Analysing BitTorrent's Seeding Strategies

Xinuo Chen
Department of Computer Science
University of Warwick
Coventry, U.K.
xinuo.chen@dcs.warwick.ac.uk

Stephen A. Jarvis
Department of Computer Science
University of Warwick
Coventry, U.K.
saj@dcs.warwick.ac.uk

Abstract— **BitTorrent is a typical peer-to-peer (P2P) file distribution application that has gained tremendous popularity in recent years. A considerable amount of research exists regarding BitTorrent's choking algorithm, which has proved to be effective in preventing freeriders. However, the effect of the seeding strategy on the resistance to freeriders in BitTorrent has been largely overlooked. In addition to this, a category of selfish leechers (termed exploiters), who leave the overlay immediately after completion, has never been taken into account in the previous research. In this paper two popular seeding strategies, the Original Seeding Strategy (OSS) and the Time-based Seeding Strategy (TSS), are chosen and we study via mathematical models and simulation their effects on freeriders and exploiters in BitTorrent networks. The mathematical model is verified and we discover that both freeriders and exploiters impact on system performance, despite the seeding strategy that is employed. However, a selfish-leechers threshold is identified; once the threshold is exceeded, we find that TSS outperforms OSS – that is, TSS reduces the negative impact of selfish lechers more effectively than OSS. Based on these results we discuss the choice of seeding strategy and speculate as to how more effective BitTorrent-based file distribution applications can be built.**

Keywords-BitTorrent; Seeding strategy; peer-to-peer; file distribution

I. INTRODUCTION

In a traditional client/server file distribution paradigm, a server takes responsibility for transmitting data to all clients. This service model is limited in scalability, especially when the files are large. As a successful Peer-to-Peer file-sharing system, BitTorrent [3], solves this problem by dividing a large file into many small sized blocks and encouraging clients to exchange blocks during their downloading processes. This mechanism reduces load on the server and improves the system service capacity.

Like many other peer-to-peer systems, BitTorrent faces the challenge of *freeriders* [4, 5], which are peers who never upload blocks to others. A peer can act as a freerider by setting the upload rate to a very low value or even zero. Fortunately, BitTorrent can effectively penalise those freeriders using its Tit-For-Tat (TFT) policy that determines how peers with incomplete files (called *leecher*s) exchange blocks. With the TFT policy, all leechers exchange blocks only with those who upload to them at a higher rate, thus freeriders cannot obtain blocks because they never upload. While most previous research [3, 5-7] focuses on the behaviour of leechers, the role of the *seed*s (peers with complete files), in the process of preventing freeriders has been largely overlooked.

Since seeds own all the data blocks, they are dedicated to uploading to others. Seeds use a seeding strategy to decide which leechers to serve. Currently, a widely used seeding strategy, the *Original Seeding Strategy* (OSS), ensures that seeds upload to leechers which have the highest download rates, in the hope that new seeds can be produced quickly, which can then serve others. However, when freeriders with relatively high download bandwidths exist in the overlay, due to the mechanism of OSS, there is a possibility that those freeriders dominate the resources of seeds and delay the downloading processes of other unselfish leechers. Thus the OSS strategy may benefit freeriders rather than necessarily fostering contribution. In order to solve this problem caused by freeriders, a new seeding strategy, called the *Time-based Seeding Strategy* (TSS), was proposed in [8]. By employing this strategy, seeds serve each leecher in turn and for the same amount of time so that no single leecher (including freeriders) can dominate the resources of seeds. However, the negative side of TSS is that the speed of producing fresh seeds in the overlay is slowed down and eventually the overall performance may be impacted. Due to the lack of a comprehensive analysis of TSS in [8], it is not clear whether TSS is better than OSS in preventing freeriders and this remains an open question.

Furthermore, an issue largely ignored is that the newly generated seeds can choose not to act as expected – they may stay for only a short time [9, 10] or simply quit the system once they have obtained the whole file. In this paper, we term these types of peers *exploiters*, i.e., they serve others while downloading, but quit the overlay immediately after its completion. So far little attention has been paid to the influence of exploiters, therefore it is unclear how OSS and TSS are resilient to their behaviour.

In order to answer these questions and direct the selection of seeding strategy for BitTorrent clients, we conduct a comprehensive analysis of OSS and TSS. First we establish a mathematical model to present OSS's effectiveness in reducing the impact of selfish leechers (freeriders and exploiters) in a homogeneous environment where all peers have identical downlink and uplink bandwidths. We then introduce BitTorrent simulation experiments and provide experimental results that verify our model and compare TSS with OSS. The investigation is then extended to a heterogeneous environment. We show that under either OSS or TSS, freeriders and exploiters degrade system performance. If the number of

selfish leechers increases, the performance of OSS-led BitTorrent drops faster but is still better than a TSS-led version. However, there is a threshold for the scale of the selfish leechers. Once the threshold is met, TSS performs better than OSS and presents a predominate resistance to freeriders and exploiters.

The remainder of this paper is organised as follows. Section II presents an overview of BitTorrent and documents related work; Section III analyses the different seeding strategies using a mathematical model; Section IV describe our simulation methodology; Section V and VI discuss the simulation results; finally Section VII concludes the paper and describes future work.

II. BACKGROUND AND RELATED WORK

BitTorrent employs a series of sophisticated mechanisms to encourage peers to upload data to each other, and thus achieves scalable and highly efficient content distributions. In this section, we first give an overview of BitTorrent. Several key factors behind the success of the BitTorrent protocol, such as the *choking algorithm* and the *local-rare-first mechanism*, are already described in previous research [2, 3, 7, 8, 11]. We thus present only the two popular *seeding strategies*, OSS and TSS, in detail. Throughout this paper, we use terminology first introduced in [8].

