Analyzing and Improving a BitTorrent Network's Performance Mechanisms

Ashwin R. Bharambe
Carnegie Mellon University

Cormac Herley
Microsoft Research

Venkata N. Padmanabhan
Microsoft Research

***Abstract*— In recent years, BitTorrent has emerged as a very scalable peer-to-peer file distribution mechanism. While early measurement and analytical studies have verified BitTorrent's performance, they have also raised questions about various metrics (upload utilization, fairness, etc.), particularly in settings other than those measured. In this paper, we present a simulation-based study of BitTorrent. Our goal is to deconstruct the system and evaluate the impact of its core mechanisms, both individually and in combination, on overall system performance under a variety of workloads. Our evaluation focuses on several important metrics, including peer link utilization, file download time, and fairness amongst peers in terms of volume of content served.**

Our results confirm that BitTorrent performs near-optimally in terms of uplink bandwidth utilization, and download time except under certain extreme conditions. We also show that low bandwidth peers can download more than they upload to the network when high bandwidth peers are present. We find that the rate-based tit-for-tat policy is not effective in preventing unfairness. We show how simple changes to the tracker and a stricter, block-based tit-for-tat policy, greatly improves fairness.

I. INTRODUCTION

BitTorrent [1] has recently emerged as a very scalable P2P content distribution tool. In a P2P system like BitTorrent, peers not only download content from the server but also serve it to other peers. Thus the serving capacity of the system grows with the number of nodes, making the system potentially self-scaling. In BitTorrent, a file is broken down into a large number of blocks and peers can start serving other peers as soon as they have downloaded their first block. Peers preferentially download blocks that are rarest among their local peers so as to maximize their usefulness to other peers. These strategies allow BitTorrent to use bandwidth *between* peers (*i.e., perpendicular bandwidth* [2]) effectively and handle flash crowds well. In addition, BitTorrent incorporates a tit-for-tat (TFT) incentive mechanism, whereby nodes preferentially upload to peers from whom they are able to download at a fast rate in return.

The soundness of these architectural choices is borne out by the success of the system in actual deployment. Recent measurement and analytical studies [3–5] (discussed in Section III) indicate that BitTorrent scales well with system size and has performed well in real large-scale file distributions. However, we believe that these studies leave a number of questions unanswered. For example:

- Izal et al. [3] reported that clients observed high download rates. However, was this optimal? Could BitTorrent have achieved even higher bandwidth utilization in this setting?
- Does BitTorrent's Local Rarest First (LRF) policy for picking new blocks to download from peers effectively avoid the last block problem?
- How effective is BitTorrent's TFT policy in avoiding unfairness *i.e.,* a node uploads much less than it downloads? Fairness can be important to encourage peer participation, for instance, in settings where the ISP charges users based on upload traffic volume.
- Previous studies have assumed that at least a fraction of nodes perform altruistic uploading even after finishing their downloads. However, if nodes depart as soon as they finish, is the stability or scalability of the system hurt significantly?

The answers depend on a number of parameters that BitTorrent uses. It would be difficult, if not impossible, to control such a large space of possibilities in an analytical or live measurement setting. Hence, in this paper, we attempt to answer these questions using a simulator which models the data-plane of BitTorrent.[1] We believe our study is complementary to previous BitTorrent studies. Our principal findings are as follows:

First, we find BitTorrent to be remarkably robust and scalable at ensuring high uplink bandwidth utilization. It scales well as the number of nodes increases, keeping the load on the origin server bounded, even when nodes depart immediately after completing their downloads. The LRF policy performs better than alternative block-choosing policies in a wide range of environments (e.g., flash crowd, post-flash crowd situations, small network sizes, etc.) By successfully getting rid of the last block problem, it provides a simpler alternative to previously proposed source coding strategies, e.g., Digital Fountain [6].

Second, we find that BitTorrent shows sub-optimal behavior in certain "extreme" workloads:

1) The bandwidth of the origin server is a precious resource, especially when it is scarce. It is important that server deliver *unique* packets to the network at least as quickly as they can be diffused among the peers. We present a simple way of ensuring this.
2) BitTorrent's rate based TFT mechanisms do *not* prevent unfairness in terms of the data served by nodes, especially in node populations with heterogeneous bandwidths. We demonstrate that clustering of similar nodes using *bandwidth matching* is key to ensuring fairness

[1]We do not consider control-plane issues such as the performance of the centralized *tracker* used for locating peers.

without sacrificing uplink bandwidth utilization.

3) BitTorrent's LRF policy "equalizes" the replication rate of all blocks. Hence, in a setting where there is a wide range in the fraction of a file that different nodes already possess (say because nodes that have downloaded a file partially rejoin during a subsequent flash crowd), BitTorrent is not very effective at allowing nodes who have most of a file to rapidly find the few blocks that they are missing.

In addition to elaborating on these findings, we present simple yet effective techniques to alleviate the unfairness in BitTorrent.

The main focus of our evaluation is on BitTorrent's performance in flash-crowd scenarios. We believe that such scenarios are a critical test for systems such as BitTorrent, since it is such overload conditions that make provisioning a server-based content distribution system hard and make peer-to-peer alternatives such as BitTorrent seem appealing. That said, we do consider a few alternative scenarios as well.

The rest of the paper is organized as follows: in Section II, we present a brief overview of the BitTorrent system. Section III discusses related analytical and measurement-based studies. Section IV describes our simulation environment and the evaluation metrics. More details about our methodology can be found in [7]. The main contribution of our work appears in Sections V through VIII, where we present a simulation-based evaluation of BitTorrent under a variety of configurations and workloads. We conclude in Section IX.

