

## **Multi-torrent: a performance study and applications**

---

Yan Yang\*, Alix L.H. Chow  
and Leana Golubchik

Department of Computer Science,  
University of Southern California,  
Los Angeles, CA 90089, USA  
E-mail: yangyan@usc.edu  
E-mail: lhchow@usc.edu  
E-mail: leana@usc.edu  
\*Corresponding author

**Abstract:** Empirical evidence suggests that most nodes in BitTorrent (BT) participate in *multiple torrents*, but surprisingly little research exists on this topic. Here, we focus on a multi-torrent system, and specifically on (a) what incentives could be provided for nodes to contribute resources as seeds in a multi-torrent environment, and (b) what are the resulting performance consequences. We show why the current BT lacks incentives for nodes to stay as seeds. Motivated by that, we propose a cross torrent based method to encourage nodes to stay as seeds and present an extensive performance study to illustrate the benefits of our approach.

**Keywords:** BT; BitTorrent; file sharing; P2P; peer-to-peer; content distribution.

**Reference** to this paper should be made as follows: Yang, Y., Chow, A.L.H. and Golubchik, L. (2010) 'Multi-torrent: a performance study and applications', *Int. J. Advanced Media and Communication*, Vol.

**Biographical notes:** Yan Yang is a PhD student in the Computer Science Department at USC. He received his BS Degree in Computer Science and Information System from Donghua University, Shanghai, China in 2000 and the MS Degree in Computer Science from USC in 2005, respectively. His research interests include computer communication networks and P2P systems.

Alix L.H. Chow is a PhD candidate in the Computer Science Department at USC. He received his BEng in Computer Engineering and MPhil in Computer Science and Engineering from the Chinese University of Hong Kong in 2001 and 2003, respectively. His research interests include computer communication networks and distributed multimedia systems.

Leana Golubchik is an Associate Professor in the Computer Science Department, with a joint appointment in Electrical Engineering, at USC. Prior to that she was on the faculty at the University of Maryland at College Park and at Columbia University. She received several awards,

including the IBM Faculty Award, the NSF CAREER award, and the Okawa Foundation award. She received her PhD from UCLA; she is a member of the IFIP WG 7.3.

---

## 1 Introduction

In recent years BT (described by Cohen (2003)) has become an extremely popular and successful P2P file sharing system. According to CacheLogic (2005), BT accounts for an astounding 35% of all the traffic on the internet, which is more than all other P2P applications combined. The success of BT as well as its interesting design has resulted in a number of research efforts, aiming at understanding its characteristics and performance, e.g., Qiu and Srikant (2004), Bharambe et al. (2006), Fan et al. (2006), and Piatek et al. (2007), to name a few. These works provide significant insight into the performance, fairness, scalability (and other characteristics) of BT, and we give an overview of related work and its relationship to our work in Section 6.

Most of these studies (with a notable exception of Guo et al. (2005)) are focused on a single-torrent system – they assume that a node joins a single torrent in a system, all nodes in that system download from the same torrent and eventually leave the system. In practice, many published torrents are related (e.g., different episodes of a TV show or movies/music from the same genre), which results in many peers downloading data from multiple torrents. Specifically, from the trace analysis given in Guo et al. (2005), more than 85% of peers participate in multiple torrents. However, in the current BT system, there is no mechanism to ‘relate’ the multiple downloads of the torrents in which a particular node is participating – we term this a *multi-torrent approach*. There are a number of applications which could benefit from such an approach. For instance, fast delivery of patches in popular multi-player online games (e.g., Blizzard (2008)), is an important problem. Such games have millions of subscribers which (periodically) need to download patches which results in substantial internet traffic. Such downloads need to be completed fairly quickly, as users cannot continue to play the game *until all patches are downloaded*. Since each user (potentially) downloads multiple patches, this indicates that not only is P2P technology useful for this application, but also that there is an opportunity for a multi-torrent approach. An interesting application in this context would be a software installer, e.g., such as the Google Pack installer or the Microsoft Live installer, where users would typically download some subset of the available software modules simultaneously, providing another opportunity for exploring a multi-torrent approach. Similar applications include software update installers, such as Microsoft Windows updates, Mac OS X updates, Adobe updates, and so on. Multimedia online applications, such as iTunes Movie Store, NetFlix Video-on-Demand service, and YouTube, can benefit from a multi-torrent approach as well. In these applications, users normally have sufficient storage space to keep previously viewed content. This makes multi-torrent a natural approach in such applications. (A more detailed description of such applications is given in Section 4.)

Intuitively, a multi-torrent approach might have the following advantages. Typically, it is desirable to have ‘seeds’ in a torrent, (i.e., nodes which have completed their download but continue to contribute their uploading capacity to the system). Briefly, seeds can contribute to:

- helping newly joined nodes ramp up so that they can become contributing peers faster
- helping nodes nearing the end of their downloads find the last few file pieces/chunks faster
- keeping a torrent ‘alive’ (i.e., making sure that all pieces of a file are available in the system).

(A more detailed discussion of these benefits is given in Section 2.) Although current studies, e.g., as in Guo et al. (2005) and Izal et al. (2004), suggest that seeds do exist in BT, there are no incentives in the current BT system for nodes to provide seeding capacity (e.g., by staying around as seeds after their download in a particular torrent is complete or by sharing files downloaded earlier). However, if such incentives were provided – for instance, if a node was to receive better service in torrent *A* because it acts as a seed in torrent *B* – then, intuitively, seeding behaviour of nodes in BT might increase. *Exploring an approach to providing such an incentive and evaluating the resulting BT performance characteristics is the main topic of this work.*

Surprisingly, little exists in the literature which considers relating multiple torrents. To the best of our knowledge, Guo et al. (2005) is the only work which studies this topic. Specifically, in Guo et al. (2005) the authors develop an analytical model of multiple torrents where it is assumed that a node participating as a ‘leecher’ (node which has not completed its download yet) in a particular torrent is willing to serve as a seed in torrents in which it has participated some time earlier in its lifetime. This model is then used to illustrate that this multiple torrent approach can extend the lifetime of a torrent. Here, the lifetime of a torrent is the time until some piece/chunk is lost from a torrent due to the departure of one of its nodes. (A more detailed description of Guo et al. (2005) is given in Section 6.) While the work in Guo et al. (2005) provides intriguing ideas and results, it leaves a number of open questions. Two such questions are:

- 1 how to provide appropriate incentives for nodes to contribute resources as seeds (preferably) in a distributed manner and through simple/local changes
- 2 what are the resulting performance consequences of such behaviour, both on the nodes which are willing to be seeds and on the overall system.

In this paper, we focus on the above stated questions using our proposed multi-torrent approach. We first illustrate how nodes staying around in the system as seeds after finishing their downloads helps improve the overall system performance. We then show why the current BT unchoking algorithm lacks incentives for nodes to stay around as seeds in a multiple torrent environment. Motivated by that, we propose a cross torrent based method to encourage nodes to stay around as seeds and perform an extensive performance study of our techniques. Thus, the contributions of this work are as follows:

- We propose a multi-torrent BT system which can be easily implemented through fairly small modifications of the current BT protocol (refer to Section 2), unlike the approach proposed in Guo et al. (2005) which requires modifications to the BT tracker. Thus, we believe that our approach is scalable and easily deployable.

- We propose a “cross-torrent-based” tit-for-tat (CTFT) strategy motivated by providing incentives for nodes to act as seeds (refer to Section 3). Unlike the exchange-based incentives suggested in Anagnostakis and Greenwald (2004) which require nodes to maintain and search through a request tree before transmitting file chunks, CTFT only uses peer transmission rate information and does not require additional modifications to the current BT system. We believe it is a more efficient, scalable, and easily deployable approach.
- We perform an extensive simulation-based study which illustrates that:
  - a our approach does improve the overall performance of the system
  - b our approach does provide incentives for nodes to act as seeds by providing better performance for such nodes (refer to Section 4).

## 2 Background and motivation

As background and to establish terminology, we briefly describe how BT works currently.

### 2.1 BT system

In BT, nodes join the system (after receiving ‘start-up’ information from the tracker) and begin requesting chunks of data from their neighbours. The tracker maintains a list of nodes which are currently participating in the corresponding torrent. It is responsible for assisting in peer discovery and is not involved in any data transfer or data scheduling. Nodes which do not have a complete copy of the file are termed ‘leechers’ and those which do are termed ‘seeds’. Each leecher  $i$  picks a number (typically 5) of nodes to whose requests it will respond with an upload of an appropriate chunk, i.e., these nodes are ‘unchoked’. A subset of these nodes (typically 4) are picked based on the Tit-for-Tat (TFT) mechanism, i.e., those neighbours which have provided the best service (in terms of download rate) to node  $i$  recently. And a subset (typically 1) is picked randomly, i.e., they are ‘optimistically unchoked’ (to explore better neighbours). The TFT mechanism in BT is basically a “*local selfish behaviour*” where leechers try to upload to the most reciprocative neighbours. Seeds also pick a subset of neighbours (typically 5), and upload data to them. In past versions of BT, seeds chose neighbours with the highest download rates. In a more recent protocol, Legout et al. (2007), the seeding capacity is distributed more uniformly among the neighbouring peers. All these choices are re-evaluated periodically.

