (e.g., $\mathbb{C}$) to the donor (e.g., $\mathbb{A}$). If IP address spoofing is considered to be a threat, there are standard ways to authenticate communication between these two parties, and to prevent replay attacks (refer to RFC4953).

finish downloading the file and leave the swarm, all chains must eventually terminate. In the most extreme case of a swarm consisting of a single seeder and a single leecher, the seeder will simply upload the complete, unencrypted file directly to the leecher: the leecher cannot reciprocate to any other peer. Although this is a form of free-riding, it cannot be exploited by selfish participants unless they are willing to wait until all leechers depart and hope that no one else will join the swarm.

B. Incentives in T-Chain

In this section, we show that each party in Figure 1(a) (i.e., $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$) has an incentive to follow the T-Chain protocol.

Proposition 2.1 (Incentive compatibility): Assume that $\mathbb{A}$ receives a positive utility from distributing each file piece, $\mathbb{B}$ receives a positive utility from receiving piece $i1$ from $\mathbb{A}$, and $\mathbb{C}$ receives a positive utility from receiving piece $i2$ from $\mathbb{B}$. If $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ do not collude and the cost of transmitting encryption keys is negligible, then all three players increase their utility by participating in T-Chain.

Proof: $\mathbb{A}$ clearly benefits from uploading a file piece and corresponding encryption key to $\mathbb{B}$; in fact, $\mathbb{A}$ will initiate as many chains with as many participants as possible, so as to distribute more file pieces. $\mathbb{B}$ is incentivized to reciprocate to $\mathbb{C}$, as otherwise $\mathbb{C}$ will not send a notification to $\mathbb{A}$ and $\mathbb{A}$ will refuse to send $\mathbb{B}$ the key $K^{i1}_{\mathbb{A},\mathbb{B}}$. The payee $\mathbb{C}$ has an incentive to report its reception of an encrypted piece from $\mathbb{B}$ to $\mathbb{A}$. Otherwise, $\mathbb{B}$ will not release its decryption key $K^{i2}_{\mathbb{B},\mathbb{C}}$ to $\mathbb{C}$ in the next transaction. ■

Upon uploading to $\mathbb{C}$, $\mathbb{B}$ can also benefit from direct reciprocity, i.e., designating itself as the next payee in the chain, if it needs a file piece that $\mathbb{C}$ possesses. If direct reciprocity is not possible, $\mathbb{B}$ will designate some other payee $\mathbb{D}$ at no cost to itself. Designating $\mathbb{D}$ as the payee reduces $\mathbb{B}$'s potential competition with $\mathbb{D}$ in other chains and in future transactions in the same chain, benefitting $\mathbb{B}$ through an *offloading effect* [17].

The above mentioned incentives for the donor $\mathbb{A}$, requestor $\mathbb{B}$, and payee $\mathbb{C}$ apply to all participants in each transaction of a chain. Since there are strong incentives for all participants to follow the given protocol so as to maximize their benefits, we claim that T-Chain is incentive compatible.

C. Additional Features of T-Chain Protocol

T-Chain's basic protocol can be improved through newcomer bootstrapping, flow control (adaptive receiver selection), and opportunistic seeding. Each of these is motivated and described below.

1) Newcomer Bootstrapping: In order to reciprocate, requestors must have at least one completed (i.e., decrypted) file piece needed by the payee. This may not be the case for newcomers, however. For instance, suppose $\mathbb{B}$ in Figure 1(a) is a newcomer. $\mathbb{B}$ is required by $\mathbb{A}$ to reciprocate for p_{i1} by uploading another encrypted file piece p_{i2} to $\mathbb{C}$. Since $\mathbb{B}$ has no completed file pieces yet, it has difficulty in complying.

In this case $\mathbb{A}$ must select a piece p_{i1} that both $\mathbb{B}$ and $\mathbb{C}$ need, which is the only case in which the LRF policy is not used in T-Chain. Now $\mathbb{A}$ uploads the piece p_{i1} after encryption (i.e., $K^{i1}_{\mathbb{A},\mathbb{B}}[p_{i1}]$) to $\mathbb{B}$. Then $\mathbb{B}$ will be able to reciprocate by simply forwarding the encrypted piece $K^{i1}_{\mathbb{A},\mathbb{B}}[p_{i1}]$ or by uploading it after re-encryption using its own key to $\mathbb{C}$.

Note that this procedure makes no change in the basic protocol, except for the piece selection scheme. No system resources need to be set aside for newcomer bootstrapping, in contrast with other schemes (e.g., PropShare [7]).

A significant innovation of T-Chain is that this method for bootstrapping newcomers is difficult for free-riders to exploit. Newcomers, like all file requestors, must reciprocate to other leechers in order to receive decryption keys for the (encrypted) pieces they receive. We believe the combination of immediate, barrier-free entry of newcomers into the swarm, without risk of free-riding, is unique in the literature (c.f. Section V).

2) Flow Control (Adaptive Receiver Selection): In the basic protocol described above, qualified neighbors have a uniform probability of being designated as the payee of an encrypted file piece upload. However, in a real swarm, some neighbors may have heterogeneous upload bandwidth capacities, making this policy sub-optimal. A leecher with low upload bandwidth can accumulate a backlog of encrypted file pieces that need to be reciprocated, while a leecher with high upload bandwidth may download pieces at a rate too slow to use its full upload capacity while reciprocating.

To prevent these scenarios, each leecher in T-Chain can maintain a *local* history of its neighbors that records the number of *pending* file pieces, defined as the number of encrypted file pieces uploaded to that neighbor for which it has not yet received notification of reciprocation. A neighbor who has received more than k pending file pieces from $\mathbb{A}$ will be neither selected by $\mathbb{A}$ to receive pieces nor designated as a payee until its number of pending file pieces drops below k. This procedure also helps participants identify uncooperative or malfunctioning neighbors. If a neighbor does not reciprocate its uploads, its number of pending file pieces will increase and it will be banned as a payee. Note that this adaptive selection requires no centralized monitoring or information sharing between participants.

The value of k determines how many pending file pieces a leecher is permitted to buffer. A higher value of k helps smooth out variations in system capacity, processing and networking delays, upload bandwidths, etc., but increases the probability that some leechers are over- and some underloaded. An alternative option for $\mathbb{A}$ is to choose a neighbor with the smallest number of pending pieces. In the experiments described in Section IV, k was set to 2.

3) Opportunistic Seeding: In the basic T-Chain protocol, only a seeder may initiate a chain. However, if too many chains are terminated, e.g., due to leecher failure, departure of the leecher from the swarm without completing the file download, temporary network problems, or free-riding, then the number of chains in the swarm may not fully utilize all of the available upload capacity, degrading system performance. Yet the seeder may not be able to keep up with the rate at which existing chains terminate.

