

Treat-Before-Trick : Free-riding Prevention for BitTorrent-like Peer-to-Peer Networks

Kyuyong Shin, Douglas S. Reeves, Injong Rhee

Department of Computer Science

North Carolina State University

Raleigh, NC 27695, USA

{kshin2, reeves, rhee}@ncsu.edu

Abstract

In P2P file sharing systems, free-riders who use others' resources without sharing their own cause system-wide performance degradation. Existing techniques to counter free-riders are either complex (and thus not widely deployed), or easy to bypass (and therefore not effective). This paper proposes a simple yet highly effective free-rider prevention scheme using (t, n) threshold secret sharing. A peer must upload encrypted file pieces to obtain the subkeys necessary to decrypt a file which has been downloaded, i.e., subkeys are swapped for file pieces. No centralized monitoring or control are required. This scheme is called "treat-before-trick" (TBeT). TBeT penalizes free-riding with increased file completion times (time to download file and necessary subkeys). TBeT counters known free-riding strategies, incentivizes peers to donate more upload bandwidth, and increases the overall system capacity for compliant peers. TBeT has been implemented as an extension to BitTorrent, and results of experimental evaluation are presented.

1 Introduction

Peer-to-Peer (P2P) systems are increasingly popular as an alternative to the traditional client-server model for many applications, including file sharing and streaming. In a P2P system, peers directly collaborate with one another to reduce download time, by sharing resources such as upload bandwidth [1, 2, 3]. Unlike the client-server model, upload bandwidth potentially scales as the number of peers in a P2P system increases. Despite the importance of fair cooperation among peers (for scalability), without incentives for sharing, many users tend not to share their resources. Peers that benefit from P2P file-sharing, but who do not contribute resources to the overall system capacity, are termed *free-riders* [4, 5, 6, 7, 8]. As the fraction of free-riders increases, the overall system capacity per peer decreases.

Much research has been done to prevent free-riders in P2P systems [6, 8, 9, 10, 11]. No technique to date has been notably successful in preventing or discouraging free-riders [12, 13]. Reasons include the complexity of imple-

mentation, the performance penalty imposed on *compliant* users (i.e., non-free-riders), and, particularly, the ease of bypassing (without penalty) the existing mechanisms.

In this paper, we consider the free-riding problem in a major P2P file-sharing system, *BitTorrent*. We propose a unique scheme to penalize free-riding and reward compliant behavior. This scheme makes use of the secret sharing, or (t, n) threshold, algorithm [14]. Files are divided and encrypted by the owner, or *seeder*. The key used for encryption is divided into n subkeys, any t of which are sufficient to recover the key and decrypt the file pieces. The file owner uploads both the file pieces and subkeys to a set of requesting peers, called *leechers*. Leechers barter with each other by exchanging subkeys for file pieces (and vice versa); that is, a peer must upload an intact encrypted file piece *before* receiving a subkey, which is termed the "data first, key later" (DFKL) rule. We call this scheme "*treat-before-trick*", or *TBeT*. TBeT has several benefits:

- It penalizes free-riders. Free-riders who do not contribute upload bandwidth to the system will be able to obtain subkeys only from seeders. This effectively increases the download completion times for free-riders. This incentive mechanism should reduce the fraction of free-riders, so that system capacity scales with the number of peers (resulting in decreased file download times).
- Existing methods (e.g. cheating, whitewashing [15], Sybil attacks [16], and large-view-exploits [4, 5, 17]) for bypassing free-riding restrictions are not effective with TBeT.
- It incentivizes peers to donate more upload bandwidth. TBeT allows peers donating more bandwidth to complete their file downloads faster. This incentive encourages peers to donate more resources for their own good, which eventually leads to increasing the overall system capacity.
- It is easy to implement, completely distributed, and scalable. TBeT does not require trusted third parties, centralized servers, use of currency or tokens, accounting,

or computation of reputation. It requires only simple algorithms for encryption and decryption, and secret sharing for subkey generation and key restoration. In TBeT, peers barter subkeys for data pieces independently and directly among themselves, with no requirement for arbitration by any other party, including the seeder.

The computational overhead of file encryption and decryption, as well as subkey generation and key restoration, is shown to be more than compensated for by the reduced file download completion times experienced by compliant peers.

The remainder of this paper is organized as follows. Section 2 briefly describes and analyzes existing methods of free-rider prevention. Section 3 summarizes the operation of BitTorrent, existing free-riding strategies, and (t, n) threshold secret sharing, which are all the basis for TBeT. Section 4 details our method, and section 5 presents the results of simulating this method. Section 6 discusses limitations of the proposed method and possible solutions. Finally section 7 concludes the paper.

2 Related Work

A number of studies have shown that free-riding is a problem in current P2P systems, with resulting serious performance degradations. For instance, Hughes et al. [12] investigates free-riding in Gnutella and find that 85% of peers do not share any files. So, prevention of free-riding in P2P has been recognized as an important goal for the continued growth of P2P systems. Varying incentive schemes have been proposed recently to encourage peers to cooperate by sharing their resources. These schemes may be classified as monetary-based [10], reciprocity-based [1, 18, 15, 6], and reputation (or credit)-based [9], which is well summarized by Feldman et al. [8].

Among these incentive schemes, only the reciprocity-based scheme is widely deployed in current P2P systems, primarily because of its simplicity. BitTorrent currently uses a reciprocity-based scheme, based on the well-known strategy of *tit-for-tat* [1, 4], or TFT. This simple, fairness-enforcing mechanism of BitTorrent is shown to achieve a Nash equilibrium under certain conditions [19], and is widely believed to strongly discourage free-riding [17].

Recent work [4, 5, 17, 7], however, shows that this mechanism is ineffective in discouraging free-riders. There exist multiple ways (including cheating, whitewashing, sybil attacks, and the “large-view-exploit”) for free-riders to circumvent the mechanism. As a result, free-riders receive similar (or, in some cases, even better) download speeds to compliant peers. There has not yet been an effective solution for such circumventions of the BitTorrent TFT.

Recently, Locher et al. [11] proposed a scheme to protect peers from free-riding *seeders*, based on source coding. The method, however, still relies on use of TFT to prevent

free-riding between leechers, which is a much more serious problem in BitTorrent. TBeT, in contrast, effectively discourages free-riding between leechers. We believe that their source coding technique is complementary to TBeT.

