

# A Novel Free-riding Prevention Technique Based on Secret Sharing in BitTorrent-like Peer-to-Peer Networks \*

Kyuyong Shin

kshin2@ncsu.edu

Department of Computer Science  
North Carolina State University  
Raleigh, North Carolina, USA

June 12, 2007

## Abstract

Collaboration is the key premise of Peer-to-Peer (P2P) systems because its success highly relies on the voluntary contribution of resources by individual peers. Selfish peers or free-riders who attempt to benefit from others' resources without sharing their own, however, cause system-wide performance degradation in P2P systems. To address this problem, much research has been done in this area, however existing works are not widely used due to either their complexity or inefficiency. In this paper, we introduce a simple but highly efficient free-rider prevention technique which forces a peer to cooperate based on  $(t, n)$  threshold secret sharing. Our simulation results show that free-riders take much more time to complete their download <sup>1</sup> compared to those of compliant peers, which means that it's nearly impossible for peers to freely ride on the system without any contribution under the proposed framework. Another unique benefit of our framework is that the system can now provide fairness. Our experiment shows that there is a strong correlation between the upload rate and the download completion time of a peer.

## 1 Introduction

Peer-to-Peer systems have gained momentum as an alternative approach to the traditional client-server model for various applications, including file sharing. In a P2P system, peers can directly communicate and collaborate with one another to reduce the download time by sharing resources such as upload bandwidth and CPU capacity [4, 3, 8]. As opposed to the traditional client-server model, the larger the number of users, the higher the bandwidth that is available in P2P networks.

Despite the importance of fair cooperation among peers, it is still difficult to achieve the fairness since many users tend not to share their resources without any incentives or penalties given in a system. In fact, many P2P systems lack such a mechanism which makes them suffer from free-riding. As a consequence,

---

\*This topic has been concurrently progressed as the CSC 773 course project with the team members including Chen Zhao, Harsha Girish, and Greg Neujahr. They contribute the implementation of a simulator. However, the original idea, writing, and the evaluation parts of this paper were purely done by the author. Thanks to Chen for helpful discussion on the subject and thanks to Harsha and Greg for proofreading.

<sup>1</sup>If a peer,  $P_A$ , sends data to a peer,  $P_B$ ,  $P_A$  will be an uploader and  $P_B$  will be a downloader. Throughout this paper, we use terms "uploading" which means sending data to others and "downloading" which means receiving data from others.

as the number of free-riders increases the overall system capacity per peer decreases which means that it will eventually experience the “Tragedy of the commons [9],” manifested as system-wide performance degradation. Much research has been done to prevent free-riders, however existing works [12, 20] are not widely used due to either their complexity or inefficiency.

In this paper, we propose a unique method to address free-riders problem based on the  $(t, n)$  threshold algorithm [16] in file sharing P2P networks. The main idea is that if a seeder (who wants to upload its data file) encrypts the file with a key,  $k$ , using simple and inexpensive symmetric encryption algorithm, such as DES. The seeder then generates  $n$  pieces of sub keys, using the well known  $(t, n)$  threshold algorithm so that any peer who gets  $t$  sub keys out of total  $n$  can figure out what the original key,  $k$ , is hence it can decrypt the data file using the recovered key. The seeder divides the data file into  $m$  pieces, as BitTorrent does. Upon request, the seeder distributes a piece of the data file and a sub key to the requesting peer (We assume different pieces go to different peers, and always the unit is a pair, like a piece of data and a sub key). Now, each peer can barter its sub keys for pieces of a data file, and vice versa. In this case, the peers need to follow “*data first, key later*,” rule because otherwise, the same problem of Prisoner’s Dilemma will be encountered. We can easily show that there is no benefit for cheaters not to give sub keys or pieces of data at the end because only the peer who gets all the pieces of data file and more than  $t$  sub keys can recover the original data file uploaded by the seeder.

Our method brings unique benefits as listed follow:

- We provide a new incentive mechanism using sub keys. In our method, each peer should try as many as sub keys possible to reach the required the number of sub keys, which means that the system enforces peers to contribute to each other. This also reduces an user’s temptation to cheat other peers since cheating peers will not get any benefits from them without contributing its resources to the system.
- Enforce the fairness. Our experimental result shows that there is strong correlation between the upload rate and the download completion time in our system, which means that our approach enforces a strong fairness on peers so that if they upload fast, they can download fast in compensation for their efforts.
- Easy to implement. Our method does not require any centralized servers, monetary schemes, or reputation schemes that are difficult to implement in P2P systems. Our method only require a simple encryption/decryption and sub key generation algorithms that can be easily implemented with low overhead.

The remainder of this paper is organized as follows. Section 2 discusses existing free-rider prevention techniques and their limitations. Section 3 briefly describes how BitTorrent and  $(t, n)$  threshold scheme work which is the foundation for our scheme. Section 4 details our method and section 5 presents our simulation results. Section 6 discusses the possible weakness of our method and future directions. Section 7 concludes the paper.

## 2 Related Work

Traditional client-server model has low maintenance overhead because only the server has the resources to be allocated to many requesters, which means that the system manager needs to only care about the server to maintain the entire system. This advantage, however, is also a fatal drawback in that nobody can get services from the system if the server collapses. In addition, since the system capacity is fixed regardless of

the number of participants, the capacity per participant decreases as the number of clients increases, which means that this approach does not scale well.

Contrary to the traditional client-server model, P2P systems highly rely on the resources of the participating peers rather than a single server. So, there is no strict distinction between the client and the server that a peer can function as both client and server at the same time to other peers in a system. Since all peers provide their resources such as storage, bandwidth, and computing power to other peers, the overall system capacity increases as the number of peers increases, making the systems potentially self-scaling. Moreover, since the resources including data are distributed among multiple participants, there is no single point of failure, which increases robustness and reliability. However, collaboration should be the key premise of P2P systems because its success highly relies on the voluntary contribution of resources by individual peers. In other words, if nobody contributes its resources to the system, P2P systems cannot survive.

Despite the importance of cooperation among peers in P2P system, it is difficult to achieve because many users tend not to share their resources. There can many reasons including followings. (1) Free-riding is a rational behavior of mankind. Suppose that one can get services or goods for free from others. In that case, who do want to pay for them no matter how cheap they are? It is rational for everyone to maximize its utility. (2) Everyone has the limited amount of resources. If one has sufficient resources, it would not join the P2P systems. In this sense, every peer wants to save its resource as much as possible. (3) Generally the upload bandwidth is insufficient compared to the download bandwidth in most peers because many Internet service providers provide relatively low bandwidth for outgoing traffics. Therefore it is difficult for peers to contribute more than what they receive and thus free-riding already exists whether they intend to or not to.

Free-riding is a one of serious problems in P2P systems and many studies show that there are a lot of free-riders in the current P2P systems which makes the system suffer from performance degradation. For example, Hughes et al. [10] revisited the free-riding issue in Gnutella and found that 85% of the peers do not share any files. Therefore they point out that preventing free-riding is one of the most important issues in the P2P systems. To address this problem, different incentive schemes have been proposed in recent years to encourage peers to cooperate with each other. Such approaches can be categorized as follows: 1) monetary based scheme, 2) reciprocity based scheme, and 3) reputation (or credit) based scheme [6].

- **Monetary based scheme** adopts simple payment mechanisms to encourage peers to collaborate with each other. By introducing micro-payment methods into P2P systems, the system can keep track of the resource consumption and contribution of each peer. For example, each peer receives a fixed amount of e-money from the system when it arrives the system. If  $P_A$  receives a service from  $P_B$ ,  $P_A$  pays  $P_B$  back for the service with the e-money. The more service a peer provides, the more e-money it can get from others. If a peer has used up all e-money, it cannot get services from others until it earns some money by serving others. In this sense, how much e-money a peer has represents its standing in the system. Vishnumurthy et al. [20] developed a virtual currency, called KARMA, which captures the amount of resources that a peer has contributed and consumed. In their system, each transaction is monitored by the system and each peer's activities can be controlled based on the accounted information.

