

Multi-Torrent: a Performance Study*

Yan Yang
Univ. of Southern California

Alix L.H. Chow
Univ. of Southern California

Leana Golubchik
Univ. of Southern California

Abstract

BitTorrent (BT) has become an extremely popular and successful peer-to-peer (P2P) file sharing system. Although empirical evidence suggests that most nodes participate in multiple torrents, surprisingly little research literature exists on this topic. In this paper, we focus on a multi-torrent system, and specifically on the questions of (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 of such behavior, both on the nodes which are willing to be seeds and on the overall system. We give evidence of the current system lacking incentives for nodes to stay around as seeds in a multi-torrent environment. Motivated by that, we propose a cross torrent based method and then present a simulation-based performance study to illustrate the benefits of our approach.

1. Introduction

In recent years BitTorrent [5] (BT) has become an extremely popular and successful peer-to-peer (P2P) file sharing system. According to [4], BT accounts for about 35% of all the traffic on the Internet, more than all other P2P applications combined.

Briefly, in BT, nodes join the system, after receiving “start-up” information from a tracker, and begin requesting chunks of data from their neighbors¹. 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 neighbors 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 neighbors). The TFT mechanism in BT is an example of “local selfish behavior”, where leechers try to upload to the most reciprocal neighbors. Seeds also pick a subset of neighbors (typically 5), and upload data to them. In past versions of BT, seeds chose neighbors with the highest download rates. In a more recent protocol, [11], the seeding capacity is distributed more uniformly among the neighboring peers. All these choices are re-evaluated periodically.

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., [2, 6, 13], to name a few. These works provide significant insight into the performance, fairness, scalability (and other characteristics) of BT; we give an overview of such research efforts and their relationship to ours in Section 4. Most of these studies (with a notable exception of [8]) 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, movies/music from the same genre, games under the same theme, software and game patches, and so on. Specifically, from the trace analysis given in [8], such related interests result in most peers (> 85%) participating in multiple torrents. Moreover, since (a) in BT users obtain start-up information from a particular source (e.g., a forum or a web-site) and (b) there is correlation in content (e.g., video from the same TV series, music from the same category, etc.), we expect there to be a reasonably high probability that users would download from multiple common torrents simultaneously.

However, in the current BT system, there is no mechanism for “relating” 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 is an important problem. Such games have millions of subscribers which (periodically) need to download

*This research was funded in part by the USC Annenberg Graduate Fellowship, the NSF 0091474, 0417274, 0540420 grants and by IMSC, an NSF ERC, Cooperative Agreement No. EEC-9529152.

¹The 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.

patches; this 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 [3], but also that there is an opportunity for a multi-torrent approach. A more detailed description of this application is given in Section 3. Other applications include movie rentals and software installers; due to lack of space, we do not discuss them here and refer an interested reader to [15].

Intuitively, a multi-torrent approach might have the following advantages. Typically, it is desirable to have “seeds” in a torrent. Briefly, seeds can contribute to (a) helping newly joined nodes ramp up so that they can become contributing peers faster, (b) helping nodes nearing the end of their downloads find the last few file pieces/chunks faster, and (c) keeping a torrent “alive” (i.e., making sure that all pieces of a file are available in the system). Although current studies, e.g., as in [8, 9], 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 behavior 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.* We note that acting as a seed in one torrent while downloading in another might hurt a node’s performance, as 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 TBT unchoking from peers which could provide data it does need. That is, in the absence of proper mechanisms, a selfish peer might be better off using all its uploading capacity in the torrents where it is a leecher.

Surprisingly, little exists in the literature which considers relating multiple torrents. To the best of our knowledge, [8] is the only work which studies this topic. Specifically, in [8] 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. While the work in [8] 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, and (2) what are the resulting performance consequences of such behavior, 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 to 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. Lastly, we extend an existing BT simulator [2] with our multi-torrent approach and use it to 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 [8] 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 2). Unlike, e.g., the exchange-based incentives suggested in [1], 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 and (b) our approach does provide incentives for nodes to act as seeds by providing better performance for such nodes (refer to Section 3).