II. BitTorrent Overview

BitTorrent [1] is a P2P application whose goal is to enable fast and efficient distribution of large files by leveraging the *upload* bandwidth of the downloading peers. The basic idea is to divide the file into equal-sized *blocks* (typically 32-256 KB in size) and have nodes download the blocks from multiple peers concurrently. The blocks are further subdivided into *sub-blocks* to enable *pipelining* of requests so as to mask the request-response latency [8].

Corresponding to each large file available for download (called a *torrent*), there is a central component called the *tracker* that keeps track of the nodes currently in the system. The tracker receives updates from nodes periodically (every 30 minutes) as well as when nodes join or leave the torrent.

Nodes in the system are either *seeds*, i.e., nodes that have a complete copy of the file and are willing to serve it to others, or *leechers*, i.e., nodes that are still downloading the file but are willing to serve the blocks that they already have to others.[2] When a new node joins a torrent, it contacts the tracker to obtain a list containing a random subset of the nodes currently in the system (both seeds and leechers). The new node then attempts to establish connections to about 40 existing nodes, which then become its *neighbors*. If the number of neighbors of a node ever dips below 20, the node contacts the tracker again to obtain a list of additional peers it could connect to.

[2]In the rest of the paper, unless otherwise specified, we will use the terms "node" and "leecher" interchangeably.

Each node looks for opportunities to download blocks from and upload blocks to its neighbors. In general, a node has a choice of several blocks that it could download. It employs a *local rarest first (LRF)* policy in picking which block to download: it tries to download the block that is least replicated among its neighbors.

A *tit-for-tat (TFT)* policy is employed to guard against free-riding: a node preferentially uploads to neighbors that provide it the best download rates. Each node limits the number of concurrent uploads to a small number, typically 5. Seeds have nothing to download, but they follow a similar policy: they upload to up to 5 nodes that have the highest download rate.

The mechanism used to limit the number of concurrent uploads is called *choking*, which is the temporary refusal of a node to upload to a neighbor. Only the connections to the chosen neighbors (up to 5) are *unchoked* at any point in time. A node reevaluates the download rate that it is receiving from its neighbors every 10 seconds to decide whether a currently unchoked neighbor should be choked and replaced with a different neighbor.

BitTorrent also incorporates an *optimistic unchoke* policy, wherein a node, in addition to the normal unchokes described above, unchokes a randomly chosen neighbor regardless of the download rate achieved from that neighbor. Optimistic unchokes are typically performed every 30 seconds. This mechanism allows nodes to discover neighbors offering higher download rates, as well as enables new nodes to obtain their first block.

III. Related Work

There have been analytical as well as measurement-based studies of the BitTorrent system. At the analytical end, Qiu and Srikant [5] have considered a simple fluid model of BitTorrent. Their main findings are: (a) the system scales very well, i.e. the average download time is not dependent on the node arrival rate, and (b) file sharing is very effective, i.e. there is a high likelihood that a node holds a block that is useful to its peers.

Izal et al. [3] and Pouwelse et al. [4] present measurement-based studies of BitTorrent based on tracker logs of different torrents. The main findings of these studies are: (a) the average download rate is consistently high; (b) as soon as a node has obtained a few chunks, it is able to start uploading to its peers (the local rarest first policy works); (c) the node download and upload rates are positively correlated (tit-for-tat policy works); and (d) nodes might not stay on in the system (as seeds) after completing their download. So it is the few long-lived seeds that are critical for file availability. Accordingly, in our experiments, we simulate a small number of long-lived seeds, with leechers departing as soon as they have finished downloading.

Gkantsidis and Rodriguez [9] present a simulation-based study of a BitTorrent-like system. They show results indicating that the download time of a BitTorrent-like system is not optimal, especially in settings where there is heterogeneity in node bandwidth. They go on to propose a network coding [10] based scheme called Avalanche that alleviates these problems.

Our study differs from previous studies of BitTorrent in the following important ways: first, while the analytical study reported in [5] presents the steady state scalability properties of BitTorrent, it ignores a number of important BitTorrent parameters (e.g., node degree (d), maximum concurrent uploads (u)), and environmental conditions (e.g., seed bandwidth, etc.) that could affect uplink bandwidth utilization. Second, previous studies only briefly allude to free-riding; in this paper, we quantify systematic unfairness in BitTorrent and present mechanisms to alleviate it.

Finally, there have been a number of proposals for peer-to-peer content distribution systems that employ a BitTorrent-like swarming approach (although at least some of these were proposed independently of BitTorrent). The innovations in these proposals include the use of erasure coding (e.g., [2]), adaptive strategies for peer selection (e.g., Slurpie [11]), and a combination of the two (e.g., Bullet [12] and Bullet' [13]). While some of the techniques we evaluate here overlap with those presented in this body of work, we believe that our analysis is able to uncover the impact of each technique individually in the context of BitTorrent, something that is hard to do with new and different systems that incorporate a number of new techniques. Also, the issue of fairness that we focus on has not received much attention in this previous work.

IV. Methodology

To explore aspects of BitTorrent that are difficult to study using traces of real torrents [3, 4] or analysis [5], we use a simulation-based approach for understanding and deconstructing BitTorrent performance. Such an approach provides the flexibility of carefully controlling the configuration parameters of the various BitTorrent mechanisms, or even selectively turning off certain mechanisms and replacing them with alternatives. This would be difficult or even impossible to achieve using live Internet measurement techniques (e.g., using tracker logs [3, 4] or by participating in a live-torrent). Thus, while certain interactions specific to a real deployment will be missed, we believe the abstraction is rich enough to expose most details that are relevant to our experiments. We briefly describe our simulator and define the metrics used in our evaluation. More details can be found in [7].