3 Background

3.1 BitTorrent Overview

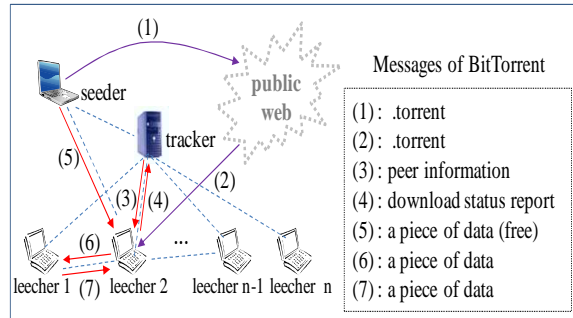


Figure 1: BitTorrent overview

Our scheme is based on a popular P2P file sharing protocol called *BitTorrent* [20, 1]. As seen in Figure 1, it is composed of three parties: a group of *seeders*, a *tracker*, and a much larger number of *leechers*. A seeder is a peer who wants to contribute or publish a large file via BitTorrent. The seeder segments the file into *file pieces* of fixed size (typically 250KB). The seeder also creates a *.torrent* file containing file meta-data, and posts it on a public web-site (1). Peers called leechers interested in downloading the file can find the *.torrent* file on the web-site (2). Once a leecher finishes downloading a file, it may become an *inherited seeder*, uploading the downloaded files for free to other leechers. The tracker maintains a list of all peers currently downloading the file, or uploading the file to other peers, and keeps track of the locations of file pieces based on the download status report from each peer (4). The seeders, leechers, and tracker of a file constitute a BitTorrent *swarm*. Each peer interested in a file will initially contact the tracker to obtain a list of up to 50 randomly-selected peers (including seeders) currently in the swarm of the file (3). The newcomer attempts to establish a TCP connection to each peer in that set; if the connection is accepted by the peer, they become *neighbors*. Then the newcomer can download file pieces from the seeders in the swarm (5), or exchange file pieces with other leechers (6 and 7). When downloading pieces, a peer will download first the piece which is rarest (fewest copies exist) among the pieces held by its neighbors, which is termed the *Local Rarest First* (LRF) policy. Each leecher periodically informs the tracker of the pieces it has downloaded (4). When the number of neighbors of a peer drops below 30, the peer contacts the tracker again to obtain a fresh list of new candidate neighbors (3).

Each peer individually attempts to maximize its aggregate download rate by downloading pieces as quickly as possible. Initially, all neighbor TCP connections are *choked*, meaning no data is sent to them. *Unchoking* a neighbor, or sending it file pieces, is controlled by rate-based *tit-for-tat* (TFT). That is, the peer unchokes those 4 neighbors who have given the best upload rate to it. Selection of which neighbors to choke/unchoke (called *rechoking*) occurs at an interval which is on the order of every 10 seconds. In addition, each peer randomly selects one additional neighbor for unchoking every 30 seconds. This process, called *optimistic unchoking*, allows new peers into the system, and helps to find peers with better download speeds.

3.2 Existing Free-riding Strategies

BitTorrent’s TFT policy is designed to penalize free-riding. There are several strategies for avoiding such penalties, including the large-view-exploit, cheating, and white-washing (or the Sybil attack).

It has been shown in [19] that under TFT, free-riders should receive about 20% of the upload bandwidth of non-free-riding leechers. This assumes the neighborhood, or *view size* of a peer does not change. Unfortunately, there is no mechanism in BitTorrent to prevent a free-rider from establishing a much larger view, by making repeated requests to the tracker for new peer IDs. Each new neighbor will optimistically unchoke the free-rider with some probability; the free-rider, as a result, will receive download speeds equal to or better than that of compliant peers.

Another exploit is due to the sequence of events in TFT. A peer wishing to download must first upload files to a neighbor, with a promise from the neighbor to upload its file pieces in return. Free-riders may trivially *cheat* the peer by refusing to upload file pieces after downloading file pieces from the peer.

TFT does eventually penalize free-riders. In the long term, free-riders are choked off by the cheated peers if IDs of the free-riders are remembered [4]. This penalty assumes that past behavior can be uniquely attributed to long-lasting peer identities. In distributed systems, however, it can be quite easy for users to change their identities, termed *white-washing* [16], or to create multiple simultaneous identities, termed the *sybil attack* [15]. In BitTorrent, for example, users can easily change their peer IDs with little effort. This allows free-riders to escape from the consequences of their past actions and to cheat other leechers again and again, without being choked off.

3.3 (t, n) threshold secret sharing

Our protocol is based on *secret sharing* [14]. This refers to a scheme for distributing shares of a secret among many users, so that the secret can only be reconstructed by combining a minimum number of shares.

The (t, n) threshold secret sharing divides a secret into n

shares, such that the original secret is easily reconstructed from *any* t shares among n . We use Shamir’s (t, n) threshold scheme [14], which is efficient and simple to implement, based on the operation of polynomial interpolation. There is a unique degree- $t - 1$ polynomial $y(x) = a_0 + a_1x + a_2x^2 + \dots + a_{t-1}x^{t-1}$ which interpolates a set of t distinct points. For purposes of secret sharing, a_0 can serve as the secret; any t points lying on this polynomial are sufficient to compute a_0 . To use the (t, n) threshold scheme, we need to (1) pick a random $t - 1$ degree polynomial in which a_0 is the secret, and (2) evaluate $n \geq t$ points from $P_1 = y(1)$ to $P_n = y(n)$; n point values will be the shares.

For example, suppose that a seeder wants to use $(3, 5)$ threshold secret sharing. It randomly selects a polynomial with degree 2. If $y(x) = 10 + 2x + 3x^2$ is the polynomial, 10 is the secret. Next, it needs to pick a random prime number greater than 10; assume the prime number in this example is 11. The seeder can then generate the shares by polynomial interpolation and modulo arithmetic, with resulting shares $(1, 4)$, $(2, 4)$, $(3, 10)$, $(4, 0)$, and $(5, 7)$, i.e., $(1, 4)$ because $y(1) = 10 + 2 * 1 + 3 * 1^2 = 15 \% 11 = 4$, and so forth. Constructing the secret from the shares is interpolation, using the same modulus; details may be found in [14].

4 Treat-Before-Trick (TBeT)

In this section, we present a new scheme for penalizing free-riders. It is assumed that peers are selfish, and are neither aware of nor concerned about aggregate system performance. The scheme is based on *secret sharing*, which is explained in 3.3. We first present the basic protocol description of the scheme and discuss some optimization strategies for the basic scheme. The analysis of the optimal threshold value and its anti-free-riding nature will follow.

4.1 Basic Protocol Description

In this section, we describe the operation of TBeT.

The steps of TBeT are the same as those of BitTorrent, with the exception of key management. Peers are still assumed to use TFT and optimistic unchoking when determining how much resources to share. The seeder who first publishes or contributes a file to the system divides the file F into m pieces, and encrypts them using a symmetric key k which it generates. Let the i th encoded piece be denoted p_i . The seeder also generates n subkeys (k_1, k_2, \dots, k_n) , using (t, n) threshold secret sharing. We assume here that $t = m \leq n$. The seeder creates a .torrent file and posts it on a public web-site, as is usual in BitTorrent. This .torrent file also includes the SHA-1 hash value of each subkey; these are used to verify the authenticity of each received subkey. Upon request, the seeder distributes one file piece and one subkey to each requesting leecher. There is no dependence of the subkey on the piece; any subkey may be distributed with any file piece. The subkey and file piece to be downloaded by a leecher are determined by the nor-

mal LRF policy. The seeder must ensure that file pieces and subkeys are more or less uniformly distributed among the leechers. This ensures that leechers can profitably barter file pieces for subkeys (and vice versa) with each other. Initially, the seeder is the only peer who has the complete set of file pieces and subkeys, among all peers in the swarm.