To compensate for too few chains in the swarm, T-Chain can use *opportunistic seeding*: a leecher $\mathbb{B}$ initiates a new chain by voluntarily uploading an encrypted file piece to another leecher $\mathbb{C}$, if $\mathbb{B}$ in possession of at least one completed file piece and has no pending (not yet reciprocated) file pieces. In such a transaction, the leecher $\mathbb{B}$ plays the role of the seeder and thus selects *both* the requestor and the payee, as is the case for normal seeders. The leecher $\mathbb{B}$ may, and probably will, designate itself as the leecher to whom $\mathbb{C}$ must reciprocate, which benefits $\mathbb{B}$ itself. Opportunistic seeding immediately increases the number of chains in which $\mathbb{B}$ is participating, benefiting both $\mathbb{B}$ and the system. We investigate the frequency and the effect of opportunistic seeding in Section IV-D.

III. Security, Performance, and Overhead

In this section, we discuss how T-Chain counteracts known strategic manipulation techniques for free-riding and increases the rate of peer bootstrapping. We also show that these benefits come with small additional overhead.

A. Countering Known Free-Riding Attacks

In this section, we first consider T-Chain's vulnerability to five previously-known free-riding attacks (exploiting altruism, cheating, the large-view exploit, whitewashing, and the Sybil attack), and then discuss its vulnerability to attacks tailored to the operation of T-Chain. Attacks with other goals, such as denial of service, content pollution, or malicious disruption of system operation, are outside the scope of T-Chain.

1) Exploiting Altruism: In BitTorrent, free-riders can exploit the altruism of seeders, who do not expect reciprocation for uploads, and optimistic unchoking [18]. In contrast, T-Chain does not use altruism: any work that is done requires reciprocation in order to be successfully completed. The only exception occurs during chain termination (Section II-A3): a seeder or a leecher $\mathbb{X}$ in a tiny swarm, who cannot find any leecher (including itself) needing a file piece from $\mathbb{Y}$, may upload an unencrypted file piece to $\mathbb{Y}$. However, this is a rare occurrence; moreover, as discussed in Section II-A3, free-riders cannot easily cause and exploit chain terminations.

2) Cheating: Cheating, or initiating transactions with other leechers and later refusing to reciprocate by uploading to those leechers, can easily occur in BitTorrent [2]. With T-Chain, however, leechers derive no advantage from refusing to reciprocate: the pieces downloaded from other peers are encrypted, and are therefore useless to the downloader without the matching decryption key, which is only released upon reciprocation.

3) Large-view-exploit and Whitewashing: Since T-Chain prevents exploiting altruism and cheating, there is little benefit for free-riders to increase their chances of receiving altruistic uploads by artificially increasing their number of neighbors (i.e., using the large-view-exploit [19], [20]) or by frequently changing their identities (i.e., engaging in whitewashing [9], [21]). Even though they can potentially increase the number of encrypted pieces received through these techniques, they must still reciprocate to decrypt the encrypted pieces.

4) Collusion and the Sybil Attack: Free-riders can collude with each other to maximize their benefits without contribution. For instance, indirect reciprocity (e.g., reputation) based schemes are vulnerable to collusive behavior such as false accusation and praise [22]. Such attacks are more difficult in T-Chain and can only occur in isolated, rare scenarios: suppose $\mathbb{D}$ uploads to $\mathbb{R}$ and designates $\mathbb{P}$ as the leecher to whom $\mathbb{R}$ must reciprocate. If $\mathbb{R}$ and $\mathbb{P}$ are in collusion (or $\mathbb{R}$ and $\mathbb{P}$ are false IDs of the same peer), leecher $\mathbb{P}$ may lie to $\mathbb{D}$, falsely stating that $\mathbb{R}$ uploaded an encrypted piece to it when in fact $\mathbb{R}$ did not. In this case, $\mathbb{D}$ will upload the matching key to $\mathbb{R}$ "for free", so free-riding will occur.

This type of collusion or Sybil attack is possible only during indirect reciprocity: in direct reciprocity, $\mathbb{D}$ designates *itself* as the payee $\mathbb{P}$ to whom $\mathbb{R}$ must upload and will not give the key to $\mathbb{R}$ for decryption unless it actually receives a reciprocal piece from $\mathbb{R}$. We now calculate the probability that a collusion (or Sybil) attack can occur during indirect reciprocity. For such attacks to be successful, the requestor and the payee of the same transaction must be colluders (or Sybils). We argue that probability of this occurring is very small, unless the colluder (or Sybil) set is very large, which is difficult to achieve.

Suppose that there are N peers in the system. Then each peer receives b randomly chosen neighbors from the tracker. Let $\mathtt{S}$ denote one colluder (or Sybil) set of m peers; typically, $m \ll N$ and $b \ll N$. We now calculate the probability of a successful collusion (or Sybil) attack, i.e., that the payee and requestor are both from $\mathtt{S}$.

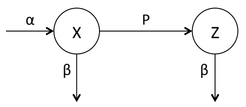
Since the requestor $\mathbb{R}_{i+1}$ of transaction $i+1$ is the payee $\mathbb{P}_i$ of the previous transaction i (i.e., $\mathbb{R}_{i+1} = \mathbb{P}_i$), the requestor $\mathbb{R}_{i+1}$ and payee $\mathbb{P}_{i+1}$ of transaction $i+1$ must have been separately chosen by the donors $\mathbb{D}_i$ and $\mathbb{D}_{i+1}$ of the previous and current transactions i and $i+1$. Both randomly choose their payees $\mathbb{P}_i$ and $\mathbb{P}_{i+1}$ from among their neighbors. To simplify the discussion, we assume that the current transaction is not terminating and that all neighbors are equally eligible to be chosen as payees. By the definition of T-Chain, $\mathbb{D}_i$ and $\mathbb{D}_{i+1}$ ($\equiv \mathbb{R}_i$, the requestor of transaction i) must be different peers. We thus compute the probability $\mathbf{P}_s$ of a successful attack as follows: we first denote by $\mathbf{P}_l$ the probability that l out of b peers returned from the tracker are colluders (or Sybils) in $\mathtt{S}$, and by $\mathbf{P}_c$ the probability that both the requestor and the payee of a "random" chain are from these l. Then, it is not hard to see that:

$$\mathbf{P}_s = \sum_{l=2}^{\min(m,b)} \mathbf{P}_l \mathbf{P}_c,$$

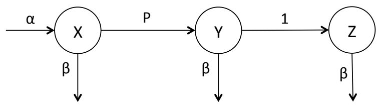
where

$$\mathbf{P}_l = \prod_{i=0}^{l-1} \frac{m-i}{N-i}, \quad \mathbf{P}_c = \prod_{j=0}^{1} \frac{l-j}{b-j}.$$

Note that when $m \ll N$, the probability $\mathbf{P}_s$ is very small. Moreover, since successful attacks require indirect reciprocity, the actual success probability is much lower than $\mathbf{P}_s$. Section IV-C experimentally investigates the effect of many



(a) BitTorrent-like model.



(b) T-Chain model.

Fig. 2: Transition diagrams for the two protocols.

leechers colluding with each other; the colluders receive very slow download times, making collusion impractical.