A. BitTorrent Mechanism

Prior to the content distribution, the provider splits the file into a number of pieces and obtains the SHA-1 hashes for all pieces. Together with the IP address and port number of the tracker, the provider encapsulates the piece information into a torrent. Normally the provider itself will then connect to the tracker and thus become the initial seed. After peers retrieve the torrent, they obtain the IP address and port number of the tracker and then join the BitTorrent overlay through the tracker. From this time point, peers become leechers or if some peers already have the complete set of pieces when they join, they become initial seeds (we do not consider this case in our research). Every peer updates its own peer set by obtaining a peer list, which contains a random set of peers (their IP addresses and port numbers), from the tracker at a certain time interval. Seeds upload data to leechers using the *seeding strategy*, while leechers interact with each other, i.e., decide who to download data from or who to upload data to, by executing the *choking algorithm*. When starting to download data from others, leechers decide which file piece to retrieve by following the guidance of the *local-rarest-first algorithm* (LRF). Once a leecher finishes downloading, it either rejoins the overlay to be a seed or leaves immediately so becoming an exploiter.

B. Seeding Strategy

The initial seed and the regular seeds that come from leechers all have a complete set of file pieces and thus do not need any data from others. Note that the choking algorithm uses uploading rates of leechers to decide whom to upload to, thus it is not applicable for seeds anymore because seeds cannot calculate other leechers' uploading rates. In order to contribute the distribution overlay optimally, seeds employ seeding strategies. There are currently two deployed seeding strategies: the *original seeding strategy* (OSS) and the *time-based seeding strategy* (TSS). These are described in the following text and further analysed in Section III.

Algorithm 1: OSS, invoked by seeds every 10 seconds

```
remove optimistically unchoked leecher from the interested_leecher_list
for every leecher in the interested_leecher_list do
    calculate the rate (download_rate) at which the leecher downloads from this_local_seed
end for
sort interested_leecher_list in descending order based on the leecher's download_rate
for i ← 1 to 3 do
    unchoke interested_leecher_list(i)
end for
```

Algorithm 2: TSS, invoked by seeds every 10 seconds

```
global variable: t
sort the interested_leecher_list based on the leecer's last unchoke time, with the most recently unchoked leecher last
if t = 0 then
    for i ← 1 to 3 do
        unchoke interested_leecher_list(i)
    end for
    int r ← random integer between 4 and n (n is the number of interested leechers)
    unchoke interested_leecher_list(r)
    t ← t + 1
else if t = 2
    for i ← 1 to 4 do
        unchoke interested_leecher_list(i)
    end for
    t ← 0
end if
```

OSS (see algorithm 1) has been employed since BitTorrent was invented [3]. In the BitTorrent distribution process, there are seeds that always stay in the overlay for a limited time period [7, 11]. Thus, how to force seeds to maximally contribute to the overlay before they leave becomes a problem that OSS aims to solve. In following OSS, seeds upload data to leechers whose downloading rates are highest. In other words, seeds intend to upload data as quickly as possible. There are three aspirations behind this process: 1. seeds can deliver a maximum number of pieces to leechers; 2. leechers that download from seeds can become new seeds quickly; 3. new seeds can continue to serve the remaining leechers.

TSS (see algorithm 2) was introduced in the official BitTorrent client 4.0.0 [8]. In following TSS, seeds upload data to leechers uniformly. In other words, seeds serve each of their neighbour leechers in turn based on the time stamp of last service, regardless of the leechers' download rates. A seed can perform *s* parallel uploads. After a seed has been uploading data to a leecher for sixty seconds (typically), it chokes the leecher and selects another leecher to serve. In this manner, all leechers that are connected to a seed will be served for similar time period. The purposes of this strategy

is: 1. To prevent any single leecher from monopolising seeds; 2. To reduce the amount of duplicate data a seed needs to upload before it contributes a full set of file pieces to the overlay.

C. Related Work in the Analysis of BitTorrent

There is a large body of literature on analysing BitTorrent's mechanism.

Qiu *et al.* [7] construct a fluid model for BitTorrent and indicates that the mean download completion time of leechers does not relate to the peer arrival rate. However, the success of the distribution of a file is related to the number of freeriders in the overlay.

Fan *et al.* [12] investigate how BitTorrent might implement incentives and thus prevent freeriders. Felber *et al.* [13] conduct a simulation to investigate how BitTorrent-like protocols handle flash-crowds.

Bharambe *et al.* [2] use an event-driven simulator to comprehensively evaluate the performance impacts from the core algorithms that `BitTorrent employs. They find that the choking algorithm is very effective and the local-rarest-first algorithm outperforms alternative piece selection strategies. Their work also makes clear that the initial seed is very important and it should distribute a full set of pieces into the overlay as quickly as possible.

Iza *et al.* [9] conduct a comprehensive analysis on the data that is derived from the tracker log of the most popular Redhat 9 torrent. They observe that the mean downloading rate of leechers is 50KB/s, which indicates the good connectivity that most leechers have.

Pouwelse *el al.* [10] present a measurement study on BitTorrent. The major contribution of their work is the analysis of the flashcrowd effect, i.e., the phenomenon of the sudden popularity of a new file distribution.

Massoulie *et al.* [14] introduce a probability model of *coupon replication systems*. The major conclusions, which are directly related to BitTorrent, are 1. the peer arriving or departing rate does not affect the system performance significantly; 2. the efficiency of the distribution does not critically depend on the local-rarest- first algorithm.

Our work on BitTorrent differs from previous work in the following respects. First, we identify a special kind of selfish leecher, which we term an *exploiter*. We show the impact of these on system performance; degradation is less sever than that caused by freeriders. Second, we analyse two seeding strategies (OSS and TSS) in detail and focus on their resistance to the selfish lechers, which no previous research has conducted. Finally, we show through experimentation that choosing seeding strategy to adapt to the scale of the selfish leechers in the overlay is very important. We thus propose in our future work that a mechanism is needed to detect the scale of the selfish leechers so that the seeding strategy can be loaded or changed dynamically.