A. Simulator Details

Our discrete-event simulator models peer activity (joins, leaves, block exchanges) as well as many of the associated BitTorrent mechanisms (local rarest first, tit-for-tat, etc.) in detail. We have made the software available [14]. The network model associates a downlink and an uplink bandwidth for each node, which allows modeling asymmetric access networks. The simulator uses these bandwidth settings to appropriately delay the blocks exchanged by nodes. The delay calculation takes into account the number of flows that are sharing the uplink or downlink at either end, which may vary with time. Doing bandwidth-sensitive delay computation for each block transmission is expensive enough that we have to limit the maximum scale of our experiments to 8000 nodes on a P4 2.7GHz, 1GB RAM machine.

Given the computational complexity of even the simple model above, we decided to simplify our network model in the following ways. First, we do not model network propagation delay, which is relevant only for the small-sized control packets (*e.g.*, the packets used by nodes to request blocks from their neighbors). We believe that this simplification does not have a significant impact on our results because (a) the download time is dominated by the data traffic (*i.e.*, block transfers), and (b) BitTorrent's pipelining mechanism (Section II) masks much of the control traffic latency in practice. Second, we do not model the packet-level dynamics of TCP connections. Instead, we make the assumption that connections traversing a link share the link bandwidth equally, with the share of each connection fluctuating as the number of connections varies. Note that BitTorrent's store-and-forward mode of operation at the granularity of blocks is unaffected by this simplification. Although this simplification means that TCP "anomalies" (e.g., timeouts) are not modeled, we believe that the relatively long length of the connections mitigates this issue. Note that, while an individual block in BitTorrent may not be very long (32-256 KB), data flow is kept as continuous as possible using pipelining of block requests. Finally, we do not model shared bottleneck links in the interior of the network. We assume that the bottleneck link is either the uplink of the sending node or the downlink of the receiving node. While Akella et al. [15] characterize bandwidth bottlenecks in the interior of the network, their study specifically ignores edge-bottlenecks by conducting measurements only from well-connected sites (*e.g.*, academic sites). The interior-bottlenecks they find are generally fast enough ($\geq$ 5 Mbps) that the edge-bottleneck is likely to dominate in most realistic settings. Hence we believe that our focus on just edge-bottlenecks is reasonable.

We also make one simplification in modeling BitTorrent itself. We ignore the *endgame mode*[8], which is used by BitTorrent to make the end of a download faster by allowing a node to request the sub-blocks it is looking for in parallel from multiple peers. This is inconsequential to our study because: (a) the endgame mode has no effect on steady-state performance, and (b) it does not help with the "last block" problem which we study in Section VI-C. Recall that the last block problem occurs when a node has difficulty *finding* a peer that possesses the last block, increasing the overall download time significantly in many distribution systems. Since the endgame mode assumes the availability of the last block at multiple peers it plays no role in removing this potential bottleneck.

B. Metrics

We quantify the effectiveness of BitTorrent in terms of the following metrics:

Link utilization: We use the mean utilization of the peers' uplinks and downlinks over time as the main metric for evaluating BitTorrent's efficacy. The utilization at any point in time is computed as the ratio of the aggregate traffic flow on all uplinks/downlinks to the aggregate capacity of all uplinks/downlinks in the system; *i.e.,* the ratio of the actual flow to the maximum possible.

Notice that if all the uplinks in the system are saturated, the system as a whole is serving data at the maximum possible rate. While downlink utilization is also an important metric to consider, the asymmetry of most Internet access links makes the uplink the key determinant of performance. Also, by design, duplicate file blocks (*i.e.*, blocks that a leecher already has) are never downloaded. Note that a small amount of uplink bandwidth could be "wasted" in BitTorrent due to duplicate sub-block requests during endgame mode; however, as noted above, we do not model this mode. Hence, the *mean download time* for leechers is inversely related to the average uplink utilization. Because of this and the fact that observed uplink utilization is easier to compare against the optimal value (i.e., 100%), we do not explicitly report the mean download time for most of our experiments.

Fairness: The system should be fair in terms of the number of blocks served by the individual nodes. No node should be *compelled* to upload much more than it has downloaded. Nodes that *willingly* serve the system as seeds are, of course, welcome, but deliberate free-riding should not be possible. Fairness is important for there to be an incentive for nodes to participate, especially in settings where ISPs charge based on uplink usage or uplink bandwidth is scarce.

We quantify fairness using the *normalized* count of file blocks uploaded by a node, i.e., the number of blocks uploaded divided by the number of blocks in the file. So, for example, a normalized load of 1.5 means that the node serves a volume of data equivalent to 1.5 copies of the file. We also use the normalized count of blocks served to quantify the load on the seed(s) in the system.

Optimality: Throughout this paper we will refer to a system as having optimal utilization if it achieves the maximum possible link utilization, and having complete fairness if every leecher downloads as many blocks as it uploads. We will refer to the system as being optimal on the whole if it has optimal utilization *as well as* complete fairness. Note that a heterogeneous setting can have differing degrees of fairness for the same level of bandwidth utilization.

V. Experiment Overview

A. Workload Derived from a Real Torrent

In order to set the stage for the experiments to follow, we first examine how our simulator performs under a realistic workload. We consider two important workload parameters: (a) node arrival pattern, and (b) uplink and downlink bandwidth distribution. To derive realistic arrival patterns, we use the tracker log for the Redhat 9 distribution torrent [3]. Table I describes the distribution of peer bandwidth, which was derived from the Gnutella study reported in [16]. While discretizing the CDFs presented in [16], we excluded the tail of the distribution. This means that (a) dial-up modems are eliminated, since it is unlikely that they will participate in such large downloads, and (b) very high bandwidth nodes are eliminated, making the setting more bandwidth constrained. We set the seed bandwidth to 6000 kbps.

Downlink (kbps)	**Uplink** (kbps)	**Fraction of nodes**
784	128	0.2
1500	384	0.4
3000	1000	0.25
10000	5000	0.15

TABLE I: Bandwidth distribution of nodes derived from the actual distribution of Gnutella nodes [16].