A leecher needs to download the .torrent file from the public web-site to join the swarm. It contacts the tracker of the associated swarm to obtain a list of peers in that swarm, and contacts a subset of those peers to exchange file pieces and subkeys. Information about the subkeys and file pieces held by a neighbor are obtained directly from the neighbor. The leecher can obtain file pieces and subkeys from the seeder, or by bartering with other leechers.

While bartering, each peer uses the rate-based TFT and optimistic unchoking policies. In addition, neighboring peers exchange subkeys and file pieces according to the *data first, key later* (DFKL) rule. Under DFKL, a peer i wishing to obtain a subkey of F must first upload a file piece of F to peer j . Peer j then reciprocates by sending a subkey to peer i . Since both file piece hashes and subkey hashes are contained in the .torrent file, any peer can easily verify that it has received a valid piece, or a valid subkey. Therefore, an invalid file piece will not be rewarded with a subkey, and a neighbor who refuses to upload a subkey after receiving a valid piece may be choked off.

When a peer downloads all the file pieces of F , and at least t subkeys out of n , the peer has completed the file download. Then, the peer either remains in the system as an *inherited seeder* or leaves the system at its own discretion. Note that inherited seeders need not to re-encrypt the file pieces, nor do they re-generate the subkeys because they already have sufficient information (after their download completion).

4.2 Protocol Enhancements

We now discuss three techniques for improving the performance of the basic TBeT protocol. These are the one-time cheat, the use of semi-seeders, and the neighbor pruning.

4.2.1 One-Time-Cheat

In TBeT, a leecher can initiate its first barter only after getting at least one file piece or a subkey from the seeder. If many peers initiate a file request at the same time, the seeder will potentially be a bottleneck. This problem is alleviated by a technique called the *one-time cheat*, or OTC, which allows a new neighbor to request and obtain a file piece *without* bartering a subkey in exchange, but one time only. Any further requests for file pieces from that same neighbor will be bartered for subkeys of that file. With OTC, the near-simultaneous arrival of a large number of leechers will not overload the seeder, as the leechers can very quickly start swapping file pieces. Although OTC would seem to encour-

age use of the large-view-exploit, it only allows *file pieces* to be exchanged quickly. Downloading of subkeys (which are also needed for file decryption) still requires the peer to upload file pieces to other leechers.

4.2.2 Semi-Seeders

The basic TBeT protocol does not specify the actions to be taken by a leecher once it has downloaded at least t subkeys but has not finished downloading all m file pieces. Since TBeT still adopts the TFT mechanism of BitTorrent, its remaining file piece download times will elongate if it decides to stop uploading to other peers (which is well indicated in subsection 5.6). Thus, it is in the best interest of such a peer to continue uploading file pieces and subkeys to others, in order to receive its remaining file pieces faster. We call such a peer a *semi-seeder*. A semi-seeder needs to upload only to those neighbors who have uploaded file pieces to it in the past (based on its local history), in proportion to their own upload speed, so that free-riders cannot benefit from the extra capacity.

4.2.3 Neighbor Pruning

In TBeT, free-riders must rely solely on seeders to provide them with subkeys (OTC will only upload file pieces, not subkeys). This could have the side effect of increasing the population of free-riders in the system at any given time since it will take longer for free-riders to download desired subkeys. The increased population of free-riders increases the chances of having free-riders in the neighbor set of a compliant peer, which would limit the available upload capacity in the view of a compliant peer.

To mitigate this side effect, we propose that each peer keeps track of the amount of data uploaded from each neighbor for some interval of time (default 10 minutes). If a neighbor has not uploaded any file pieces during that interval, it is suspected of free-riding. The peer will remove it from the neighbor list, and terminate the connection to the neighbor. When the view size of a peer becomes too small, a compliant peer requests a new peer list from the tracker. This allows free-riding neighbors to be replaced by compliant neighbors.

4.3 Optimal Threshold Value

The secret-sharing threshold t is a critical parameter of the proposed method. We present below a simple mean value analysis of the contribution of t . We use this to derive an optimal value which minimizes the download completion time of peers. The parameters used in the analysis are explained in Table 1. To simplify the analysis, we assume the following:

- All leechers have the same upload capacity, and the download capacity of each leecher is unbounded.
- There is only one seeder in the system.

m	the number of file pieces
b	the size of each piece, in bytes
t	the threshold (minimum # of subkeys needed)
N	the number of leechers in the swarm
μ_s	the upload capacity of the seeder
μ_l	the upload capacity of a leecher
T_o	the optimal download completion time
T_a	the total download completion time
T_k	the subkey download completion time
T_p	the file piece download completion time

Table 1: Notation in this section

- The initial start-up time needed for all leechers to download at least one file piece and one subkey is negligible; this means each leecher can initiate its first barter immediately upon joining the system.
- The total number of subkeys in the system is assumed to be infinite, so each subkey uploaded by the seeder to a leecher is unique.

We analyze only the basic protocol without the enhancements described above in subsection 4.2.

4.3.1 Optimal Download Completion Time (T_o)

The minimum download completion time of a peer in a system using BitTorrent combined with TBeT can be defined as follows:

$$T_o = \frac{mb}{\mu_l + \frac{\mu_s}{N}} = \frac{mbN}{\mu_l N + \mu_s} \quad (1)$$

Since the total data bytes a peer needs to receive from the system is $m * b$ and we assume that there is no limit on the download bandwidth of a peer, the download completion time of a peer depends only on the upload bandwidth of other leechers and the seeder. The completion time is minimized when the upload bandwidth of all leechers and the seeder is fully utilized as shown in Eq. 1.

Note that the total download completion time (T_a) of a leecher in our system is equal to the *maximum* of the subkey download completion time (T_k) and the piece download completion time (T_p), since only the peers who collect more than t subkeys and all m pieces complete their downloads. Below, we analyze T_k and T_p separately.

4.3.2 Subkey Download Completion Time (T_k)

The maximum amount of file upload by a leecher is tb , but as some amount is supplied by the seeder, each peer roughly contributes $(tb - T_k \frac{\mu_s}{N})$ bytes during T_k on average. Since the upload bandwidth of each leecher is μ_l , $T_k = \frac{tb - T_k \frac{\mu_s}{N}}{\mu_l}$. Solving this equation gives

$$T_k = \frac{tbN}{\mu_l N + \mu_s} \quad (2)$$

Based on Eq. 2, we conclude that T_k increases proportionally to the threshold size t .