III. Model for Seeding Strategy

As described in Section II, there are currently two kinds of seeding strategies, the *original seeding strategy* (OSS) and the *time-based seeding strategy* (TSS), which have been deployed through different BitTorrent client applications. In this section, we present a mathematical analysis to investigate the impact that freeriders or exploiters have on the mean download completion time of unselfish leechers if OSS is employed.

A. Metrics, Assumptions and Scenarios

Leechers join a BitTorrent network in order to obtain a complete shared file from the overlay. They have one major concern - the time they need to spend to complete the downloading. Since uploading data is the driving force of a BitTorrent system, unselfish leechers are important to the overlay and the quality of the service that they get from the distribution should be kept at a certain level; otherwise, the fairness in the system is violated. Thus, we choose the *mean download completion time* of unselfish leechers, denoted by t_u, in the system as the metric of our mathematical analysis. Using this metric, we can investigate how freeriders and exploiters affect the fairness and the benefits that unselfish leechers obtain if BitTorrent employs OSS.

Another concern that leechers have is whether a complete set of blocks of the file can be finally obtained. [7] suggests that the probability of a leecher finding a desired block among its neighbours is very close to one, therefore, for simplicity of analysis, the first assumption that we make is that at any point in time, the network owns the complete set of blocks of a file.

The environment in which a file is distributed is assumed to be homogeneous. This is the second assumption that we make. Initially, there are N leechers and one initial seed. All peers have identical uplink bandwidth of u, but various downlink bandwidth of d_i. The size of the file being distributed is A.

Two other assumptions are made:

- A peer's downloading bandwidth is not the bottleneck of data transfer. The experimental results in [7] suggest that the mean utilization of d_i is practically quite low (20-40%). Therefore, it is reasonable to assume that the downlink bandwidth of each leecher will not constrain the downloading process.
- Peers have the same uplink bandwidths u, and the mean utilisation of peers' uplink bandwidths is 100% [2]. We will consider the heterogeneous setting where the uplink bandwidths are not equal in our simulation study.

Three scenarios are considered in the model. All leechers in the first scenario are unselfish leechers. In the second scenario, a number F of leechers are freeriders and the remainder of the leechers are unselfish. In the final scenario, only unselfish leechers and a number E of exploiters are present in the network. The different t_u for the three scenarios are calculated to show whether OSS can guarantee fairness in the system.

B. The Model

In all three scenarios, unselfish leechers act as seeds for a mean time T_s after finishing downloading, regardless of whether they are being served by seeds. T_s is assumed to be a constant value, which is long enough to let new seeds contribute sufficient blocks to the network [2, 11]. Let T_{s0} de-

note the staying time of the initial seed. Freeriders and exploiters have higher downlink bandwidths than unselfish leechers. The mean download time of exploiters is denoted by T_E.

In the first scenario, all the unselfish leechers keep uploading to others before they become seeds. The amount of data that they upload is uNt_u. After that they become seeds, they stay for a mean period of time T_s, hence, the amount of data they send out to the system is uNt_u. The initial seed contributes to the network with the amount of data uT_{s0}, where T_{s0} is long enough to ensure that at least one copy of the entire file blocks are distributed into the network.

In the distribution process of the whole file (i.e., within the period of the torrent lifetime), each of the N leechers eventually obtains a complete copy of the file. Therefore, an amount NA of data has been uploaded by all peers. Thus, we have $NA = uNt_u + uNT_s + uT_{s0}$ and t_u is given by

$$t_u = \frac{NA - uNT_s - uT_{s0}}{Nu} = \frac{A}{u} - T_s - \frac{T_{s0}}{N} \tag{1}$$

When the F freeriders with the highest download rates join the system, the seeds pass on all of their contributions to the freeriders (following the OSS policy) before they, or the freeriders, leave the network. Because the freeriders do not upload blocks at all, all data are distributed through only the unselfish leechers and the seeds; that is, $NA = (N-F)ut_u^{'} + (N-F)uT_s + uT_{s0}$.

In the second case, $t_u^{'}$ is given by

$$\begin{aligned} t_u^{'} &= \frac{NA - (N-F)uT_s - uT_{s0}}{(N-F)u} \\ &= \frac{N}{N-F} \cdot \frac{A}{u} - T_s - \frac{T_{s0}}{N-F} \end{aligned} \tag{2}$$

To show the performance degradation caused by the freeriders, the increase rate (IRD) of mean download completion time of unselfish leechers is

$$IRD = \frac{t_u^{'} - t_u}{t_u} = \frac{F}{N-F} \cdot \frac{\frac{A}{u} - \frac{T_{s0}}{N}}{\frac{A}{u} - \frac{T_{s0}}{N} - T_s} > \frac{F}{N-F} \tag{3}$$

In [4], it is discovered that nearly 70% of Gnutella users share no files. If we assume F to be 70%, IRD will be more than 230%, which implies that unselfish leechers will have to spend 230% more time on downloading because of the existence of freeriders. At an early stage, freeriders dominate the resources of initial seeds, and unselfish leechers seldom exchange blocks with each other, because new blocks are rarely delivered to them. This blank transfer period causes a significant increase to the download completion time of unselfish leechers. Note that when $T_{s0} = NA/u$, $P = 0$. This implies that when the initial seed stays in the network for a sufficiently long period of time, the mean download completion time of unselfish leechers will not be increased.

If an alternative strategy, TSS (see algorithm 2), is applied, the freeriders will not be the only consumers of the initial seed. The seed tries to equalise its contributions to every leecher in the system. The unselfish leechers have equal chance to obtain new blocks from the initial seeds and the blank transfer period no longer exists. The mean download completion time of the unselfish leechers is thus reduced. This result will be further demonstrated in the simulation experiments.