In order to make the simulations tractable, we made two changes. First, we used a file size of 200 MB (with a block size 256 KB), which is much smaller than the actual size of the Redhat torrent (1.7 GB). This means the download time for a node is smaller and the number of nodes in the system at any single point is also correspondingly smaller. Second, we present results only for the *second* day of the flash crowd. This day witnesses over 10000 node arrivals; however, due to the smaller file download time, the maximum number of active nodes in the system at any time during our simulations was about 300.

The simulation results can be summarized as follows: The observed uplink utilization was 91%; this means that the overall upload capability of the network is almost fully utilized. However, this comes at the cost of considerable skew in load across the system. The seed serves approximately 127 copies of the file into the network. Worse, some clients uploaded 6.26 times as many blocks as they downloaded, which represents significant unfairness. These results lead to a number of interesting questions:

1) How robust is the high uplink utilization to variations in system configuration and workload, e.g., differing number of seeds and leechers, join-leave patterns, bandwidth distribution, etc.?
2) Can the fairness of the system be improved without hurting link utilization?
3) How well does the system perform when there is heterogeneity in terms of the extent to which leechers have completed their download e.g., new nodes coexisting with nodes that have already completed most of their download?
4) How sensitive is system performance to parameters such as the node degree (i.e., the number of neighbors of a node) and the maximum number of concurrent uploads?

To answer these questions, we present a detailed simulation-based study of BitTorrent in the sections that follow.

B. Road-map of Experiments

Unless otherwise specified, we use the following default settings in our experiments:

- File size: 100 MB (400 blocks of 256 KB each)
- Number of initial seeds: 1 (the origin server, which stays on throughout the duration of the experiment)
- Seed uplink bandwidth: 6000 Kbps

- Number of leechers that join the system (n): 1000
- Leecher downlink/uplink bandwidth: 1500/400 kbps
- Join/leave process: a flash crowd where all leechers join within a 10-second interval. Leechers depart as soon as they finish downloading.
- Number of neighbors of each node (degree d): 7.
- Maximum number of concurrent upload transfers (u): 5

The key parameters that affect the evolution of a torrent are: (1) the number of seed(s) and their serving capacity, (2) the number of leechers that wish to download, (3) the policies that nodes use to swap blocks among themselves, (4) the distribution of node upload/download capacities, and (5) the arrival/departure pattern of the leechers.

We start in Section VI by examining only (1), (2) and (3). That is, we consider a homogeneous setting where all leechers have the same downlink/uplink bandwidth (1500/400 Kbps by default, as noted above), and arrive in a flash crowd. We explore the impact of the number of leechers, the number of initial seeds, aggregate bandwidth of seeds, etc. This section also evaluates BitTorrent's LRF policy for picking blocks. We wish to point out that, although we use a small node degree ($d = 7$), our results are not affected at higher node degrees. We present results with a small node degree to emphasize BitTorrent's resilience with respect to this parameter.

Then in Section VII we examine (4) and turn to a heterogeneous flash-crowd setting where there is a wide range in leecher bandwidth. We consider 3 kinds of connectivity for leechers: high-end cable (6000/3000 Kbps), high-end DSL (1500/400 Kbps), and low-end DSL (784/128 Kbps).

Finally, in Section VIII, we turn to (5) and consider workloads other than a pure flash-crowd scenario. In particular, we consider cases where leechers with very different download "objectives" coexist in the system. For instance, new nodes in the post-flash crowd phase compete with nodes that have already downloaded most of the blocks. Likewise, an old node that reconnects during the start of a new flash crowd to finish the remaining portion of its download competes with new nodes that start their downloads.

VI. Homogeneous Environment

In this section, we study the performance of BitTorrent in a setting consisting of a homogeneous (with respect to bandwidth) collection of leechers. Figures presented in this section and the next (Sec. VII) represent the results of a single simulation run, since the differences across multiple runs were found to be statistically insignificant.

A. Number of nodes

First we examine the performance of the system with increasing network size. We vary the number of nodes that join the system from 50 to 8000. All nodes join during a 10 second period, and remain in the system until they have completed the download. Figure 1a shows that upload capacity utilization (see Section IV) is close to 100% regardless of system size. Utilization is a little short of 100% because of the start-up phase when nodes are unable to utilize their uplinks effectively. The high uplink utilization indicates that the system is performing almost optimally in terms of mean download time. The downlink utilization, on the other hand, is considerably lower. This is expected given the asymmetric access links of the leechers.

Another important measure of scalability is how the work done by the seed varies with the number of leechers. We measure this in terms of the normalized number of blocks served, *i.e.*, the number of blocks served divided by the number of blocks in one full copy of the file. Ideally, we would like the work done by the seed to remain constant or increase very slowly with system size. Figure 1b shows that this is actually the case. The normalized number of blocks served by the seed rises sharply initially (as seen from the extreme left of Figure 1b) but then flattens out. The initial rise indicates that the seed is called upon to do much of the serving when the system size is very small, but once the system has a critical mass of 50 or so nodes, peer-to-peer serving becomes very effective and the seed has to do little additional work even as the system size grows to 8000.

In summary, BitTorrent performance scales very well with increasing system size both in terms of bandwidth utilization and the work done by the seed.

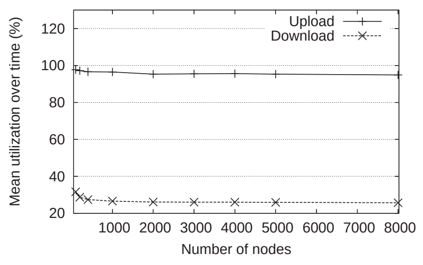
B. Number of seeds and bandwidths of seeds

Next we consider the impact of numbers of seeds and aggregate seed bandwidth on the performance of BitTorrent. We first consider the case where there is a single seed, and then move on to the case of multiple seeds.