This scheme, however, has some drawbacks. First, it requires a special infrastructure for accounting with authentication and billing, which is nearly impractical, if not impossible, to implement in P2P systems. Second, it is vulnerable to the whitewashing attack [7] where free-riders repeatedly rejoin the system with new identities whenever they use up all the e-money they have. It is really hard for the system to tell whitewashers from legitimate newcomers because changing identities is very easy to perform in P2P systems.

- In **reciprocity based scheme**, each peer maintains history of other peers' activities only if they are directly related to the peer and uses this information to make a decision. For example, if there is a service request from  $P_B$  to  $P_A$ ,  $P_A$  checks the history for  $P_B$  to see whether the peer has provided any services for him or her in the past. If the peer has provided services, then the recipient accepts the request, otherwise rejects<sup>2</sup>. The main advantage of this scheme is the simplicity of implementation because each peer only needs to maintain the local information on symmetric transactions between two peers. The *tit-for-tat (TFT)* and its variations [4, 19, 7, 1] are good examples of this scheme.

There are several disadvantages of the reciprocity based scheme. First, it cannot capture the overall contribution of peers to the system. For example, even though  $P_A$  has provided sufficient services to other peers in the system but not  $P_B$ ,  $P_A$  cannot get service from  $P_B$  because it has not served  $P_B$  yet. Second, they are only appropriate for applications with long term relationship between peers. If there is no long lasting history between peers, reciprocation will never happen.

- In contrast to the reciprocity based scheme, in **reputation based scheme**, the decision whether a peer  $P_A$ , accepts the request from  $P_B$  depends on the services that  $P_B$  has provided to other peers in the system. Therefore even though  $P_B$  has not provided any service directly to  $P_A$  but  $P_B$  has provided some services to other peers in the system,  $P_A$  accepts the request based on the service level of  $P_B$ . This scheme is more scalable than the reciprocity based scheme because the decision in the reputation based schemes is made based on the overall contribution level of each peer to the entire system and not based on only local symmetric information. [12] is a good example of these schemes.

This scheme, however, relies on second-hand observations to reach the global view of all transaction results that can cause nontrivial memory spaces and calculation overhead to manage the system-wide reputation information, which is hard to implement. It is also vulnerable to the sybil attack [7] where a peer creates a large number of pseudonymous entities and reports false information to the system to gain a good reputation or to deprecate others' reputation.

Among these incentive schemes, only the reciprocity based scheme is actually used to current P2P systems because of its simplicity of implementation than other proposed schemes. BitTorrent [2, 4], a P2P file distribution system, which is widely used in the current Internet, is a good example of a reciprocity based scheme. BitTorrent incorporates the rate-based TFT incentive mechanism, whereby each peer preferentially uploads its data to other peers, from whom it's able to download what it wants as fast as possible. This simple but fair sharing mechanism of BitTorrent is shown to achieve a Nash equilibrium under certain conditions [15] and is widely believed to strongly discourage free-riding [13].

Contrary to popular belief, however, BitTorrent's incentive mechanism does not provide sufficient incentives to prevent the desire of peers for free-riding, and many research [11, 17, 13] show that there exist possible manipulation attacks, whereby free-riders still achieve better download speed than those of compliant peers. Among them, the most devastating free-riding attack is the cheating accompanied by whitewashing described in 4.3.1, 4.3.4. We explain the negative effects of those attacks in detail later in this paper.

### 3 Background

In this section, we describe the principal ideas behind BitTorrent and secret sharing, in order to better explain our scheme detailed in section 4. We first summarize how BitTorrent works.

---

<sup>2</sup>Exception occurs if and only if  $P_B$  is a newcomer in the system and the exception degree will be implementation dependent.

### 3.1 BitTorrent Overview

BitTorrent is a P2P file sharing protocol designed to facilitate file transfer among multiple peers across unreliable networks [2, 4]. It is composed of three parties: the tracker, the seeders (including original seeder), and the leechers. Initially, the original seeder who wants to publish a large file via BitTorrent cuts the file into pieces of fixed size (typically 250KB) and creates a *.torrent* file which contains meta-data information (such as name, length of each piece, SHA1 hash value of each piece, tracker's URL, etc.) of the file and posts the *.torrent* file on a public website. Peers who are interested in downloading the file, can find the *.torrent* file on the public website.

The tracker maintains a list of all clients currently downloading a certain file (leechers) or only uploading the file to other peers (seeders), and keeps track of which peers have certain pieces. Based on this information the tracker helps peers find each other and coordinates their activities. In fact, the newcomers can only find other peers who are downloading the same file, by contacting the tracker.

In order to participate in the torrent swarm<sup>3</sup>, a client must download the corresponding *.torrent* file from the public website, and contact the tracker to get a list containing a random subset of the peers (up to 50 peers including leechers and seeders) currently in the swarm. Then it downloads pieces from the seeders or exchanges pieces with other leechers via TCP connections. Each peer needs to send requests to the tracker to obtain the list of peers on a regular basis and update the piece information downloaded so far on the tracker.

Each peer attempts to maximize its aggregate download rate by downloading as much as it can. When exchanging pieces, each peer uploads pieces to a fixed number (e.g. 4) of peers from which it can get some pieces in return, in a process called unchoking<sup>4</sup>. The decision on which peers to choke/unchoke is based on downloading rate of the peer and is reevaluated on a regular basis for every 10 seconds. This maximize the downloader's download rate. There is a peer which is unchoked regardless of its upload rate, in order to discover an unused connection better than the current ones every 30 seconds. This process is called optimistic unchoking. In summary, a peer uploads its data up to 5 neighbors and it selects 4 high upload rate peers among its neighbors who are currently uploading data to him/her (unchoking) and randomly selects one of its neighbors as an optimistic unchoking peer. There is no limitation on the number of download connections. Above mentioned reciprocation and choking/unchoking could maximize the client's download rate, through so called rate-based tit-for-tat (TFT).

### 3.2 $(t, n)$ Threshold Secret Sharing

Secret sharing refers to any method of distributing a secret among many users so that the secret can only be reconstructed by combining the shares. Individual shares cannot be directly used to recover the original secret. For example,  $(t, n)$  threshold secret sharing divides a secret into  $n$  pieces, so that the original secret is easily reconstructed from any  $t$  pieces among  $n$ , but with even  $t - 1$  pieces we cannot reconstruct the original key. As mentioned in [16], threshold schemes can be used in situations wherein some suspicious individuals with conflicting interests need to cooperate. This was a strong motivation for us to apply this scheme to P2P systems to solve the free-riding problem.

There are many different secret sharing schemes, but we use Shamir's  $(t, n)$  threshold scheme [16], because it is very simple and easy to use. Shamir's threshold scheme is based on polynomial interpolation. As we know, any distinct  $t$  points of a given polynomial,  $y(x) = a_0 + a_1x + a_2x^2 + \dots + a_{t-1}x^{t-1}$ ,

---

<sup>3</sup>A group of peers (seeders and leechers) downloading the same file and the tracker constitute a BitTorrent swarm.

<sup>4</sup>Each peer has TCP connections with its all neighbors and initially all connections are choked, which means that no data is sending via the connections. Unchoking means that the peer sends its data to one of its neighbors via the TCP connection between the peer and the neighbor

represent the polynomial. In this case, we can take  $a_0$  as the secret, and any  $t$  points of this polynomial as the threshold. Therefore, if we want 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  $n$  points from  $P_1 = y(1)$  to  $P_n = y(n)$  ( $n$  point values will be the shares). For a real implementation, modular arithmetic will be used instead of real arithmetic with some conditions.