5) Failure to Complete the Exchange: In Figure 1(b), the participant $\mathbb{B}$, having previously uploaded an encrypted file piece to $\mathbb{C}$, may later fail to upload the decryption key to $\mathbb{C}$. There is minimal gain for $\mathbb{B}$ in failing to upload the key, however, since uploading the key requires several orders of magnitude less bandwidth than uploading the file piece did.

B. Newcomer Bootstrapping Speed

We compare T-Chain's bootstrapping speed with that of a simplified BitTorrent-like protocol in which each peer selects a random peer to unchoke every $1/\delta$ timeslots (normally $1/\delta$ is 5, i.e., 20% of bandwidth is used for BitTorrent's optimistic unchoking). We consider a discrete-time system with $t = 0, 1, 2, \ldots$ indexing the timeslot and suppose that one file piece per chain is uploaded in each timeslot. Thus, each T-Chain transaction spans two timeslots: one in which the donor uploads a file piece to the receiver, and one in which the receiver uploads a piece to the payee. We do not explicitly consider the time for transmitting file piece receipts and decryption keys, since they are much smaller than the file pieces (c.f. Section III-C). All proofs are given in [23].

We define three state variables to track the number of peers in the swarm at each time t: $x(t)$, the number of completely un-bootstrapped peers; $y(t)$, the number of partially bootstrapped peers (i.e., they have received an encrypted file piece in their first transaction but have not yet reciprocated), and $n(t)$, the total number of peers. The total number of un-bootstrapped peers is then $y(t) + x(t)$. For ease of notation, we also define $z(t) = n(t) - x(t) - y(t)$ as the number of fully bootstrapped peers.

We define β as the peers' departure rate and α as the newcomer arrival rate, as shown in Figure 2's state transition diagrams. Here P represents the probability of bootstrapping. We use M to denote the total number of file pieces, and assume that each (bootstrapped) peer in T-Chain participates (i.e., uploads a file piece and designates a payee) in on average K chains per timeslot. Peers participate in direct reciprocity if they require any file pieces possessed by the designated recipient of the current transaction; we use ω to denote the fraction of chains in which bootstrapped peers do *not* participate in direct reciprocity.

1) BitTorrent-like Dynamics: In this simplified BitTorrent-like method, a peer uploads a piece to a randomly selected peer with probability δ in each timeslot and otherwise uploads based on the peers' contributions (generally, $\delta = 0.2$ for BitTorrent as discussed above). In this case, $y(t) \equiv 0$, since no peers are partially bootstrapped. We now calculate the probability that an un-bootstrapped peer will be bootstrapped at time t (Figure 2(a)):

$$P = \frac{1}{n(t)} + \left(1 - \left(1 - \delta + \frac{\delta\,(n(t)-2)}{n(t)-1}\right)^{z(t)}\right) - \left(1 - \left(1 - \delta + \frac{\delta\,(n(t)-2)}{n(t)-1}\right)^{z(t)}\right)\frac{1}{n(t)},$$

where the first term is the probability that the seeder bootstraps the peer, the second term is the probability that another downloader (i.e., leecher) bootstraps the peer, and the third term accounts for the fact that it is possible that the peer will be chosen by both the seeder and a downloader. We then have the dynamical equation for the expected values of x:

$$x(t+1) = \frac{x(t)(1-\beta)(n(t)-1)}{n(t)}\left(\frac{n(t)-1-\delta}{n(t)-1}\right)^{z(t)} + \alpha n(t), \tag{1}$$

where we have simplified the expression for P. Moreover, we find that $n(t+1) = (1 - \beta + \alpha)n(t)$, allowing us to solve for $n(t) = (1 - \beta + \alpha)^t\, n(0)$. Thus, if $\beta = \alpha$, i.e., the arrival and departure rates are the same, then the number of peers in the system remains constant: $n(t) = n$.

2) T-Chain Dynamics: We now formulate the dynamics for T-Chain, with the state transition diagram in Figure 2(b). We suppose that $t > 1$ and find that

$$P = 1 - \left(\frac{n(t)-1}{n(t)}\right)\left(\frac{n(t)-2}{n(t-1)-1}\right)^{K\omega(z(t-1))} \tag{2}$$

in Figure 2(b), accounting for both the seeder's probability of choosing a given un-bootstrapped peer and the probability that a fully bootstrapped peer at time $t-1$ designated a currently un-bootstrapped peer as the next recipient in the chain. We next calculate ω, or the probability that a bootstrapped peer engages in indirect reciprocity in a given chain. This occurs if (i) the peer must upload to another bootstrapped peer and does not need any of its file pieces, or (ii) the peer must upload to a fully un-bootstrapped peer. Thus,

$$\omega = \frac{x(t-1) + \omega' y(t-1) + \omega''(z(t-1)-1)}{n(t-1)-1}, \tag{3}$$

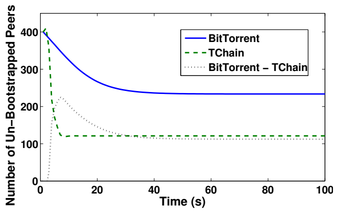
where $\omega' = \sum_{n_j=1}^{M-1} p_j n_j / M$ is the probability that the peer already has the single file piece possessed by a partially bootstrapped peer and ω'' is the probability that the peer already has all the file pieces of another fully bootstrapped peer. Here we define p_m as the probability that a given bootstrapped peer has m file pieces. To calculate ω'', we find the probability that bootstrapped peer j does not need any pieces from bootstrapped peer i, i.e.,

$$\omega'' = \sum_{m_j=1}^{M-1} p_{m_j} \sum_{m_i=1}^{m_j} p_{m_i} \frac{(M-m_i)!\,m_j!}{M!\,(m_j-m_i)!}. \tag{4}$$

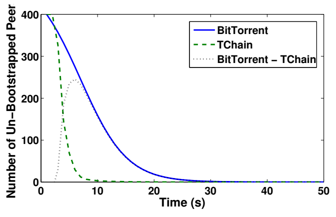
If M is large and the p_m are uniform, then we have $\omega'' \approx \log(M)/M$. Note that ω' and ω'' are independent of the state variables and may be taken as fixed constants. Thus, we can write down equations for the state variables using (2) and (4):

$$x(t+1) = \alpha n(t) + x(t)(1-\beta)\,(1-P) \tag{5}$$

$$y(t+1) = x(t)(1-\beta)P \tag{6}$$



(a) $(\alpha, \beta) = (0.1, 0.1)$.



(b) $(\alpha, \beta) = (0, 0)$.

Fig. 3: Evolution of the number of un-bootstrapped peers with $K = 5$, $\omega'' = \log(100)/100$, $\omega' = 0.495$, $\delta = 0.25$, $n = 500$.

where $n(t) = (1 - \beta + \alpha)^t n(0)$, P is given by (2), and ω is given by (3) and (4). Again, if $\beta = \alpha$, $n(t)$ is a constant n.

