T-Chain: A General Incentive Scheme for Cooperative Computing

Kyuyong Shin, Carlee Joe-Wong, *Member, IEEE,* Sangtae Ha, *Senior Member, IEEE,* Yung Yi, *Member, IEEE,* Injong Rhee, *Member, IEEE,* and Douglas Reeves, *Member, IEEE*

***Abstract*—In this paper, we propose a simple, distributed, but highly efficient fairness-enforcing incentive mechanism for cooperative computing. The proposed mechanism, called Triangle Chaining (T-Chain), enforces reciprocity to avoid 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, allowing them to immediately benefit, 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.**

***Index Terms*—Cooperative communication, reciprocity, symmetric encryption**

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, application-layer multi-cast, file sharing, streaming, and recent work in delay-tolerant networks [1] and cloud [2], mobile [3], and fog computing [4].

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 (the "tragedy of the commons" [5]). *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 [6]–[8].

A previous version of this work appeared at IEEE ICDCS 2015.

K. Shin is with the Korea Military Academy, C. Joe-Wong is with Carnegie Mellon University, S. Ha is with the University of Colorado–Boulder, Y. Yi is with KAIST, and I. Rhee and D. Reeves are with NCSU.

A large range of incentive mechanisms to enforce reciprocity in cooperative computing have been proposed, which can be categorized into three groups: direct reciprocity [9]–[12], indirect reciprocity [13]–[17], and the use of coding or encryption [6], [11], [14], [18]. 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 [6].

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 [19] for details). This trade-off occurs because the goals of ensuring fairness and increasing 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 [20]. 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 an 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. Moreover, resources for bootstrapping are dynamically adjusted to the arrival rate and demands of newcomers. We call the proposed solution Triangle Chaining (T-Chain), and emphasize several of its 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-C.
- *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, the amount of system resources dedicated to newcomer bootstrapping is automatically adjusted according to the system needs instead of being pre-allocated.
- *Fairness*: In order to complete the received resource, the recipient must reciprocate with the same amount of work or resource contribution. This ensures excellent fairness among participants.[1]
- *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.
- *Simplicity*: T-Chain is easy for peers to understand and does not require a trusted third party or any token exchanges. Thus, it can be easily implemented and deployed.

We illustrate T-Chain's benefits and practicality by applying it to BitTorrent [9], 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 [9], PropShare [11], and FairTorrent [12]) 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, T-Chain adds only a 1% overhead to BitTorrent's normal bandwidth and storage requirements.

Section II briefly describes the operations of BitTorrent and details our method, T-Chain. Security considerations, newcomer bootstrapping performance, and implementation overhead 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.

[1]In rare cases, the recipient may not find a participant to which he can reciprocate, hurting fairness. We address this concern in Section II-B3.

TABLE I: Summary of Notation

$\mathbb{A}$, $\mathbb{B}$, ...	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 of a chain
$K[p_{ik}]$	File piece p_{ik}, encrypted with symmetric key K
$F_{\mathbb{A}}$	The set of pieces completed (downloaded and decrypted) by $\mathbb{A}$
$K^{ik}_{\mathbb{A},\mathbb{B}}$	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

II. Design of T-Chain

In this section, we briefly sketch the operations of BitTorrent to better explain our approach. We then present a new method, T-Chain, for minimizing or preventing free-riding. We finally discuss incentives for cooperation and suggest some performance enhancement mechanisms.

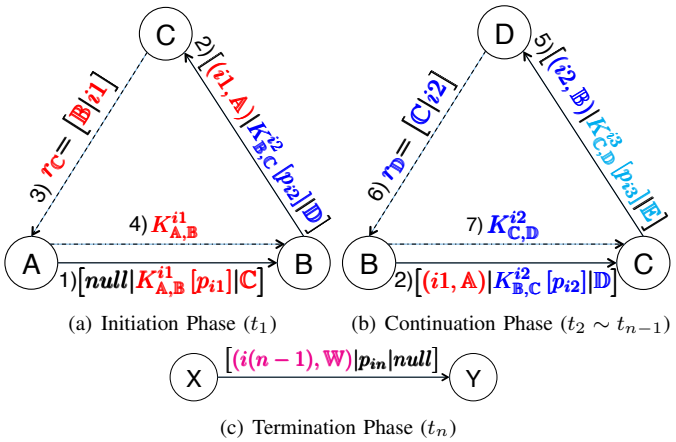
Like most incentive mechanisms, T-Chain does not address network communication failures. We assume that participants do not upload corrupted or falsified file pieces; such pieces can be detected, and their senders blacklisted, with the usual BitTorrent mechanisms. Such an attack is distinct from free-riding: uploading a false piece requires just as much upload bandwidth as uploading a valid file piece.

A. BitTorrent Overview

In BitTorrent, participants form a *swarm* sharing a single file divided into many fixed size pieces, which are further subdivided into blocks. Participants logically 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., 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) [9], in which a leecher initially *chokes* all upload connections to its neighbors, uploading 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



(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.

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.

B. Basic T-Chain Protocol

We apply T-Chain to BitTorrent and assume that the resource being shared (and the limiting factor on the system performance) is upload bandwidth. As mentioned above, we use some BitTorrent terminology and concepts (refer to Cohen's original paper [9] for details). Table I introduces our notation. 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.

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_{ij} to $\mathbb{R}_j$, who is one of its requesting neighbors. This file piece is first encrypted by $\mathbb{D}_j$ with key $K^{ij}_{\mathbb{D}_j,\mathbb{R}_j}$ to ensure payment (i.e., reciprocation). To receive the decryption key $K^{ij}_{\mathbb{D}_j,\mathbb{R}_j}$ 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.[2] 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 respectively 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.

[2]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. However, using new keys ensures that the recipient cannot guess the key from previous transactions.

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 $\mathbb{B}$.[3] $\mathbb{A}$ 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.[4] 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.