### 2.2 Multiple torrents

As noted in Guo et al. (2005), most BT peers (>85%) participate in multiple torrents; however, in the current BT these downloads are performed without an attempt to ‘relate’ the multiple torrents. We also note that:

- a users in BT typically obtain start-up information (.torrent file) from a similar source (e.g., a forum or a website)
- b there is some correlation in content of interest (as noted in Section 1)

Thus, we expect there to be a reasonably high probability that users obtaining start-up information from the same source would download from multiple common torrents simultaneously. Motivated by this, we first focus on the opportunity to have nodes act as seeds in the torrents they have already completed while they are downloading as leechers in the torrents they have not completed.<sup>1</sup> The potential benefits of this include the following three categories.

- *Seeds helping newly joined nodes ramp up faster.* It usually takes a newly joined node some time to ramp up its download rate, due to lack of data chunks which it can offer/upload to other nodes – this prohibits it from being unchoked via TFT, e.g., as discussed in Chow et al. (2008). Therefore, at the beginning it receives its chunks mostly through optimistic unchoking of other leechers as well as from seeds. Thus, having a larger number of seeds in the system would allow a newly joined node to ramp up faster. This faster ramp up would also allow new nodes to contribute their uploading resource to the system earlier, thus improving not only their performance but the overall system capacity.
- *Seeds helping improve ‘end-game’ behaviour.* When a node nears the end of its download process in BT, it has a lower probability of finding peers which carry the few remaining data chunks it needs to complete its file download – this would result in a decrease in its download rate, e.g., as discussed in Chow et al. (2008). Having more seeds in the system would increase the probability of a leecher finding peers with those last few chunks (as seeds have the complete file), and hence would improve the download rate. In the current BT system, this is addressed by having an ‘end-game’ mode in which nodes, toward the end of their download process, attempt to download the same chunks from multiple peers, in order to improve their probability of getting those last few chunks from someone. This however, has the potential of wasting system resources as a node might receive duplicate chunks. (Our approach would be orthogonal to the current end-game mode.)
- *Seeds keeping a torrent alive.* Having nodes participate longer in the system, and particularly seeds, also extends the lifetime of a torrent. That is, it reduces the probability of a node departure resulting in some data chunks disappearing from the torrent (i.e., if the departing node had the last copy of a particular chunk and left before another node downloaded it) – this helps maintain data availability of the less popular content. As this benefit is well studied in Guo et al. (2005), we do not explore it further in this work.

Although these potential benefits to the overall system performance are significant, in the current BT system, a node does not have an incentive to contribute its resources as a seed in some torrent while its downloading from another torrent. Moreover, doing this might hurt that node’s performance – it would use part of its upload capacity to help peers from which it does not need any data while reducing its ability to ‘compete’ for TFT unchoking from peers which could provide data it does need. That is, a selfish peer might be better off using all its uploading capacity in the torrents where it is a leecher. In what follows we give a motivating example which illustrates a potential for incentivising peers to remain as seeds in some torrents while they are completing their downloads as leechers in other torrents.

### 3 Proposed approach

We begin with a simple (simulation-based) example which motivates the use of multiple torrents and the need for incentivising peers to remain as seeds in some torrents while they are completing their downloads as leechers in other torrents. We then suggest an approach for providing such incentives.

#### 3.1 Motivating example

Here, we have slow and fast nodes (with node capacities and mix given in Table 1) where each node arriving to the system joins 2 (randomly chosen) torrents out of 10 available ones. (The details of the simulator used, to generate the results of experiments described below, are described in Section 4, with the simulation settings given in Table 2.) When a node completes a download in one of its torrents, depending on the experiment, it does or does not ‘stay around’ as a seed in that torrent while it completes its download in the remaining one; specifically, we perform the following experiments:

- original BT where none of the nodes stay around
- original BT where all nodes stays around
- original BT where each node stays around with probability  $\frac{1}{2}$
- BT with our simple modification to the current TFT mechanism, which we term cross-torrent TFT (CTFT), where each node stays around with probability  $\frac{1}{2}$  (the details of our approach are given below).

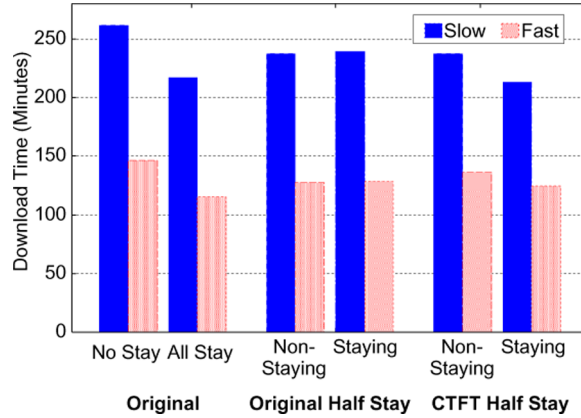
**Table 1** Classes bandwidth distribution

<i>Class</i>	<i>Fraction (%)</i>	<i>Download capacity (kbps)</i>	<i>Upload capacity (kbps)</i>
Slow	40	1500	128
Fast	60	5000	512

**Table 2** Simulation settings

File size	200 MB (800 Chunks, 256 KB each)
Simulation time	12h (+3h Warmup)
Avg. node inter-arrival time ( $\frac{1}{\lambda}$ )	45s
Peer set size	40
Leecher unchokings	4 Regular + 1 Optimistic
#Seed unchokings	5

The resulting average download times for these experiments are depicted in Figure 1 – here, the first group corresponds to experiments (1) and (2), the second group corresponds to experiment (3) where we depict download times of nodes that did and did not stay around, and the third group corresponds to experiment (4), where again we depict download times for nodes that did and did not stay around.

**Figure 1** Motivating example (see online version for colours)

Comparing results of experiments (1) and (2) we can note that nodes ‘staying around’ does improve the average download time, in this case by  $\approx 17\%$  for the slow nodes and by  $\approx 21\%$  for the fast nodes. However, as noted the ‘stay around’ in this simulation is ‘forced’, and the current BT system has no incentives for the nodes to stay around after they finish their downloads. To illustrate this further, note that in Figure 1 when half of the nodes stay around, i.e., experiment (3), nodes that do and do not stay around experience similar improvements in download times (as compared to experiment (1)).

That is, there is no incentive for nodes to stay around voluntarily, and a mechanism for encouraging ‘staying around’ behaviour is needed. To this end, we propose a simple modification to the current TFT mechanism (as detailed in Section 3), and employ it in experiment (4). As depicted in Figure 1, nodes which do stay around experience better average download times (for both, fast and slow classes) than those which do not stay around – this is in contrast to experiment (3), which has the same settings except for our proposed CTFT mechanism. Specifically, in this example, when using CTFT, staying nodes download  $\approx 10\%$  faster than their non-staying counterparts. This motivates the need for a mechanism to provide incentives for nodes to stay around voluntarily. We describe our proposal for such a mechanism next.

In the current system, a node which does stay around may end up experiencing poorer performance than one which does not stay around (refer to Section 2). In that case, nodes are not likely to stay around, *unless* proper incentives can be provided. Therefore, the question we consider here is *how to encourage nodes to stay around* in order to reap the potential benefits of multiple torrents.

There are a number of approaches one could take to try to exploit the potential benefits of multiple torrents. In this section we present one such simple approach and explore its benefits through a simulation-based performance study in Section 4. (A discussion of possible other directions is given in Section 5.) In devising this approach we are motivated by the following. Firstly, we would like to provide incentives for a node to stay around as a seed in torrents it has finished while it is downloading from other torrents as a leecher. Specifically, we would like such nodes to experience shorter download times than nodes which do not stay around.

Secondly, we would like to achieve this while maintaining the original spirit of BT where nodes exhibit “local selfish behaviour” (refer to Section 2). Lastly, we would like our scheme to be easily implementable and deployable. To this end we focus on local modifications (not centralised approaches), and without the need of assistance from (or modifications to) the tracker(s) as in Guo et al. (2005). Given this, our scheme modifies the TFT unchoking mechanism of leechers, by considering multiple torrents and favouring nodes which stay around as seeds. We term this approach *Cross-Torrent Tit-for-Tat* (CTFT).