Suppose that a seeder wants use (3,5) threshold secret sharing. It can randomly selects a polynomial with degree 2. When picking a polynomial,  $a_i$  should be greater than zero.  $y(x) = 10 + 2x + 3x^2$  will be a possible polynomial and ,in this case, 10 is the secret. Next, it needs to pick up a random prime number greater than the secret 10. Let's assume that the prime number is 11. Now, it can generate the shares by polynomial interpolation and modular arithmetic and the shares will be  $(1, 4)$ <sup>5</sup>,  $(2, 4)$ ,  $(3, 10)$ ,  $(4, 0)$ , and  $(5, 7)$ .

Constructing the secret from the shares is the reverse way of the generating. Suppose a peer has collected the first three shares:  $(1, 4)$ ,  $(2, 4)$ , and  $(3, 10)$ . By interpolating the three points into an unknown polynomial  $y(x) = a_0 + a_1x + a_2x^2$ , the peer can get three polynomial expressions with three variables;  $y(1) = 4 = a_0 + a_1 + a_2$ ,  $y(2) = 4 = a_0 + a_1 * 2 + a_2 * 2^2$ , and  $y(3) = 10 = a_0 + a_1 * 3 + a_2 * 3^2$ . By solving the equations with the same modular arithmetic, the peer comes to know the original secret  $a_0 (=10)$ . Note that since there are three unknown variables, one cannot figure out what the original secret,  $a_0$ , is without gathering at least three shares. Detailed mechanisms are described in [16].

## 4 Secret Sharing Based Free-rider Prevention Method

In this section, we first describe our assumptions with relative requirements of incentive mechanisms about P2P systems and present the overview of our method to a secret sharing based incentive mechanism.

### 4.1 Assumptions

1. **Application** : We consider file sharing P2P systems, in which each peer cooperates to download relatively large files such as movies, even though file sharing is not the only tangible application of our approach. In this sense, the BitTorrent network is a good target example to which we can apply our approach.
2. **Transient System** : We assume P2P systems are virtual ad-hoc networks because they are formed by autonomous peers with a common goal of sharing their resources with a frequent churn rate. Peers form a P2P network for their own purpose, but after getting what they want, they will leave the network, which means the network will eventually disappear.
3. **Anonymity** : In most P2P systems, user identifications (IDs) can be easily obtained with no cost, and it is difficult to find the mapping between the IDs and real users. This is, on the one hand, a benefit of P2P system, but on the other hand, makes it hard to implement some applications. In our scheme, we consider the situation where there is no authentication mechanism, system-wide trusted third party, or unique ID for a certain user, which is a common situation in P2P systems.
4. **Selfish Users** : The basic nature of peer is to be selfish. Peers want to use resources of other peers as much as possible without contributing their own, unless they are denied service from others. We do

---

<sup>5</sup> $y(1) = 10 + 2 * 1 + 3 * 1^2 = 15 \% 11 = 4$ . The rests will be calculated in the same way

not consider malicious attacks, such as Denial of Service (DoS) attack to break the system, because there are no gains for the peers who do such kinds of attacks.

Above mentioned assumptions, stemming from the P2P systems, bring the goal of designing our scheme. First, it should be simple to implement and incurs low overhead. Second, it should be scalable so as to endure large populations and high churn rate. Third, it needs to be implemented in a completely distributed manner without the need of any centralized parties who manage the global information of participants on the incentive mechanism. Last but most importantly, there must be a strong regulation for binding the participating peers, so that deviating from the regulation would give a penalty to the peer in any manner.

## 4.2 Overview

As we mentioned in the assumption, we consider a swarm based P2P system like BitTorrent from which peers download a large file by sharing their resources. Similar to BitTorrent there are three parties; the tracker, seeders (original seeders and inherited seeders), and leechers. The role of each party is described below.

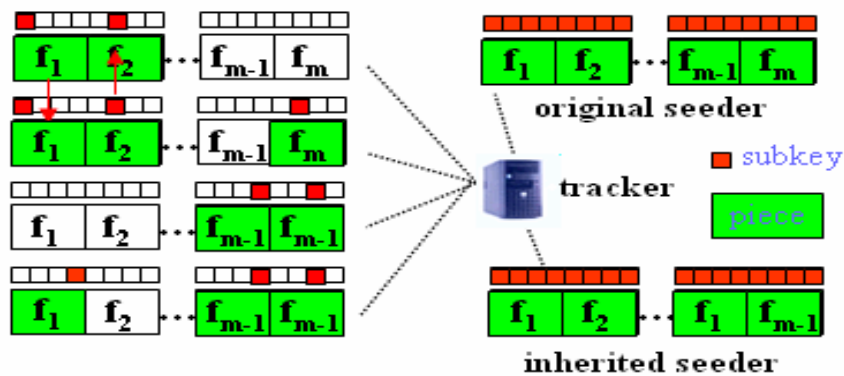


Figure 1: System Overview

### 4.2.1 Seeders

The original seeder first (1) encrypts the file with a key,  $k$ , using a simple encryption/decryption algorithm such as DES with 64 bits. Since the purpose of the encryption is not to protect the encrypted data file ( $E_k(F)$ ), it is not necessary for using a strong encryption algorithm. After encryption, (2) it divides the encrypted data file into  $m$  small pieces ( $f_1, f_2, \dots, f_m$ ) as BitTorrent does. Now, (3) it needs to divide the encryption key,  $k$ , into  $n$  sub keys ( $k_1, k_2, \dots, k_n$ ) based on the  $(t, n)$  threshold secret sharing described in 3.2. If the threshold is  $t$ , then it needs to pick a random  $t - 1$  degree polynomial. Finally, (4) upon request, it distributes a pair  $\langle f_i, k_j \rangle$  to the requesting peer. While distributing a pair, it must be ensured that different pairs go to different peers, so each peer can be interested in each other. As shown in figure 1, the original seeder is the only peer who initially has complete pieces of  $E_k(F)$  and all sub keys in the swarm. The inherited seeder who becomes a seeder by finishing downloading, altruistically uploads data to other peers by doing only step (4). In contrast to the original seeder, inherited seeders have only  $t$  sub keys. However they can generate more sub keys up to  $n$  because they already know what the original polynomial is.

## 4.2.2 Leechers

A peer who wants to download the file needs to (1) download the *.torrent* file from an ordinary public website<sup>6</sup> to join the swarm. Next, (2) it contacts the tracker to get a list containing a random subset of the peers currently in the swarm. Based on the information (3) it tries to contact seeders and other leechers to get pieces of  $E_k(F)$  and sub keys. By contacting, it can either download a pair,  $\langle f_i, k_j \rangle$ , from seeders, or barter a sub key for a piece of  $E_k(F)$  with other leechers and vice versa. During the bartering, each peer should keep the barter rule, “*data first, key later*”, to prevent undesirable side effects, such as key disclosure with free-riding. The rate-based TFT mechanism of BitTorrent can be applied to the bartering between leechers with some more considerations. Exception occurs when a leecher completes its piece download but not sub keys, so called key-hunting mode. In this case, it needs to find a peers who have more sub keys because the peer needs not to download anymore data pieces. The insight behind this is that by doing so the peer can reduce the sub key completion time, which also minimizes the overall download completion time. After, and only after, downloading whole pieces of  $E_k(F)$  and more than  $t$  sub keys, (4) it can figure out what the original encryption key,  $k$ , is and decrypt  $E_k(F)$  with the key. Finally, (5) it may leave the swarm or stay there as an inherited seeder to serve other leechers solely at the peer owner’s discretion.