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,[5] $\mathbb{A}$ 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, as indicated in Figures 1(a) and 1(b).

2) Continuation Phase (Figure 1(b)): As described above, the fulfillment of one transaction starts another. In the second transaction, $\mathbb{B}$ acts as $\mathbb{A}$ did in the first transaction: step 2 in Figure 1(a) is shown again in Figure 1(b), since it initiates Figure 1(b)'s transaction. Along with uploading $K^{i2}_{\mathbb{B},\mathbb{C}}[p_{i2}]$ to $\mathbb{C}$, $\mathbb{B}$ selects a qualified participant $\mathbb{D}$ from $\mathbb{B}$'s 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.[6]

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, which can continue as usual. Similarly, one leecher may take part in multiple chains simultaneously. In general, this is desirable to fully utilize the available upload bandwidth of all the leechers (see Section II-D3 for details).
- *Indirect reciprocity.* If direct reciprocity is not possible, the donor $\mathbb{B}$ randomly chooses a payee $\mathbb{D}$ among its neighbors who need at least one of $\mathbb{C}$'s file pieces

[3]To expedite the file distribution, the seeder will likely initiate as many chains as possible given its upload and leechers' download capacities.

[4]A different method is used when $\mathbb{B}$ is a newcomer (Section II-D1).

[5]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).

[6]Note that if steps 3 or 4 in Figure 1(a) are interrupted, e.g., due to link failures between $\mathbb{A}$ and $\mathbb{C}$, the next transaction can still take place as usual.

(including the file piece p_{i2} about to be uploaded). The donor does not specify the exact piece that $\mathbb{C}$ should upload to $\mathbb{D}$; as long as $\mathbb{D}$ requires at least one piece from $\mathbb{C}$, reciprocation can occur. If no such neighbor exists in the donor's (not requestor's) neighbor set, then the chain terminates, as discussed in the next section.

Chains can form cycles if a designated payee served as a donor in a previous transaction of the chain. For instance, in direct reciprocity, a two-member cycle is formed. As for direct reciprocity, however, cycles do not affect T-Chain's transactions. Additionally, we 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 a payee $\mathbb{Z}$ for $\mathbb{Y}$ to continue the chain. However, if neither $\mathbb{X}$ nor any of $\mathbb{X}$'s neighbors needs a 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.

A chain terminates if and only if $\mathbb{X}$ cannot find a payee to whom $\mathbb{Y}$ can reciprocate. In practice, this is rare: newcomers to the swarm will always need a piece from $\mathbb{Y}$, and LRF piece selection ensures that pieces are well-distributed, with most peers needing a piece from another. When newcomers stop joining a swarm, all leechers will eventually finish downloading the file and leave the swarm, forcing all chains to 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 to the leecher. Free-riders cannot exploit this vulnerability as they do not control newcomers' arrivals.

4) Effect of Peer Departures: Chain transactions can be interrupted if a participant unexpectedly departs the swarm before the transaction is completed. Using the notation in Figure 1(a) as an example, $\mathbb{A}$ may depart before receiving the reciprocation report from $\mathbb{C}$ and releasing the decryption key to $\mathbb{B}$; or $\mathbb{C}$ may depart before $\mathbb{B}$ can reciprocate to $\mathbb{C}$. Neither event, however, is fatal to T-Chain. $\mathbb{A}$ can easily send its decryption key to $\mathbb{C}$ before leaving the swarm; then $\mathbb{C}$ can send it to $\mathbb{B}$ upon reciprocation. While $\mathbb{A}$ has no concrete incentive to do so, sending a decryption key to $\mathbb{C}$ has negligible overhead for $\mathbb{A}$. If $\mathbb{C}$ leaves the swarm before $\mathbb{B}$ reciprocates, then $\mathbb{A}$ can simply designate another payee, $\mathbb{C}'$, to whom $\mathbb{B}$ should reciprocate. If $\mathbb{C}$ does not leave the swarm but requires no pieces from $\mathbb{B}$, e.g., due to out-dated lists of required file pieces at $\mathbb{A}$, then $\mathbb{A}$ can similarly choose a new payee. Free-riders cannot take advantage of this re-assignment by falsely claiming that $\mathbb{C}$ has left the swarm, since $\mathbb{C}$ is a neighbor of $\mathbb{A}$ and would directly notify $\mathbb{A}$ of its departure.

C. 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 II.1 (Incentive compatibility with direct reciprocity)**.** *Consider two leechers, $\mathbb{A}$ and $\mathbb{B}$, who each need a file piece from the other. Suppose that $\mathbb{A}$ has been asked to reciprocate a previous transaction with $\mathbb{C}$ by uploading a piece to $\mathbb{B}$. Then if either $\mathbb{A}$ or $\mathbb{B}$ do not follow their prescribed actions in the next stage of the chain (cf. Figure 1(b)), the other can retaliate by preventing $\mathbb{A}$ or $\mathbb{B}$ from receiving a file piece.*

Proof. There are three steps that should occur in the next transaction of the chain: $\mathbb{A}$ should send an encrypted piece $K_{\mathbb{A}}[p_{\mathbb{A}}]$ to $\mathbb{B}$ designating itself as the payee; $\mathbb{B}$ should send an encrypted piece $K_{\mathbb{B}}[p_{\mathbb{B}}]$ to $\mathbb{A}$, designating itself as the payee; and $\mathbb{A}$ should send the decryption key $K_{\mathbb{A}}$ to $\mathbb{B}$. Note that $\mathbb{A}$ does not need to verify that $\mathbb{B}$ has reciprocated its sending of $K_{\mathbb{A}}[p_{\mathbb{A}}]$, as $\mathbb{A}$ is the recipient of $\mathbb{B}$'s reciprocation.