### 3.2 *Cross-Torrent TFT (CTFT)*

CTFT modifies the unchoking part of leechers’ behaviour as follows. When a leecher is choosing peers for TFT unchokes, instead of choosing peers with the fastest downloading rates in a particular torrent (as is currently done), we look at the peers’ aggregate downloading rate in all torrents in which they participate. Moreover, this aggregation is done in a weighted manner, where higher weight is given to the downloading rate from torrent(s) where a peer is a seed, i.e., in order to favour nodes which stay around as seeds. That is, the total contribution, which we use to rank peers for TFT unchoking, of a peer  $N_y$  with respect to a peer  $N_x$  is  $\sum_{i=1}^{\#Torrents} w_i(y) \times D_i(x, y)$ , where  $D_i(x, y)$  is the downloading rate of node  $N_x$  from node  $N_y$  and  $w_i(y)$  is the weight we assign to that downloading rate. If  $N_y$  is *not* a seed in torrent  $i$ , then  $w_i(y) = 1$ , otherwise,  $w_i(y)$  can be set to a value that is larger than one. (The effect of different weights is studied in Section 4.)

For example, let us consider two nodes  $N_x$  and  $N_y$  which are both participating in torrents  $T_a$  and  $T_b$ . Node  $N_y$  has finished downloading the file in torrent  $T_a$  and is staying as a seed in that torrent. Suppose that  $D_a(x, y)$  and  $D_b(x, y)$  are the downloading rates of node  $N_x$  from node  $N_y$  in torrents  $T_a$  and  $T_b$ , respectively. When node  $N_x$  is selecting peers to unchoke via TFT in torrent  $T_b$ , Node  $N_x$  ranks the peers in torrent  $T_b$  (which are interested in downloading from  $N_x$ ) based on their weighted total contribution. As node  $N_y$  is a seed in torrent  $T_a$  and a leecher in torrent  $T_b$ , its total contribution to node  $N_x$  will be  $W \times D_a(x, y) + D_b(x, y)$ , where  $W > 1$  is the weight assigned to seeds. This in a sense gives ‘credit’ to node  $N_y$  in torrent  $T_b$  for both, contributions it is making as a leecher in  $T_b$  *and* contributions it is making as a seed in  $T_a$ . Note that this ‘credit’ is given locally – that is, node  $N_x$  is only concerned with the benefit *it* gets from node  $N_y$  in both torrents, not with the benefit other nodes might receive from  $N_y$ . This is in line with the original spirit of BT, as described in Section 2.

Through an extensive performance study (in Section 4) we illustrate benefits of the proposed approach. Although these benefits can be significant, we also show that these benefits are not uniform when there is sufficient heterogeneity in the file sizes of the different torrents. Specifically, the decrease in download times of the smaller files is achieved at the cost of the increase in download times of the larger files (e.g., as illustrated in Figure 6). Intuitively, this is due to the fact that the distribution of seeding capacity does not take into consideration where such capacity is needed more.

There are a number of approaches that can be taken to mitigate this problem. One approach (which we evaluate in Section 4) is to estimate the ‘need’ for seeding capacity in the different torrents and then only participate as seeds in those with

‘greater need’. Thus we modify the above CTFT approach as follows. Each node does a local<sup>2</sup> estimate (i.e., based on its neighbours only) of the ratio of seeds to leechers, and then only participates as a seed in those torrents where the seeds to leechers ratio is below  $R$ , where  $R$  is a system parameter. This local estimate and hence the choice of in which torrents to seed is re-evaluated every  $S$  time units, where  $S$  is another system parameter.<sup>3</sup> We term this adaptation of the CTFT approach as *CTFT with dynamic seeding* (CTFT-DS).

We also note that another way to ‘shift’ the seeding capacity to larger files is to give higher weights to torrents with larger files in the CTFT approach described above. This can also be combined with the CTFT-DS. (Evaluating the combined effects of these approaches is part of our future efforts.) We note that we explored a number of possible approaches to controlling seeding in torrents, e.g., based on information about the number of seeds and leechers obtained from the tracker, based on comparing the total amount of upload bandwidth to the total amount of download bandwidth of leechers, and so on. The results using these approaches are similar to those of the approach presented here. However, these approaches require tracker side assistance and are less scalable. We omit these results here due to space limitations.

#### 4 Simulation study

In the following performance study we use the BT simulator provided by Bharambe et al. (2006) (this simulator is also used by other groups for BT research). This is an event-based simulator which simulates<sup>4</sup> the chunk exchange mechanism in the BT protocol. To explore our proposed approach, we modify the simulator in Bharambe et al. (2006) as follows:

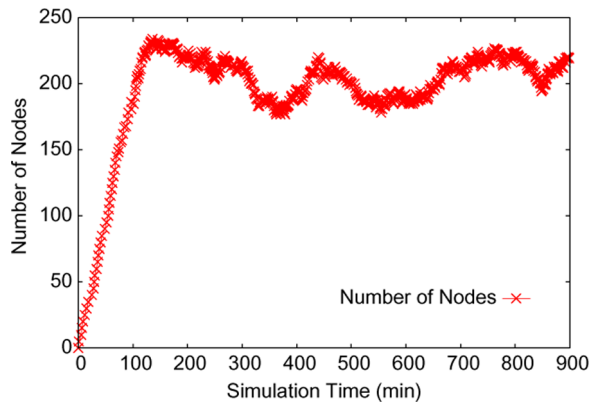
- We extend the simulator to support multiple torrents. Each torrent has one initial seed when the simulation begins. An arriving node chooses which torrents it wants to join, either as a leecher or a seed (as described in detail below).
- We allow nodes to stay around in torrents in which they finished downloads (based on techniques described above). Nodes leave the system when they finish downloads in all joined torrents.
- We add support for our CTFT and CTFT-DS mechanisms (refer to Section 3).
- We allow node arrivals; in what follows we use a Poisson arrival process with rate  $\lambda$ .
- We update the seed uploading algorithm to be like the current BT protocol, i.e., more uniform rather than uploading to the fastest peers as in older versions of BT.

Unless specified otherwise, the following results correspond to the simulation settings in Table 2. The system starts with one origin seed per torrent, each with 1000 kbps upload capacity and each staying in the system for the duration of the simulation. Arriving nodes are assigned to a particular class according to a given distribution. The classes differ in their upload and download capacities. The default classes

and corresponding distribution<sup>5</sup> are given in Table 1. The fast class is used when simulating a homogeneous system.

For tractability of simulations, our experiments include up to ten torrents, where each node joins a subset of these torrents, depending on the experiment (as described below). To obtain a fair comparison between approaches, we use the same node arrival sequence for each simulation with a given arrival rate and class distribution. In experiments where nodes randomly select which torrents to join (as leechers or as seeds), we also use the same torrent selection sequence. In the following simulations, we look at the steady state behaviour of the system. Each simulation run corresponds to 15h, and we only compute our results over the last 12h. Figure 2 depicts the number of nodes in the system as a function of simulation time; this indicates that the system has passed the initial ‘ramp up’ stage during those first 3h.

**Figure 2** Number of nodes in the system (see online version for colours)



In what follows, unless otherwise stated, we focus on two metrics:

- the average download time over all torrents
- the average download time for the last torrent (i.e., the average amount of time it takes a node to complete all its downloads).

These metrics are computed either over all nodes, or on a per-class basis, depending on the experiment.

#### 4.1 Design space exploration

To illustrate the performance consequences of our approach, we first explore the effects of different parameters using simple scenarios, where seeding capacity is due only to nodes staying around (as described above), except for the original source. Below we use the following notation for the various schemes being simulated:

- ‘No Stay’ refers to all nodes leaving each torrent as soon as the download in that torrent is complete
- ‘Stay’ refers to (some fraction) of the nodes (depending on the experiment) staying around as seeds in each of their torrents until they complete the last of their downloads, at which point they leave the system

- ‘Original’ refers to the use of the original BT TFT mechanism
- ‘CTFT’ refers to the use of our proposed CTFT mechanism (with  $W = 4$  as default).

#### 4.1.1 Different number of torrents

In this experiment we study how a multi-torrent system performs as a function of the number of torrents each node joins, where the performance improvements are due to staying around only, i.e., here we use the original BT TFT mechanism. For clarity of presentation, we consider a homogeneous system with only the fast nodes in Table 1. Figure 3 depicts the percentage improvement in download time, as compared to the ‘No Stay’ case, for two experiments:

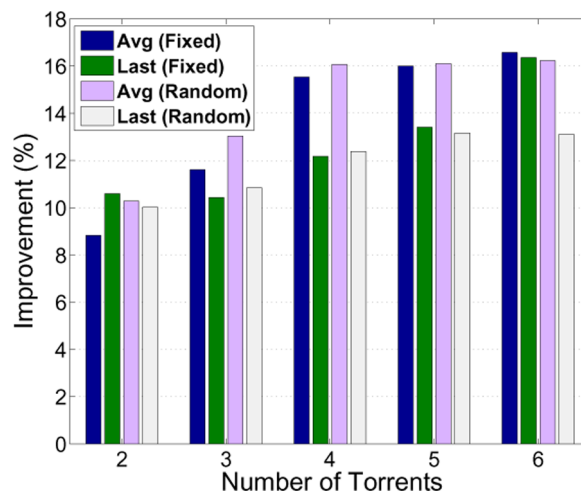
- when the choice of torrents to join is *fixed*, i.e., all nodes join the same  $X$  torrents
- when this choice is *random*, i.e., each node uniformly selects  $X$  torrents (out of 10) to join.