## 4.2.3 Tracker

The roles of the tracker are exactly the same with BitTorrent network, except it needs to manage the sub key information of each peer in addition to the data information. Please refer to 3.1 for detailed role of the tracker.

## 4.3 Counter Free-riding Features

In this subsection, we describe how the proposed scheme precludes existing free-riding techniques in BitTorrent system. Since free-riders can exploit not only seeders but also leechers, we need to consider both cases.

### 4.3.1 Cheating between leechers

The most common and devastating attack in BitTorrent is cheating between leechers, by which free-riders exploit other leechers’ resources without sharing their own. In rate based TFT, free-riders profit directly from cheating because pieces obtained are portions of the original file. Even though the rate-based TFT adopts a retaliation scheme, by which the deceived leecher can take revenge on the cheater by doing the same thing in future, it is not effective enough. This is because normally there are a relatively large number of peers in most P2P systems, and they usually encounter each other accidentally and make just one-time interactions.

Contrary to the rate-based TFT, in our scheme, there is little benefit from cheating, because the gain from deceiving is almost zero. Suppose a free-rider repeatedly cheats other leechers to collect pieces of  $E_k(F)$  without giving sub keys. Eventually, it can get all pieces of  $E_k(F)$ , but cannot decrypt the file, because of insufficient number of sub keys. Note that the only way a leecher can get sub keys from other leechers is by giving pieces of  $E_k(F)$  to and getting sub keys in return them because of the barter rule; “*data first, key later*”.

---

<sup>6</sup>We do not concern how peers get the *.torrent* file because it is out of scope of this paper.

### 4.3.2 Large View Exploit

Another possible attack is the large view exploit [11, 17, 13] which is a mechanism to enhance the success probability of cheating. In the original BitTorrent implementation, each peer can get a list containing a random subset of the peers of up to 50 in the swarm. But a peer can repeatedly request the list from the tracker and obtain more partial views, which enables the peer to get a larger view compared to normal peers. By getting a larger list, a free-rider can benefit more often from other leechers' optimistic unchoking periods.

The larger view exploit does not affect the soundness of our approach, because free-riders cannot get the sub keys by cheating. To get the sub keys, they should upload their pieces of  $E_k(F)$ , otherwise they cannot.

### 4.3.3 Downloading from only seeders

As described in the cases mentioned above, in our approach, there is little benefit for free-riders to cheat other leechers. They can, however, exploit seeders, instead of exploiting other leechers, because seeders tend to distribute a  $\langle f_i, k_j \rangle$  pair to each requesting peer. Therefore continuous requests for pairs from seeders enable free-riders eventually get all the pieces of  $E_k(F)$  and sub keys, without help from other leechers.

However, if higher priority is given to newcomers and the seeders' capacities are evenly distributed to leechers in the swarm, it takes a relatively longer time for free-riders to complete their downloading compared to that of compliant peers, which suffices as the penalty of free-riding.

### 4.3.4 Whitewashing

In most P2P systems, identities are zero-cost [7], which means that users can easily change their identity with little cost. This is desirable for system growth as it gives some advantages to newcomers to encourage their participation. This, however, allows free-riders to misbehave by switching to new identities to escape the consequences of their actions, which is known as whitewashing [7]. It is hard to deal with the whitewashing in most incentive schemes because players cannot distinguish newcomers from whitewashers. For example, in rate-base TFT, free-riders cannot cheat a leecher twice because of the retaliation mechanism. If the free-riders change their identities, however, they can cheat the same leecher again, because there is no way for the leecher to distinguish them.

In our approach, whitewashers get little advantage out of cheating other leechers because of the reason mentioned 4.3.1. They can benefit from seeders' capacities, because whitewashers can be treated as different users by the seeders. However, we can easily frustrate their intents by giving them the same sub key with different pieces of  $E_k(F)$  based on their IP address. Suppose there are two peers,  $P_A$  and  $P_B$ , with the same IP address. In that case, the seeder treats them as whitewashers and give them  $\langle f_i, k_j \rangle$  and  $\langle f_j, k_j \rangle$  respectively. If they are the same user, it can get two different pieces of  $E_k(F)$  but acquire only one sub key, which means that there is little benefit for the user. However if they are two different users behind same NAT, either pair will be helpful to each of them.

### 4.3.5 Collusion

Reputation based schemes are vulnerable to collusive behavior such as false praise or false accusation, and the negative effects could be critical if the collusion is in conjunction with whitewashing or Sybil attack. It is hard to deal with a collusion in reputation based approaches because there should be not only a system-wide global overview of reputation, but also complicated detection mechanisms mentioned in [7]. What makes it worse is that the result of collusion could affect the entire system.

In our approach, collusion could happen among different peers <sup>7</sup> who are interested in the same file, because they can associate with each other to exchange the sub keys they have. If they are true free-riders, who tend not to upload even a single piece to others, the collusion will be successful, if and only if, the total number of unique sub keys they have is more than  $t$ , which with a large file is a very unlikely situation. If they are partial free-riders who contribute their resources to others to some limited extent, each of them can freely ride with  $\frac{1}{c}$  effort when  $c$  is the number of colluders with different sub keys. In this case, however, the colluders should coordinate each other's effort to get proper  $t$  sub keys.

As a consequence, even though collusion is possible in our approach, it enforces a much stricter condition from free-riders to accomplish, and the effects are limited only to a specific file and not the entire system.

## 5 Evaluation

We evaluate the effectiveness of our method by using various simulations. Since there are not publicly available simulators for a BitTorrent-like system, we have developed our own simulation system that comprises two simulators written in C. We first developed a simulator that behaves an event-driven BitTorrent P2P system. It models almost all peer activities (join, leave, choke, unchoke, optimistic unchoke, and piece exchanges) as well as the key mechanisms of original BitTorrent (rarest first, rate-based TFT, etc) based on the BitTorrent Protocol Specification v1.0 [4] in a simplified way<sup>8</sup>. Note that the results from the simulator totally coincide with other previous results done by many other related works. In addition to the original BitTorrent simulator (Original BT), we also developed a simulator to measure the effectiveness of our method. The underlying mechanisms of our key version BitTorrent (KeyVersion BT) simulator are exactly the same as the original BitTorrent except the seeder's behavior. In the old versions of original BitTorrent, seeders prefer leechers to whom they upload fast, but we modified the seeders' behaviors so that their upload bandwidths are evenly distributed among all leechers in both systems. The reason is that the majority of current BitTorrent implementations follow this approach and newcomers can easily get the first pair and initiate the first barter with other leechers by doing that.

### 5.1 Experiments Setup

The simulation starts with only one seeder and the seeder runs throughout the simulation. Leechers arrive and start to download by contacting other peers (the seeder and other leechers) and exit immediately after they complete their download. We assume a flash crowded situation in which all leechers join the system at the same time in the early state of the simulation which gives the worst case scenario. The seeder's upload bandwidth is 6,000 Kbps and other leechers' upload bandwidths are equal of 800 Kbps. There is no limitation on the download bandwidth of each peer. The target file size is 128MB and the number of pieces ( $m$ ) is 1000. The threshold value ( $t$ ) is the same as the number of pieces and the number of total number of sub keys ( $n$ ) in the system is twice the number of pieces. Unless otherwise specified, we use the above setting in our simulations and all results show the average of 5 runs.

---

<sup>7</sup>We don't need to consider whitewashing because whitewashers can be easily frustrated by our approach as discussed in 4.3.4.

<sup>8</sup>We did not model network propagation delay and model piece level data exchange rather than sub-piece level data exchange with 30 second rechoke and 90 second optimistic unchoke interval.