We now show that $\mathbb{A}$ and $\mathbb{B}$ risk retaliation from the other if they do not complete each step. If $\mathbb{A}$ does not send the encrypted piece $K_{\mathbb{A}}[p_{\mathbb{A}}]$ to $\mathbb{B}$, it cannot initiate the next transaction in which it receives $p_{\mathbb{B}}$ from $\mathbb{B}$. If $\mathbb{B}$ does not reciprocate $\mathbb{A}$'s sending of $K_{\mathbb{A}}[p_{\mathbb{A}}]$, then $\mathbb{A}$ can withhold $K_{\mathbb{A}}$ from $\mathbb{B}$, preventing $\mathbb{B}$ from accessing $p_{\mathbb{A}}$. Finally, if $\mathbb{A}$ does not send its key $K_{\mathbb{A}}$ to $\mathbb{B}$, then $\mathbb{B}$ can withhold its key $K_{\mathbb{B}}$ in the next transaction, preventing $\mathbb{A}$ from accessing $p_{\mathbb{B}}$. □

Thus, if both $\mathbb{A}$ and $\mathbb{B}$ value gaining a piece above the cost of sending encrypted pieces and decryption keys to each other, as well as encrypting and decrypting the pieces, they have an incentive to participate in the chain.[7] We show in Section III-C that the costs of encryption, decryption, and sending decryption keys are negligible compared to the cost of uploading an encrypted piece; and if this uploading cost exceeds $\mathbb{A}$'s or $\mathbb{B}$'s utility in receiving an additional piece, they would not participate in the swarm in the first place.

From the proof of Proposition II.1, a non-seeder $\mathbb{A}$ even has an incentive to initiate a chain instead of simply continuing an existing one, since $\mathbb{A}$ anticipates receiving a piece from $\mathbb{B}$. This chain initiation is called opportunistic seeding and is discussed in Section II-D3. If $\mathbb{A}$ is a seeder, it needs no incentive to start a chain: the existence of the swarm presupposes that the seeder's objective is to altruistically disseminate the file to leechers.

Incentive compatibility for non-seeders with indirect reciprocity (i.e., $\mathbb{B}$ reciprocates to a leecher $\mathbb{C} \neq \mathbb{A}$) is less clear. If $\mathbb{A}$ is continuing a chain, then $\mathbb{A}$ has an incentive to upload an encrypted piece to $\mathbb{B}$, in order to receive a piece in the previous transaction. We argue that $\mathbb{A}$ would also have an incentive to initiate a chain, as doing so reduces its potential competition with $\mathbb{C}$ in future transactions, benefitting $\mathbb{A}$ through an *offloading effect* [21]. $\mathbb{A}$ may also become a payee in later stages of the chain, acquiring pieces faster than if the chain had not existed. This type of chain initiation, however, is relatively rare (c.f. Section II-D3).

The remainder of our proof for the direct reciprocity case works as before except for the last step, in which $\mathbb{C}$ acknowledges receipt of an encrypted piece from $\mathbb{B}$ and $\mathbb{A}$ sends the decryption key to $\mathbb{B}$. $\mathbb{C}$ has an incentive to acknowledge receipt; otherwise, $\mathbb{B}$ will retaliate by not sending the decryption

[7] Of course, $\mathbb{A}$ and $\mathbb{B}$ may not retaliate against each other. However, $\mathbb{A}$ and $\mathbb{B}$ are unlikely to have a previous collusion arrangement (Section III-A4) and thus could not ensure the other's non-retaliation.

key to $\mathbb{C}$.[8] While $\mathbb{B}$ cannot retaliate against $\mathbb{A}$ for failing to send the decryption key, $\mathbb{B}$ may be able to retaliate in the future. Since the cost of sending $K_{\mathbb{A}}$ to $\mathbb{B}$ is negligible (cf. Section III-C), unless $\mathbb{A}$ judges the probability of needing a piece from $\mathbb{B}$ in the future to be similarly negligible, e.g., if $\mathbb{A}$ already has nearly all of the file pieces, $\mathbb{A}$ would prefer to send $K_{\mathbb{A}}$ to $\mathbb{B}$ rather than risk future retaliation.

The above mentioned incentives apply to all participants in each transaction of a chain. Since all participants have an incentive to follow the given protocol or risk retaliation by others that deprives them of receiving pieces, we claim that T-Chain is incentive-compatible.

D. 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, as 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 [11]).

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.

[8]If $\mathbb{B}$ and $\mathbb{C}$ collude, $\mathbb{B}$ may simply upload an unencrypted piece to $\mathbb{C}$, saving $\mathbb{C}$ the cost of reciprocation and $\mathbb{B}$ the cost of encrypting this piece. However, $\mathbb{B}$ could not then retaliate if $\mathbb{C}$ does not report $\mathbb{B}$'s reciprocation to $\mathbb{A}$. $\mathbb{B}$ is unlikely to trust $\mathbb{C}$ enough to give up this leverage unless they are in a pre-determined group of colluders, which is unlikely (cf. Section III-A4), especially since $\mathbb{B}$'s cost of encryption is negligible (Section III-C).

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 with 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, it will accumulate pending file pieces and eventually 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 a moderate value of 2, which balances smoothing out variations between leechers with over- or underloading some leechers. Since flow control is not a core component of T-Chain, we fix $k = 2$ in order to study other dimensions of T-Chain's performance.

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 remaining chains may not utilize leechers' full upload capacity, degrading system performance. Yet the seeder may not be able to keep up with the rate of chain termination.

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-G.

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, 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 [22]. 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-B3): 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-B3, free-riders cannot easily cause or exploit chain terminations.

2) Cheating: Cheating, or initiating transactions with other leechers and later refusing to reciprocally upload to those leechers, can easily occur in BitTorrent [6]. 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 [23], [24]) or by frequently changing their identities (i.e., engaging in whitewashing [13], [25]). 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 [26]. 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 the probability of this occurring is very small, unless the colluder (or Sybil) set is very large, which is difficult to achieve.