3) Protocol comparison: First, we take $\beta = \alpha$, so that both BitTorrent and T-Chain have a constant number of peers n. We use $x_b(t)$ to denote the number of un-bootstrapped BitTorrent peers and $(x_t(t), y_t(t))$ to denote the numbers of fully and partially un-bootstrapped T-Chain peers; equilibria are denoted by omitting the (t) argument. We then find that:

Proposition 3.1 (Equilibrium un-bootstrapped peers.): Suppose that $\alpha = \beta$. The system converges to an equilibrium under either method, $y_t = \beta(n - x_t)$, and a necessary condition for $x_b \geq x_t + y_t$ (i.e., BitTorrent to have more un-bootstrapped peers than T-Chain in equilibrium) is that

$$\frac{n}{n-1} \leq 1 + (1-\beta)\left(1 - \left(\frac{n-1}{n}\right)\left(\frac{n-2}{n-1}\right)^{K(1-\beta)^2 n}\right). \tag{7}$$

A sufficient condition for $x_b \geq x_t + y_t$ is that

$$\frac{n}{n-1} \leq \left(\frac{n-1-\delta}{n-1}\right)^{(1-\beta)n} \left(1 + (1-\beta) \times \left(1 - \left(\frac{n-1}{n}\right)\left(\frac{n-2}{n-1}\right)^{\frac{K(1-\beta)^2 n\overline{\omega}}{n\beta+1-\beta}}\right)\right), \tag{8}$$

$\overline{\omega} = \left(n\left(\beta + \omega'\beta(1-\beta) + \omega''(1-\beta)^2\right) - \omega''\right)/(n-1)$. Intuitively, as $K\overline{\omega}$ grows, both (7) and (8) are more likely to be satisfied, since T-Chain peers are more likely to engage in indirect reciprocity and bootstrap newcomers.

We now focus on the case $\alpha = \beta = 0$, i.e., peers neither depart nor enter the system, to examine the relative convergence rates of both methods. It is easy to see from (1) and (5) that the system converges exponentially fast. Thus, if a flash crowd of newcomers arrives, the largest decrease in the number of un-bootstrapped peers comes at the beginning of the time interval. We therefore examine how quickly these newcomers are bootstrapped in the first few timeslots:

Proposition 3.2 (Short-term convergence rates): Suppose that $\alpha = \beta = 0$. Then at the equilibrium, $x_b = x_t = y_t = 0$. If $n - x_b(t) << n$ for BitTorrent and $z_t(t-1) << n$ for T-Chain, then T-Chain converges faster to the equilibrium if

$$K z_t(t-1)\left(\frac{x_t(t-1) + \omega' y_t(t-1) + \omega''(z_t(t-1) - 1)}{n-1}\right) \geq \delta(n - x_b(t)). \tag{9}$$

Intuitively, at time t, $K z_t(t-1)\omega$ peers are chosen for indirect reciprocity in T-Chain, and $\delta(n - x_b(t))$ peers are chosen for optimistic unchoking by BitTorrent. Thus, in order for T-Chain to converge faster than BitTorrent, T-Chain should choose more peers, giving it a larger probability of bootstrapping newcomers. For instance, if $x_t(t-1) + y_t(t-1) \leq x_b(t)$ and $x_t(t-1) + y_t(t-1) \geq n\mu$ (T-Chain has fewer un-bootstrapped peers than BitTorrent, and a fraction μ of peers are not bootstrapped), then a sufficient condition is $K\omega'\mu \geq \delta$, which holds, e.g., if $\delta = 0.2$, $\omega' = 0.495$ (approximating ω' with $M = 100$ and $p_m = 1/M$), $\mu = 0.5$, and $K = 2$.

Figure 3 illustrates Propositions 3.1 and 3.2's results when a flash crowd of newcomers arrives at time $t = 0$. T-Chain has consistently fewer un-bootstrapped peers than BitTorrent, both when $\alpha = \beta = 0.1$ (peers arrive and depart, Figure 3(a)) and when $\alpha = \beta = 0$ (no arrivals or departures, Figure 3(b)).

C. *Overhead of T-Chain*

To implement T-Chain on top of BitTorrent, we must add some additional mechanisms (e.g., symmetric key encryption, reception reports, etc), which yield some additional overhead.

1) Encryption Overhead: Each leecher in T-Chain must decrypt and encrypt the equivalent of the entire file once. Sirivianos et al. [10] have shown that the encryption of a 128KB piece with a symmetric key takes only 0.715 milliseconds. Thus, a 1GB file, for instance, requires only 12 seconds for encryption and decryption, compared to the 1024 seconds required to transfer the file at 1MBps. The encryption and decryption time yields an overhead of less than 1.2%.

2) Report Overhead: T-Chain file transfers experience additional delay due to the reciprocal upload of a file piece, transmission of a reception report, and key uploading that must occur before a transaction completes. However, the reception report and the key uploaded are very small in size compared to file pieces, and thus the transmission time for those messages is negligible. Moreover, consecutive transactions in a chain are interleaved as seen in Figure 1, so (without encryption) the total completion time of n transactions in a single chain of T-Chain takes no more than the time for $n+2$ piece uploads in BitTorrent. More importantly, each leecher may engage in multiple uploads and downloads simultaneously. Leechers waiting for a key upload can still participate in other chains until the transaction is complete.

3) Required Space: T-Chain requires space to store pending file pieces (i.e., the encrypted pieces received but not yet reciprocated) and their decryption keys, as well as encrypted file pieces before transmission. The space used for pending file pieces, however, can be reused to store the decrypted pieces once the key to a pending file piece is received. Encrypted files prepared for transmission can be deleted after transmission; the leecher only needs to store the matching key to complete the transaction. Thus, each leecher would require only 256KB of additional space for a 1GB file if 128KB file pieces and 256-bit encryption keys are used, representing a 0.02% overhead.

IV. EVALUATION

We evaluate T-Chain's effectiveness through event-driven simulations in a wide range of scenarios and present the results here. To perform our experiments, we use a BitTorrent simulator [2] to measure the performance of a standard BitTorrent system as the basis of the experiments. This simulator models all of the usual BitTorrent protocol functions, including joining and leaving the swarm, neighbor choking, normal and optimistic unchokings, seeding, piece exchanges, etc.

A. Simulation Setup

Each experimental run started with a swarm consisting of a single seeder (without leechers). This seeder remained in the swarm throughout the simulation run. A leecher joining the swarm was assumed to begin downloading file pieces, remain in the swarm until its download of the file completed, and exit the swarm immediately upon completion. We initially model the leechers' arrival as a flash crowd in which all leechers joined the swarm within the first 10 seconds. For instance, a file may attract high interest prior to its release [24], representing a demanding test case for file sharing. We later use a real trace arrival model taken from the RedHat 9 Torrent tracker trace [25], which represents 5 months of activity in an actual BitTorrent swarm.