2. Proposed Approach

We begin with a simple (simulation-based) example which motivates the use of multiple torrents and the need for incentivizing 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.

Motivating Example: here, we have slow and fast nodes (with node capacities and mix given in Table 2) where each node arriving to the system joins 2 (randomly

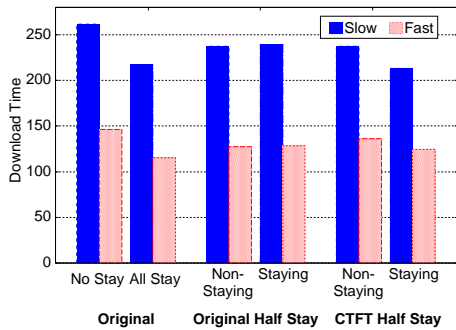


Figure 1. Motivating Example

chosen) torrents out of 10 available ones. (Refer to Section 3 for the simulator details, used to generate results described below, with corresponding simulation settings given in Table 1.) 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: (1) original BT where none of the nodes stay around, (2) original BT where all nodes stays around, (3) original BT where each node stays around with probably $\frac{1}{2}$, (4) BT with our simple modification to the current TFT mechanism, which we term cross-torrent TFT (CTFT), where each node stays around with probably $\frac{1}{2}$.

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.

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, the “stay around” in this experiment 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” behavior is needed. Therefore, a question we consider here is *how to encourage nodes to stay around* in order to reap the potential benefits of a multi-torrent approach. To this end, we propose a simple modifica-

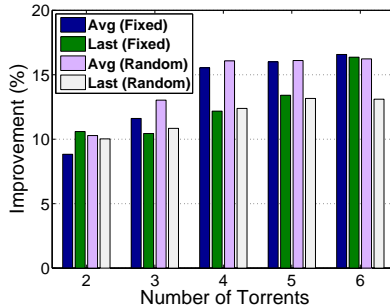


Figure 2. Staying Around Improvement

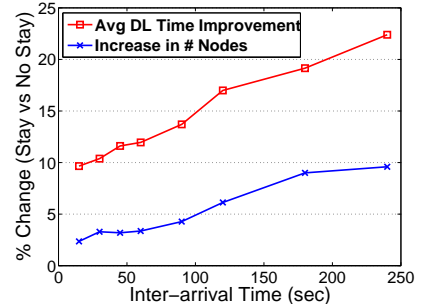


Figure 3. Node Inter-Arrival Time

tion to the current TFT mechanism (as detailed below), and employ it in experiment (4). As depicted in Figure 1, nodes which do stay around experience better average download times (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.

There are a number of approaches one could take - in this section we present one such simple approach in order to explore and illustrate potential benefits of a multi-torrent approach. 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 behavior”. Lastly, we would like our scheme to be easily implementable and deployable. To this end we focus on local modifications (not centralized approaches), and without the need of assistance from (or modifications to) the tracker(s) as in [8]). Given this, our scheme modifies the TFT unchoking mechanism of leechers, by considering multiple torrents and favoring nodes which stay around as seeds. We term this approach *Cross-Torrent Tit-for-Tat* (CTFT).

Cross-Torrent TFT (CTFT): CTFT modifies the unchoking part of leechers’ behavior 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 favor 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 3.)

For example, 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.

We illustrate benefits of the proposed approach through an extensive performance study in Section 3. 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 3) is for a nodes 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 local² estimate (i.e., based on its neighbors 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 esti-

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

mate and hence the choice, of in which torrents to seed, is re-evaluated every S time units, where S is another system parameter³. 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. Due to lack of space, we do not evaluate the combined effects of these approaches here.

3. Performance Study

In the following performance study we use an event-based simulator provided by [2] which simulates BT’s chunk exchange mechanism. To explore our proposed approach, we modify the simulator in [2] 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 (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 2).
- We allow node arrivals; in what follows we use a Poisson arrival process with rate λ .
- 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 given in Table 1. The system starts with one origin seed per torrent, each with $1000kbps$ 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 distribution⁴ are given in Table 2. When simulating a homogeneous system, we use the fast class.