4.3.3 Piece Download Completion Time (T_p)

Since the file size is $m * b$, and the download bandwidth of each peer depends only on the upload bandwidth of other leechers and the seeder, the following three cases are possible:

$t < m$: In this case, since leechers cannot download faster than their upload rates, the download completion time is determined by T_p . The extreme case is when t is zero (i.e., only the seeder uploads) and in this case $T_p = \frac{m * b}{\frac{\mu_s}{N}} \gg T_o$, which is equivalent to a client-server model. Otherwise, since $t < m$, each leecher can download file pieces from other leechers only during T_k . After that, however, leechers can download the remaining pieces ($m - t$ pieces) only from the seeder, since all leechers have received all the required subkeys and therefore will not participate in uploading. (Reminder: for this analysis, we do not consider enhancements to the basic protocol, such as the semi-seeder mode of operation). So,

$$\begin{aligned} T_p &= T_k + \frac{(m - t)bN}{\mu_s} \\ &= \frac{tbN}{\mu_l N + \mu_s} + \frac{(m - t)bN}{\mu_s} \\ &> \frac{mbN}{\mu_l N + \mu_s} = T_o. \end{aligned} \quad (3)$$

Eq. 3 tells us T_p always greater than T_o on the assumption that $t < m$.

$t = m$: If $t = m$, then based on Eq. 3 or Eq. 4 (putting m instead of t), T_p and T_k are the same, which are again equal to T_o . This achieves the optimal download completion time as it allows the upload bandwidth of peers to be fully utilized by the system.

$t > m$: For this case, the total download completion time will be dominated by the subkey download completion time, since T_k will be greater than T_p . In this case,

$$\begin{aligned} T_k &= \frac{tbN}{\mu_l N + \mu_s} \\ &= \frac{mbN}{\mu_l N + \mu_s} + \frac{(t - m)bN}{\mu_l N + \mu_s} \\ &> \frac{mbN}{\mu_l N + \mu_s} = T_o. \end{aligned} \quad (4)$$

From Eqs. 1, 2, 3, and 4, we conclude the optimal download completion time occurs when the threshold value t is equal to m , which is equal to the number of file pieces in the system.

4.4 Countering Free-Riding Strategies

Peers in TBeT need to download both m file pieces, and at least t subkeys to complete the file download. Although

free-riders may cheat other peers to download file pieces, they will be penalized while attempting to download subkeys. Note that, in TBeT, leechers provide subkeys only in response to file piece uploading by other leechers. Only the seeder will provide subkeys without expecting a file piece upload beforehand. However, the seeder bandwidth, which is limited, is shared among all the peers in the system. The result will be a very slow rate of obtaining subkeys (and therefore long completion times) for the free-rider. We believe that free-riding seeders might be well addressed if the source coding technique introduced by Locher et al. [11] is incorporated in TBeT.

Free-riders in BitTorrent may use whitewashing or sybil attacks to enhance their ability to cheat others. In TBeT, however, changing identities will not allow a free-rider to obtain subkeys from other leechers; the free-rider must upload a valid file piece first, regardless of what identity or how many different identities it uses.

The large-view-exploit has turned out to be a highly effective free-riding technique in BitTorrent [4, 5, 17]. With the modifications proposed in TBeT, the large-view-exploit is not effective. An increase in the number of neighbors will not result in obtaining subkeys any quicker, since each subkey request must be preceded by an intact file piece upload to a neighbor.

5 Evaluation

We evaluate the effectiveness of TBeT by simulation. Since no standard simulators for a BitTorrent-like system are publicly available, we developed our own BitTorrent/TBeT simulator, based on the BitTorrent Protocol Specification [20]. This simulator models the peer activities of joining, leaving, choking, unchoking, optimistic unchoking, and piece exchanges. It also captures the essential policies of BitTorrent, including local-rarest-first, rate-based TFT, etc. The network delay model considers only transmission delays but not propagation/queuing delays.

5.1 Experimental Setup

Each simulation run initially starts with only one seeder in the swarm. This seeder remains a part of the swarm throughout the simulation. Leechers arrive and start downloading from other peers (the seeder and other leechers), and exit the swarm immediately after they complete both file downloading and decoding. The arrival process of peers is modeled in either of two ways: as a *flash crowd*, or as a *continuous stream*. The flash crowd model mimics the behavior of many leechers requesting a file at the same time. The continuous stream model is based on the RedHat 9 Torrent tracker trace [21], which represents 5 months of activity in a real BitTorrent.

The seeder's upload bandwidth is set to 6,000 Kbps and the upload bandwidth of leechers is either homogeneous, or heterogeneous. In the homogeneous case, the upload band-

width of all leechers is set to 800 Kbps. In the heterogeneous case, the upload bandwidth of other leechers varies from 400 Kbps to 1,200 Kbps according to the distribution adapted from [6, 7]. There is no limit on the download bandwidth of peers. The number of peers from which a peer concurrently downloads is determined by the number of unchoked peers, as defined by the TFT policy of BitTorrent. The file size is assumed to be 128MB. The number of file pieces m and the subkey threshold value t are set to 1,000, whereas the total number of subkeys n is set to $2m$ to ensure a relatively high availability of subkeys in the system. All measurements show the mean and the 95% confidence intervals, resulting from 5 runs using different random seeds. Unless otherwise specified, we use the above settings in our simulations.

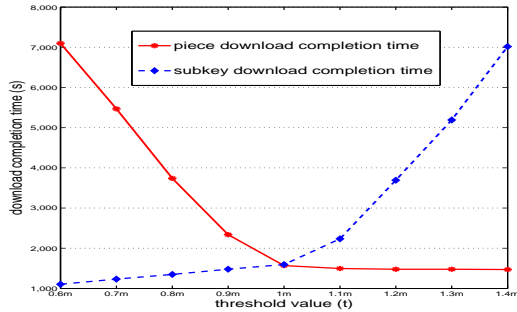
One final concern is the lifetime of free-riders. A scheme that is truly effective at penalizing free-riders means their download times will be much greater than that of compliant users. It is unrealistic, however, to expect that free-riders will remain in the system no matter how long their download times become. To capture this effect, we measured the *average download completion time* of a file, using BitTorrent, in a system consisting of 500 compliant peers under a flash crowd, and no free-riders. Let this time be denoted as C . In each experiment, a peer that has been downloading for more than C seconds will compute the predicted file download time, based simply on the file size and the amount of the file downloaded so far, and the predicted subkey download time, based simply on the threshold t and the number of subkeys downloaded so far. For simulations of TBeT, the predicted download completion time is the greater of these two. For simulations of BitTorrent, the predicted download completion time is just the predicted file download time.

If the predicted download completion time is at least $10 * C$, the free-rider will leave the system with probability $L(t) = 1 - e^{-(t-C)/C}$, where t is the elapsed time since the free-rider joined the system. In this model, $L(t)$ approaches 1 with exponentially decaying probability as t increases beyond C . This model of peer departures is applied only under the continuous stream model, and is applied to both BitTorrent and TBeT, for fair comparison. Other models of departure probability were tested, with very similar results. So, we omitted the results here.