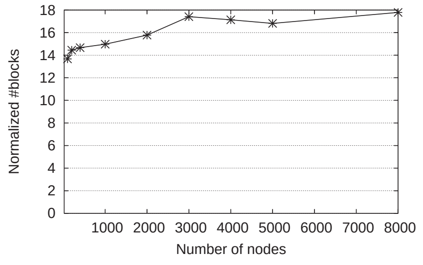
Figure 1c shows the mean upload utilization as the bandwidth of a single seed varies from 200 Kbps to 1000 Kbps. The "nosmartseed" curve corresponds to default BitTorrent behavior. We see that upload utilization is very low (under 40%) when the seed bandwidth is only 200 Kbps. This is not surprising since the low seed bandwidth is not sufficient to keep the uplink bandwidth of the leechers (400 Kbps) fully utilized, at least during the start-up phase. However, even when the seed bandwidth is increased to 400 or 600 Kbps, the upload utilization is still considerably below optimal.

Part of the reason for poor upload utilization is that seed bandwidth is wasted serving duplicate blocks prematurely *i.e.*, even before one full copy of the file has been served. We find that about 50% of the blocks are served prematurely when the seed bandwidth is 400 kbps. Thus, despite the LRF policy, multiple nodes connected to the seed can operate in an uncoordinated manner and independently request the same block.

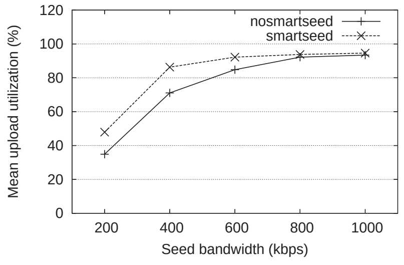
Once identified, there is a simple fix for this problem. We have implemented a *smartseed* policy, which has two components: (a) The seed does not choke a leecher to which it has transferred an incomplete block. This maximizes the opportunity for leechers to download and hence serve complete blocks. (b) For connections to the seed, the LRF policy is replaced with the following: among the blocks that a leecher is looking for, the seed serves the one that it has served the least. This policy improves the diversity of blocks in



(a)



(b)



(c)

Fig. 1: (a) Mean upload and download utilization of the system as the flash-crowd size increases. (b) Contribution of the seed node for different sized flash-crowds. (c) Change in upload utilization as bandwidth of the seed is varied.

the system, and also prevents premature serving of duplicate blocks. This results in a noticeable improvement in upload utilization, especially when seed bandwidth is limited and precious (Figure 1c). It is interesting to note that some BitTorrent clients have independently and recently incorporated a similar fix, called the "super-seed" mode [17].

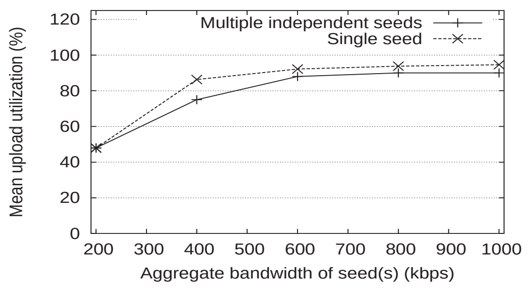


Fig. 2: Upload utilization for a single seed versus multiple independent seeds.

Finally, Figure 2 compares the cases of having a single seed and having multiple independent seeds, each with 200 Kbps bandwidth, such that the aggregate seed bandwidth is the same in both cases. All seeds employ the smartseed policy. The upload utilization suffers in the case of multiple seeds because the independent operation of the seeds results in duplicate blocks being served by different seeds, despite the smartseed policy employed by each seed.

In summary, we find that seed bandwidth is a precious resource and it is important not to waste it on duplicate blocks until all blocks have been served at least once. The "smartseed" policy, which modifies LRF and the choking policy for the seeds' connections, results in a noticeable improvement in system performance.

C. Block choosing policy and Node degree

Next we address the question of the block choosing policy. As mentioned earlier, the LRF policy is considered as one of the key aspects of BitTorrent. Here, we investigate its importance under various conditions. We assume that the seed employs the *smartseed* strategy introduced in the previous section.

Before describing our experiments let us quickly revisit the intuition behind the LRF policy. Since any rare block will automatically be requested by many leechers, it is unlikely to remain rare for long. For example, if a rare block is possessed by only one leecher, it will be among the first blocks requested by any nodes unchoked by that leecher. This, of course, decreases its rareness until it is as common in the network as any other block. This should reduce the coupon collector or "last block problem" that has plagued many file distribution systems [6]. These arguments are qualitative. The goal of this section is to measure how well LRF actually performs.

We investigate 3 issues. First, we compare LRF with an alternative block choosing policy in which each leecher asks for a block picked at random from the set that it does not yet possess but that is held by its neighbors. Second, we examine how the effectiveness of LRF varies as the seed bandwidth is varied. Since a high-bandwidth seed delivers more blocks to the network, the risk of blocks becoming rare is lower. Third, we examine the impact of varying the node degree, d, which defines the size of the neighborhood used for searching in the LRF and random policies.

Figure 3a summarizes the results with regard to the following issues: (a) random *vs.* LRF, (b) low seed bandwidth (400 kbps) *vs.* high seed bandwidth (6000 kbps), and (c) node degree, $d = 4, 7,$ and 15. In all cases, the leechers had down/up bandwidths of 1500/400 kbps. Observe that the low bandwidth seed has only as much upload capacity as one of the leechers.