In the final scenario, the E exploiters keep exchanging blocks with the leechers while they receive data from the seeds. However, the exploiters will leave the network as soon as they finish downloading; that is, they do not stay as seeds for a period of time T_s. Therefore, $NA = (N-E)ut_u^{''} + Eut_E + (N-E)uT_s + uT_{s0}$ and $t_u^{''}$ is given by

$$\begin{aligned} t_u^{''} &= \frac{NA - Eut_E - (N-E)uT_s - uT_{s0}}{(N-E)u} \\ &= \frac{N}{N-E} \cdot \frac{A}{u} - \frac{E}{N-E} \cdot t_L - T_s - \frac{T_{s0}}{N-E} \end{aligned} \tag{4}$$

where t_E is the mean download time of the E exploiters.

To show the performance degradation, $IRD^{'}$ is calculated

$$\begin{aligned} IRD^{'} &= \frac{t_u^{''} - t_u}{t_u} = \frac{E}{N-E} \cdot \frac{\frac{A}{u} - \frac{T_{s0}}{N} - t_E}{\frac{A}{u} - \frac{T_{s0}}{N} - T_s} \\ &= \frac{E}{N-E} \cdot \left(1 - \frac{t_E - T_s}{\frac{A}{u} - \frac{T_{s0}}{N} - T_s}\right) \end{aligned} \tag{5}$$

Note that $t_E < t_u$ because exploiters tend to be served by seeds with a higher priority, thus we have

$$IRD^{'} > \frac{E}{N-E} \cdot \frac{T_s}{\frac{A}{u} - \frac{T_{s0}}{N} - T_s} \tag{6}$$

The negative impact that the exploiters bring to the system is less than that of the freeriders. When t_E increases, $IRD^{'}$ drops. The reason for this is that if the exploiters have to stay as leechers in the network for a longer period of time, they serve more blocks to the unselfish leechers and the download rates of the unselfish leechers therefore increase. OSS does not try to force exploiters to stay as leechers for a longer period of time; on the contrary, it helps the exploiters finish downloading in a faster manner if exploiters have higher download rates.

IV. Simulation Methodology

In this section we conduct simulations to 1) verify our mathematical model which is presented in Section III; 2) analyse seeding strategies in both homogeneous and heterogeneous network environments; 3) further explore the performance impact of seeding strategies using more comprehensive metrics.

A. Simulator Details

A discrete-event-based simulator written in Java is implemented to simulate peer activities, such as joining, leaving

and block exchanging, as well as most of the BitTorrent mechanisms (the local-rare-first algorithm, the choking algorithm, OSS, TSS, etc). In the simulator, we treat every BitTorrent or network related operation as an event. Every event is associated with an event timestamp. The event timestamp does not fit the real time and it is a relative time indicator for events. In the simulation, we preset some particular launching events and event chains will be formed because one event may lead to another. For example, when a peer becomes able to serve others, the first choking round occurs and a preset choking event for the peer is generated. When the event finishes, it will generate or schedule the next choking round which will occur 10 seconds later. Note that the *10 seconds* is the timestamp inside the simulation. All events, including both the preset and the newly generated ones, are pushed into a priority queue, sorted by event time. The events in the event queue are polled one by one and executed.

Each peer in the simulator is associated with a downlink and an uplink bandwidth, which resemble asymmetric network access as widely observed today. Based on these bandwidth settings, the simulator calculates the block transfer delay in the following way. When peer *A* is going to send a block to peer *B*, the simulator retrieves *A*'s upload bandwidth and *B*'s download bandwidth. The bandwidth with the least value is selected and the time delay is calculated through dividing the block size by the selected bandwidth value. The simulator then schedules a block-received event with the time delay for peer *B*. When the event is executed, peer *B* receives the block and its file block set is updated.

Since the time delay computation for each block transmission is expensive, we make a number of simplifications that have negligible impact on the performance aspects we are considering. For example, we ignore the interaction of BitTorrent control packets, which are normally very small compared with data block size. Following Akella et al. [15], network bottlenecks are assumed to mainly occur in the edges of networks.

We simulate BitTorrent mechanisms, such as its choking algorithm, local-rare-first algorithm, OSS, TSS, as comprehensively as possible. However, BitTorrent performance can be influenced by many configuration parameters at client side or the network condition of peers. Since our study is to find the performance impact of seeding strategies, we do not try to find a set of optimal parameters for BitTorrent performance but use those parameters proposed in related research [2, 8].

B. Metrics

In term of fairness, we focus on the benefits that the unselfish leechers can obtain in the BitTorrent distribution process. Hence the metrics we select for the simulations are all for unselfish leechers.

Mean download completion time of unselfish leechers: In the mathematical model, we use the *mean download completion time* of unselfish leechers as the metric to compare seeding strategies where freeriders or exploiters exist. Since the purpose of BitTorrent is to distribute a file among a number of peers, the mean download completion time indicates the efficiency of the distribution. Using this metric we can investigate the overall system performance. In the simulation experiments we continue to validate our model and extend the BitTorrent overlay environment from a homogenous to a heterogeneous setting. The download completion time of every unselfish leecher is summed and a mean value calculated.

Download rate and bandwidth utilisation: In the experiments with homogeneous settings, we use download rate and bandwidth utilisation of unselfish leechers as metrics to indicate performance differences between OSS-led and TSS-led BitTorrent overlays where selfish leechers exist.

Cumulative distribution of unselfish leechers' download completion times: For the experiments with heterogeneous settings, we plot all cumulative distributions of the download times of all unselfish leechers. This metric will help us to understand how individual leechers perform.

TABLE I. BANDWIDTH DISTRIBUTION OF LEECHERS (DERIVED FROM ACTUAL DISTRIBUTION OF THE GNUTELLA NETWORK [1, 2])

Downlink (KB/s)	Uplink (KB/s)	Fraction of leechers
78	12	0.3
150	38	0.6
300	100	0.1

C. Setup of Experiments

Since our study focuses on the seeding status of the distribution, i.e. the finishing period, we assume a steady-state network where all leechers are already present. Bharambe et al. [2] also use a set of peer bandwidths, which was derived from the Gnutella study presented in [2]. We borrow the bandwidth values from [2] and assign them to peers in our simulation experiments.