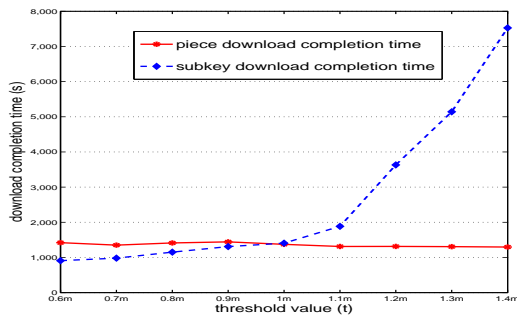
5.2 Optimal Threshold Value (t)

Section 4.3 showed that the optimal value of the threshold t is equal to the number of file pieces m under some simplifying assumptions. To test this analysis, an experiment was conducted using differing values for t . In this experiment, 300 peers with homogeneous upload rates arrived in a flash crowd, and joined the swarm.

As shown in Figure 2(a), results from the simulation of the basic protocol confirm the analysis. In TBeT, the peer



(a) TBeT basic protocol



(b) TBeT with enhancements

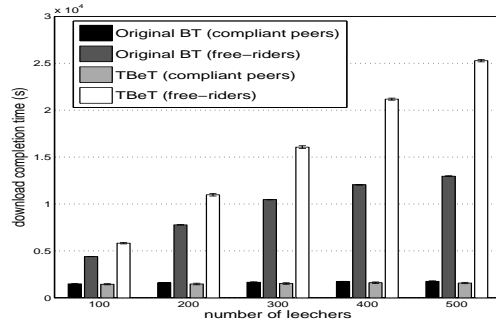
Figure 2: Simulation of the basic protocol (a) and the protocol with enhancements (b); the piece download completion time (T_p) and the subkey download completion time (T_k) are plotted for a flash crowd of 300 peers. It shows that the download completion times of compliant peers are minimized when $t = m$.

completion time is the *maximum* of the piece completion time and subkey completion time. The simulation results agree with the analysis in that (1) when $t = m$, the peer download completion time is a minimum; (2) when the threshold value decreases, the subkey download completion time also decreases. However, the piece download completion time sharply increases as the subkey download completion time decreases; and (3) when the threshold value increases beyond m , the reverse is true (subkey download time dominates piece download time).

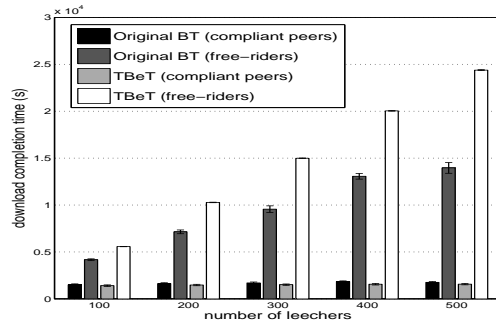
Figure 2(b) shows the results of simulating TBeT combined with the protocol enhancements of section 4.2. With these enhancements, leechers that have completed subkey downloading become semi-seeders and continue uploading, so that the remaining leechers complete earlier. When the threshold value is low, this leads to a greatly decreased piece download time, T_p (thanks to the semi-seeders). But the threshold value should not be reduced to less than m , because a low threshold value would encourage free-riders to stay for a long time in the system. Note that any results for TBeT hereafter reflect the use of TBeT with the protocol enhancements.

5.3 Penalty for Free-Riding

The next experiment investigated the penalty suffered by free-riders for their failure to share resources. In the experiment, peers arrived in a flash crowd and the number of peers and the ratio of free-riders were varied. The upload bandwidth of free-riders was set to zero (the most selfish assumption). As a result of this selfish assumption, each free-rider uploads zero pieces (but uploads subkeys). The experiment measured and compared the average completion times of free-riders and compliant peers.



(a) Homogeneous

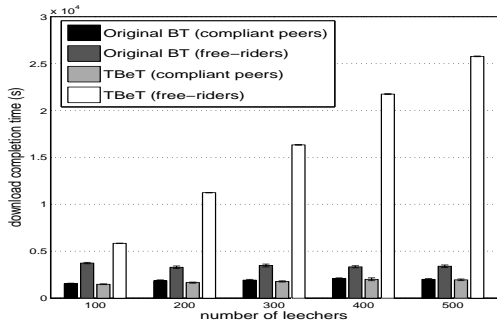


(b) Heterogeneous

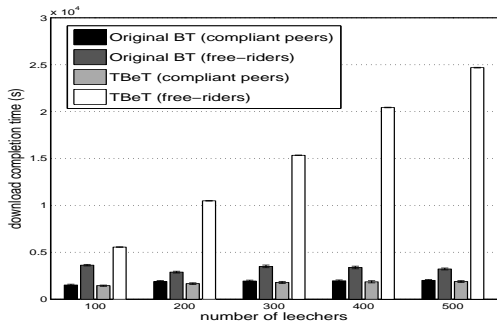
Figure 3: Average download completion time of compliant peers and free-riders. 25% of peers are free-riders in both systems. Results are shown for the homogeneous (a) and heterogeneous (b) upload bandwidth cases. The error bar indicates 95% confidence intervals. In this simulation, free-riders make no attempt to avoid penalties or increase their download rate.

Figures 3(a) and 3(b) show the average download completion times of compliant peers and free-riders, for both the original BitTorrent, and for TBeT. In the experiment, 25% of the total peer population was assumed to be free-riders. In this experiment, the free-riders did *not* use any techniques to improve their performance, such as the large-view-exploit or whitewashing. It can be seen that download completion time of compliant peers is very close in both systems, and is independent of the number of leechers. For both systems, free-riders (not attempting to avoid detection

or penalties) incur far deprecated download times, with the penalty increasing as the number of peers increases. However, the penalty for free-riders is greater in TBeT than in BitTorrent, with the difference increasing significantly as the system size increases. The results are similar for both the heterogeneous and homogeneous cases.



(a) Homogeneous

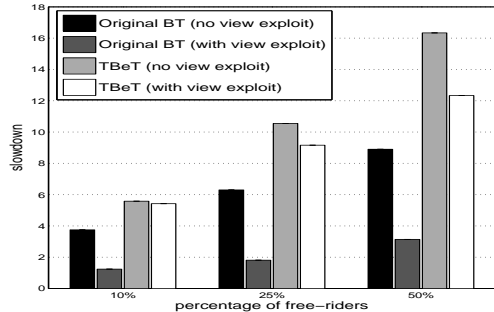


(b) Heterogeneous

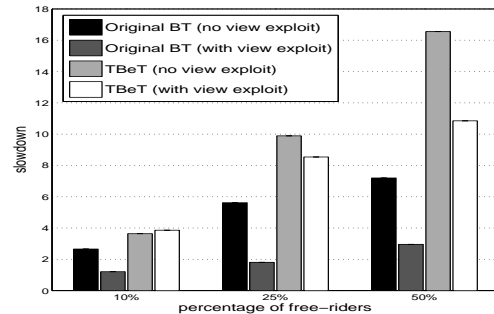
Figure 4: The same setup as in figure 3 except that free-riders use the large-view-exploit. While use of this free-riding technique improves the performance of free-riders in the original BitTorrent system, it has no effect when TBeT is used.