The general trend is that uplink utilization improves with increases in both seed bandwidth and node degree. When node degree is low ($d = 4$), leechers have a very restricted local view. So LRF is not effective in evening out the distribution of blocks at a global level, and performs no better than the random policy. However, when node degree is even moderately large ($d = 7$ or 15) and seed bandwidth is low, LRF outperforms the random policy by ensuring greater diversity in the set of blocks held in the system. Finally, when the seed bandwidth is high, the seed's ability to inject diverse blocks into the system improves utilization and also eliminates the performance gap between LRF and the random policy. Thus, LRF makes a difference only when node degree is large enough to make the local neighborhood representative of the global state and seed bandwidth is low.

In Figure 3b, we graph the average number of *interesting* connections available to each leecher in the network for the case of $d = 7$. The connection between a node and its peer is called *interesting* if the node can send useful data to its

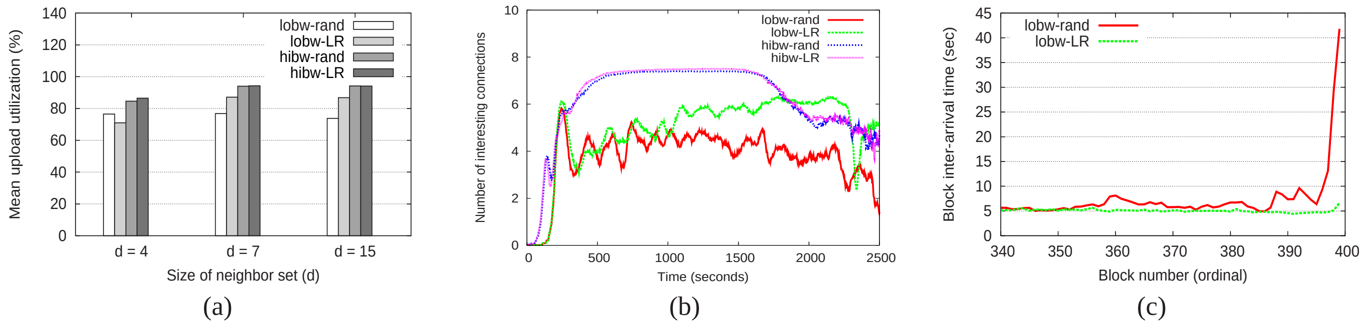


Fig. 3: (a) Upload utilization for LRF and Random policies for different values of the node degree, d. (b) Variation of the number of *interesting* connections over time for $d = 7$. (c) Inter-arrival times for blocks at the tail end of the file; each point represents the mean time to receive the k^{th} block, where the mean is taken over all nodes.

peer. As stated in the caption, each point here represents the mean number of interesting connections (averaged over all the nodes in the system) at a particular point in time. Observe that in the high seed bandwidth case there is little difference between the LRF and the random block choosing policies (the top 2 curves in Figure 3b). In the low seed bandwidth case the difference is very pronounced. With the LRF policy, the number of *interesting* connections is significantly higher, especially towards the end of the download. This underlines the importance of the LRF policy in the case where seed bandwidth is low.

Next we plot in Figure 3c the inter-arrival time between blocks in the case of a low-bandwidth seed. This is the time between the receipt of consecutive distinct blocks, averaged across all nodes. We plot this for both the LRF and the random block choosing policies, with $d = 7$ in both cases. Recall that the file size is 400 blocks, so the figure only shows the inter-arrival time of the last few blocks. The sharp upswing in the curve corresponding to the random policy clearly indicates the last-block problem. There is no such upswing with LRF.

In summary, our results indicate that the LRF policy provides significant benefit when seed bandwidth is low and node degree is large enough for the local neighborhood of a node to be representative of the global state. Nevertheless, we find that the node degree needed for LRF to be effective is quite modest relative to the total number of nodes in the system. Specifically, in a configuration with 8000 nodes, we find that LRF is effective for $d = 7$, which corresponds to each node having direct visibility to a neighborhood that represents only 0.09% of the system. However, given the scaling limitations of our simulator, we are not in a position to extrapolate this result to larger system sizes.

D. Concurrent Uploads

In BitTorrent, each node uploads to no more than a fixed number of nodes ($u = 5$, by default) at a time. This fixed upload degree limit presents two potential problems. First, having too many concurrent uploads delays the availability of full blocks to the network. That is, if a leecher's upload capacity is divided between u nodes, there can be a considerable delay before any of them has a complete block that they can start serving to others. Second, low peer downlink bandwidth can constrain uplink utilization. That is, a leecher uploading to a peer can find its *upload* pipe underutilized if the receiving node actually becomes the bottleneck on the transfer (*i.e.*, has insufficient available *download* bandwidth to receive as rapidly as the sender can transmit).

Figure 4 graphs the mean upload utilization as a function of the maximum number of concurrent uploads permitted (*i.e.*, u) for low and high bandwidth seeds. We show the results both with and without the *smartseed* fix. (Since u can be no more than d, we used $d = 60$ rather than 7 in this experiment, to allow us to explore a wide range of settings for u.) Without the *smartseed* fix, as u increases, the probability that duplicate data is requested from the seed increases causing link utilization to drop. The drop in utilization is very severe when seed bandwidth is low, since in such cases, as we have seen before, good performance critically depends on the effective utilization of the seed's uplink. We see utilization dropping gradually even when the *smartseed* fix is applied. The reason is that a large u causes the seed's uplink to get fragmented, increasing the time it takes for a node to fully download a block that it can then serve to others.

Thus, an adaptive strategy for maintaining the number of concurrent connections such as in [11, 13] may be able to achieve the right balance.