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

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

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-D experimentally investigates the effect of many leechers colluding with each other; the colluders receive very slow download times, making collusion impractical.

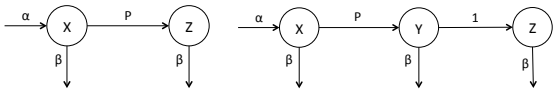
5) Failure to Complete the Exchange: In Figure 1(b), the participant $\mathbb{B}$, having previously uploaded an encrypted file piece to $\mathbb{C}$, may later fail to upload the decryption key to $\mathbb{C}$. There is minimal gain for $\mathbb{B}$ in failing to upload the key, however, since uploading 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 to that of a BitTorrent-like protocol in which each peer unchokes a random peer every $1/\delta$ timeslots (normally $1/\delta = 5$, i.e., 20% of bandwidth is used for BitTorrent's optimistic unchoking). We consider a discrete-time system with $t = 0, 1, \ldots$ 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 for the donor's upload to the requestor, and one for the requestor's to the payee. We do not consider the time to send file piece receipts and decryption keys, since they are much smaller than the file pieces (c.f. Section III-C).

We define three 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 a single encrypted file piece that has not been 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 define $z(t) = n(t) - x(t) - y(t)$.

We define β as the peers' departure rate and α as the newcomer arrival rate (arrivals and departures are assumed to follow Poisson distributions), 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



(a) BitTorrent-like model. (b) T-Chain model.

Fig. 2: Transition diagrams for the two protocols.

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 a transaction's designated payee; we suppose this happens in a fraction $1-\omega$ of transactions.

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, $y(t) \equiv 0$: no peers are partially bootstrapped. We now calculate the probability that a 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:

$$\mathbb{E}\left[x(t+1)|x(t), n(t)\right] = x(t)\,(1-P) + \alpha n(t). \tag{1}$$

Moreover, we find that $\mathbb{E}\left[n(t+1)|n(t)\right] = (1-\beta+\alpha)n(t)$, allowing us to solve for $\mathbb{E}\left[n(t)|n(0)\right] = (1-\beta+\alpha)^t\, n(0)$. Thus, if $\beta = \alpha$, i.e., the arrival and departure rates are the same, then the expected number of peers in the swarm remains constant.

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. Thus, we generally have $\omega'' \leq \omega'$, which we will assume throughout the rest of the paper. 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):

$$\mathbb{E}\left[x(t+1)|x(t), y(t), z(t)\right] = \alpha n(t) + x(t)(1-\beta)\,(1-P) \tag{5}$$

$$\mathbb{E}\left[y(t+1)|x(t), y(t), z(t)\right] = x(t)(1-\beta)P \tag{6}$$

where 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: We now focus on the case $\alpha = \beta = 0$, i.e., peers neither depart nor enter the system, to examine the bootstrapping rates of both methods. To facilitate comparisons between T-Chain and BitTorrent, we use x_t to denote the number of un-bootstrapped T-Chain peers, and x_b to denote the number of un-bootstrapped BitTorrent peers. We also define the *bootstrapping rate* at a given time t as the ratio of the expected number of un-bootstrapped peers at time $t+1$, given the number at time t, to the number of un-bootstrapped peers at time t: $\frac{\mathbb{E}[x(t+1)|x(t)]}{x(t)}$. We first consider a flash-crowd scenario, in which several newcomers have recently arrived, resulting in a large number of un-bootstrapped peers. All proofs are in the Appendix. We derive sufficient conditions for T-Chain to bootstrap newcomers faster than BitTorrent in the first few timeslots:

Proposition III.1 (Short-term bootstrapping rates)**.** *Suppose that $\alpha = \beta = 0$. If $n - x_b(t) \ll n$ for BitTorrent and $z_t(t-1) \ll n$ for T-Chain, then T-Chain has a higher bootstrapping rate than BitTorrent at time t if*

$$Kz_t(t-1)\left(\frac{x_t(t-1) + \omega' y_t(t-1) + \omega''(z_t(t-1)-1)}{n-1}\right) \geq \delta(n - x_b(t)). \tag{7}$$

Intuitively, at time t, $Kz_t(t-1)\omega$ 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 bootstrap peers at a faster rate 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 $K = 2$. Note that this condition is weaker than that given in Proposition III.2, which assumes most peers are bootstrapped. When there are many bootstrapped peers in the swarm, T-Chain's number of un-bootstrapped peers falls slower than when few peers are already bootstrapped.

Over time, more peers are bootstrapped and the assumption $z_t(t-1) \ll n$ in Proposition III.1 will not hold. Thus, we next examine the bootstrapping rate when many peers are already bootstrapped. Once again, we can derive sufficient conditions for which T-Chain is faster than BitTorrent:

Proposition III.2 (Long-term bootstrapping rates)**.** *Suppose that $\alpha = \beta = 0$ and that $x_t(t) + y_t(t) \leq \mu n$, $x_b(t) \geq \nu n$. Then T-Chain's bootstrapping rate at time t is faster than BitTorrent's if*

$$\left(1 - \frac{\delta}{n-1}\right)^{n(1-\nu)} \geq \left(1 - \frac{1}{n-1}\right)^{Kn(1-\mu)\omega''}. \quad (8)$$

In the limit for large n, this condition becomes $\delta(1-\nu) \leq K\omega''(1-\mu)$. From Proposition III.1's result, we can generally take $\nu > \mu$ since $x_b(t) \geq x_t(t) + y_t(t)$ (T-Chain bootstraps peers faster than BitTorrent shortly after many of them arrive), implying that $K\omega'' > \delta$ is a sufficient condition to ensure (8).

Our analysis assumes that all leechers are neighbors, in order to focus on comparing BitTorrent and T-Chain. We do not expect this assumption to have a significant impact on our results: in fact, limiting the number of each peer's neighbors would have less effect on T-Chain than on BitTorrent, since T-Chain peers can overcome piece availability limitations among their neighbors with indirect reciprocity. In Section IV's evaluation, leechers are connected to at most 55 of their neighbors, and Propositions III.1's and III.2's results on T-Chain's bootstrapping faster than BitTorrent still hold.