In term of peers' bandwidths, our experiments have two settings: homogenous and heterogeneous. In the homogenous setting, all leechers have the same network bandwidths that we choose from the bandwidth values shared by [1, 2]. In the heterogeneous setting, the link bandwidths of leechers follow the distribution which is presented in [2].

The experimental setup is summarised as follows:

- File size: A = 200MB; piece size: 256KB; block size: 16KB
- Number of initial seeds: 1
- All unselfish seeds stay in the overlay for 1000 seconds
- Peers' bandwidth:
  - Initial seed uplink: 50KB/s
  - Leechers' bandwidth. Homogeneous: downlink = 150KB/s, uplink = 38KB/s; Heterogeneous: see Table I
  - for freeriders and exploiters, the downlink remains 150KB/s and the uplink remains 38KB/s
- Number of leechers: N = 1000 and all leechers join the network simultaneously at the beginning of the distribution
- Unchoking slots for each peer: s = 5

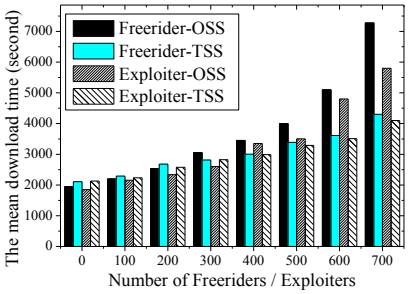

Figure 1. the mean download times of unselfish leechers in OSS and TSS when freeriders / exploiters exit.

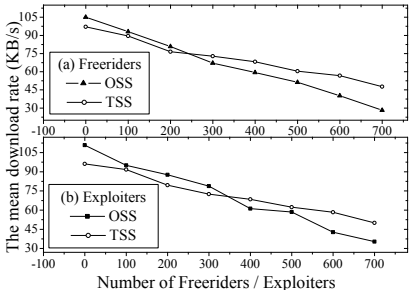

Figure 2. the mean download rate of unselfish leechers in OSS and TSS when (a) freeriders / (b) exploiters exist.

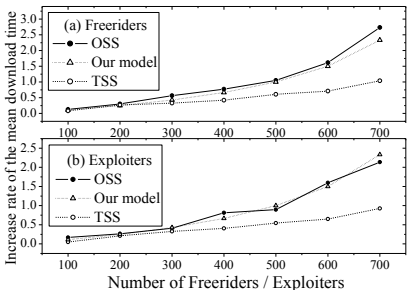

Figure 3. the increase rates of the mean download time of unselfish leechers in OSS, TSS and our model where (a) freeriders / (b) exploiters exist.

D. *Roadmap of Experiments*

We begin in Section V by applying OSS and TSS, respectively, to BitTorrent and examining their mean download times of unselfish leechers in a homogeneous environment (all leechers have the same uplink and downlink bandwidths) while freeriders or exploiters exist. For each of the seeding strategies, there are three experiments performed with this setting. In the first all leechers are unselfish. In the second and last we vary the number of freeriders / exploiters from 100 to 700 and record the mean download times. Through the three sets of mean download times we calculate the *Increase Rates* of the mean download times to study the impact of OSS and TSS on unselfish leechers while freeriders or exploiters exist. We finally use our mathematical model to predict the increase rates and compare them with OSS and TSS to verify the accuracy of our model.

In Section VI, we continue the experiments with heterogeneous settings. In addition to the mean download time of unselfish leechers, we use the cumulative distribution of download times for the three classes of unselfish leechers whose network bandwidths are different. This metric helps us investigate the details of individual completion time. We use the set of 100, 300, and 700 for the numbers of freeriders or exploiters, as the trend of the change of the download time is the important characteristics that we are studying and we believe this set is enough to highlight this trend.

V. RESULTS: THE HOMOGENOUS SETTING

In this section, we present the experimental results in a homogenous setting. For each of the experiments, the figures are plotted from the mean values of the results which are provided from ten simulation runs.

A. *Impact of Freeriders*

Fig. 1 shows the mean download times of OSS and TSS while freeriders exist. When there is no freerider in the network, the mean download time when OSS is applied is 1938.4 seconds, while applying TSS leads to a download time of 2114.6 seconds. The download process therefore costs 8.3% more time to finish if TSS is employed; OSS clearly outperforms TSS when all leechers are unselfish. The advantage of OSS is kept, although debasing, along with the increment of the number of freeriders. When the number of freeriders reaches 300, TSS starts to out-perform OSS. Its mean download time is 2812.6 seconds representing a 7.8% improvement over OSS. When the number of freeriders is 700, which means 70% of the leechers are freeriders, the mean download time of TSS is 40.9% less than OSS and the downloading performance of OSS becomes very poor (its mean downloading time is 7275.9 seconds). Clearly, TSS performs better in resisting freeriders than OSS, but OSS out-performs TSS when the number of freeriders is below a certain value.

We plot the mean download rates of OSS and TSS in Fig. 2(a). We can see that the download rate of OSS is higher than that of TSS before the number of freeriders reaches 300, but drops quickly as the number of freeriders increases. When 70% of the leechers are freeriders, the mean download rate of OSS drops to 28.1KB/s where the mean download bandwidth utilisation is only 18.7%. On the other hand, the mean download rate of TSS drops smoothly and slowly. Even in the worst case, the mean download rate of TSS is 47.6KB/s, although when there are no freeriders in the overlay, TSS performs 17.3% worse than OSS.

From Fig. 1 and Fig. 2(a), it can be concluded that before the number of freeriders reaches a certain value, OSS leads to a higher distribution performance than TSS. This is because OSS encourages leechers, who have higher download rates, to be seeds more quickly. Although there may be a number of freeriders in the overlay, a portion of unselfish leechers still have the chance to be served by seeds, and continue to serve afterwards and boost the whole distribution process. We know that freeriders will not serve others after they finish downloading and even while they are being served by seeds, they are not sharing file blocks with others. Thus, when the number of freeriders grows, the possibility for more freeriders dominating the resources of seeds will also increase, and the downloading performance of unselfish leechers are negatively and significantly impacted.