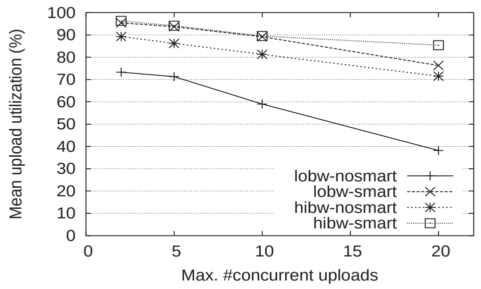


Fig. 4: Utilization for different values of the maximum number of concurrent uploads (u).

VII. Heterogeneous Environment

In this section, we study the behavior of BitTorrent when node bandwidth is heterogeneous. As described in Section IV-B, a key concern in such environments is fairness in terms of the volume of data served by nodes. Recall that in the workload derived from Redhat torrent log (Section V-A), some nodes uploaded 6.26 times as many blocks as they downloaded. Such unfairness is undesirable, especially since uplink bandwidth is generally a scarce resource. BitTorrent implements only a *rate-based* TFT policy, which can still result in unfairness in terms of the volume of data served. This section quantifies the extent of the problem and presents mechanisms that enforce stricter fairness without hurting uplink utilization significantly.

A node in BitTorrent unchokes those peers from whom it is getting the best download rate. The goal of this policy is to match up nodes with similar bandwidth capabilities. For example, a high-bandwidth node would likely receive the best download rate from other high-bandwidth nodes, and so would likely be uploading to such high-bandwidth nodes in return. To help nodes discover better peers, BitTorrent also incorporates an optimistic unchoke mechanism. However, this mechanism significantly increases the chance that a high bandwidth node unchokes and transfers data to nodes with poorer connectivity. Not only can this lead to decrease in uplink utilization (since the download capacity of the peer can become the bottleneck), it can also result in the high bandwidth node serving a larger volume of data than it receives in return. This also implies that the download times of lower bandwidth nodes will improve at the cost of higher bandwidth nodes.

We consider two simple mechanisms that can potentially reduce such unfairness: (a) Quick bandwidth estimation (QBE), and (b) Pairwise block-level TFT. Note that enforcing fairness implies that the download time of a node will be inversely related to its *upload* capacity (assuming that its uplink is slower than its downlink).

A. Quick Bandwidth Estimation

In BitTorrent, optimistically unchoked peers are rotated every 30 seconds. The assumption here is that 30 seconds is a long enough duration to establish a reverse transfer and ascertain the upload bandwidth of the peer in consideration. Furthermore, BitTorrent estimates bandwidth only on the transfer of blocks; since all of a node's peers may not have interesting data at a particular time, opportunity for discovering good peers is lost.

Instead, if a node were able to quickly estimate the upload bandwidth for all its d peers, optimistic unchokes would not be needed. The node could simply unchoke the u peers out of a total of d that offer the highest upload bandwidth.

In practice, a quick albeit approximate bandwidth estimate could be obtained using lightweight schemes based on the packet-pair principle [18] that incur much less overhead than a full block transfer. Also, the history of past interactions with a peer can be used to estimate its upload bandwidth.

In our experiments here, we neglect the overhead of QBE and effectively simulate an idealized bandwidth estimation scheme whose overhead is negligible relative to that of a block transfer.

B. Pairwise Block-Level Tit-for-Tat

The basic idea here is to enforce fairness directly in terms of blocks transferred rather than depending on rate-based TFT to match peers based on their upload rates. Suppose that node A has uploaded U_{ab} blocks to node B and downloaded D_{ab} blocks from B. With pairwise block-level TFT, A allows a block to be uploaded to B if and only if $U_{ab} \leq D_{ab} + \Delta$, where Δ represents the unfairness threshold on this peer-to-peer connection. This ensures that the maximum number of *extra* blocks served by a node (in excess of what it has downloaded) is bounded by $d\Delta$, where d is the size of its neighborhood. Note that with this policy in place, a connection is (un)choked depending on whether the above condition is satisfied or not. Also, there is no need for the choker to be invoked periodically.

Thus, provided that Δ is at least one (implying that new nodes can start exchanges), this policy replaces the optimistic unchoke mechanism and bounds the disparity in the volume of content served. However, it is important to note that there is a trade-off here. The block-level TFT policy may place a tighter restriction on data exchanges between nodes. It may so happen, for example, that a node refuses to upload to any of its neighbors because the block-level TFT constraint is not satisfied, reducing uplink utilization. We quantify this trade-off in the evaluation presented next.

C. Results

We now present performance results for vanilla BitTorrent as well as the new mechanisms described above with respect to three metrics: (a) mean upload utilization (Figures 5 and 7); and (b) unfairness as measured by the (normalized) maximum number of blocks served by a node (Figures 6 and 8). All experiments in this section use the following settings: a flash-crowd of 1000 nodes joins the torrent during the first 10 seconds. In each experiment, there are an equal number of nodes with high-end cable modem (6000 Kbps down; 3000 Kbps up), high-end DSL (1500 Kbps down; 400 Kbps up), and low-end DSL (784 Kbps down; 128 Kbps up) connectivity. We vary the bandwidth of the seed from 800 Kbps to 6000 Kbps. Seeds always utilize the *smartseed* fix.

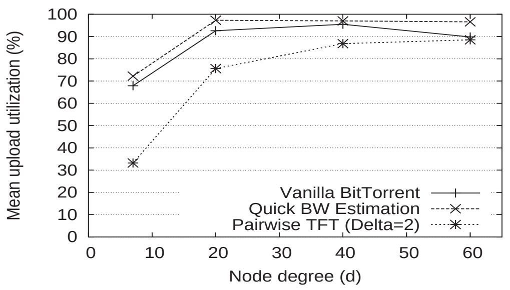


Fig. 5: Mean upload utilization for (a) vanilla BitTorrent, (b) BitTorrent with QBE, and (c) with the pairwise block-level TFT policy.