Figures 4(a) and 4(b) show the results for the same conditions, except that the free-riders were assumed to use the large-view-exploit. This is the most damaging free-riding technique currently being used in BitTorrent. In this experiment, we assume that free-riders request new peer information from the tracker at every optimistic unchoking period in order to increase their view size. The graphs show that free-riders in BitTorrent substantially reduce their penalty, and finish almost as quickly as compliant peers. TBeT, however, is virtually unaffected by the large-view-exploit.

Figures 5(a) and 5(b) show the ratio of the average completion time of free-riders over that of compliant peers for the homogeneous and heterogeneous cases as we vary the percentage of free-riders. In the figures, we find that the slowdown ratio for the free-riders in TBeT is consistently higher than for free-riders in BitTorrent. When free-riders



(a) Homogeneous



(b) Heterogeneous

Figure 5: The ratio of the average completion time of free-riders over that of compliant peers (i.e. slowdown ratio) under the homogeneous (a) and heterogeneous (b) upload speed case. With the large-view-exploit, the penalty imposed on free-riders is substantially larger in TBeT than in BitTorrent.

use the large-view-exploit, the slowdown penalty of free-riders for TBeT is more than 5 times of that for BitTorrent. This ratio increases as the population of free-riders increases, and is unaffected by the assumption of homogeneous or heterogeneous upload bandwidth.

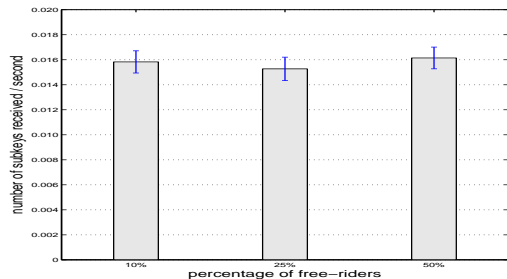


Figure 6: The rate at which free-riders in TBeT get subkeys, for the continuous arrival process with 600 heterogeneous upload rate peers. At this rate, a free-rider would take approximately 60,000 seconds to download 1,000 subkeys.

A related experiment investigated the affect of free-riding in TBeT when peers arrived continually, rather than in a flash crowd. Figure 6 shows the subkey download rate for free-riders under the continuous stream model. It indicates that each free-rider requires, on average, approximately 60,000 seconds (more than 25 times greater than compliant peers) to download the needed 1,000 subkeys. Under these circumstances, based on the peer departure assumption described in subsection 5.1, almost all free-riders in TBeT depart the system without successfully completing the download.

5.4 Performance of Compliant Peers

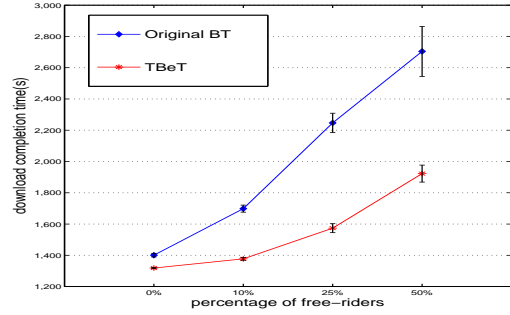
From section 5.3, we find in TBeT that free-riders experience much longer completion times than compliant peers even with the large-view-exploit. This is because TBeT limits the amount of subkeys that selfish free-riders can get from the system so that their only source of subkey download is the seeder. The completion time of compliant users will be highly variable under the flash crowd model. Since free-riders are not assumed to depart in this model, as compliant users complete and leave, the percentage of free-riders in the system grows higher and higher. So, the remaining compliant users have great difficulty in getting data. We view these assumptions and behavior as leading to system collapse in either BitTorrent or TBeT. Therefore we investigate the continuous stream model with free-riders who depart before their download completion if predicted download times become too great (as assumed in subsection 5.1), which is more realistic.

Figure 7 shows the results for continuously arriving peers as the percentage of free-riders is varied. The completion times of compliant peers are much less in TBeT, than in BitTorrent. At 25% free-riders with homogeneous upload rate peers, for instance, the completion time of compliant users is about 33% less for TBeT than for BitTorrent. As the relative population of free-riders increases, the difference is accentuated, and is unaffected by the assumption of homogeneous or heterogeneous upload bandwidth. The result of discouraging or penalizing free-riders yields the improved performance for compliant users in TBeT.

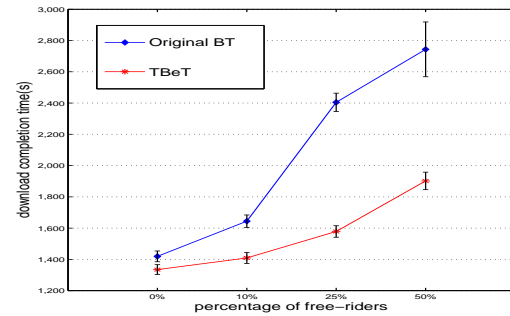
5.5 Fairness

TBeT and BitTorrent both allocate more resources to peers who donate more upload bandwidth to the system. The TFT policy penalizes free-riders in both systems if no attempt at evasion occurs. In TBeT, the DFKL policy further penalizes free-riders regardless of whether or not evasion is attempted. So, we claim that the combination of the two policies in TBeT further provides a powerful incentive for compliant users to increase their upload bandwidth.

To investigate this claim, an experiment was conducted. In this experiment, free-riders are assumed to use the large-view-exploit and peers arrive the system in a flash crowd.



(a) Homogeneous



(b) Heterogeneous

Figure 7: The completion time of compliant peers. The simulation is done using a trace-driven arrival process mimicking the arrivals of peers seen in a real P2P system. All peers are assumed to have the same upload bandwidth in (a) and heterogeneous upload bandwidth peers are assumed in (b). TBeT improves the performance of compliant peers.

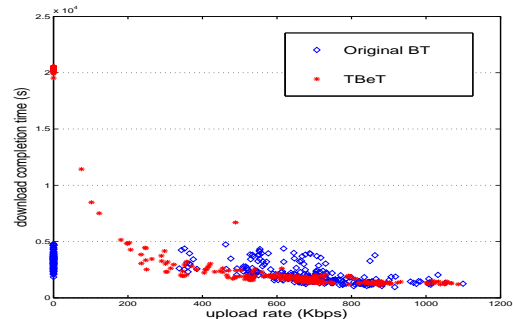


Figure 8: Correlation between the download completion time and the donated upload rate in both systems with the large-view-exploit and 25% free-riders out of 400 leechers. In this experiment, peers arrive the systems in a flash crowd.

The download bandwidth received by a peer was then measured, and compared with the upload bandwidth donated by that peer. The results for BitTorrent and for TBeT are shown in figure 8. This figure indicates that BitTorrent under

the large-view-exploit fails to effectively reward generous peers by providing proportionally better download speeds to them. The result is that overall system performance of BitTorrent is limited by the number of high bandwidth altruistic peers. In TBeT, there is a clear correlation between upload and download speeds, which provides a powerful incentive to peers to contribute resources to the system.