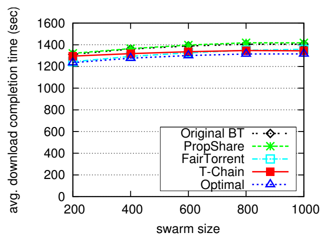
We compared the performance of four protocols: original BitTorrent [5], PropShare [7], FairTorrent [8], and T-Chain. The latter three protocols were implemented as BitTorrent extensions in our simulator. In each, we set the seeder's upload bandwidth to 6,000 Kbps. The upload bandwidth of leechers was assumed to be heterogeneous, varying from 400 Kbps to 1,200 Kbps, in accordance with the assumptions of [2], [18], [24]. There was no limit on the download bandwidth of leechers; upload bandwidth was assumed to be the limiting factor or resource [26]. The file size was taken as a fixed size of 128 MB (1 Gb) unless otherwise stated. The block sizes of BitTorrent and PropShare were set to 16 KB, and the piece size was set to 256 KB (i.e., one piece = 16 blocks), agreeing with the values used by most actual BitTorrent clients. A piece size of 64 KB was used for T-Chain and FairTorrent without further subdivision; this is the basic exchange unit of FairTorrent [8].

Data points in each graph show the mean and 95% confidence intervals of the average file download completion time over 30 runs, using different random number seeds.

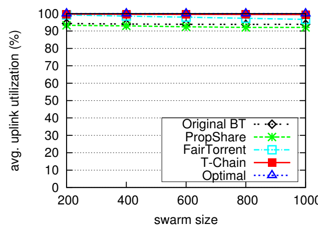
B. Effect of Free-Riding

T-Chain is designed to enforce reciprocity and thereby prevent free-riding in file sharing applications, without degrading the system performance. To evaluate whether this goal has been achieved, we first measure the system performance when there are only *compliant* leechers (i.e., leechers that comply with all the normal requirements of each protocol) and then compare it to the performance with free-riders.

Figure 4 shows the results without free-riders. Figure 4(a) shows that all methods perform similarly (in terms of average download completion time) and close to optimal (cf. [24]). T-Chain and FairTorrent have slightly smaller download completion times than the other methods due to their better uplink utilization, which can be observed from Figure 4(b). This improvement comes from the fact that T-Chain and FairTorrent

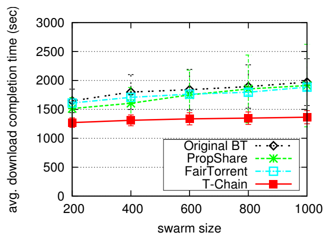


(a) Avg. download completion time.

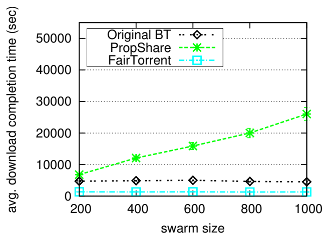


(b) Avg. uplink utilization.

Fig. 4: (a) Average download completion time and (b) average uplink utilization for leechers in BitTorrent, PropShare, FairTorrent and T-Chain under a flash crowd leecher arrival model without free-riders.



(a) Compliant leechers.



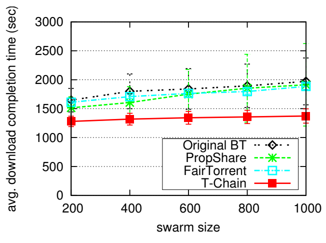
(b) Free-riders.

Fig. 5: Average download completion time for (a) compliant leechers and (b) free-riders in BitTorrent, PropShare, FairTorrent and T-Chain in a flash crowd (with the large-view-exploit and whitewashing).

dynamically adjust the system resources to bootstrap more newcomers soon after their arrival, compared to the fixed 20% of system resources that are pre-allocated for bootstrapping in BitTorrent and PropShare. All methods are scalable; as the swarm size and demand on each peer's upload bandwidth increase, the download completion times stay relatively constant.

We next show the results of an experiment in which 25% of the leechers were free-riders. Each free-rider engaged in the worst possible behavior and provided zero upload bandwidth to other leechers. In addition, it was assumed that each free-rider attempted to avoid penalties for its behavior by using the large-view-exploit. Leechers requested a new list of neighbors from the tracker at every rechoking period (10 second intervals), more frequently than in normal BitTorrent operations, and accepted all neighboring requests. We further assumed that the free-riders employed whitewashing and the Sybil attack [18]–[20]. A free-rider in FairTorrent may want to employ whitewashing to neutralize the deficit-based approach by disconnecting and reconnecting its TCP connection as soon as it gets one (free) piece from one of its neighbors. This effectively restores its deficit value (to zero), allowing it to be treated as another newcomer by the deceived neighbor.

Figure 5(a) indicates that the addition of the free-riders lengthens the average download completion time for compliant leechers by as much as 33%, 29% and 28% for BitTorrent, PropShare, and FairTorrent respectively. T-Chain, in contrast, effectively protects compliant leechers from this performance degradation. Figure 5(b) shows that free-riders are successful in BitTorrent, PropShare, and FairTorrent, with FairTorrent delivering the best and PropShare the worst (i.e., longest completion times) performance for the free-riders. Simple whitewashing thus enables free-riders in FairTorrent to finish their downloads as fast as compliant leechers. The


(a) Compliant leechers.

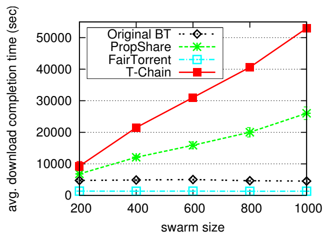

(b) Free-riders.

Fig. 6: The effects of collusion in T-Chain under the flash crowd arrival model (the same setting as in Figure 5 except with collusion in T-Chain).

figure contains no line for T-Chain, since not a single free-rider completed the download of the unencrypted file. The almost-fair exchange protocol and designated reciprocation effectively prevent successful free-riding, even under these challenging conditions. Leechers in T-Chain can easily identify uncooperative neighbors through adaptive receiver selection (Section II-C2), minimizing the amount of system resources allocated to free-riders.

C. *Impact of Collusion*

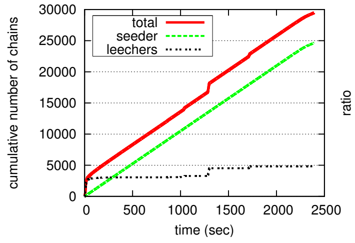
We next evaluate T-Chain's performance when free-riders collude with each other, i.e., lie on each other's behalf. While T-Chain is designed to eliminate incentives for collusion, it cannot absolutely prevent it from occurring; collusion opportunities are highly limited (Section III-A4) but do exist. We investigate collusion under the same experimental settings with free-riders as in Section IV-B. We assumed that *all* free-riders in T-Chain colluded, sending false reception reports on behalf of other colluding free-riders.

Figure 6 shows the impact of collusion against T-Chain. Since collusion only affects T-Chain, the results for the other methods are as before. As seen in Figure 6(b), with collusion free-riders *are* able to complete their downloads. However, the average download completion time for a free-rider is almost 40 times longer than for a compliant leecher when the swarm size is 1,000. Free-riders' average download speeds are thus less than 20Kbps (slower than dial-up). The average download completion times for free-riders are 103%, 1,066%, and 3,497% higher with T-Chain than with PropShare, BitTorrent and FairTorrent, respectively. In addition, collusion has little effect on the average download completion times for T-Chain's compliant leechers (compare Figures 5(a) and 6(a)).