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. [14] 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 8Mbps. 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 [6] 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

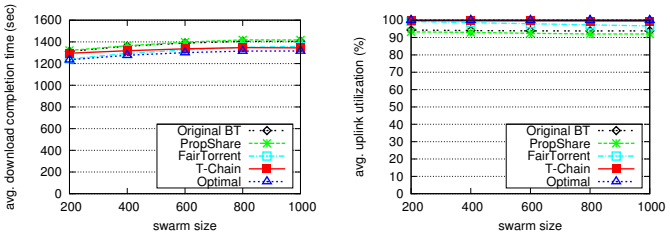
Each experimental run started with a swarm consisting of a single seeder. This seeder remained in the swarm throughout the simulation run. A leecher joining the swarm was assumed to begin downloading file pieces, remain in the swarm until its download of the file completed, and exit the swarm immediately 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 [27], representing a demanding test case for file sharing. We later use a real trace arrival model taken from the RedHat 9 Torrent tracker trace [28], which represents 5 months of activity in a BitTorrent swarm. Each leecher requests a list of 50 randomly selected neighbors from the tracker upon arrival, and whenever its list of neighbors falls below 30. Leechers maintain at most 55 neighbors throughout their time in the swarm.

We compared the performance of four protocols: original BitTorrent [9], PropShare [11], FairTorrent [12], 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 6000 Kbps. The upload bandwidth of leechers was assumed to be heterogeneous, varying from 400 Kbps to 1200 Kbps, in accordance with the assumptions of [6], [22], [27]. There was no limit on the download bandwidth of leechers; upload bandwidth was assumed to be the limiting factor or resource [29]. 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 [12].

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. Performance without 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.

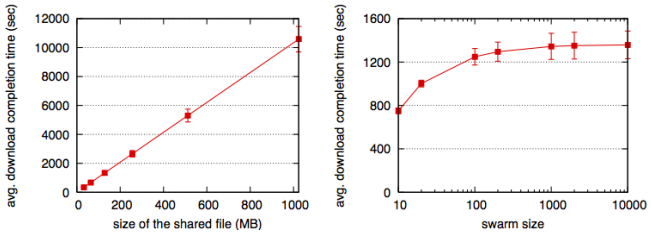


(a) Avg. download completion time. (b) Avg. uplink utilization.

Fig. 3: (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.

Figure 3 shows the results without free-riders. Figure 3(a) shows that all methods perform similarly (in terms of average download completion time) and close to optimal (cf. [27]). 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 3(b). This improvement comes from the fact that T-Chain and FairTorrent 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.

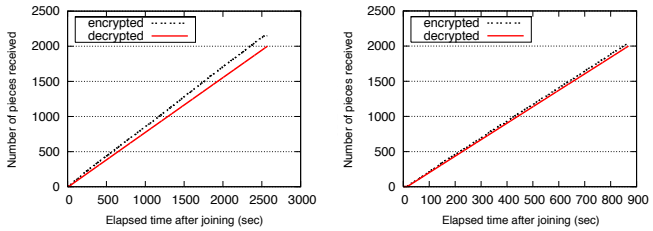
The next two experiments investigate the effects of the file size and the swarm size on system performance in T-Chain.



(a) File size effects. (b) Swarm size effects.

Fig. 4: The effects of (a) file size and (b) swarm size under T-Chain.

Figure 4(a) shows the effects of the size of the file shared in the system. In this experiment, the number of compliant leechers joining the system was fixed at 600 (without free-riders), but the size of the shared file varied from 32MB to 1024MB. The average download completion time increases linearly with the file size, indicating that T-Chain performs well even when large files are shared. Figure 4(b) shows the effects of the swarm size. In this experiment, the file size was fixed at 128MB, but the number of compliant leechers joining the system varied from 10 to 10,000. No free-riders are considered in this experiment. As seen in the figure, T-Chain is highly scalable in that the average download completion time converges and stays nearly constant regardless of the number of leechers joining the system in a flash crowd fashion. The average download completion time is low with fewer than 200 leechers because the seeder's upload bandwidth dominates the system performance when there are few leechers in the system. The upload bandwidth of the seeder was set to 6000Kbps. T-Chain thus has good average performance when there are no free-riders in the system.

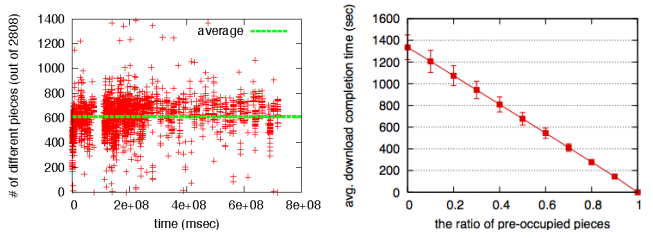


(a) 400Kbps leecher. (b) 1200Kbps leecher.

Fig. 5: Transfer times of individual file pieces for two specific leechers with the lowest and the highest upload rates.

To further understand T-Chain's effects for individual leechers, Figure 5 shows the transfer times of individual file pieces for two specific leechers with the highest and lowest upload rates. In each figure, one line (set of data points) represents the times at which each encrypted file piece was received by the leecher, and the other line represents the times at which the matching decryption key for each encrypted file piece was received. The upload of file pieces to the leecher proceeds at a very steady rate in each case, and the delay before a downloaded piece's decryption key becomes available is generally quite small.