To have 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, we also use the same torrent selection sequence. In the following simulations, we look at the steady state behavior of

³Specific parameter setting for R and S are discussed in Section 3.

⁴We 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 [15].

Table 1. Simulation Settings

Filesize	200 MB (800 Chunks, 256 KB each)
Simulation Time	12 hours (+3 hours Warmup)
Avg node inter-arrival time ($\frac{1}{\lambda}$)	45 sec
Peer Set Size	40
Leecher Unchokings	4 Regular + 1 Optimistic
#Seed Unchokings	5

Table 2. 2 Classes Bandwidth Distribution

Class	Fraction	Download Capacity	Upload Capacity
Slow	40%	1500kbps	128kbps
Fast	60%	5000kbps	512kbps

the system. Each simulation run corresponds to 15 hours, and we only compute our results over the last 12 hours⁵.

Unless otherwise stated, we focus on two metrics (1) the average download time over all torrents and (2) 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.

To illustrate the performance consequences of our approach, we first explore the effects of different parameters using simple scenarios. Below we use the following notation for the various schemes being simulated: (a) “No Stay” refers to all nodes leaving each torrent as soon as the download in that torrent is complete; (b) “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; (c) “Original” refers to the use of the original BT TFT mechanism; and (d) “CTFT” refers to the use of our proposed CTFT mechanism with $W = 4$ by default.

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 2. Figure 2 depicts the percentage improvement⁶ in download time, as compared to the “No Stay” case, for two experiments (1) when the choice of torrents to join is *fixed*, i.e., all nodes join the same X torrents, and (2) 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 2. From these results we observe the following: (a) having nodes stay around as seeds improves average and last download times for both fixed and random selection experiments, and (b) in most

⁵We examine the number of nodes in the system to make sure the system passes the “ramp up” stage during the first 3 hours. Due to lack of space we omit the details of this; they can be found in [15].

⁶Due to lack of space, we only present percentage improvements; the specific values of our metrics are given in [15].

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. Although there is not a monotonic trend in the *percentage* improvement as a function of number of torrents, the actual improvements are approximately linear [15].

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 2 and each node joining the same 3 torrents. Figure 3 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 size⁷ for each node, and, in general, larger peer set sizes result in better BT performance [14].

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 2 where each node randomly select 3 torrents to join out of 10. Figure 4 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 fast 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 observe:

- 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 favors nodes that stay around as seeds, which illustrates that CTFT has the desired effect of providing incentives for nodes to stay around as seeds.

Different Weight of CTFT: Here, we study the effect of using different CTFT weights, as described in Section 2. We consider a heterogeneous system in Table 2 where

⁷We 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).

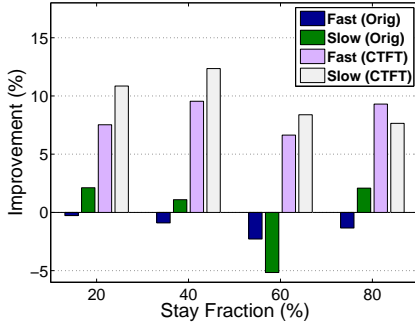


Figure 4. Staying Fraction

each node randomly selects 3 torrents to join out of 10. We experiment with CTFT weights from 1 to 20, where a weight of 1 corresponds to giving no special consideration for seeding. In this experiment, each node stays around with probability $\frac{1}{2}$. In Figure 5, we depict the resulting percentage improvement in download times⁸ 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:

- Staying nodes have better performance than non-staying nodes when CTFT uses weights larger than 1 (i.e., when we do give “credit” to nodes for seeding), which illustrates that CTFT provides proper incentives for nodes to stay around.
- Slow nodes benefit more from staying around (percentage-wise) than fast nodes. This is reasonable as the download times of slow nodes is longer and hence the effect of CTFT on them is more pronounced. Similarly, slow nodes are more sensitive to the weight values than the fast nodes. However, effects of the weight level off fairly quickly. The fact that the system performance is not very sensitive to the weight settings is a desirable characteristic, i.e., we can achieve desirable effects without necessarily determining optimal settings.