The value of  $X$  is depicted on the  $x$ -axis of Figure 3. From these results we observe the following:

- having nodes stay around as seeds improves average and last download times for both fixed and random selection experiments
- in most cases, performance improvements increase with  $X$ , due to larger variances in download times (as a function of  $X$ ) which results in longer stay around times.

For example, when the choice of torrents to join is fixed, the standard deviation of average download times increases from  $\approx 11$  for 2 torrents to  $\approx 45$  for 6 torrents.

**Figure 3** Staying around improvement (see online version for colours)

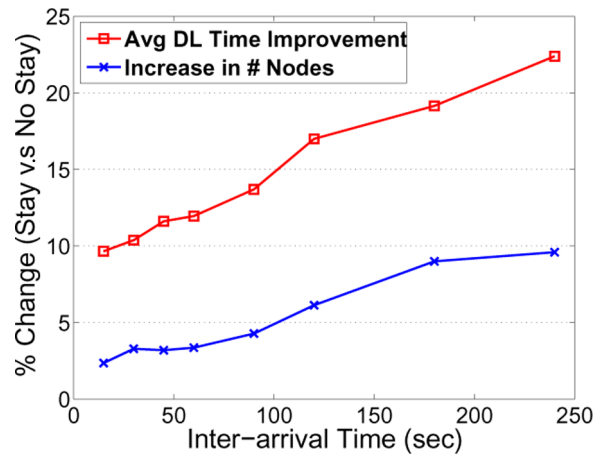


Although there is not a monotonic trend in the *percentage* improvement as a function of number of torrents, the actual improvements are approximately linear. We note that when the number of torrents each node joins is relatively few, the average download time using random torrent selection is a bit faster than that using fixed torrent selection, e.g., the difference is  $\approx 10\%$  for the case when nodes join 2 torrents. Their respective download times are closer when node joins a relatively large number of torrents, e.g., only  $\approx 1\%$  difference in the case when nodes join 5 torrents. This can be explained as follows – the network size in the random selection experiment, becomes more similar to that of the network size in the fixed selection experiment as the number of torrents joined grows (as also discussed below).

#### 4.1.2 Different node arrival rates

In this experiment we observe the effects due to the arrival rate. For clarity of presentation, we consider a homogeneous system with only the fast nodes given in Table 1 and each node joining the same 3 torrents. Figure 4 depicts the percentage improvement in average download time and percentage increase in network size, both due to nodes staying around (as compared to the ‘No Stay’ case), as a function of node inter-arrival time. From these results, we observe that staying around results in larger improvements when the inter-arrival time is longer (lower arrival rate). This is due to staying around being more helpful in torrents with fewer nodes. When the arrival rate is lower, we have a smaller network size and staying around results in greater relative increases in network size (as shown in the figure). This also increases the peer set size<sup>6</sup> for each node, and, in general, larger peer set sizes result in better BT performance (see Sirivianos et al. (2007)).

**Figure 4** Inter-arrival time (see online version for colours)



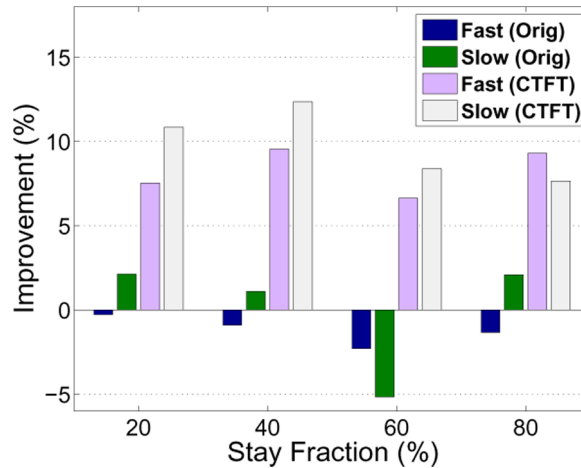
#### 4.1.3 Different fraction of nodes staying around

Here, we study how CTFT performs with different fractions of nodes staying around. We consider a heterogeneous system in Table 1 where each node randomly selects 3 torrents to join out of 10. Figure 5 presents the average download time improvement of the staying nodes as compared to the non-staying nodes when

we vary the fraction of staying nodes from 20% to 80%. Results for both the fast nodes and slow nodes are shown for the ‘Original’ and ‘CTFT’ schemes. Negative percentages indicate cases where staying nodes download slower than the non-staying ones. We make the following observations.

- In ‘Original’ fast and slow staying nodes have about the same performance as the non-staying nodes (staying nodes are doing a bit better in some cases and in most cases they are doing a bit worse). This indicates that the original TFT does not provide appropriate incentives for nodes to stay around as seeds.
- In all cases of ‘CTFT’, staying nodes download faster than non-staying nodes because CTFT favours nodes which stay around as seeds, which illustrates that CTFT has the desired effect of providing incentives for nodes to stay around as seeds.

**Figure 5** Staying fraction (see online version for colours)



## 4.2 Multi-torrent applications

Above, we focused on potential performance benefits of our multi-torrent approach, where for clarity of exposition we made simplifying assumption about user and file characteristics. However, in the real world, file sizes can vary significantly and depending on application, users can exhibit different behaviour. Thus, in this section we explore our approach in the context of the following potential multi-torrent applications in which users have natural common interest and which we believe can benefit from a multi-torrent approach:

- game patches
- online movie rentals
- software installers.

Focusing on specific applications, allows us to consider realistic file sizes as well as user behaviour,<sup>7</sup> e.g., we allow each node to download from a different number of torrents, and we also allow ‘re-seeding’. By ‘re-seeding’ we mean that nodes can join the system and immediately become seeds in some torrents by sharing files which they downloaded at some earlier time. The ability and willingness to share files downloaded earlier may be appropriate for some applications, as we explore below. This is similar to the multi-torrent characteristics considered in Guo et al. (2005), with the main difference being that in our approach we do this using local (to a node) information and in a distributed manner, i.e., without the aid of trackers or additional protocol modifications as in Guo et al. (2005).

Each simulation run below corresponds to 30h, with the initial 10h being warmup time; we increase our simulation time due to the increase in file sizes as well as greater heterogeneity, e.g., in file sizes and torrent popularity.

#### 4.2.1 *Game patches*

The first application we consider is an online game patch system. With the popularity of online games, fast delivery of game patches is difficult because:

- the online game community is huge (e.g., World of Warcraft (WoW), reached 10 M subscribers in March 2008 (see Blizzard (2008)))
- game patches such as bug fixes, feature enhancements, and map updates are released fairly frequently – while minor patches are on the order of 10 MB, major patches can be several hundred MBs
- when players leave the game, they need to install all available patches before being able to play again – since purchased game software installation discs are not (typically) patched up-to-date, new installations require downloads of all current patches as well.

All this indicates that game patch distribution results in a huge amount of internet traffic, and thus can benefit from use of BT-like systems. In fact, some game software companies have already started delivering game patch software in a P2P fashion (see Blizzard (2008)).