Instead of letting a particular set of leechers dominate the seed resources, TSS forces seeds to treat every leecher equally (despite their download rates) and serve them one by one for a certain period of time. When the number of freeriders is below a certain value, the TSS-led distribution process has lower performance than OSS-led because it does not try to boost the whole process by making more seeds quickly (unlike OSS). While the number of freeriders is growing, the

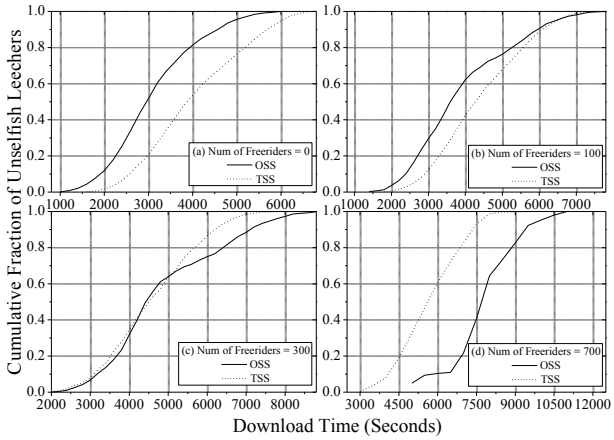

Figure 4. The cumulative distribution of the download times for all unselfish leechers when freeriders exist.

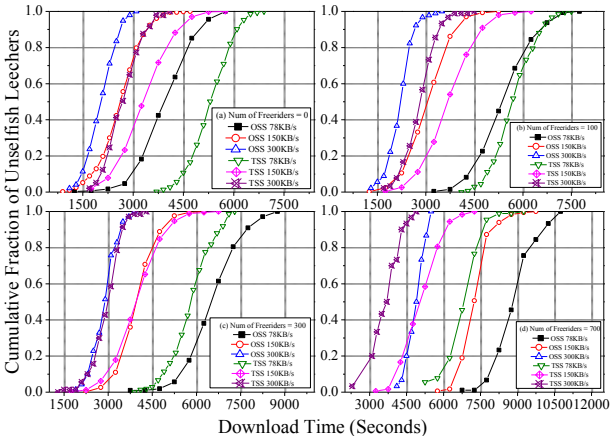

Figure 5. The cumulative distribution of the download times for each class of unselfish leechers when freeriders exit.

performance is negatively impacted but does not drop sharply like an OSS-led distribution, because freeriders will never dominate the seed resources and unselfish leechers still can be served by seeds even if the number of freeriders is large.

We compare the *IRD*s of OSS-led BitTorrent and our model in Fig. 3(a). The reason for calculating the IRD is to see how much the mean download time grows if the number of freeriders changes from zero to a certain value. Through the growth we can conclude which seeding strategy (OSS or TSS) performs better in resisting the freeriders. In addition, by calculating the IRD from the experimental results, we can justify how practical our mathematical model is. Note that our mathematical model is only for an OSS-led BitTorrent distribution process. We can see from Fig. 3(a) that the mathematical model matches the experimental results, the trends are approximately the same and our experience is that it is not uncommon for a mathematical model to underestimate values as it is, after all, an approximation of what is actually performed in practice. We can confirm that in real-world experiments the OSS-led BitTorrent performs worse than our model predicts, although the model still provides a reliable indication of IRDs, and that the mean download time grows at a slightly faster rate.

If we focus on TSS-led BitTorrent, we can see that the *IRD*s from this are very low; from the best case (0.08) to the worst case (1.04). When comparing *IRD*s from a TSS-led implementation with the OSS-led implementation, we find that the TSS-led version is on average 41.66% better than the OSS-led version, and the difference is consistent ranging from -62.01% to -10.75% (the negative sign indicates that the TSS-led version is lower than the OSS-led version). This shows that with TSS, the mean download time increases in a slow and steady manner, while more freeriders emerge in the overlay.

B. *Impact of Exploiters*

Fig. 1 also shows the mean download times of the OSS-led and the TSS-led version when there are exploiters in the overlay. The OSS-led BitTorrent performs better than the TSS-led one before the number of exploiters reaches 400 after which the reverse is true. Exploiters in a TSS-led BitTorrent overlay cannot dominate the seeds; however, it is similar for unselfish leechers whether they receive file blocks from seeds or exploiters. Hence, there are no large differences despite employing OSS or TSS when the number of exploiters is below a certain value (400 in our experiments).

Fig. 2(b) plots the mean download rate of the unselfish leechers. When there are no exploiters in the overlay, the OSS-led BitTorrent distribution shows a high download rate of 110.73KB/s and the download bandwidth utilisation is 0.74. Although it is predictable that TSS-led BitTorrent performs worse than the OSS-led version, the mean download rate remains high at 96.12KB/s. When the number of exploiters is 100, the download rate of the OSS-led BitTorrent decreases and is close to that of TSS. OSS keeps performing better than TSS until the number of exploiters reaches 400. When the number of exploiters equals 500, OSS is worse but close to TSS; however, the performance of OSS drops significantly afterwards while TSS's degradation is slow. When the number of exploiters is 700, the download bandwidth utilisation of OSS is 0.24, lower than that of TSS (0.33).

From Fig. 3(b), we can see that the increase rate of the mean download time for OSS matches our model and the average difference is 0.091. Before the number of exploiters reaches 400, the increase rates of the mean download time of OSS and TSS are close. This indicates that before the number of exploiters reaches a certain value, the negative impact on both seeding strategies are similar. After this, OSS suffers more acutely than TSS. In other words, TSS performs better than OSS in resisting exploiters.