D. *Opportunistic Seeding*

As discussed previously in Section II-C3, opportunistic seeding (i.e., initiation of a new chain by a leecher) is helpful when the number of chains in the swarm is insufficient to fully utilize all of the available upload capacity of the system. Leechers can benefit from opportunistic seeding through direct reciprocity (i.e., designating themselves as transaction payees). We conducted two experiments to investigate the frequency of opportunistic seeding in T-Chain.

Figure 7(a) shows the cumulative number of chains created by the seeder (green dotted line) and leechers (black dotted line) in a flash crowd model. In this graph, it was assumed that


(a) Under a flash crowd.

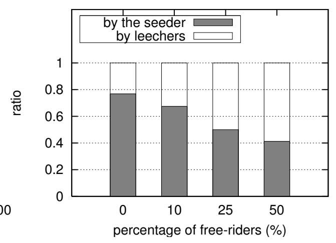

(b) Under a real trace.

Fig. 7: (a) The cumulative number of chains created by the seeder and by leechers in a flash crowd, and (b) the fraction of chains resulting from opportunistic seeding in a real trace, as a function of the fraction of free-riders in the system.

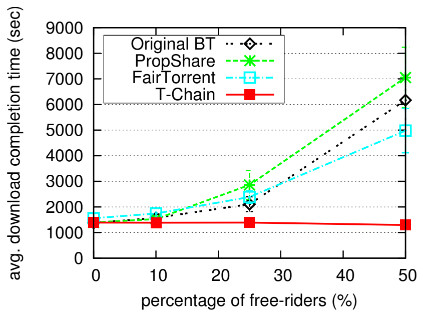


Fig. 8: The average download completion times for compliant leechers in BitTorrent, PropShare, FairTorrent and T-Chain under a continuous stream model.

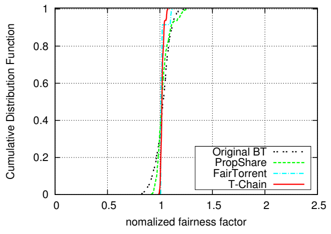
600 compliant leechers without free-riders join the system. As seen in the figure, the amount of opportunistic seeding is high when the system is newly initiated and the seeder cannot satisfy the demands of all newcomers, resulting in under-utilization of newcomers' available upload bandwidth without opportunistic seeding. After several dozens of seconds, the rate of opportunistic seeding is nearly zero, as reciprocation fully utilizes the upload capacity.

Figure 7(b) shows the fraction of chains resulting from opportunistic seeding under a real trace model, as a function of the fraction of free-riders in the system. As the number of free-riders increases, opportunistic seeding creates more chains, since each instance of free-riding will terminate a chain. The under-utilization of upload capacity caused by such chain termination is immediately compensated by leechers using opportunistic seeding with T-Chain. However, free-riders are still unlikely to successfully download the file.

E. *Real Swarm Performance*

We next consider conditions more gradual than flash crowds by examining the system performance when arrivals mirrored the behavior of leechers in a single BitTorrent swarm that downloaded the RedHat 9 release [25]. Free-riders provided no upload bandwidth and attempted to avoid penalties by means of the large-view-exploit and whitewashing. We measured completion times for the first 1,000 compliant leechers that successfully completed their downloads for each method, but excluded the first 500 compliant leechers from the average performance in order to avoid startup transients and focus on the steady state performance.

Figure 8 demonstrates the average download completion


(a) No free-riders.

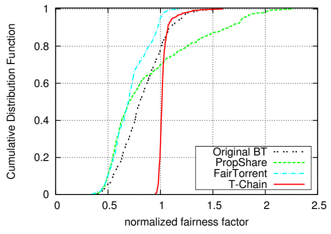

(b) 25% free-riders.

Fig. 9: Fairness enforced by each method.

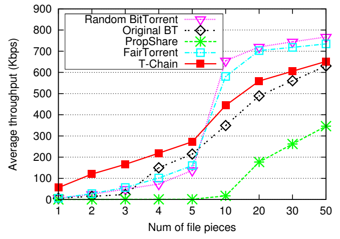
times for compliant leechers with each method. These times are quite similar until the fraction of free-riders exceeds 10%, at which point T-Chain clearly delivers better results. When the fraction of free-riders is 50%, the average download completion time for compliant leechers with BitTorrent, PropShare, and FairTorrent is roughly 5 times longer than with T-Chain. Compliant leechers in T-Chain can effectively detect and penalize free-riders through adaptive receiver selection (Section II-C2), and opportunistic seeding enables better utilization of the system resources (Section IV-D) than BitTorrent's and PropShare's fixed resource allocations for newcomer bootstrapping. Simple whitewashing severely deteriorates FairTorrent's performance for compliant leechers.

We next evaluate T-Chain's performance in terms of *fairness*, or the ratio of leechers' benefits (i.e., received piece downloads) to their contributions (piece uploads). In a fair system, this ratio would be 1: leechers would benefit according to their contributions, encouraging participation in a cooperative system. To evaluate the fairness offered by each method, we define the fairness factor as the ratio of the pieces downloaded to the pieces uploaded by each leecher during its participation in the swarm.

We use the same experimental conditions as in Figure 8 and show the resulting fairness in Figures 9(a) and 9(b). These figures plot the Cumulative Distribution Function (CDF) of the fairness factors of the last 500 compliant leechers in each system. Figure 9(a) demonstrates that all four methods are quite fair when there is no free-riding in the system, with T-Chain and FairTorrent being slightly more fair than the others. However, Figure 9(b) shows that when free-riding increases to 25% of leechers, only T-Chain continues to achieve a high level of fairness, with only a few leechers receiving more pieces than they contribute. BitTorrent, PropShare, and FairTorrent show a marked divergence from fairness. Compliant leechers are therefore likely to conclude that T-Chain is fairer than BitTorrent, PropShare, or FairTorrent.

F. Performance with Small Files

We finally measure T-Chain's performance when transferring small files. We compare the four methods above (BitTorrent, PropShare, FairTorrent, and T-Chain) to Random BitTorrent, in which all leechers' and seeders' bandwidth was only used for optimistic unchoking, for different file sizes ranging from 64 KB to 3.2 MB. In this experiment, it was assumed that 1,000 leechers join the system as a flash crowd and that a leecher leaves the system upon completing its download, but is then immediately replaced by a newcomer.


(a) No free-riders.

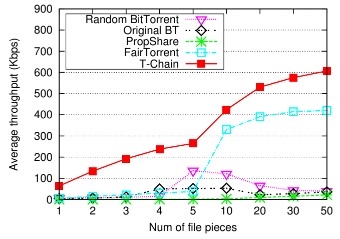

(b) 50% free-riders.

Fig. 10: The average download throughput of compliant leechers in Random BitTorrent, BitTorrent, PropShare, FairTorrent, and T-Chain under various file sizes with different numbers of free-riders.