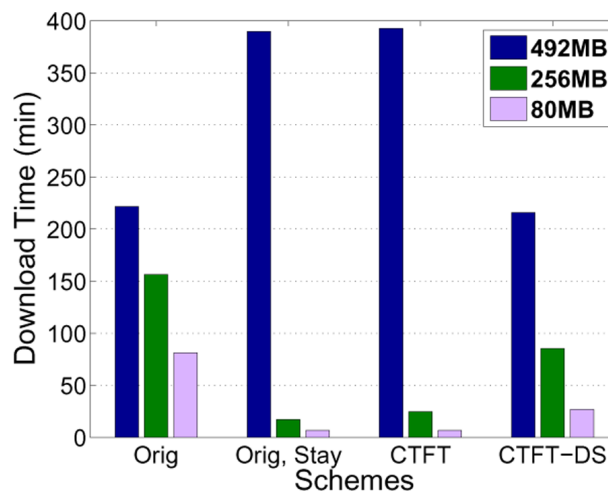
We collect three game patch file sizes from WoW’s major patch versions 2.0, 2.1, and 2.2; these are 492 MB, 256 MB, and 80 MB, respectively. For clarity of presentation, we use a homogeneous system with fast node characteristics given in Table 1, and we assume that each arriving node requires all three patches.<sup>8</sup> When CTFT-DS is used, we set the seed-to-leecher ratio ( $R$ ) to 1 and the re-evaluation interval ( $S$ ) to 3 min. We explore the effects of ( $R$ ) and ( $S$ ) in later experiments. Figure 6 depicts the resulting average download times using the following schemes:

- ‘Original’
- ‘Original, Stay’
- ‘CTFT’
- ‘CTFT-DS’,

where we make the following observations:

- Nodes staying around results in a significant decrease in the average download time of the smaller files at the expense of large files. This is due to the following – when nodes finish downloading smaller files, they continue to seed in those torrents for a long time, while they complete large file downloads. Thus, more of the overall system capacity ends up being used by the smaller file downloads (rather than larger file downloads) in the ‘Stay’ case as compared to the ‘No Stay’ case. This motivates the CTFT-DS scheme (as described in Section 3).
- The average large file download times of ‘CTFT-DS’ are as good as those of ‘Original’ and are much better than those of ‘CTFT’ or ‘Original, Stay’; at the same time smaller files are downloaded faster under ‘CTFT-DS’ than under ‘Original’ and only slightly slower than under ‘CTFT’. That is, ‘CTFT-DS’ (as compared to ‘Original’) does not hurt large file downloads while significantly improving smaller file downloads. This is due to its dynamic seeding approach which balances the number of seeds and leechers and hence shifts some of the upload capacity towards larger files. The statistics of average seed-to-leecher ratio, for each file size, can be found in Table 3. We observe that for 80 MB and 256 MB file, the seed-to-leecher ratio is quite high without dynamic seeding and is close to 1 with dynamic seeding, which is due to dynamic seeding’s re-allocations.
- We also note that:
  - 1 large file torrents lack sufficient seeds in all cases (this can be seen in Table 3 where the seed-to-leecher ratio is still quite low even with dynamic seeding) and hence further improvements are possible there
  - 2 although in Figure 6 ‘CTFT’ has similar performance to ‘Original, Stay’, it does provide incentives for nodes to stay around for seeding.<sup>9</sup>

**Figure 6** Game patch system (see online version for colours)



**Table 3** Average seed-to-leecher ratio

<i>Scheme</i>	<i>80 MB</i>	<i>256 MB</i>	<i>492 MB</i>
Original, Stay	60.553	22.001	0.002
CTFT-DS	0.995	0.922	0.004

#### 4.2.1.1 *Effect of CTFT weight*

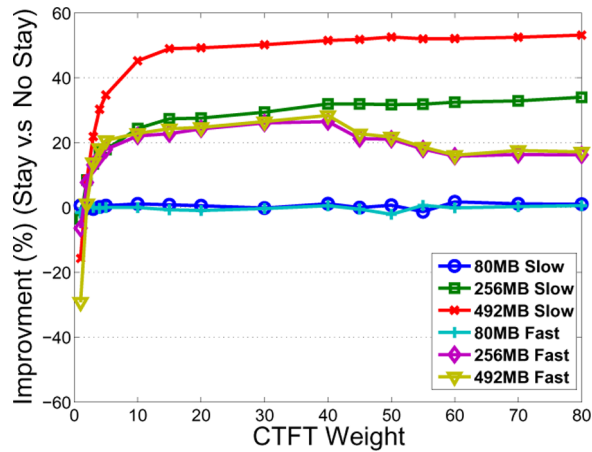
To study the impact of CTFT weight on dynamic seeding, we use a heterogeneous system with fast and slow nodes given in Table 1. We use the same file settings and assume each arriving node requires all three patches. To compare the performance of staying nodes and non-staying nodes, we let each node stay around with probability  $\frac{1}{2}$ . We experiment with CTFT weight from 1 to 100, where a weight of one corresponds to giving no special consideration for seeding. In Figure 7, we depict the resulting percentage improvement in download times experienced by staying nodes, as compared to the non-staying ones. Note that negative improvements correspond to cases where the staying nodes download slower than the non-staying ones. We observe:

- For both node classes, staying nodes under large and medium files have better performance than non-staying nodes when CTFT uses weights larger than one (i.e., when we do give ‘credit’ to nodes for seeding), which illustrates that CTFT provides proper incentives for nodes to stay around.
- For both node classes, staying nodes and non-staying nodes have similar performance under the small file, regardless of CTFT weight. This is due to:
  - 1 the download time for the small file is too short
  - 2 plenty of seeds for the small file also makes the CTFT effect less obvious.
- Slow nodes benefit more from the stay around (percentage-wise) than fast nodes, which is also true for nodes downloading the large file. This is expected, as the download times in these cases are longer.
- Staying nodes’ improvement due to CTFT is quite sensitive under small weights, but is reasonably insensitive when higher weights are used. We also note that a very high weight setting could hurt the clustering characteristics typically exhibited by BT systems. Such clustering of nodes with *similar* bandwidth capabilities serves as an incentive to contribute capacity in the original BT protocol and is studied, e.g., in Legout et al. (2007). Specifically, when the CTFT weight is high, slow nodes (which stay around) would have a higher probability of ‘clustering’ with fast nodes after they finish some downloads. This can be observed in fast nodes for the 492 MB and the 256 MB file when CTFT weight is greater than 40. Therefore, we believe that very high weights should not be used, especially since similar performance improvements can be achieved with lower weights.

In a real implementation, there are multiple approaches for nodes to determine the actual weight. One such approach is to use the tracker side assistance:

Tracker collects nodes average download time and determines the actual weight based on this. Tracker gradually increases the weight, if average download times decrease. Otherwise, tracker gradually decreases the weight. Another possible approach is to let nodes determine the weight locally by estimating peers' downloading rates. Peers' download rates can be estimated using the 'having' messages rates. We note that an accurate weight is not required as the system performance is not very sensitive to the weight setting, as shown in our experiments.

**Figure 7** Different CTFT weight (see online version for colours)

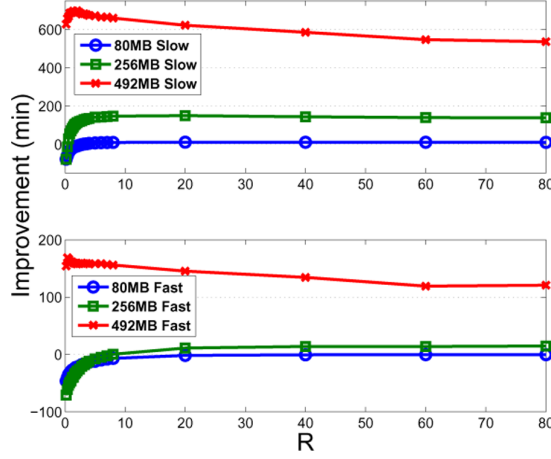


#### 4.2.1.2 Effect of seed-to-leecher ratio ( $R$ )

To study the impact of choice of  $R$  on dynamic seeding, we use a heterogeneous system with fast and slow nodes given in Table 1. We use the same file settings and assume that each arriving node requires all three patches. We let all nodes stay around and perform dynamic seeding. We experiment with  $R$  from 0.2 to 80, where a value of 1 corresponds to equal number of seeding nodes and leechers. In Figure 8, we depict the resulting improvement in download times (in number of minutes) as compared to CTFT without dynamic seeding. Note that negative improvement corresponds to cases where nodes download slower. We observe:

- For both node classes, dynamic seeding improves the performance of the large file downloading significantly. For example, the improvements of fast and slow nodes are  $\approx 688$  min and  $\approx 162$  min when  $R = 1$ , respectively. This is because dynamic seeding shifts the seeding capacities from the small and medium files to the large file. As expected, it also results the performance of the small and medium files downloading slower than without dynamic seeding.
- For both node classes, increasing  $R$  results in a faster download time for the small and the medium size file. This is due to more nodes seeding as a result of a larger  $R$ . However, increasing  $R$  hurts the performance of the large file because more seeding capacity is used for the small and the medium files and hence less is shifted towards the large file.

In a real implementation, nodes can determine the value of  $R$  using similar approaches to determining the CTFT weight (as discussed earlier).