5.6 Performance of Rational Peers

As indicated above, TBeT exploits the selfishness of peers to increase the system capacity. No peers are relied upon to altruistically donate their resources; peers share their resources with others only when such sharing results in added benefits for them. This is accomplished by the difficult to evade penalties for free-riding, and by the correlation of download (experienced as faster download times) and upload bandwidth.

It is desirable for there to be a similar incentive for leechers to become semi-seeders when they have finished downloading the number of subkeys they require. A semi-seeder will continue to upload file pieces and subkeys to other peers, but only if the semi-seeder has a history of direct, satisfactory experience with those peers.

In this subsection, we compare the completion times of semi-seeders with other leechers who elect not to become semi-seeders after their subkey completion. Such leechers are termed here *rational peers*. A rational peer, upon successful downloading of t subkeys, will continue to upload subkeys to other leechers in the attempt to obtain the remaining file pieces it lacks. A rational peer will not, however, upload file pieces to other leechers after this point.

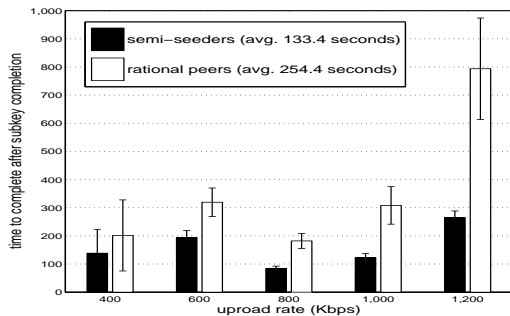


Figure 9: The average download completion time of semi-seeders and rational peers after subkey completion under the continuous arrival process. Half of 500 semi-seeders were randomly selected as rational peers.

A test of download speeds for rational peers and semi-seeders was conducted under the continuous arrival process without free-riders. We randomly select half of semi-seeders as rational peers so that they do not upload pieces after their subkey completion. Figure 9 shows the result for peers with varying upload speeds. File download comple-

tion times following the subkey download completion were measured for the two groups. In this test, we find that the times to complete their downloads after subkey completion of rational peers are almost double that of semi-seeders. This demonstrates there is significant incentive to switch to semi-seeder mode after completion of the subkey download.

6 Discussion

In this section, we analyze the overhead for implementing TBeT on BitTorrent-like systems. We also discuss the limitations of our approach and possible solutions to those limitations.

6.1 Computational Overhead

The computational cost of TBeT includes the time for the seeder to generate subkeys and encrypt the file, and the time for each leecher to reconstruct the key and decrypt the file pieces. Based on an experiment (run on an Intel 2.53-GHz Pentium 4 processor and 512MB of DDR SDRAM), the time for the seeder to generate subkeys for a 1GB file with 4,000 pieces was several minutes; this was using the original SSSS implementation [22] and a 64 bit key. In exchange, the file completion time for compliant peers with TBeT was found to decrease by more than two hours on average for the 1GB file. File decryption time was less than 1% of the average file download time, and the extra bandwidth required for subkey exchange was insignificant compared with the bandwidth needed for the exchange of file pieces.

6.2 Collusion

Collusion between free-riders (exchanging subkeys) is not prevented in current TBeT. However, the performance impact is limited, for two reasons. First, as each file is encoded with a different key, colluding is only beneficial for files that are of common interest to the free-riders. Second, unless each free-rider has a large number of subkeys, a large number of free-riders must collude in order to succeed. As seen in figure 6, since the subkey download rate of free-riders is extremely low in TBeT, collusion is unlikely to be effective in practice. In some cases, free-riders might try to get some amount of subkeys by uploading file pieces (i.e. contributing their resources) and then collude with each other by exchanging subkeys. Note that in BitTorrent, no such contribution is required for free-riders.

6.3 Key Disclosure

In describing our work, the assumptions of TBeT to this point have been that users are selfish, but not malicious. This may not hold true under real conditions. A possible attack by a malicious peer is described as follows: The malicious peer participates as a compliant user in TBeT, and thereby obtains in a legitimate way at least t subkeys. During this stage, the malicious peer contributes upload band-

width to the system, and does not act as a free-rider. From the t subkeys, the encryption key k may be found. The malicious peer may then post the key for this file to a public web site. Once disclosed, *other* peers with the key can freely ride on TBeT using the large-view-exploit, as is currently the case in BitTorrent. We believe, however, this is a small risk for several reasons. First, the discloser should contribute its resources to get the key. Second, the discloser has to make this information public; both the discloser and the free-rider risk exposure. Third, and most importantly, there are *no* benefits to the discloser itself; only other leechers will benefit. Finally, each file targeted by free-riders must have its own, unique key exposed by a malicious peer.

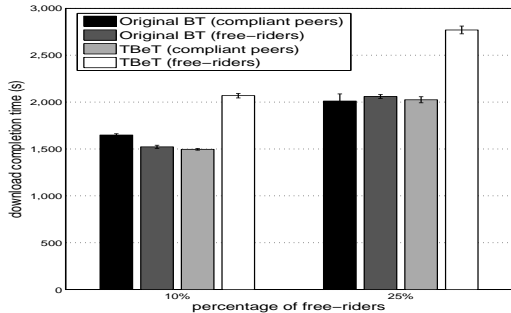


Figure 10: The completion times of compliant peers and free-riders with heterogeneous upload rates in both systems after a key disclosure (i.e. free-riders know all the subkeys upon joining TBeT) under the continuous stream model. The completion times of 100 free-riders using the large-view-exploit and contemporaneous compliant peers in both systems are investigated. This clearly shows that the average download completion time of free-riders in TBeT is much longer (at least 30%) than that in BitTorrent whereas the average download completion time of compliant peers in TBeT is shorter than that in BitTorrent even under the key disclosure.

Nonetheless, key disclosure is a serious threat if not properly addressed. We investigated the effect of key disclosure. Figure 10 demonstrates the completion times of compliant peers and free-riders with heterogeneous upload rates in both systems, under the continuous stream model. The average download completion time of 100 free-riders and that of contemporaneous compliant peers in both systems was measured. In this case, it was assumed that free-riders joining TBeT know all the subkeys, due to key disclosure by some other peers. Even with the key disclosure, the average download completion time of compliant peers in TBeT is shorter than in BitTorrent. More importantly, TBeT more effectively penalizes free-riders with much longer (at least 30%) completion times than BitTorrent. This is due to the semi-seeders, and the neighbor pruning technique in TBeT (i.e. the protocol enhancements explained in 4.2). That is, free-riders cannot benefit from the upload capacity

of semi-seeders without contributing bandwidth, thanks to the local histories managed by each semi-seeder. In addition, neighbor pruning enables compliant peers to eliminate possible free-riders from their neighbor sets and helps them to find other compliant peers. TBeT therefore outperforms BitTorrent even under this scenario of key disclosure.