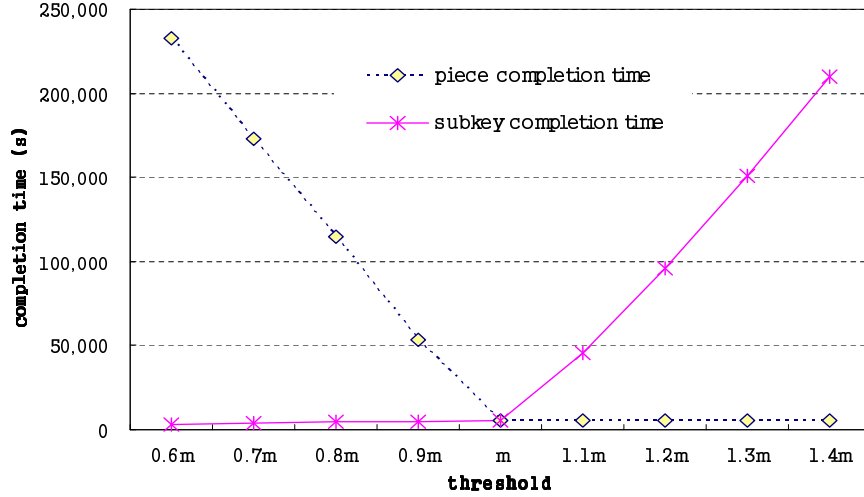


Figure 2: Optimal threshold value

## 5.2 Optimal threshold value, $t$ , of KeyVersion BT

Based on the initial analysis in Appendix, we come to a conclusion that the threshold value ( $t$ ) should be equal to the number of pieces of the file ( $m$ ) to get the optimal system performance. Note that the threshold value is the contribution level how much each peer uploads before it completes its download. Therefore if the threshold is high, each peer should upload more to the system and, in such case, it can get more available bandwidth from the system in return. If the threshold is low, the system imposes less upload overhead on peers, but each peer will suffer from the low available bandwidth because the total system capacity highly depends on the contribution of peers in the system.

Figure 2 illustrates the relationship between the threshold value and the download completion time with 100 leechers. As expected, when the the threshold value is decreased, the sub key completion time (time to get sufficient sub keys) also decreases. However, the piece completion time (time to get all file pieces) sharply increases in inverse proportion to the sub key completion time and vice versa. Therefore we get an optimal download completion time when we set the threshold as the same as the number of pieces.

## 5.3 Relative performance of our approach

As we mentioned before, the underlying mechanisms of the KeyVersion BT are exactly the same as those of the Original BT except the sub key mechanism, which means that the overall performance should be almost the same in both systems.

Figure 3 shows the average download completion times in both systems when there is no free-riders. As shown in the figure, Original BT is highly scalable in that the average completion time of leechers does not increase much even though the number of leechers increases. However the average completion time of leechers in KeyVersion BT sharply increases compared to that of Original BT as the number of leechers increases. The performance degradation is more severe as the number of leechers increases.

This is not surprising. Note that, differ from Original BT, in KeyVersion BT, each leecher can initiate its first barter only after getting at least one pair of  $\langle f_i, k_j \rangle$  from the seeder. Due to the flash crowded situation, however, some leechers should wait long in the seeder's service queue until they get the first pair from the seeder, which causes the huge performance degradation.

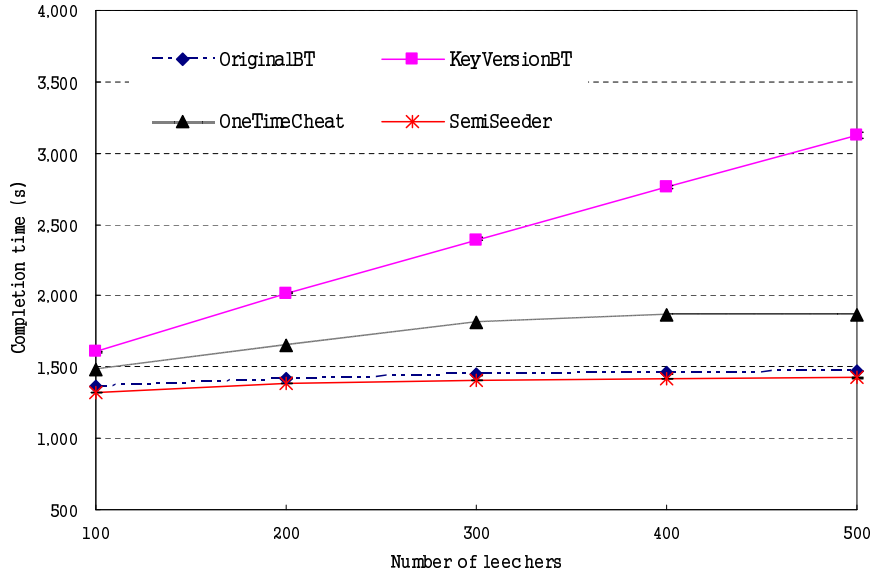


Figure 3: Average download completion time

Once identified, there is a simple solution to address this problem. The major reason for the poor performance was due to the long delay in the seeder’s service queue after a newcomer joined the system. To fix this problem, we introduce “one time cheat” mechanism by which each peer cheats one of other leechers in the early stage to get the first piece so that it can start barter the piece for a sub key with other leechers. By doing that newcomers can initiate their first transactions without such a long delay in the seeder’s queue. As shown in the figure 3, “OneTimeCheat” significantly reduces the initial overhead caused by the seeder bottleneck and also scales well compared to the KeyVersion BT. Even though we allow newcomers to cheat others just one time in the early stage, the soundness of our approach does not deteriorate because there is no benefit for the newcomers to keep cheating others in the long run as explained in 4.3.1.

Even with OneTimeCheat, there still exists a performance gap between our approach and Original BT. This is because of the threshold value,  $t$ , of our approach. The threshold value enforces a certain level of contribution on peers so that each peer should upload certain amount of data to others to complete its download. This threshold value, however, also restricts the altruism of each peer. That is, a peer that has already uploaded that amount of data would not upload anymore, which is different from Original BT in which a peer keeps uploading before it completes the download.

Not to restrict the altruism of peers, we introduce a “Semi-Seeder mode” in which a peer completes its upload but not its download<sup>9</sup> altruistically sends  $\langle f_i, k_j \rangle$  pairs to its neighbors. In this case, free-riders can take advantage of these bandwidths from the semi-seeders, if there is no proper mechanism to prevent it. To avoid such an undesirable situation, each peer maintains contribution histories of its all neighbors and determines to whom it freely sends its  $\langle f_i, k_j \rangle$  pairs based on this information when it becomes a semi-seeder. Contribution level of each neighbor of a peer increases if and only if the neighbor uploads data pieces to the peer. Since these contribution histories are locally maintained by each peer, we do not need to care about the scalability problem.

As shown in the figure 3, the relative performance of “Semi-Seeder” is almost the same as that of Original BT with some improvement. The improvement is due to the threshold value. That is, in Original

<sup>9</sup>We call this type of peer as a semi-seeder in this paper.

BT, a peer can leave the system as soon as it completes its download, but, in Semi-Seeder mode, it cannot leave the system until it completes its upload, which means that the overall system capacity is somewhat higher in our approach than in Original BT. Hereafter whole results of our approach are based on this “Semi-Seeder” approach unless otherwise specified.