**Figure 8** Different seed-to-leecher ratio ( $R$ ) (see online version for colours)

#### 4.2.1.3 Effect of re-evaluation interval ( $S$ )

The purpose of having nodes re-evaluate the seed-to-leecher ratio is to let them adapt to the possible changes in the network, e.g., a sudden burst of arrivals. With a sudden network change, the seed-to-leecher ratio for a node's neighbouring peers can change dramatically. Seeding without re-evaluation can inefficiently allocate the system resources, resulting in degradation in system performance. To study the impact of re-evaluation interval ( $S$ ) on dynamic seeding, we use a homogeneous system with fast nodes given in Table 1. For a clearer presentation, we let each node download the small (80 MB) and the large (492 MB) files only. We insert a burst of arrivals in the simulation, between the 900th and the 960th min. The arrival rate during the bursty period is five times greater than the normal arrival rate. We compare the average download rate using 10, 180, 360 s re-evaluation intervals with that of a system without re-evaluation. We depict the average download rates for the 492 MB and the 80 MB files in Figures 9 and 10, respectively. We only depict the results starting with 840th min to focus on the burst's effect. We observe:

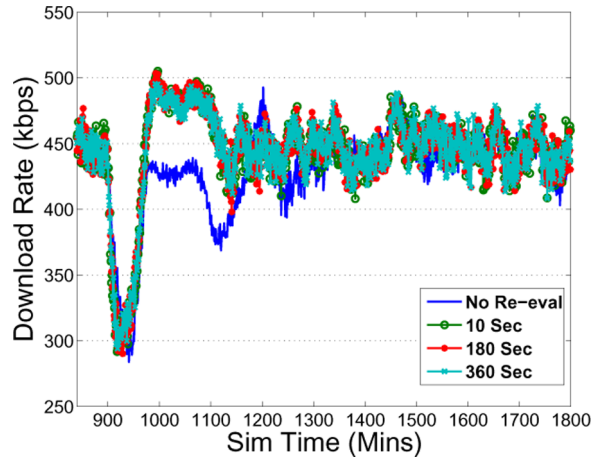
- For the 492 MB file, with re-evaluation, the download rate recovers faster from the performance drop between 900 and 950 min and maintains a higher rate after that. This is because with higher arrivals nodes become seeds for the 80 MB file and thus the seeding capacity for the 80 MB file experiences a surplus. With re-evaluation, the system is able to re-allocate resources from the 80 MB file to the 492 MB file faster.
- For the 80 MB file, with re-evaluation, the system is able to maintain a higher download rate when the burst occurs. There is a smaller performance drop between 900 and 950 min than in the system of without re-evaluation. This is because more system resources are allocated to the 80 MB file when the burst occurs. We also observe that without re-evaluation, the average download rate is much higher than that with re-evaluation. This is because with bursty arrivals, more nodes become seeds for the 80 MB file and the seeding capacity

for 80 MB file becomes significantly larger than needed. For example, with re-evaluation, the 80 MB file downloads only  $\approx 1$  min slower than without re-evaluation. This indicates that system resources are not used as efficiently without re-evaluation.

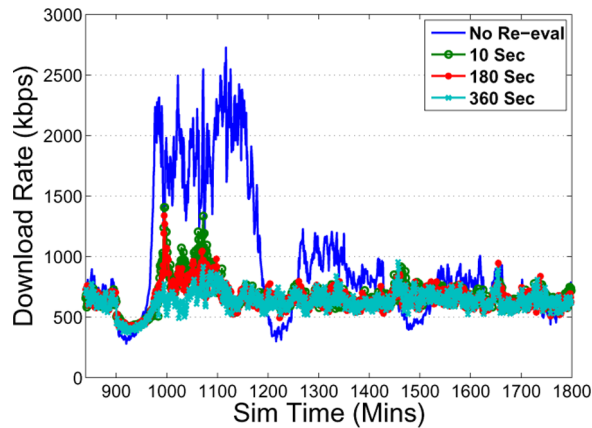
- Without re-evaluation, the system experiences some oscillations after the arrival burst. This can be observed in the performance drops of the 80 MB file around 1200, 1500 min. The reason is that there are more node departures than arrivals in these periods, due to the burst. This also shows that the system's behaviour is more stable with re-evaluation.

The above observations demonstrate that the system adapts to bursty arrivals better with re-evaluation. We also note that the performance is, in general, not very sensitive to  $S$  and therefore an accurate value of  $S$  is not required in a real implementation.

**Figure 9** 492 MB file with bursty arrival (see online version for colours)



**Figure 10** 80 MB file with bursty arrival (see online version for colours)



#### 4.2.2 Online movie rentals

Next, we consider an online movie rental system, e.g., such as Apple’s iTunes Store or Netflix. The large file sizes typical of this application (e.g., a 720p movie on iTunes is  $\approx 1.2$  GB) result in long download times and high costs to providers for hosting the movies. Thus, this is another application where a BT-like system would be useful. For our experiments, we collected the length and ratings for the Top 10 rented movies from NetFlix (a popular online movie rental site in USA). For tractability of experiments, we assume a movie’s bitrate to be 450 kbps. Due to the difficulty in obtaining real statistics of movie popularities, we use the number of times each movie is rated by customers as its popularity indicator; we normalise this number by the total number of customer ratings to obtain the resulting popularity distribution. The collected movie file sizes and popularity distribution are given in Table 4.

**Table 4** NetFlix Top 10 Movie, 3/18/08

<i>Size (MB)</i>	<i>Len (min)</i>	<i># Rating</i>	<i>Pop (%)</i>
333	101	1920102	10
363	110	1677314	9
369	112	2394935	13
372	113	2283232	12
386	117	1480414	8
396	120	1344481	7
402	122	1751443	9
405	123	2535077	13
448	136	1943487	10
498	151	1790922	9

We consider heterogeneous nodes in Table 1 where each arriving node randomly (uniformly) selects  $Y$  torrents (movies) to join, where  $Y$  is between 1 and  $M$ . It will participate as a leecher in one of them (representing the movie it wants to download) and as a seed (i.e., as described earlier it ‘re-seeds’) in the remaining  $Y - 1$  of them (representing movies it downloaded previously and is willing to share). The specific  $Y$  torrents (movies) are chosen based on movie popularity (in this case using the distribution given in Table 4). And, the choice of which one (out of  $Y$ ) is the movie of interest (i.e., the one to be downloaded) is chosen randomly (uniformly). The results of our experiments are depicted in Figures 11 and 12, where  $M = 4$ . Here, we compare the average download times for the following schemes:

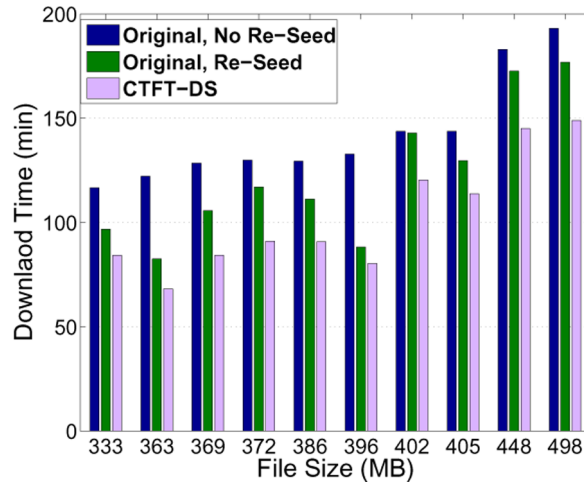
- 1 ‘Original, No Re-Seed’
- 2 ‘Original, Re-Seed’
- 3 ‘CTFT-DS’.

We observe the following:

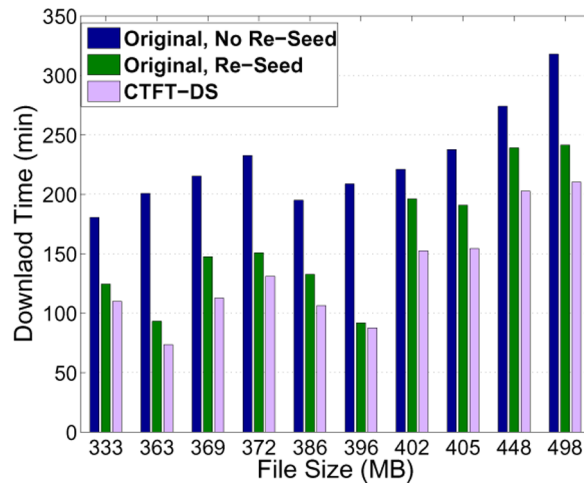
- 1 Under ‘Original, No Re-Seed’, nodes download much slower than under ‘Original, Re-Seed’ and ‘CTFT-DS’, i.e., re-seeding helps

- 2 'CTFT-DS' has better performance than 'Original, Re-Seed'; we also observe significant reduction in download times for large files under 'CTFT-DS'
- 3 Moreover, as in the case of 'CTFT', 'CTFT-DS' provides incentives for nodes to stay around while 'Original, Re-Seed' lacks such incentives.

**Figure 11** Online Movie Rental (Fast Nodes) (see online version for colours)



**Figure 12** Online Movie Rental (Slow Nodes) (see online version for colours)



#### 4.2.3 Software installers

The last application we consider is a software installer, e.g., such as Google Pack installer and Microsoft Live installer. In such applications, software companies put together software bundles for user downloads; this can be software from the same company or popular software packages. We consider this application as we believe

it can also benefit from a BT-like system. In our experiments, we collect file sizes and average numbers for software downloads from “http://www.mininova.org”, a major BT sites in USA, which are listed in Table 5. Due to the difficulty in obtaining real statistics about software popularity, we estimate it using our collected average number of downloads for each module (normalised by the total number of downloads).