Nevertheless, the system performance of TBeT is degraded by key disclosure. The effect of key disclosures can be mitigated by changing the file encryption key periodically, and re-encoding the file using this new key. The seeder must then distribute the newly-encoded file pieces and the new set of subkeys. Let the period between two consecutive key changes be called a *key session*. Changing the key will result in an increase in the download completion time of compliant peers who are currently part of the swarm. To reduce the impact of rekeying, we propose an *overlapping* subkey generation scheme. The scheme reuses some of the current subkeys in generating the next full key, rather than starting from scratch. With even one subkey change, the corresponding key will be different. Let $R (< t)$ be the number of subkeys which are reused, then $n - R$ new subkeys must be generated and made available by the seeder. For each new key, the seeder must publish (in the new .torrent file) what the new set of subkey hash values are. Leechers then must download sufficient subkeys to decrypt file pieces encrypted with the old file key and sufficient subkeys to decrypt file pieces encrypted with the new file key, with some subkeys serving for both purposes. This overlapping key scheme allows transition to a new key session with less impact on the performance of compliant users.

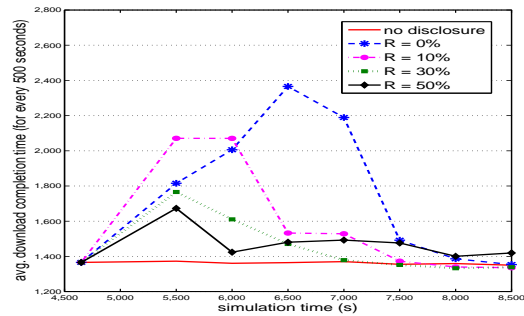


Figure 11: The instantaneous completion time of compliant peers in TBeT with rekeying, assuming continuous peer arrivals. Results are averaged over every 500 seconds, after a rekeying event at time 4,650 seconds, for varying percentages of subkey overlap (R). All compliant peers converge to the average completion times before rekeying after a brief increase in download times. In this experiment, we assume 10% of the peers are free-riders.

We conducted a trace-driven experiment to measure the impact of rekeying. The experiment measured the download completion time of compliant users after rekeying for

various values of R . Homogeneous upload rate peers with 10% free-riders are considered and we assume that after 4,600 seconds (the amount of time for the average download completion time of compliant peers to become stable), a malicious peer who leaves the system discloses the encryption key.

Figure 11 presents the results. As expected, completion time for compliant peers is increased by key disclosure and the needed rekeying. As R increases, this increase in the completion time is reduced, but the effects of key disclosure remain for a longer period of time. For this experiment, the reuse of 30% of the subkeys from the previous key provides a good balance. With 30% R , the average download completion time of compliant peers temporarily increase and reaches a maximum of about 1,670 seconds, which is about the same as BitTorrent with 10% free-riders. However, the average time keeps decreasing (thanks to the rekeying) and stabilizes after about 3,000 seconds (i.e., under one hour).

7 Conclusion

We have proposed a method of preventing free-riding in P2P systems. This method, called TBeT, is based on (t, n) secret sharing. A file is divided into pieces, encrypted with a symmetric secret key, and then distributed to requesting peers along with subkeys generated. Peers must then swap file pieces for subkeys, which are needed in order to decrypt the file pieces. Use of secret sharing effectively counters known free-riding techniques, such as the large-view-exploit, and whitewashing.

Simulations demonstrate the benefits of TBeT when compared with a popular P2P system, BitTorrent. Free-riding is heavily penalized or eliminated under TBeT, while download time for compliant peers is reduced by as much as 50%. The system provides the proper incentives to peers to contribute bandwidth, to a much greater degree than BitTorrent, as shown experimentally.

TBeT is straightforward, has modest computational overhead, and minimal bandwidth overhead. It works well in combination with existing policies used by BitTorrent such as rate-based tit-for-tat (TFT). While handling free-riding, TBeT is subject to attack by malicious peers, who potentially disclose keys but do not profit themselves. Periodic or on-demand rekeying in response to key disclosure, as well as other factors, reduce the likelihood and impact of such attacks. We leave the key disclosure problem as our on-going future work.

8 Acknowledgments

We thank Chen Zhao, Harsha Girish, and Greg Neujahr for their helpful discussions on this work and the anonymous reviewers for their fruitful feedbacks. This work was partly supported by the Secure Open Systems Initiative (SOSI) at North Carolina State University.

References

- [1] B. Cohen, "Incentives build robustness in bittorrent," in *P2PECON'03*. Berkeley, May 2003.
- [2] B.-G. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz, "Chunkcast : An anycast service for large content distribution," in *IEEE P2P'06*, February 2006.
- [3] P. Garbacki, A. Iosup, D. Epema, and M. van Steen, "2fast: Collaborative downloads in p2p networks," in *IEEE P2P'06*, 2006.
- [4] S. Jun and M. Ahamad, "Incentives in bittorrent induce free riding," in *P2PECON'05*, Philadelphia, PA, August 2005.
- [5] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, "Free-riding in bittorrent networks with the large view exploit," in *IPTPS'07*, Bellevue, WA, February 2007.
- [6] A. R. Bhambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *IEEE INFOCOM'06*, Spain, April 2006.
- [7] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent?" in *USENIX NSDI'07*, May 2007.
- [8] M. Feldman and J. Chuang, "Overcoming free-riding behavior in peer-to-peer systems," in *ACM Sigecom Exchanges*, vol. 5, July 2005.
- [9] H. T. Kung and C.-H. Wu, "Differentiated admission for peer-to-peer systems: Incentivizing peers to contribute their resources," in *P2PECON'03*, June 2003.
- [10] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "Karma: A secure economic framework for peer-to-peer resource sharing," in *P2PECON'03*, June 2003.
- [11] T. Locher, S. Schmid, and R. Wattenhofer, "Rescuing tit-for-tat with source coding," in *IEEE P2P'07*, September 2007.
- [12] D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: The bell tolls?" in *IEEE Distributed Systems Online*, June 2005.
- [13] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *MMCN'02*, 2002.
- [14] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
- [15] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *EC'04*, May 2004.
- [16] J. R. Douceur, "The sybil attack," in *IPTPS'02*, Cambridge, MA, March 2002.
- [17] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *HotNets'06*, November 2006.
- [18] K. Tamilmani, V. Pai, and A. Mohr, "Swift: A system with incentives for trading," in *P2PECON'04*, June 2004.
- [19] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *ACM SIGCOMM'04*, Portland, OR, August 2004.
- [20] "The bittorrent protocol specification," February 2008. [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html
- [21] "Redhat 9 torrent tracker trace." [Online]. Available: http://mikel.tlm.unavarra.es/~mikel/bt_pam2004/
- [22] "Shamir's secret sharing scheme implementation." [Online]. Available: <http://point-at-infinity.org/ssss/>