The results for the 400Kbps leecher in Figure 5(a) are particularly instructive. In this figure, the slope of the downloaded file pieces' line is higher than the slope of the downloaded decryption keys' line. The slope of the first line is determined by the upload bandwidth of the leecher's neighbors, while the slope of the second line is determined by the upload bandwidth of the leecher itself. Since this leecher has a lower upload bandwidth than the average for the swarm, the second slope is lower. Eventually, the adaptive receiver selection rule will apply if the leecher buffers too many pending pieces, in which case the slope of the line for the file pieces will decrease.



(a) Number of different pieces. (b) Effect of initial piece differences.

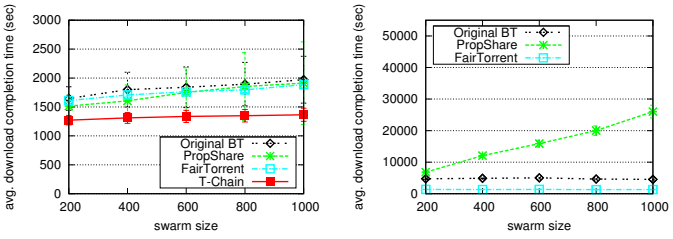
Fig. 6: (a) The number of different pieces between each pair of neighbors in a real BitTorrent swarm over 7 days and (b) the effect of initial piece differences among leechers.

T-Chain leverages leechers' needs for different file pieces in order to facilitate piece exchanges. If leechers do not require different pieces, the chain created by the seeder is hard to grow.

To investigate this effect, we measure the number of different pieces between pairs of leechers in a real BitTorrent swarm. We insert a crawler (i.e., our own leecher) into the swarm and measure the piece differences between all pairs of the crawler's neighbors at different times; new leechers became neighbors to the crawler throughout the swarm duration. Figure 6(a) shows the results. The average number of different pieces between two neighbors over the 7-day measurement period was 612 out of 2808 total pieces, indicating that leechers have an interest in growing chains with little restriction. To further verify these results, we additionally conduct an experiment in which 600 compliant leechers (without free-riders) join the system with a certain number of randomly selected pieces, ranging from 0% to 100% of the total pieces. Figure 6(b) shows the results. T-Chain's download completion times decrease linearly with the fraction of pre-occupied initial pieces, indicating that T-Chain by itself creates a strong incentive for leechers to exchange pieces, successfully growing chains.

C. Performance under Free-Riding

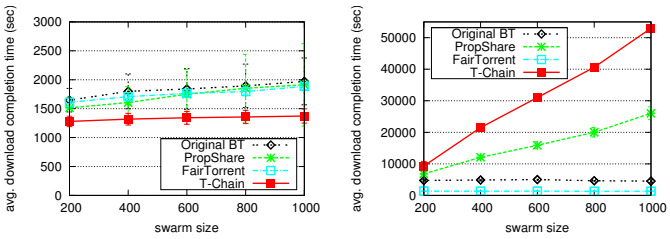
We next show the results of an experiment in which 25% of the leechers were free-riders. Each free-rider engaged in the worst possible behavior and provided zero upload bandwidth to other leechers. In addition, it was assumed that each free-rider attempted to avoid penalties for its behavior by using 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 [22]–[24]. 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.



(a) Compliant leechers. (b) Free-riders.

Fig. 7: 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).

Figure 7(a) indicates that the addition of 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 7(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 time) performance for the free-riders. Simple whitewashing thus enables free-riders in FairTorrent to finish their downloads as fast as compliant leechers. The figure contains no line for T-Chain, since not a single free-rider completed the download of the unencrypted file, even under these challenging conditions. Leechers in T-Chain can easily identify uncooperative neighbors through adaptive receiver selection (Section II-D2), minimizing the amount of system resources allocated to free-riders.



(a) Compliant leechers. (b) Free-riders.

Fig. 8: The effects of collusion in T-Chain under the flash crowd arrival model (adding collusion in T-Chain to Figure 7's setting).

D. 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 IV-C) but do exist. We investigate collusion under the same experimental settings with free-riders as in Section IV-C. We assumed that *all* free-riders in T-Chain colluded, sending false reception reports on behalf of other colluding free-riders.

Figure 8 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 8(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 time for T-Chain's compliant leechers (compare Figures 7(a) and 8(a)).

E. Real Swarm Performance

We next consider conditions more gradual than flash crowds by examining the system performance when arrivals mirror the behavior of leechers in a single BitTorrent swarm that downloaded the RedHat 9 release [28]. 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.

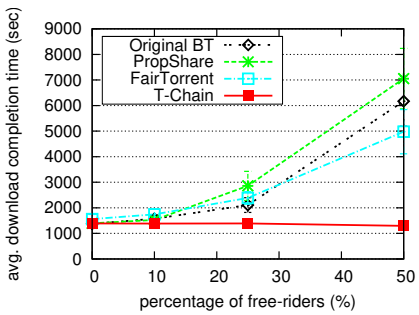
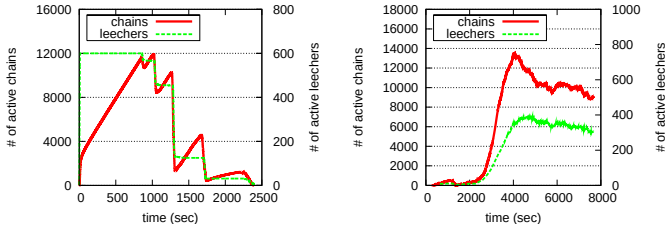


Fig. 9: The average download completion times for compliant leechers in BitTorrent, PropShare, FairTorrent and T-Chain under a continuous stream model.

Figure 9 demonstrates the average download completion time 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-D2), and opportunistic seeding enables better utilization of the system resources (Section IV-G) than BitTorrent's and PropShare's fixed resource allocations for newcomer bootstrapping. Simple whitewashing severely deteriorates FairTorrent's performance for compliant leechers.

F. Chain Characteristics

T-Chain's performance is strongly related to the growth of chains in the system: to saturate leechers' upload bandwidth, they must participate in a sufficient number of chains. To investigate this effect, we next examine the number of active chains and length of each chain formed.