We thus capture the performance of each approach under high churn rates (in a swarm sharing a small file). We measured the average download throughput of compliant leechers (i.e., the average amount of data successfully downloaded per second) in each system during the first 1,000 seconds. We consider one case with no free-riding and one with 50% of leechers being free-riders.

Figures 10(a) and 10(b) show the download throughput of leechers in a swarm with different file sizes (i.e., different number of file pieces). As seen in Figure 10(a), if the shared file has relatively few pieces (e.g., below 5), the average throughputs of BitTorrent, Random BitTorrent, PropShare, and FairTorrent are extremely low, even without free-riding. The lack of file pieces reduces opportunities for reciprocation, so seeding is the primary method of distribution. In an extreme case with only one shared piece, every compliant leecher leaves the system as soon as the piece is attained, and the system effectively functions as a client-server model with the seeder as the server and the leechers as the clients. T-Chain achieves better performance than do other methods in this scenario, since leechers are forced to reciprocate.

When the number of file pieces ranges between 5 and 30, Random BitTorrent and FairTorrent outperform T-Chain due to the overhead of the piece encryption and key exchange. The effect of this overhead, however, is quickly diluted as the file size grows, as seen from the previous results with large files (Sections IV-B – IV-E). BitTorrent and PropShare continue to perform worse than T-Chain: their fixed bandwidth allocation for newcomer bootstrapping is insufficient for small files and high churn rates. If 50% of the leechers are free-riders, T-Chain performs better than all the other methods, regardless of file size (Figure 10(b)).

V. Related Work

The success of cooperative computing depends on effectively motivating or incentivizing participants to voluntarily donate their resources to the system. If reciprocity can be *enforced*, the problem of free-riding can be greatly reduced or eliminated. To achieve cooperation, many incentive schemes have been proposed in the literature. These schemes encourage or enforce reciprocity based on direct experience, on information indirectly obtained (e.g., reputation), or on the use of encryption. Table II summarizes the advantages provided by T-Chain when compared to other major direct and indirect reciprocity schemes.

TABLE II: Comparison of major incentives under possible attacks (√ : Good, (blank) : Medium, × : Bad)

Features		Direct Reciprocity				Indirect Reciprocity	
		BitTorrent	PropShare	FairTorrent	**T-Chain**	EigenTrust	Dandelion
Simplicity & Scalability		√	√	√	√	×	×
Fairness		×		√	√	√	√
Flexible Newcomer Bootstrapping		×	×	√	√	×	×
Immunity to	Exploiting Altruism	×	×	×	√	√	√
	Cheating	×			√	√	√
	Large-view-exploit	×		√	√	√	√
	Sybil or Whitewashing		√	×	√		
	Collusion	√	√	√			√
	False Praise (Accusation)	√	√	√	√		
Asymmetric Interest				√	√	√	√
Work with small files		×	×		√		

Direct reciprocity [5]–[8] is a straightforward approach in which the willingness of two participants to cooperate is influenced by the quality of their past direct interactions. The rate-based TFT policy of BitTorrent is a well known example. BitTyrant [18], PropShare [7], and FairTorrent [8] attempt to improve TFT's fairness by tweaking resource allocation to unchoked neighbors. The most significant advantage of direct reciprocity is its simplicity of implementation, in that any decision depends only upon local observations. However, it is difficult for these approaches to accommodate asymmetric interests, capabilities, and state among participants, with the result that some of the system capacity may be wasted. Additionally, these approaches have been shown to be vulnerable to cheating, in that a file piece upload by one participant may not be reciprocated by the other party. Note that approximately 20% of the system resources in BitTorrent and PropShare are reserved for transaction initiation, which has been shown to be a vulnerable target of strategic free-riding. The simple elimination of such initiation mechanisms to prevent free-riding, however, would significantly hurt overall system performance [27]. FairTorrent [8] uses a deficit-based distributed algorithm to achieve strong fairness, but its immunity to whitewashing and the Sybil attack is questionable in that it still uploads unencrypted blocks for transaction initiation (as seen in Sections IV-B – IV-F). In addition, FairTorrent cannot prevent seeders from being exploited by free-riders.

There also exist variants of direct reciprocity. Give-to-Get [6] uses a TFT policy based on forwarding pieces to other participants; this policy is somewhat similar to T-Chain's pay-it-forward reciprocity. Give-to-Get does not, however, designate recipients or validate their feedback, and is therefore readily susceptible to collusion or Sybil attacks. Accelerated Chaining [28] was proposed to let peers in a Video-On-Demand (VOD) application forward video data to their children in a chain at a rate slightly faster than the rate they receive from their parents, virtually eliminating the server workload. The concept of pay-it-forward with chaining in Accelerated Chaining is very similar to Give-to-Get and T-Chain, but they did not consider the strategic manipulation techniques that free-riders could take.

Indirect reciprocity schemes (e.g., reputation or monetary approaches [9]–[13]) base decisions about cooperation on past interactions that are direct (mutual), or that are indirect (involving other participants). Therefore, these schemes can potentially lead to better decisions about cooperation. EigenTrust [9] is a representative example. The largest drawback of these approaches is the complexity of their implementations. Reputation schemes are frequently complicated by the opportunity for participants to spread false information, while monetary systems require significant infrastructure to monitor credit, account for individual transactions, and prevent counterfeiting. One-hop reputation [11], PledgeRoute [12], and Dandelion [10] attempt to reduce this complexity by either limiting the scope of reputation calculations or by relying on a central (trusted) server. Unfortunately, newcomer bootstrapping (which is a critical factor in large, dynamic systems) is ignored in some of these approaches. When considered, bootstrapping commonly relies on some sort of altruism (e.g., in EigenTrust, 10% of each participant's resources are allotted for newcomers with no previous reputation). Those resources have been the target of strategic free-riders.

Encryption-based approaches [2], [7], [10], [14] are attractive because they attempt to prevent altruism (allocated for newcomer bootstrapping) from being exploited by free-riders. Existing schemes, however, have several limitations. Dandelion [10] uses both indirect reciprocity, as discussed above, and file piece encryption to force reciprocity, yet also relies on a trusted third party. This has scalability issues, and represents a single point of failure or compromise. In addition, the issue of seeding, or newcomer bootstrapping, is avoided by assuming that newcomers start with some initial credit, earned by some means outside the scope of the file-sharing system. The authors of PropShare [7] suggested a bootstrapping scheme using encryption (without providing details). However, T-Chain bootstraps newcomers very differently, and uses encryption throughout for enforcing reciprocity. We can in fact identify several differences between T-Chain and PropShare: (a) PropShare uses only a fixed amount (i.e., 20%) of total system resources for bootstrapping; (b) it provides *no* direct incentives to the two participants involved in bootstrapping; (c) it assumes that two participants $\mathbb{A}$ and $\mathbb{B}$ have had multiple interactions in the past; and (d) it assumes $\mathbb{A}$ and $\mathbb{B}$ have mutual (i.e., symmetric) interests. PropShare also has no chaining effect (i.e., propagation of reciprocation). The secret sharing approach of TBeT [2] is not applicable to streaming applications and is also vulnerable to the key disclosure problem.The hierarchical chunk cipher scheme [14] is another encryption-based approach, based on hierarchical circular shifting of bits. The scheme is, however, designed to prevent the fake chunk attack rather than free-riding.