We also conjecture 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 [11]. 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 the larger improvement of the “Last Download Time” relative to that of the “Average

⁸For clarity of presentation, we only depict results for weights 1 to 5; higher weight results are similar.

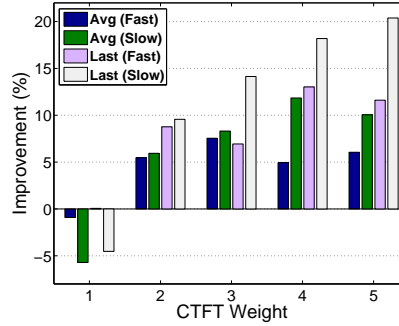


Figure 5. CTFT Weight

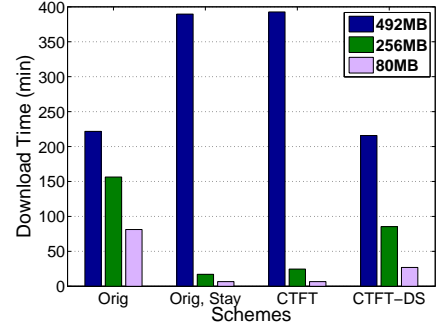


Figure 6. Game Patch System

Download Time” (of the slow nodes). Therefore, we believe that very high weights should not be used, especially since similar performance improvements can be achieved with lower weights.

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 behavior. Thus, next we explore our approach in the context of a potential multi-torrent application, namely a game patch system, in which users have natural common interest and which we believe can benefit from a multi-torrent approach. Focusing on specific applications, allows us to consider realistic file sizes as well as user behavior⁹. Other applications we have explored include online movie rentals and software installers¹⁰. Due to lack of space, results for these applications are not included here and can be found in [15].

Each simulation run below corresponds to 30 hours, with the initial 10 hours 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. When CTFT-DS is used, we set $R = 1$ and $S = 3$ mins¹¹.

Game Patches: We now consider an online game patch system. With the popularity of online games, game patch distribution results in a huge amount of Internet traffic, and thus can benefit from the use of BT-like systems. In fact,

⁹In the real world, nodes may start new downloads while other downloads are in progress; this should increase the stay around time and result in a greater benefit from our approach. Studying this would require modeling of user behavior and is outside the scope of this paper.

¹⁰There, we allow nodes to download from a different number of torrents and to “re-seed”, i.e., join the system and immediately become seeds in some torrents by sharing files which they downloaded at some earlier time. This is similar to [8], with the main difference being that we only use local information and in a distributed manner, without the aid of trackers or additional protocol modifications as in [8].

¹¹We experimented with different values of R and S ; a more thorough exploration of the resulting tradeoffs is part of on-going efforts.

some game software companies have already started delivering game patch software in a P2P fashion [3].

We collect 3 game patch file sizes from World of Warcraft (WoW)'s [3] major patch versions 2.0, 2.1, and 2.2; these are 492MB, 256MB, and 80MB, respectively. For clarity of presentation, we use a homogeneous system with fast node characteristics given in Table 2, and we assume that each arriving node requires all three patches¹². Figure 6 depicts the resulting average download times using the following schemes: (1) "Original"; (2) "Original, Stay"; (3) "CTFT"; and (4) "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 2).
- 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 numbers of seeds and leechers, for each file size, can be found in [15].
- Also: (1) large file torrents lack sufficient seeds in all cases and hence further improvements are possible there, and (2) although in Figure 6 "CTFT" has similar performance to "Original, Stay", it does provide incentives for nodes to stay around for seeding¹³.

We now 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.

Performance in the Wild: While our simulation-based study demonstrates the benefits of multi-torrents, we expect the improvements to be even greater in the real world.

¹²Future work includes investigation of effects of patch release dates and other characteristics of game systems.

¹³This 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.

One reason is that new nodes in BT have a much longer initial ramp-up period because of various delays (e.g., due to acquiring peer list, connection establishment, clustering) which do not exist in the simulator. Thus, having more seeds in the system could improve the ramp-up period of new nodes significantly. 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 should result in more significant improvements in the overall system performance. Lastly, nodes staying around helps keep torrents alive. This effect is not illustrated in our simulation study as we use constant node arrival rates, but is explored in [8].