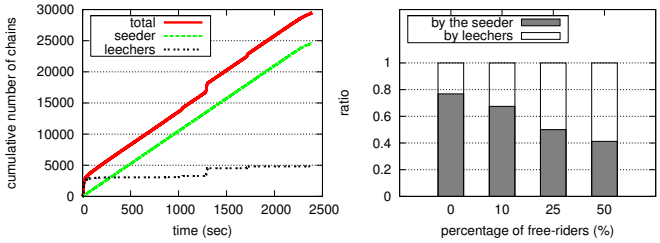


(a) Under a flash crowd. (b) Under a real trace.

Fig. 10: The number of active chains as a function of time in (a) a flash crowd and (b) a continuous stream model without free-riders.

We first consider the number of active chains in flash crowd and continuous stream models. In the flash crowd model, 600 leechers with heterogeneous upload bandwidths joined the system within 10 seconds, and no free-riders were present. In the continuous stream model, leechers with heterogenous upload bandwidths continuously joined the system, mimicking the arrivals of leechers in a real P2P system.

Figure 10(a) shows the results in a flash crowd model. As seen in the figure, the number of active chains in this model keeps increasing until the leechers with the highest upload bandwidth finish their downloads, after around 830 seconds.



(a) Under a flash crowd. (b) Under a real trace.

Fig. 11: (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.

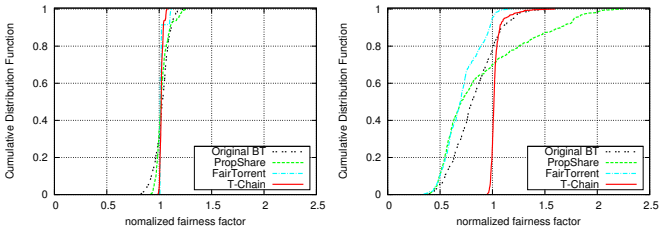
The saw-toothed decrease follows as each bandwidth group of leechers leaves the system, indicating that chain termination is strongly related to leecher departure. Figure 10(b) shows the number of active chains in a continuous stream model without free-riders. The number of chains sharply increases as the system grows and stabilizes, in conjunction with the number of active leechers in the system. The number of chains is dynamically adjusted to the number of active leechers in the system as leechers join or leave the swarm.

G. Opportunistic Seeding

As discussed previously in Section II-D3, 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). This happens when the system is just initiated with many newcomers (so the seeder cannot afford all of them by itself) or when many existing chains terminate at the same time (e.g., many leechers leave the system). In such situations, leechers initiate new chains to compensate the under-utilization of their upload bandwidth. We conducted two experiments to investigate the frequency of opportunistic seeding in T-Chain.

Figure 11(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 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 11(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.



(a) No free-riders. (b) 25% free-riders.

Fig. 12: Fairness enforced by each method.

H. Fairness

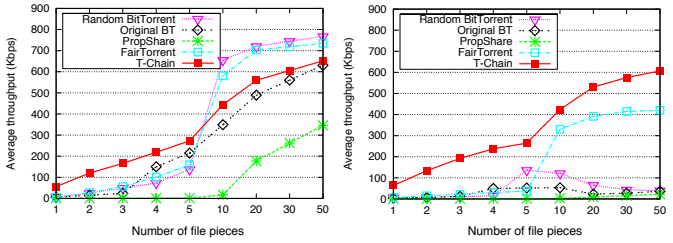
We next evaluate T-Chain's performance in terms of *fairness*, or the ratio of leechers' benefits (i.e., received piece downloads) to their contributions (piece uploads). In a fair system, this ratio would be 1: leechers would benefit according to their contributions, encouraging participation in a cooperative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

We use the same experimental conditions as in Figure 9 and show the resulting fairness in Figures 12(a) and 12(b). These figures plot the Cumulative Distribution Function (CDF) of the fairness factors of the last 500 compliant leechers in each system. Figure 12(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 12(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.

I. 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. 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 13(a) and 13(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 13(a), if the shared file has relatively few pieces (e.g., below 5), the average throughputs of BitTorrent, Random BitTorrent, PropShare, and



(a) No free-riders. (b) 50% free-riders.

Fig. 13: 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.

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-H). 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 13(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.

Direct reciprocity [9]–[12] 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 [22], PropShare [11], and FairTorrent [12] 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

TABLE II: Comparison of major incentives under possible attacks ($\surd$: Good, (blank) : Medium, $\times$: Bad)

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

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 [30]. FairTorrent [12] 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-C – IV-I). In addition, FairTorrent cannot prevent seeders from being exploited by free-riders.

There also exist variants of direct reciprocity. Give-to-Get [10] 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 [31] 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 [13]–[17], [32]) 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 [13] 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 [15], PledgeRoute [16], and Dandelion [14] 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 [6], [11], [14], [18] 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 [14] 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 [11] 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 [6] is not applicable to streaming applications and is also vulnerable to the key disclosure problem. The hierarchical chunk cipher scheme [18] 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 [33]. 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 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.

We evaluate T-Chain with extensive simulations and compare its performance 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 more fair 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

This work was partly supported by NSF grant CNS-1525435. K.Shin gratefully acknowledges partial support from the 2013 Korea Military Academy Hwarangdae Research Institute. Y.Yi was supported by the Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.B0717-17-0034,Versatile Network System Architecture for Multi-dimensional Diversity) and the Korea government (MSIP) (No. 2016R1A2A2A05921755).

REFERENCES

[1] W. Gao, G. Cao, A. Iyengar, and M. Srivatsa, "Cooperative caching for efficient data access in disruption tolerant networks," *Trans. Mob. Comp.*, vol. 13, no. 3, pp. 611–625, 2014.

[2] J. Berry, M. Collins, A. Kearns, C. A. Phillips, J. Saia, and R. Smith, "Cooperative computing for autonomous data centers," in *IEEE IPDPS*. IEEE, 2015, pp. 38–47.