We consider heterogeneous nodes in Table 1 where each arriving node randomly (uniformly) selects  $Y$  torrents (software modules) to join, where  $Y$  is between 1 and  $M$ . It then randomly (uniformly) chooses  $Z$  out of  $Y$  in which it will participate as a leecher (representing software modules it wants to download); it participates as a seed in the remaining  $Y - Z$  (representing software modules it downloaded previously and is willing to share). The specific  $Y$  torrents (software modules) are chosen based on software popularity (in this case using the distribution given in Table 5). In the experiments presented in Figure 13 and 14,  $M = 4$ . This is quite similar to the movie rental application, with the exception of nodes joining as leechers in multiple torrents and with significantly larger variance in file sizes. In these experiments we compare the resulting average download times of the following schemes:

- 1 ‘Original, No Stay, No Re-Seed’,
- 2 ‘Original, Stay, Re-Seed’
- 3 ‘CTFT-DS’.

**Table 5** Adobe SW on MiniNova, 3/18/08

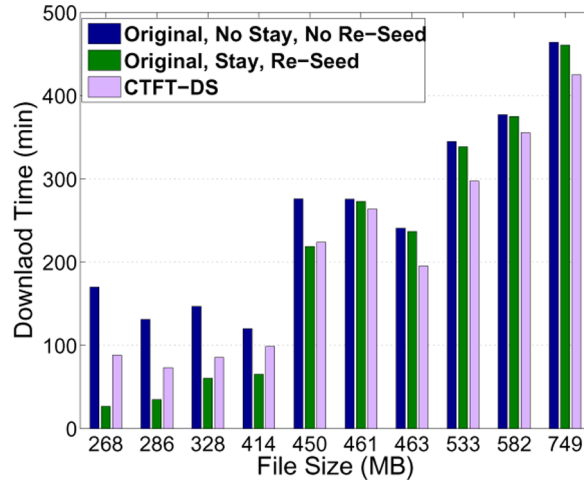
<i>Size (MB)</i>	<i>Name</i>	<i># DL</i>	<i>Pop (%)</i>
268	Acrobat	748	14
286	DreamWeaver	121	2
328	ColdFusion	79	2
414	Flex	42	1
450	Illustrator	42	1
461	Flash	425	8
463	Photoshop	2763	54
533	After Effects	658	13
582	Premiere	178	3
749	Indesign	90	2

Note that, in (1) there is no re-seeding and no staying around (i.e., this essentially corresponds to the current BT). We observe the following:

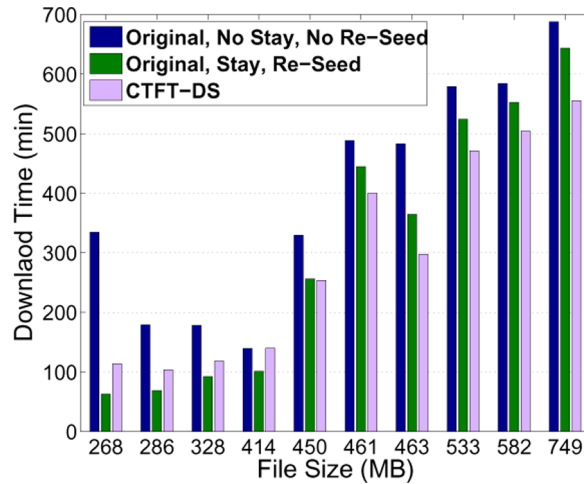
- 1 Dynamic seeding helps large file downloads, which can be observed from the difference in download times between ‘CTFT-DS’ and ‘Original, Re-Seed’, for 533 MB, 582 MB, and 749 MB files
- 2 ‘Original, No Stay, No Re-Seed’ has the worst overall performance; for fast and slow nodes it downloads significantly slower than ‘CTFT-DS’
- 3 Small file downloads do best with ‘Original, Stay, Re-Seed’, as is the case for 268 MB, 286 MB, and 328 MB files.

As before, this improvement comes at the expense of slowing down large file downloads, which is remedied by our CTFT-DS scheme. As also illustrated before, the ‘Original’ approach, unlike ‘CTFT-DS’, does not provide incentives for nodes to stay around.

**Figure 13** Software Installer (Fast Nodes) (see online version for colours)



**Figure 14** Software Installer (Slow Nodes) (see online version for colours)



## 5 Further discussion

In this section we briefly discuss on-going and future directions for further improvements as well as considerations which should be addressed when designing and developing a multi-torrent system.

### 5.1 *Performance in the wild*

While our simulation-based study of nodes acting as seeds demonstrates the benefits of multi-torrents (refer to Section 4), we expect the improvements to be even greater in the real world. One reason is that new nodes in the real world have a much longer initial ramp-up period because of various delays (e.g., for acquiring peer list, connection establishment, and clustering) which do not exist in the simulator. Thus, having seeds in the system could improve the ramp up period of new nodes significantly. In addition, a new node that joins the system with re-seeding in other torrents can boost its own initial ramp up even more in the real-world when we use our proposed CTFT algorithm. Moreover, in the real world, the download times of different torrents would have a larger variance than in the simulator. This would result in longer stay around times, which would again result in more significant improvements in the overall system performance. Lastly, as mentioned in Section 2, nodes staying around and re-seeding helps keep torrents alive. This effect is also not illustrated in our simulation study as we use constant node arrival rates. This is, however, predicted by the analytical model in Guo et al. (2005).

### 5.2 *Requirements and further improvements*

Although nodes with common torrents may exist in the same network at the same time, they may not be peers in *all* their common torrents, which could diminish the effects of our approach.<sup>10</sup> One possible enhancement would then be to have trackers assist nodes in finding peers with common torrents.

### 5.3 *System parameters*

A number of interesting system characteristics can be studied by exploring various system and proposed schemes' parameters, e.g., different approaches to weight settings in CTFT, as briefly discussed in Section 3. A number of other system parameters could also affect the performance of multi-torrents, e.g., the peer set size and the number of peers chosen for unchoking. Studying the effects of such parameters on multi-torrent performance is the topic of our on-going efforts.

### 5.4 *Practical implementation*

Our CTFT approach is completely decentralised and does not require any modifications to the BT *communication* protocol. That is, we only need to modify the BT client locally (e.g., by adding a bit more state information at each peer and changing the TFT unchoking algorithm to the CTFT algorithm given in Section 3.) Thus, our approach is easily deployable in the current BT system.

### 5.5 *Malicious behaviour*

When using our CTFT mechanism, we give a greater weight to downloading rates of peers who are seeds in some torrents. A malicious node could take advantage of this by pretending that it is a seed in some torrent. Of course, this would lead to the malicious node not being unchoked in that torrent (by the peers to which it lies) as the malicious node would have to claim to have all data. However, doing this may improve the malicious nodes probability of being unchoked in some

other torrents. Our future efforts include a study of how malicious behaviour affects system performance, in the context of multi-torrents, and what counter measures are possible. One direction for detecting a lying node could be to observe that a seed node is able to serve any requested data chunks while a lying node may not have all the chunks to serve requests. We also note that malicious behavior does require code modification which is not accessible to everyone.

### *5.6 Other multi-torrent possibilities*

Other possibilities in designing multi-torrents include exploration of better bandwidth allocation. Instead of evenly splitting upload resources among all unchoking peers (as is done in the current BT system), assigning different bandwidth to different peers is a possibility; this has been proposed in Piatek et al. (2007). Similar ideas can be used in the context of multi-torrents. Moreover, scheduling among multiple downloads may be another interesting issue in multi-torrents, e.g., when a node has a number of files to download, should it download all of them in parallel or a few at a time, and if the latter which ones and how many at a time. Answering such open questions is the topic of future work.

## **6 Related work**

BitTorrent (BT) (described by Cohen (2003)) has been the topic of a large number of research efforts. Thus, here we give a brief overview of those works most related to ours. Although measurement studies, e.g., as in Guo et al. (2005), Izal et al. (2004), indicate that seeds exist in real BT systems, there are no incentives for nodes to stay around as seeds. In our work, we focus on providing incentives for nodes to stay around as seeds in the context of a multi-torrent system.