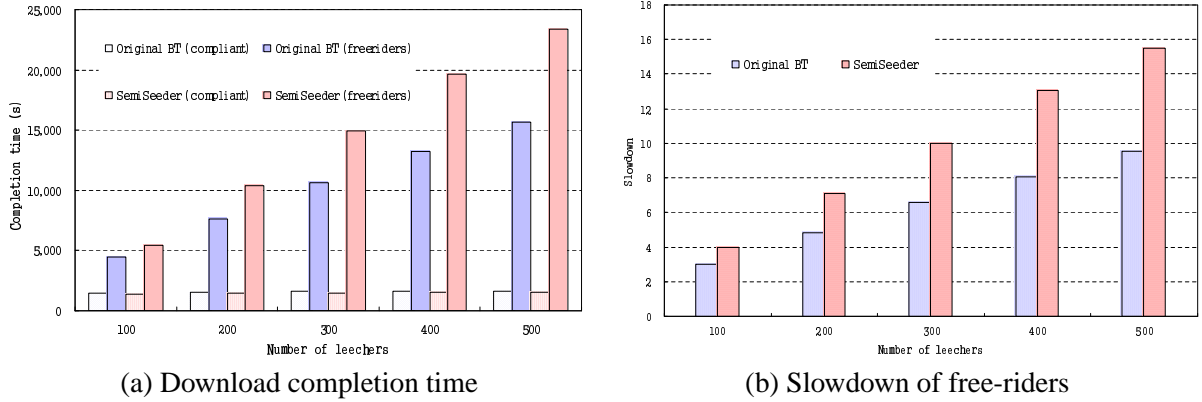


Figure 4: Without the large view exploit (25 % free-rider case)

## 5.4 Free-rider Prevention

In this section, we demonstrate how well both systems prevent free-riders in terms of download completion times of free-riders and compliant users. To do this, we used from 10%, 25 %, and 50 % of free-riders in both systems. Figure 4 (a) compares the completion times of free-riders and compliant users when there are 25% of free-riders in both systems. Figure 4 (b) shows the slowdown of download completion times of free-riders in both systems. In each experiment, the free-riders download data from the network without any contribution. We set the upload bandwidth of the free-rider as zero, which means that free-riders do not upload even a single piece to the system.

Figure 4 shows that the completion times of compliant users in both systems are almost same and the completion times of free-riders are much longer than those of compliant users and increase sharply in proportion to the number of leechers when no free-riding technique is applied to both systems. Even this case, the completion times of free-riders in our system are somewhat longer than those of free-riders in Original BT because free-riders in our system take advantage of only the seeder’s upload capacities not other leechers’, whereas free-riders in Original BT can benefit both bandwidths. Due to the optimistic unchoking of Original BT, theoretically 20% of compliant users’ upload capacities can be utilized by free-riders for free, which is nearly impossible in our approach.

As shown in the figure 4, even in Original BT, free-riders are well penalized by the system if no free-riding techniques is applied. Now, we apply a well known free-riding technique, the large view exploit [11, 17, 13, 14], by which free-riders can enhance the probability of cheating. This is the most famous free-riding technique in this literature.

With the large view exploit, the completion times of compliant users somewhat increase in proportion to the number of leechers and the completion times of free-riders sharply decrease in inverse proportion to the number of leechers in Original BT. This is because free-riders in Original BT with the large view exploit can have a larger number of neighbors compared to that of compliant users, which enables them to have a much higher probability to be selected by compliant users as optimistic unchoking peers. Eventually, as seen in

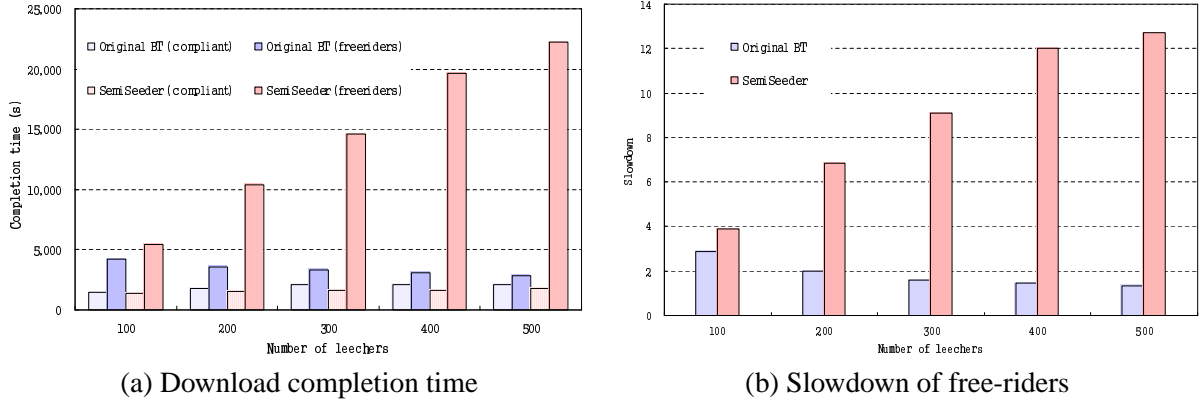


Figure 5: With the large view exploit (25 % free-rider case)

the figure 5, the completion times of free-riders are almost the same as those of compliant users, which is a strong motivation for free-riders to freely ride on the system.

Contrary to Original BT, in our approach, the completion times of free-riders tend not to decrease even though the large view exploit is applied. It's because, differ from the Original BT, in our system, free-riders cannot get any benefit from the upload capacities of compliant users without any contribution to them, but take advantage of only the seeder's upload capacity regardless of the number of leechers, which means that having a large number of neighbors does not helpful to the free-riders in reducing the completion time.

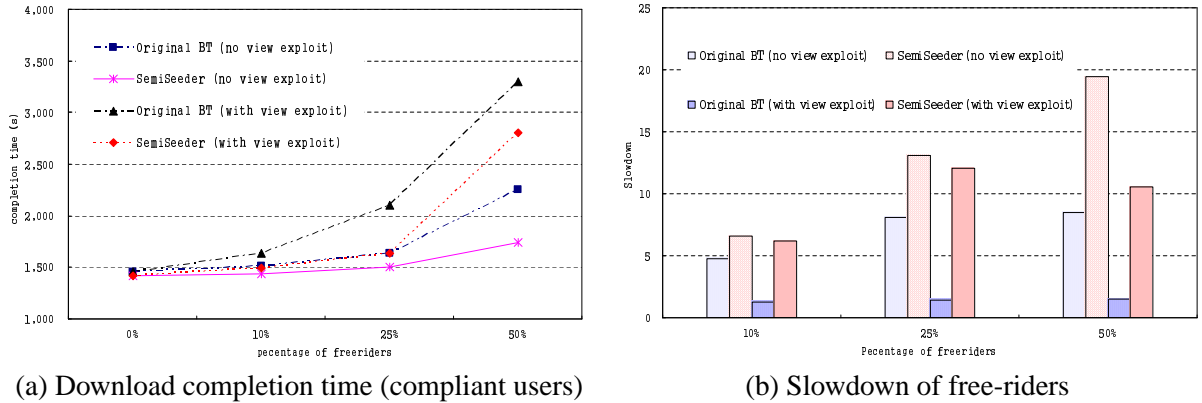


Figure 6: Free-rider effects with different ratios of free-riders

Figure 6 (a) shows the effect of the percentage of free-riders on the download completion times of compliant users where there are 400 leechers in both systems. As shown in the figure, as the number of free-riders increases, the system capacity per peer decreases, which makes the download completion times of compliant users increase as well. We observe that the download completion times of compliant users in Original BT increase faster than those of compliant users in our system even without the large view exploit. This is mainly due to the semi-seeders' behavior in our system. That is, semi-seeders in our system tend to select a compliant user as an optimistic unchoking peer based on the contribution level of their neighbors, whereas compliant users in Original BT randomly select one of their neighbors as an optimistic unchoking peer.