[3] E. Cerqueira, E. Lee, J.-T. Weng, J.-H. Lim, J. Joy, and M. Gerla, "Recent advances and challenges in human-centric multimedia mobile cloud computing," in *IEEE ICNC*. IEEE, 2014, pp. 242–246.

[4] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 44, no. 5, pp. 27–32, 2014.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

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

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

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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

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

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

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

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

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

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

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

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

[31] 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.

[32] U. E. Tahta, S. Sen, and A. B. Can, "Gentrust: A genetic trust management model for peer-to-peer systems," *Appl. Soft Comp.*, vol. 34, pp. 693–704, 2015.

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

APPENDIX A
PROOFS

Proposition III.1.

Proof: When $\alpha = \beta = 0$, the dynamics for BitTorrent and T-Chain respectively may be written as

$$\mathbb{E}_t\left[x_b(t+1)\right] = x_b(t)\left(1-\frac{1}{n}\right)\left(1-\frac{\delta}{n-1}\right)^{n-x_b(t)} \quad (9)$$

$$\mathbb{E}_t\left[x_t(t+1)\right] = x_t(t)\left(1-\frac{1}{n}\right)\left(1-\frac{1}{n-1}\right)^{Kq(t)}, \quad (10)$$

where

$$q(t) = z_t(t-1)\left(\frac{x_t(t-1) + \omega' y_t(t-1) + \omega''(z_t(t-1)-1)}{n-1}\right) \tag{11}$$

$\mathbb{E}_t\left[y_t(t+1)\right]$ is determined by $x_t(t-1)$, and the t subscripts on the expectations denote expectations taken given the number of bootstrapped and un-bootstrapped peers at times $\tau \leq t$. We consider the dynamics in (9–10) and use the approximation $(1-w)^m \approx 1-mw$ for m and w small to obtain the sufficient condition (7). If $n - x_t(t-1) - y_t(t-1) = n - x_b(t)$, we have the sufficient condition

$$\frac{x_t(t-1) + \omega' y_t(t-1) + \omega''(n - x_t(t-1) - y_t(t-1) - 1)}{n-1}$$
$$\geq \omega'' \geq \frac{\delta}{K}.$$

■

Proposition III.2.

Proof: From (9) and (10) in the proof of Proposition III.1, we see that when $\alpha = \beta = 0$, T-Chain has a higher bootstrapping rate than BitTorrent if

$$\left(1 - \frac{\delta}{n-1}\right)^{n-x_b(t)} \geq \left(1 - \frac{1}{n-1}\right)^{Kq(t)}, \tag{12}$$

where $q(t) \geq \omega''(n - x_t(t-1) - y_t(t-1)$ is given by (11) and we assume that $\omega'' \leq \omega'$. Thus, we derive (8). ■

Kyuyong Shin Kyuyong Shin is a professor in the Department of Computer Science at Korea Military Academy in Seoul, South Korea since 2009. He received his Ph.D. degree in the Department of Computer Science at North Carolina State University in 2009. His research interests lie in security issues in cooperatively managed distributed systems, which may include security issues in Peer-to-Peer (P2P), Cloud Computing, Grid Computing, etc.

Carlee Joe-Wong Carlee Joe-Wong (S'11, M'16) is an assistant professor in the ECE department at Carnegie Mellon University, working at CMU?s Silicon Valley Campus. She received her Ph.D. from Princeton University in 2016 and is primarily interested in incentives and resource allocation for computer and information networks. In 2013?2014, Carlee was the Director of Advanced Research at DataMi, a startup she co-founded from her data pricing research. She received the INFORMS ISS Design Science Award in 2014 and the Best Paper Award at IEEE INFOCOM 2012, and was a National Defense Science and Engineering Graduate Fellow (NDSEG) from 2011 to 2013.

Sangtae Ha Sangtae Ha (S?07, M?09, SM?12) is an Assistant Professor in the Department of Computer Science at the University of Colorado at Boulder. He received his Ph.D. in Computer Science from North Carolina State University. His research focuses on building and deploying practical systems. He is a co-founder and the founding CTO/VP Engineering of DataMi, a startup company on mobile networks, and is a technical consultant to a few startups. He is an IEEE Senior Member and serves as an Associate Editor for IEEE Internet of Things (IoT) Journal. He received the INFORMS ISS Design Science Award in 2014.

Yung Yi Yung Yi received his B.S. and the M.S. in the School of Computer Science and Engineering from Seoul National University, South Korea in 1997 and 1999, respectively, and his Ph.D. in the Department o Electrical and Computer Engineering at the University of Texas at Austin in 2006. From 2006 to 2008, he was a post-doctoral research associate in the Department of Electrical Engineering at Princeton University. Now, he is an associate professor at the Department of Electrical Engineering at KAIST, South Korea. He received the best paper awards at IEEE SECON 2013, ACM MOBIHOC 2013, and IEEE William R. Bennett Award 2016.

Injong Rhee Injong Rhee is the CTO and Head of R&D, Software and Services, Samsung Mobile. He was previously a Professor of Computer Science at North Carolina State University. He works primarily on network protocols for the Internet. His major contributions in the field include the development of congestion control protocols, called BIC and CUBIC. Since 2004, these protocols have been the default TCP algorithms for Linux and are currently being used by more than 40% of Internet servers around the world and by several tens millions Linux users for daily Internet communication. He also has invented several multimedia streaming and multicast technologies licensed to companies for commercial applications. He started a company based on these technologies in 2000 where he developed and launched the world's first video streaming products and push-to-talk (PTT) VoIP products for cell phones. He received NSF Career Award in 1999 and NCSU New Inventor's award in 2000.

Douglas Reeves Douglas S. Reeves is Professor of Computer Science and Associate Dean of Engineering at N.C. State University. His research interests are network and distributed systems security.