VI. Results: the Heterogeneous Setting

In this section, we study the impact of the seeding strategies on the performance of BitTorrent when peer bandwidth is heterogeneous. As described in Section IV.C, we categorise all unselfish peers into three classes whose network bandwidths are different. To emphasise the effects of freeriders and exploiters, we choose to make their bandwidths consistent: 150KB/s (down) and 38KB/s (up).

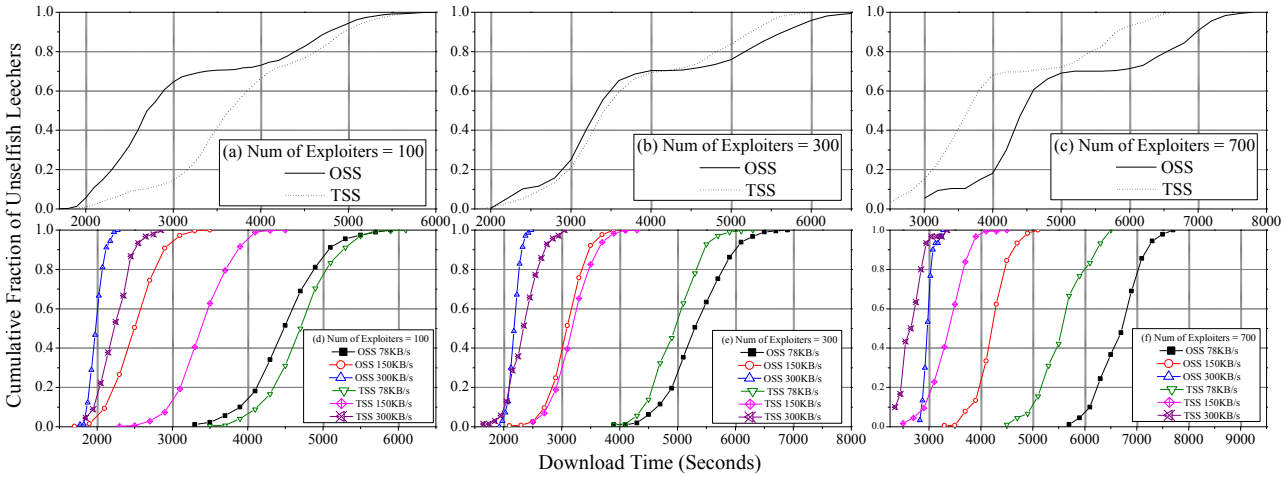

Figure 6. The cumulative distribution of the download times for each class of unselfish leechers when exploiters exit.

A. *Impact of Freeriders*

Fig. 4(a) plots the cumulative distribution of the download time when there are no freeriders in the overlay. It is clear that TSS performs much worse than OSS in this case. Using TSS, there are only 20% of the leechers able to finish the download within 3000 seconds and 90% of them need nearly 6000 seconds to complete. In the case of OSS, 52% of the leechers complete within 3000 seconds and 93% of them finish within 5000 seconds.

Recall that there are three classes of unselfish leechers in the overlay. In order to investigate the difference of the impact between OSS and TSS on each class, we plot the cumulative distribution of the download times for every class (Fig. 5(a)). The download link bandwidths of the three classes of unselfish leechers are 78KB/s, 150KB/s and 300KB/s respectively. Thus, the optimal cases for the download times of the unselfish leechers are 2625.6, 1365.3, 682.7 seconds. We can see from Fig. 4(a) that none of the leechers reach their optimal download time, despite the presence of OSS or TSS. For every class of unselfish leecher, OSS performs better than TSS. The 300KB/s class of leechers in TSS have similar download times to the 150KB/s class in OSS. The 78KB/s class of leechers, who have the lowest download bandwidths, spend the most time completing the download, particularly when TSS is employed.

When the number of freeriders is increased to 100, the two lines in Fig. 4(b), which indicate the download times under the two seeding strategies, become closer and the segments of the lines where the download time is more than 6500 overlap. In Fig. 5(b), we can see that the class of TSS 300KB/s has surpassed the class of OSS 150KB/s, while they perform similarly if the number of freeriders is 0. At the same time, the class of TSS 78KB/s and the class of OSS 150KB/s have very close download times.

This indicates that once the freeriders join the overlay, the performance of the OSS-led BitTorrent degrades towards the TSS-led version. The TSS-led BitTorrent also suffers performance degradation; however, the degree of this is less than that of the OSS. When the number of freeriders reaches 300, the lines in Fig. 4(c) are very close to overlapping, before the point where the download time equals 5000. This means that there are similar numbers of unselfish leechers having finished their download within 5000 seconds. After 5000 seconds, more unselfish leechers with TSS finish before 7000 seconds than those with OSS. Fig. 9(f) shows that the low bandwidth (78KB/s) leechers with TSS complete sooner than those with OSS

For middle and high bandwidth leechers with TSS and OSS, they perform the download similarly. This case is seen to be similar to the homogeneous setting. Where the number of freeriders is 300 and above, TSS starts to perform better than OSS. For the results of the experiments in which the number of freeriders is 700, we plot them on Fig. 4(d) and Fig. 5(d). From both figures, it is clear that when the number of freeriders is more than 300, TSS performs better than OSS. In Fig. 5(d), the class of OSS 150KB/s performs even worse than the class of TSS 78KB/s. Comparing the class of TSS 150KB/s with OSS 300KB/s, we can see that, 40% of TSS 150KB/s spent less time than OSS 300KB/s leechers.

In addition to the conclusions that we draw from the above study, we also find two interesting effects of freeriders:

- Freeriders impact simultaneously on all leechers even if they have different bandwidths;
- OSS-led and TSS-led BitTorrent overlays both suffer from freeriders. However, The TSS-led version suffers less than the OSS-led one. This is clearer when more freeriders join the overlay.

B. *Impact of Exploiters*

In this section, we study the resistance of OSS and TSS to the exploiters. When the number of exploiters equals zero, the results of the download times are the same as those where there are no freeriders in the overlay (Fig. 4(a) and Fig. 5(b)). Thus we do not repeat the results or discussion here. Instead, we continue the study from where the number of exploiters is 100.