When the large view exploit is applied, the completion times of compliant users in Original BT more

sharply increase than those of compliant users in our system. This is because there are more free-riders among the neighbors of a compliant in this scenario and the compliant users in Original BT are more easily deprived of their upload capacities by the free-riders than those in our system due to the optimistic unchoking. If the percentage of free-riders reaches 50%, compliant users even in our system severely suffer from the performance degradation. It is because, in this case, they cannot fully utilize their upload bandwidths due to the free-riders.

Figure 6 (b) demonstrates the slowdown of the completion times of free-riders. As shown in the graph, the completion times of free-riders in our system increase much faster than those of free-riders in Original BT, which means that our approach more easily discourages free-riders than Original BT. If the large view exploit is applied, the difference between Original BT and our system is obvious. With the large view exploit, the download completion times of free-riders are much longer than those of compliant users in our system (more than 10 times in 50% free-rider case), whereas the download completion times of free-riders in Original BT are only at most 1.5 times longer than those of compliant users. We believe that the much longer download completion times of free-riders in our system will effectively prevent leechers from the desire to freely ride on the system.

## 5.5 Fairness

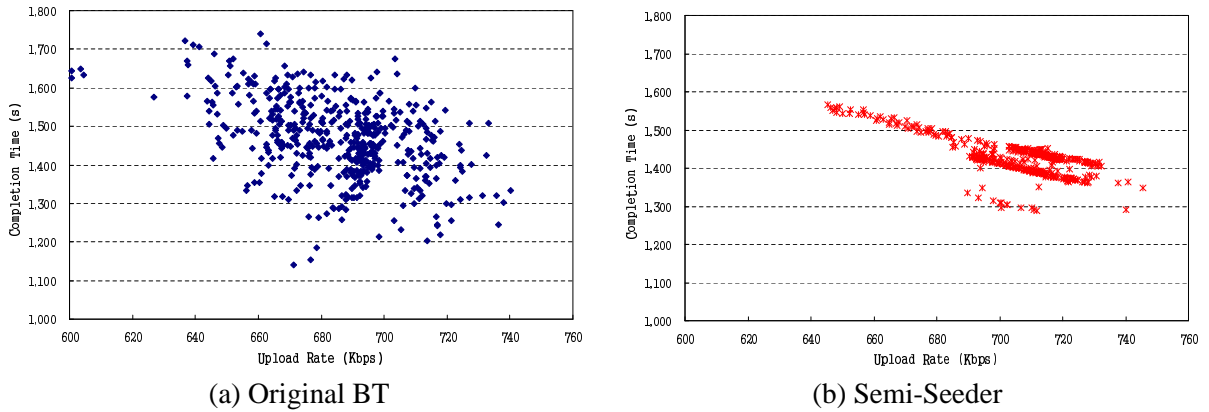


Figure 7: Correlation between download completion time and upload rate

As addressed by many previous studies, there is no strong correlation between the download completion times and the upload rates in Original BT. In our system, however, there exist a strong correlation between the download completion time and the upload rate due to the threshold value. Since a leecher cannot complete its download unless it upload a certain amount of data to others, to reduce the download completion time it should upload fast.

The results of the systems (Original BT and Semi-Seeder) are shown in Figure 7, which focuses the correlation between the upload rate and download completion time. As shown in the figure, Original BT does not have a strong correlation between the upload rate and the download completion time, in which even though a leecher uploads less, it can finish its download about the same time as that of a leecher who uploads much. Our system, however, enforces all peers strong fairness, which means that in this system, if a leecher wants to finish its download fast, it should upload much. In this sense, a free-rider in our system cannot download faster than compliant users.

## 6 Possible Weaknesses and Future Work

In this section, we list possible weaknesses of our approach, discuss how strongly the weaknesses affect the soundness of our approach, and finally present possible solutions to address the problems. The possible weaknesses we consider are computation and communication overheads, seeder bottleneck, and Denial of Service (DoS) attack.

1. **Computation and Communication Overheads** : Since we use a symmetric key algorithm to encrypt and decrypt the original data file, additional computational overheads of seeders and leechers who download the file should be considered. However, the overheads caused by encryption/decryption are quite negligible compared to the transmission delay because it takes less than 1 % of total download time when we use a fast DES implementation such as [5]. Moreover, from the users' point of view, since seeder side overhead is not directly related to their overhead, it is not a major issue.

In addition, the secret sharing scheme we use yields some computation overhead of sub key generation and restoration of original key. In fact the overhead of the secret sharing scheme totally depends on the threshold,  $t$ , because it is accompanied by polynomial interpolations up to  $t$  degree. Even though there exist some efficient algorithms dealing with this problem, it takes a long time if  $t$  is relatively big. Based on the Shamir's Secret Sharing Scheme (SSSS) implementation [18], it takes only a few seconds if  $t$  is less than 100, but it takes several minutes if  $t$  is greater than 1,000, which means that it's not negligible<sup>10</sup>. Compared to the efficiency obtained by our approach, this overhead is not harsh. We will study how we can improve our scheme to minimize the overhead induced by SSSS implementation.

Another possible overhead could be caused by communication to exchange sub keys. This communication overhead is quite negligible though considering that the size of each data piece is 250KB in standard implementation of BitTorrent and the size of each sub key is only 64 bits.

2. **Seeder Bottleneck** : In our scheme, each peer should get at least one pair of  $\langle f_i, k_j \rangle$  from the original seeder to start the exchanging between leechers. So, if there is a large number of newcomers at certain point, it takes quite long time to get the first pair. To address the problem, our scheme allows peers to cheat others just one time at the early stage, which does not impair the soundness of our approach. Suppose that a peer  $P_A$ , cheats one of other leechers to get a piece of  $E_k(F)$ . If the peer is a compliant user, then it will upload the piece to get a sub key, which means that it could get a pair of  $\langle f_i, k_j \rangle$  without helping of the original seeder. If, however, the peer is a free-rider, then as seen 4.3.1 and ??, it cannot complete the downloading until the seeder uploads a full copy of all pieces.

3. **Denial of Service Attack** : Although we do not consider DoS attacks in this paper as mentioned 4.1, it needs to be discussed because such kinds of attacks could be a good target for criticism from opponents. The most plausible and devastating DoS attack is the exposure of the original key by a peer who completes the whole downloading procedure and is about to leave the swarm. It's, however, very unlikely for normal peers to do so because there is no benefit for the peer from doing that.

Even though it is very unusual situation for normal users, malicious nodes who want to collapse the system may try that by broadcasting the original key to all other peers in the swarm. To do that, the

---

<sup>10</sup>If we consider a 1GB file with 250KB piece implementation, it needs 4,000 degree polynomial to implement (4,000, n) threshold secret sharing. In our test, it takes about 4 minutes with degree 4,000 when we use the original SSSS implementation [18] with 64 bit key.

malicious node should first involve the swarm and complete the file downloading before it broadcast the key, which is not a simple work. In addition, contrary to other approaches, in our approach, the key exposure does not mean system-wide performance degradation because the exposed key affects only the file encrypted by the key. To collapse the entire system, the malicious node should get all keys for all files shared throughout the system, which is nearly impossible.

If key exposure is detected, the system should remedy the state. One possible approach is to back to the original rate-based TFT mode and re-initiate the swarm with different key to minimize the negative effects. We will try to find other possible alternate approaches to improve the soundness of our scheme.

In addition, we need to study what happens when we apply our approach to heterogeneous environment because we consider only homogeneous environment in this paper. Moreover, we will investigate the effects of different free-riding techniques on our system in the future.