As noted, most BT studies focus on a single torrent while measurements in Guo et al. (2005) suggest that 85% of users participate in multiple torrents. To the best of our knowledge (Guo et al., 2005) is the only other effort which studies multiple torrents. Specifically (Guo et al., 2005) develop an analytical model of multiple torrents where it is assumed that a node participating as a leecher in a particular torrent is willing to serve as a seed in torrents in which it has participated some time earlier in its lifetime. This model is then used to illustrate that their multiple torrent approach can extend the lifetime of a torrent, i.e., the time until some chunk is lost from a torrent due to the departure of one of its nodes. The work in Guo et al. (2005) also proposes a multi-torrent design based on a tracker overlay system, used to facilitate information exchange among peers (e.g., which peer can participate as a seed in which torrent). Guo et al. (2005) open an interesting research problem and provide useful solutions. However, this work does leave a number of open questions (as discussed in Section 1). Specifically, it does not explore how to provide incentives for nodes to act as seeds in torrents they have completed earlier (it only briefly suggests the use of exchange-based incentives (see Anagnostakis and Greenwald, 2004) without providing details of how these would be used). In contrast, our work focuses on addressing this question. Moreover (Guo et al., 2005) do not provide a quantitative evaluation of the performance consequences to the nodes willing to be seeds as well as to the overall system – the

focus of Guo et al. (2005) is primarily on extending a torrent's lifetime (as described above). In contrast, our work illustrates a number of performance tradeoff to be considered in a multi-torrent system. Our work differs from Guo et al. (2005) in several other aspects as well. In our experiments, nodes can join their torrents of interest simultaneously and behave as seeds while they have not completed all their downloads, whereas (Guo et al., 2005) only considers what we referred to as 're-seeding'. In addition, the proposed multi-torrent design in Guo et al. (2005) requires modifications to the BT tracker, and specifically requires a tracker overlay system. In contrast, our approach makes only local modifications, while maintaining the original spirit of BT where nodes exhibit local selfish behaviour (as discussed earlier).

A number of works, e.g., Bharambe et al. (2006), Fan et al. (2006), Piatek et al. (2007), Jun and Ahamad (2005), studied various incentive-related issues in the context of *single* torrent systems. For instance, Freedman et al. (2008) use a market based approach to incentivise sharing, where efficient network resource allocation is achieved by relating file values (in the market) to their relative demand. At a high level, our dynamic seeding approach can be thought of as also attempting to match seeding capacity demand with supply; however, it does that in a distributed manner and by exploiting local information only. Provision of incentives in P2P systems is also studied in the context of virtual currency or micropayment based approaches, e.g., as in Zhong et al. (2003), Vishnumurthy et al. (2003). The main limitation of such an approach is that currency management is fairly complex and a centralised administrative authority is often needed. In contrast, CTFT requires only local knowledge and is more practical for real world deployment.

Lastly, Piatek et al. (2007) is an example of a work which develops a strategic BT client (as an illustration of what can be achieved with malicious behaviour) which attempts to get the most reciprocation through better assignment and scheduling of upload capacity to peers. They also briefly mentions how such a client would work in multiple swarms (i.e., multiple torrents). However, no details or evaluation are given. The primary focus of that work is to exploit the BT system and show that its performance can be degraded by clients which try to maximise the download capability of a malicious node, i.e., such a node would not act as a seed. In our work, we try to encourage nodes to stay as seeds by providing better service for the seeding nodes which results in an overall system performance gain. Because our work provides incentives (through better performance) for nodes to stay around as seeds, we conjecture that our CTFT mechanism would hurt the client proposed in Piatek et al. (2007) as it is not expected to be a seed.

## 7 Conclusions

We focused on a multi-torrent system, and specifically on the questions of what incentives could be provided for nodes to contribute resources as seeds in a multi-torrent environment, and what are the resulting performance consequences of such behaviour, both on the nodes which are willing to be seeds and on the overall system. Our study illustrated that performance gains are possible through consideration of multiple torrents. We believe that this is a promising research area with a number of remaining future directions.

## References

- Anagnostakis, K.G. and Greenwald, M.B. (2004) 'Exchanged-based incentive mechanisms for peer-to-peer file sharing', *Proc. ICDCS*, Tokyo, Japan, pp.524–533.
- Bharambe, A.R., Herley, C. and Padmanabhan, V.N. (2006) 'Analyzing and improving bittorrent performance', *Proc. INFOCOM*, Barcelona, Spain, pp.1–12.
- Blizzard, G.H. (2008) *World of Warcraft*, <http://www.worldofwarcraft.com>
- CacheLogic (2005) *Peer-to-Peer in 2005*, <http://www.cachelogic.com/home/pages/research>
- Chow, A.L.H., Golubchik, L. and Misra, V. (2008) 'Improving bittorrent: a simple approach', *Proc. IPTPS*, Tampa Bay, FL, USA.
- Cohen, B. (2003) 'Incentives build robustness in bittorrent', *Proc. P2PECON*, Berkeley, CA, USA.
- Fan, B., Chiu, D-M. and Lui, J.C. (2006) 'The delicate tradeoffs in bittorrent-like file sharing protocol design', *Proc. ICNP*, Santa Barbara, CA, USA, pp.239–248.
- Freedman, M.J., Aperijs, C. and Johari, R. (2008) 'Prices are right: managing resources and incentives in peer-assisted content distribution', *Proc. IPTPS*, Tampa Bay, FL, USA.
- Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X. and Zhang, X. (2005) 'Measurements, analysis and modeling of bittorrent-like systems', *Proc. IMC*, Berkeley, CA, USA, pp.35–48.
- Izal, M., Urvoy-Keller, G., Biersack, E.W., Felber, P., Al Hamra, A.A. and Garc'es-Erice, L. (2004) 'Dissecting bittorrent: five months in a torrent's lifetime', *Proc. PAM*, Antibes Juan-les-Pins, France, pp.1–11.
- Jun, S. and Ahamad, M. (2005) 'Incentives in bittorrent induce free riding', *Proc. P2PECON*, Philadelphia, PA, USA.
- Legout, A., Liogkas, N., Kohler, E. and Zhang, L. (2007) 'Clustering and sharing incentives in bittorrent systems', *Proc. SIGMETRICS*, San Diego, CA, USA, pp.301–312.
- Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A. and Venkataramani, A. (2007) 'Do incentives build robustness in bittorrent?', *Proc. NSDI*, Cambridge, MA, USA, pp.1–14.
- Qiu, D. and Srikant, R. (2004) 'Modeling and performance analysis of bittorrent-like peer-to-peer networks', *Proc. SIGCOMM*, Portland, OR, USA, pp.367–378.
- Sirivianos, M., Park, J.H., Chen, R. and Yang, X. (2007) 'Free-riding in bittorrent networks with the large view exploit', *Proc. IPTPS*, Bellevue, WA, USA.
- Vishnumurthy, V., Chandrakumar, S. and Siler, E.G. (2003) 'KARMA: a secure economic framework for p2p resource sharing', *Proc. P2PECON*, Berkeley, CA, USA.
- Yang, Y., Chow, A.L.H. and Golubchik, L. (2007) *Multi-Torrent: A Performance Study*, <http://vista.usc.edu/pub/multibt-tech.pdf>
- Zhong, S., Chen, J. and Yang, Y.R. (2003) 'Sprite: a simple, cheat-proof, credit-based system for mobile ad-hoc networks', *Proc. INFOCOM*, San Francisco, CA, USA, pp.1987–1997.

## Notes

<sup>1</sup>In Section 4, we also explore applications where nodes participate as seeds in some torrents (using previously downloaded files) while they act as leechers in torrents of current interest to them, i.e., similarly to the scenarios explored in Guo *et al.* (2005).

<sup>2</sup>The benefit of doing such estimates locally is ease of implementation and low protocol overhead.

<sup>3</sup>Specific parameter setting for  $R$  and  $S$  are discussed in Section 4.

- <sup>4</sup>It does not simulate TCP behaviour and simply shares the upload capacity of a peer evenly between its uploading sessions. The end-game mode and sub-chunk details are also not represented. This simulator is used by other groups as well, and we believe that these simplifications do not affect our results qualitatively.
- <sup>5</sup>We experimented with a broad range of distributions, and the results were qualitatively similar; due to lack of space we only present representative results with other results given in Yang *et al.* (2007).
- <sup>6</sup>We believe that this type of effect on the network and peer set sizes also contributes to the differences in performance improvements observed in the previous experiment (i.e., above settings with random selection would have smaller network/peer set sizes than those with fixed selection).
- <sup>7</sup>We still consider the case where nodes start all their downloads simultaneously. In the real world, nodes may start new downloads while other downloads are in progress. This would increase the stay around time and should result in a greater benefit from our approach. Studying this would require more detailed modelling of user behaviour and is outside the scope of this paper.
- <sup>8</sup>Future work includes investigation of effects of patch release dates and other characteristics of game systems.
- <sup>9</sup>This was illustrated in other experiments, when we depicted a breakdown of download times between staying and non-staying nodes. Due to lack of space, we do not include such a breakdown for this experiment.
- <sup>10</sup>Our scheme will degenerate into the original TFT, without additional overheads, when this commonality is not present.