The use of symmetric key cryptography to enforce reciprocity in T-Chain is reminiscent of a fair exchange protocol [29]. Such a protocol guarantees that either all or no parties benefit from shared resources, effectively preventing free-riding. Fair exchange is relatively easy to accomplish by means of an online, trusted third party, but otherwise is surprisingly difficult, slow, and complex to achieve. Note that T-chain is not a strictly fair exchange protocol, in that cheating is possible, but it removes most of the incentive for cheating. Moreover, T-Chain is extremely lightweight and simple in comparison and requires no trust or central server.

VI. Conclusions and Future Directions

This work proposes T-Chain, a general incentive mechanism to enforce cooperation among participants. T-Chain has two components: (i) an *almost-fair exchange protocol* based on symmetric key cryptography, which does not require a trusted third party; and (ii) a *pay-it-forward reciprocation scheme* that increases opportunities for multi-lateral cooperation and reduces free-riders' opportunities for collusion. We apply T-Chain to the BitTorrent protocol. Leechers download encrypted file pieces from other peers and must reciprocate each piece by uploading another (encrypted) file piece before receiving the decryption key. T-Chain thus makes cooperation mandatory and easily bootstraps newcomers by letting them upload their first received file piece to another peer. No centralized monitoring or control is required, and overhead costs are very low.

The method was evaluated with extensive simulations and compared with BitTorrent, PropShare, and FairTorrent. Under normal conditions, T-Chain provides significantly faster downloads for compliant leechers and prevents all free-riders from completing their downloads. Even under unrealistically severe collusion, free-riders can only download files with extremely slow speeds. T-Chain is also fairer than BitTorrent, PropShare, or FairTorrent for compliant leechers, incentivizing cooperation.

The simplicity of T-Chain fosters cooperation among participants and can be readily adapted to other applications or protocols. Future work will include the application of T-Chain to streaming, content distribution, overlay routing, file replication (and preservation), and name resolution services.

Acknowledgements

K.Shin gratefully acknowledges partial support from the 2013 Korea Military Academy Hwarangdae Research Institute. Y.Yi was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF- 2013R1A2A2A01067633).

References

[1] G. Hardin, "Tragedy of the commons," *Science*, vol. 162, 1968.

[2] K. Shin, D. S. Reeves, and I. Rhee, "Treat-before-trick : Free-riding prevention for bittorrent-like peer-to-peer networks," in *IPDPS'09*, Rome, Italy, May 2009.

[3] R. Guerraoui, K. Huguenin, A.-M. Kermarrec, M. Monod, S. Prusty, and A. Roumy, "Tracking freeriders in gossip-based content dissemination systems," *Computer Networks*, vol. 64, no. 8, pp. 322–338, May 2014.

[4] W. Wu, R. T. Ma, and J. C. Lui, "Distributed caching via rewarding: An incentive scheme design in p2p-vod systems," *TPDS'14*, vol. 25, no. 3, pp. 612–621, March 2014.

[5] B. Cohen, "Incentives build robustness in bittorrent," in *P2PECON*, 2003.

[6] J. J. D. Mol, J. A. Pouwelse, M. Meulpolder, D. H. J. Epema, and H. J. Sips, "Give-to-get: free-riding resilient video-on-demand in p2p systems," in *SPIE Conference Series*, 2008.

[7] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: Analyzing and improving bittorrent's incentives," *SIGCOMM*, 2008.

[8] A. Sherman, J. Nieh, and C. Stein, "Fairtorrent : Bringing fairness to peer-to-peer systems," in *ACM CoNEXT'09*, December 2009.

[9] S. D. Kamvar, M. T. Schlosser, and H. Garcia-molina, "The eigentrust algorithm for reputation management in p2p networks," in *WWW*, 2003.

[10] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecski, "Dandelion: Cooperative content distribution with robust incentives," in *USENIX*, 2007.

[11] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, "One hop reputations for peer to peer file sharing workloads," in *NSDI'08*, 2008.

[12] R. Landa, D. Griffin, R. G. Clegg, E. Mykoniati, and M. Rio, "A sybilproof indirect reciprocity mechanism for peer-to-peer networks," in *INFOCOM'09*, 2009.

[13] X. Kang and Y. Wu, "Incentive mechanism design for heterogeneous peer-to-peer networks: A stackelberg game approach," *To be appear in IEEE Transactions on Mobile Computing*, submitted on 24 Jul 2014.

[14] J. Wang, X. Hu, X. Xu, and Y. Yang, "A verifiable hierarchical circular shift cipher scheme for p2p chunk exchanges," in *Peer-to-Peer Networking and Applications*, 2013.

[15] B. Fan, J. C. Lui, and D.-M. Chiu, "The design trade-offs of bittorrent-like file sharing protocols," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 365–376, 2009.

[16] L. Jian and J. K. MacKie-Mason, "Why share in peer-to-peer networks?" in *International Conference on Electronic Commerce*, 2008.

[17] R. Krishnan, M. Smith, Z. Tang, and R. Telang, "The virtual commons: Why free-riding can be tolerated in file sharing networks?" in *ICIS*, 2002.

[18] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent?" in *USENIX NSDI'07*, May 2007.

[19] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *HotNets'06*, November 2006.

[20] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, "Free-riding in bittorrent networks with the large view exploit," in *IPTPS'07*, 2007.

[21] J. R. Douceur, "The sybil attack," in *IPTPS'02*, March 2002.

[22] M. Feldman and J. Chuang, "Overcoming free-riding behavior in peer-to-peer systems," in *ACM Sigecom Exchanges*, vol. 5, July 2005.

[23] K. Shin, C. Joe-Wong, S. Ha, Y. Yi, I. Rhee, and D. Reeves, "T-chain: A general incentive scheme for cooperative computing," Princeton University, Tech. Rep., 2014. [Online]. Available: http://www.princeton.edu/~cjoe/TChain.pdf

[24] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent network's performance mechanisms," in *INFOCOM'06*, 2006.

[25] "Redhat 9 torrent tracker trace." [Online]. Available: http://mikel.tlm.unavarra.es/~mikel/bt_pam2004/

[26] W. Wu, J. C. Lui, and R. T. Ma, "On incentivizing upload capacity in p2p-vod systems: Design, analysis and evaluation," *Comp. Netw.*, 2013.

[27] S. Jun and M. Ahamad, "Incentives in bittorrent induce free riding," in *P2PECON'05*, Philadelphia, PA, August 2005.

[28] J.-F. Pris, A. Amer, and D. D. E. Long, "Accelerated chaining: A better way to harness peer power in video-on-demand applications," in *ACM Symposium on Applied Computing(SAC)*, 2011.

[29] I. Ray, I. Ray, and N. Natarajan, "An anonymous and failure resilient fair-exchange e-commerce protocol," *Dec. Supp. Sys.*, vol. 39, 2005.