## 7 Conclusion

In this paper, we present a unique free-riding prevention technique based on secret sharing and experimentally demonstrate the effectiveness of our approach. With our approach, it's extremely hard for free-riders to freely ride on the system because there is a strong correlation between the upload rate and the download completion time of each peer. The only way a free-rider can complete its download is to download purely from seeders, but it takes long time to complete its download compared to that of compliant peers. Repeatedly cheating other leechers does not help free-riders with reducing their download completion time at all because by doing that they cannot get sufficient number of sub keys, which means that they cannot decrypt the original file.

## References

- [1] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Networks Performance Mechanisms," in *Proceeding of IEEE INFOCOM'06*, Barcelona, Spain, April 2006.
- [2] *Bittorrent Protocol Specification v1.0*. [Online]. Available: <http://wiki.theory.org/BitTorrentSpecification>
- [3] B.-G. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz, "ChunkCast : An Anycast Service for Large Content Distribution," in *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS'06)*, February 2006.
- [4] B. Cohen, "Incentives build robustness in bittorrent," in *Proceeding of Workshop on Economics of Peer-to-Peer Systems (P2PEcon'03)*, Berkeley, CA, May 2003.
- [5] F. Corella, "A Fast Implementation of DES and Triple-DES on PA-RISC 2.0." in *Proceedings of the First Workshop on Industrial Experiences with Systems Software (WIESS 2000)*, October 2000, pp. 83–84.
- [6] M. Feldman and J. Chuang, "Overcoming free-riding behavior in peer-to-peer systems," *ACM Sigecom Exchanges*, vol. 5, no. 4, pp. 41–50, July 2005.

- [7] M. Feldman, K. Lai, I. Stoica, and J. Chuang, “Robust Incentive Techniques for Peer-to-Peer Networks,” in *Proceeding of ACM Conference on Electronic Commerce (EC’04)*, May 2004.
- [8] P. Garbacki, A. Iosup, D. Epema, and M. van Steen, “2Fast: Collaborative Downloads in P2P Networks,” in *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P’06)*, September 2006, pp. 23–30.
- [9] G. Hardin, “Tragedy of the commons,” *Science*, vol. 162, December 1968.
- [10] D. Hughes, G. Coulson, and J. Walkerdine, “Free Riding on Gnutella Revisited: The Bell Tolls?” in *IEEE Distributed Systems Online*, June 2005.
- [11] S. Jun and M. Ahamad, “Incentives in BitTorrent Induce Free Riding,” in *Proceeding of 3rd ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, August 2005.
- [12] H. T. Kung and C.-H. Wu, “Differentiated Admission for Peer-to-Peer Systems: Incentivizing Peers to Contribute Their Resources,” in *Proceeding of Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, June 2003.
- [13] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, “Free Riding in BitTorrent is Cheap,” in *In HotNets*, November 2006.
- [14] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, “Do incentives build robustness in BitTorrent?” in *4th USENIX Symposium on Networked Systems Design and Implementation*, May 2007, pp. 83–84.
- [15] D. Qiu and R. Srikant, “Modeling and Performance Analysis of BitTorrent-like Peer-to-peer Networks,” in *ACM Sigcomm*, August 2004.
- [16] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
- [17] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, “Free-riding in BitTorrent Networks with the Large View Exploit,” in *Proceeding of Sixth International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2007.
- [18] *Shamir’s Secret Sharing Scheme Implementation*. [Online]. Available: <http://point-at-infinity.org/ssss/>
- [19] K. Tamilmani, V. Pai, and A. Mohr, “SWIFT: A System With Incentives For Trading,” in *Proceedings of Second Workshop of Economics in Peer-to-Peer Systems (P2PECON)*, June 2004.
- [20] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, “Karma: A Secure Economic Framework for Peer-to-Peer Resource Sharing,” in *Proceeding of Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, June 2003.

# Appendix

## A Analysis of the optimal threshold value

We assume a homogeneous system in which all leechers have the same upload capacities and the download capacity of each leecher is infinite. There is only one seeder in the system and the initial start-up time which is needed for all leechers to have at least one pair of  $\langle f_i, k_j \rangle$  is negligible, which means that each leecher can initiate its first transaction immediately after it joins the system. The total number of sub keys in the system is assumed as infinite so that each peer gets a unique sub key from the seeder whenever it gets a pair, which will simplify the analysis.

$m$	the number of data pieces
$p$	the size of each piece
$t$	the threshold (# of sub keys needed to decrypt the file)
$N$	the number of leechers
$\mu_s$	the upload capacity of the seeder
$\mu_l$	the upload capacity of a leecher
$T_k$	the sub key completion time
$T_p$	the piece completion time
$D(r)$	the available download bandwidth of each leecher

Table 1: Summary of Notations

### A.1 Download completion time

The download completion time of a leecher in our system is the maximum between the sub key completion time (time needed to collect at least  $t$  sub keys) and the piece completion time (time needed to download the  $m$  pieces). Note that a leecher can get a sub key when it uploads a piece to other leecher except when the sub key is given by the seeder. So, if there is no help of the seeder, a leecher should upload  $t$  number of pieces to others to collect sufficient sub keys.

### A.2 Sub key completion time ( $T_k$ )

When the threshold value is  $t$ , the total amount of data uploaded by all leechers will be  $t * p * N$ , but some amount will be compensated by the seeder in the meantime. So, the real amount of data uploaded by each peer during  $T_k$  is  $(t * p * N - \mu_s * T_k)$ . Since the upload bandwidth of each peer is  $\mu_l$ ,  $T_k = \frac{t * p * N - \mu_s * T_k}{N * \mu_l}$ . Solving this equation gives us

$$T_k = \frac{t * p * N}{\mu_l * N + \mu_s} \quad (1)$$

Based on the equation 1, we conclude that the sub key completion time depends on the threshold value. That is, if  $t$  is low, the sub key completion time of each peer is short, but if  $t$  increases, then the sub key completion time increases in proportion to the threshold value. So if we consider only the sub key completion time, the lower, the better.

### A.3 Piece completion time ( $T_p$ )

Since the file size is  $m * p$ , the total amount of data downloaded by all leechers in the system is  $m * p * N$ . In this case, the piece completion time of each peer is the time to download all pieces. Now, we need to consider the available download bandwidth ( $D(r)$ ) which each leecher can actually get from the system. Since we assume that the download bandwidth of each leecher is infinite, the download completion time totally depends on the available download bandwidth.

Note that  $D(r)$  depends on how much the leechers and the seeder upload to the system. We think three different situations. (1) If the threshold,  $t$ , is zero (nobody uploads except the seeder), then  $D(r) = \frac{\mu_s}{N}$ , which is a client/server model. (2) If  $t < m$ , the piece completion time,  $T_p$ , will be longer than the sub key completion time,  $T_k$ , because the leechers cannot download faster than their upload rates. In this case, since leechers upload  $(t * p * N - \mu_s * T_k)$  and the seeder uploads  $\mu_s * T_k$  during  $T_k$ , the total is  $t * p * N$ . So, during  $T_k$ ,  $D(r1) = \frac{t * p * N}{T_k * N} = \frac{t * p}{T_k}$ . Between  $T_k$  and  $T_p$ , however, since no one uploads except the seeder,  $D(r2) = \frac{\mu_s}{N}$ . Since each leecher should download  $m * p$ ,  $D(r1) * T_k + D(r2) * (T_p - T_k)$  should be  $m * p$ . Solving this equation by plugging in the equation 1 gives us

$$T_p = T_k + \frac{(m - t) * p * N}{\mu_s} \quad (2)$$

(3) If  $t \geq m$ , then the download completion time will be dominated by the sub key completion time. In this case it's not important how fast the piece completion time is.

From the equation 1 and 2, we conclude that the threshold value should be the same as the number of pieces, in which each peer achieves an optimal performance.