When the number of exploiters is increased to 300 (Fig. 6(e)), 8% of the class TSS 300KB/s leechers finish downloading before all high-level leechers in the OSS-led BitTorrent; however, the remainder of the OSS 300KB/s leechers perform better than TSS 300KB/s. The download

times of the middle level leechers are very close in both overlays, although the ones in the OSS-led overlay are still slightly better than the TSS-led version. The low-level leechers in the TSS-led overlay complete sooner than the ones in the OSS-led BitTorrent. With the emergence of more exploiters, the middle-level leechers in the OSS-led overlay significantly impact on performance (compared with Fig. 4(a)). This is because when exploiters finish downloading, they leave the network immediately.

Although high-level unselfish leechers will serve the overlay for an amount of time in an OSS-led overlay, their service targets are always other high-level leechers who have not yet finished. The services to the middle-level leechers are not therefore sufficient. In the TSS-led overlay, middle-level unselfish leechers always obtain their services from existing seeds, even if exploiters exist. Thus the performance degradation in the TSS-led overlay is not as severe. Once the number of exploiters reaches more than 300, TSS starts to transcend OSS. First, the middle- level unselfish leechers in the TSS-led overlay obtain better overall download speeds than the same class in the OSS-led version (shown in Fig. 6(c)) and 9% of them even perform better than those in the OSS-led overlay when the number of exploiters is 700 (shown in Fig. 6(f)).

The performance differences of low-level leechers between the OSS-led and TSS-led versions remain similar to the case where the number of exploiters is 300; however, their overall download times are increased. When the number of exploiters is 700, the download time window for 63% of the unselfish leechers in the TSS-led overlay is between 3000 and 5000 seconds, while this window is between 4000 and 6000 seconds for the OSS-led overlay.

VII. Conclusions and Discussion

In this paper we establish a mathematical model to analyse how the seeding strategies in BitTorrent affect system performance in the presence of selfish peers. We choose two popular seeding strategies, the Original Seeding Strategy (OSS) and the Time-based Seeding Strategy (TSS), for this study. We categorise selfish peers into two classes: freeriders and exploiters. A series of simulations are then conducted in both homogeneous and heterogeneous network settings.

First, we prove the practical uses of our mathematical model, which can be used to theoretically study the effects of the seeding strategies on a BitTorrent network. We then discover that when the number of selfish leechers is below a certain value, OSS performs better than TSS. Beyond this value, TSS outperforms OSS by equalising the contributions of seeds to every leecher. It is observed that this approach prevents freeriders from occupying the resources of seeds and successfully makes exploiters serve more blocks to other leechers.

We also discover that both freeriders and exploiters harm the system, despite the seeding strategy that is employed. TSS has better resistance (which means smaller performance degradation) to selfish leechers compared with OSS. Our experimental results are consistent with those shown in [8]: that the download rates that leechers can obtain are directly proportional to leechers' bandwidths. Furthermore, we find that the selfish leechers can impact negatively on leechers of every level of bandwidth and neither of the seeding strategies can completely eliminate this impact. TSS can reduce the impact more effectively than OSS; this can be more clearly observed when the number of selfish leechers reaches a certain threshold.

In a BitTorrent network we believe that OSS should be employed to boost system performance (higher download performance) if the number of selfish leechers is relatively small; the exact threshold depends on the BitTorrent environment. Beyond this threshold, TSS should be deployed to equalise the contributions of the seeds into all leechers to prevent selfish leechers from dominating the seeding resources. Current popular BitTorrent clients employ either OSS or TSS, and none investigate the combination of these two seeding strategies. Therefore our future research is focused on how to combine these two seeding strategies so that the seeds can deliver enhanced service to unselfish leechers despite the existence of selfish leechers. This will involve building a mechanism to detect the scale of selfish leechers in the overlay. Depending on the scale, we will direct the seeds to implement OSS or TSS dynamically.

REFERENCE

[1] S. Saroiu, P. K. Gummadi and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems", MMCN, 2002

[2] A. R. Bharambe, C. Herley and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Network's Performance Mechanisms", IEEE INFOCOM, 2006

[3] B. Cohen, "Incentives Build Robustness in BitTorrent", First Workshop on Economics of Peer-to-Peer Systems, 2003

[4] E. Adar and B. A. Huberman, "Free Riding on Gnutella", First Monday, pp.

[5] S. Jun and M. Ahamad, "Incentives in BitTorrent Induce Free Riding", ACM SIGCOMM, 2005

[6] N. Andrade, M. Mowbray, A. Lima, G. Wagner and M. Ripeanu, "Influences on Cooperation in BitTorrent Communities", The Third Workshop on Economics of Peer-to-Peer Syste, 2005

[7] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", ACM SIGCOMM, 2004

[8] A. Legout, N. Liogkas, E. Kohler and L. Zhang, "Clustering and Sharing Incentives in BitTorrent Systems", ACM SIGMETRICS, 2007

[9] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra and L. Garces-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime", PAM'04, 2004

[10] J. A. Pouwelse, P. Garbacki, D. H. J. Epema and H. J. Sips, "The BitTorrent P2P File-Sharing System: Measurements and Analysis", IPTPS, 2005

[11] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding and X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems", ACM SIGCOMM Internet Measurement Conference (IMC'05), 2005

[12] B. Fan, D. M. Chiu and J. C. Lui., "The Delicate Tradeoffs in BitTorrent-like File Sharing Protocol Design", ICNP, 2006

[13] P. A. Felber and E. W. Biersack., "Self-scaling Networks for Content Distribution", International Workshop on Self-* Properties in Complex Information Systems, 2004

[14] L. Massoulie and M. Vojnovic, "Coupon Replication Systems", Sigcomm, 2005

[15] A. Akella, S. Seshan and A. Shaikh, "An Empirical Evaluation of Wide-Area Internet Bottlenecks", IMC, 2003