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, appropriate settings of CTFT-DS parameters (i.e., R and S). For instance, a shorter S would give us finer control of seeding capacity; however, it may result in higher overheads as well as a potential adverse effect on TCP performance. Moreover, real implementations should consider including some randomness in seeding decisions, as synchronization in nodes making such decisions may result in oscillations in seeding capacity. 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 a topic of our on-going efforts.

Malicious Behavior: 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 behavior 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.

4. Related Work

BT, originally described in [5], has been the topic of a large number of a variety of research efforts. Thus, here we give a brief overview of those works most related to ours.

As already noted, most BT studies focus on a single torrent while measurements in [8] suggest that 85% of users participate in multiple torrents. To the best of our knowledge, [8] is the only other effort which studies multiple torrents. Specifically, in [8] the authors 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 [8] 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). The work in [8] opens up an interesting research problem and provides useful solutions. However, it 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 [1] without providing details of how these would be used). In contrast, our work focuses on addressing this question. Moreover, [8] does 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 in [8] is primarily on extending a torrent’s lifetime (as described above). In contrast, our work illustrates a number of performance trade-offs to be considered in a multi-torrent system. Our work differs from [8] 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 [8] only considers what we referred to as “re-seeding”. In addition, the proposed multi-torrent design in [8] requires modifications to the BT tracker, and specifically requires a tracker overlay system. In contrast, our CTFT approach is completely decentralized and only requires local client modifications (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 2). Our modifications maintain the original spirit of BT where nodes exhibit local selfish behavior (as discussed earlier)¹⁴.

A number of works, e.g., [2, 6, 10, 13], have studied various incentive-related issues in the context of *single* torrent systems. In contrast, [7] uses a market based approach to incentivize 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. Part of our future efforts is to investigate other, such as market-based and reputation-based (e.g., as in [12]) approaches to providing incentives for nodes to seed, but in the context of multi-torrent systems.

5. 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 behavior, 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

- [1] Anagnostakis and Greenwald. Exchanged-based incentive mechanisms for peer-to-peer file sharing. In *ICDCS*, 2004.
- [2] Bharambe, Herley, and Padmanabhan. Analyzing and improving bittorrent performance. In *INFOCOM*, 2006.
- [3] Blizzard. World of warcraft web site.
- [4] CacheLogic. Peer-to-peer in 2005.
- [5] Cohen. Incentives build robustness in bittorrent. In *P2PECON*, 2003.
- [6] Fan, Chiu, and Lui. The delicate tradeoffs in bittorrent-like file sharing protocol design. In *ICNP*, 2006.
- [7] Freedman, Aperijs, and Johari. Prices are right: Managing resources and incentives in peer-assisted content distribution. In *IPTPS*, 2008.
- [8] Guo, Chen, Xiao, Tan, Ding, and Zhang. Measurements, analysis and modeling of bittorrent-like systems. In *IMC*, 2005.
- [9] Izal, Urvoy-Keller, Biersack, Felber, Hamra, and Garc’erice. Dissecting bittorrent: Five months in a torrent’s lifetime. In *PAM*, 2004.
- [10] Jun and Ahamad. Incentives in bittorrent induce free riding. In *P2PECON*, 2005.
- [11] Legout, Liogkas, Kohler, and Zhang. Clustering and sharing incentives in bittorrent systems. In *SIGMETRICS*, 2007.
- [12] Lian, Peng, Yang, Zhang, Dai, and Li. Robust incentives via multi-level tit-for-tat. In *IPTPS*, 2006.
- [13] Piatek, Isdal, Anderson, Krishnamurthy, and Venkataramani. Do incentives build robustness in bittorrent? In *NSDI*, 2006.
- [14] Sirivianos, Park, Chen, and Yang. Free-riding in bittorrent networks with the large view exploit. In *IPTPS*, 2007.
- [15] Yang, Chow, and Golubchik. Multi-torrent: a performance study. Technical report, CS Dept, USC.

¹⁴Our scheme will degenerate into the original TFT, without additional overheads, when there is no commonality in participating torrents.