Figure 5 shows the mean upload utilization of BitTorrent and other policies in a heterogeneous setting, as a function of node degree. We find that utilization is sub-optimal in many cases, and especially low with pairwise block-level TFT, when the node degree is low ($d = 7$). The reason is that when the node degree is low, high-bandwidth nodes sometimes have only low-bandwidth peers as neighbors. This restricts the choice of nodes that the high-bandwidth node can serve to such low-bandwidth nodes, despite the QBE heuristic. A bandwidth bottleneck at the *downlink* of the low-bandwidth peer would reduce the uplink utilization at the high-bandwidth node. This degradation is particularly severe with pairwise block-level TFT, since in this case the high-bandwidth node is constrained to upload at a rate no greater than the *uplink* speed of its low-bandwidth peers. In all cases, uplink utilization improves as the node degree becomes larger, since the chances of a high-bandwidth node being stuck with all low-bandwidth peers decreases.

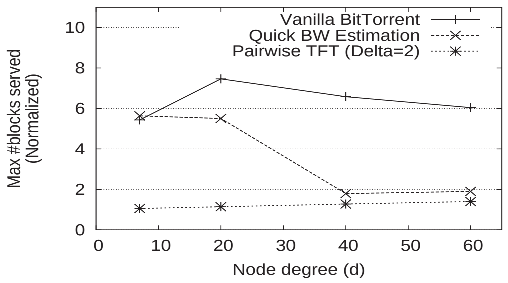


Fig. 6: Maximum number of blocks (normalized by file size) served by any node during an experiment for various policies.

The interaction between high-bandwidth nodes and their low-bandwidth peers also manifests itself in terms of a disparity in the volume of data served by nodes. Figure 6 plots the maximum number of blocks served by a node normalized by the number of blocks in the file. The seed node is not included while computing this metric. We would like to point out that Jain's fairness index [19], computed over the number of blocks served by each node, is consistently close to 1 for all schemes, implying that the schemes are fair "on the whole".

However, as Figure 6 shows, some nodes can still be very unlucky, serving more than 7 times as many blocks as they receive in certain situations. All of these unlucky nodes are in fact high-bandwidth nodes. The pairwise block-level TFT policy eliminates this unfairness by design. Figure 6 bears this out. Also, the QBE heuristic reduces unfairness significantly when the node degree is large enough that block transfers between bandwidth-mismatched nodes can be avoided.

Bandwidth-matching tracker policy: To alleviate the problems resulting from block transfers between bandwidth-mismatched nodes, we investigate a new *bandwidth-matching tracker* policy. The idea here is for the tracker to return to a new node a set of candidate neighbors with similar bandwidth to it. This can be accomplished quite easily in practice by having nodes report their bandwidth to the tracker at the time they join. (We ignore the possibility of nodes gaming the system by lying about their bandwidth.) Having bandwidth-matched neighbors would avoid the problems arising from bandwidth-mismatched pairings.

Care is needed in designing this policy. Having the tracker

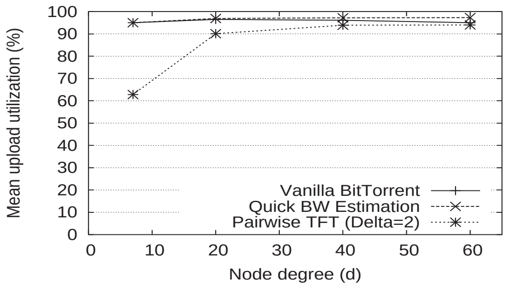


Fig. 7: Upload utilization with the bandwidth-matching tracker policy. Compare Figure 5.

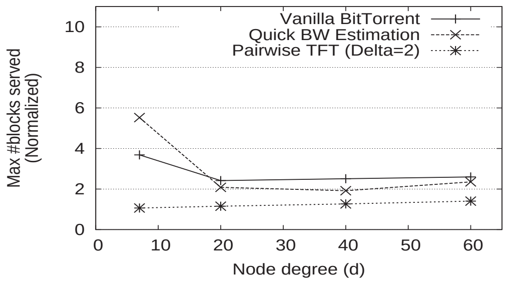


Fig. 8: Number of blocks served (normalized by file-size) with the bandwidth-matching tracker policy. Compare Figure 6.

strictly return only a list of bandwidth-matched peers runs the risk of significantly diminishing the resilience of the peer-to-peer graph, by having only tenuous links between "clouds" of bandwidth-matched nodes. In fact, we have found several instances in our experiments where groups of clients were disconnected from the rest of the network and the disconnection did not heal quickly because the tracker, when queried, would often return a list of peers that are also in the disconnected component.

To avoid this problem, we employ a hybrid policy where the tracker returns a list of peers, 50% of which are bandwidth-matched with the requester and 50% are drawn at random. The former would enable the querying node to find bandwidth-matched neighbors whereas the latter would avoid the disconnection problem.

Figures 7 and 8 show the upload utilization and fairness metrics, respectively, with the (hybrid) bandwidth-matched tracker policy in place. We find a significant improvement in both metrics across a range of values of node degree, as can be seen by comparing Figures 7 and 5, and Figures 8 and 6.

In summary, we find that a bandwidth-unaware tracker combined with the optimistic unchoke mechanism in BitTorrent results in nodes with disparate bandwidths communicating with each other. This results in lower uplink utilization and also creates unfairness in terms of volume of data served by nodes. However, it is possible to obtain a reasonable combination of high upload utilization and good fairness with simple modifications to BitTorrent. Whereas the pairwise block-level TFT policy achieves excellent fairness and good upload utilization, the QBE heuristic achieves excellent upload utilization and good fairness. The hybrid bandwidth-matching tracker policy is critical to both.

VIII. Other Workloads

Thus far we have focused on the performance of BitTorrent in flash-crowd scenarios. While a flash-crowd setting is important, it also has the property that each node is typically in "sync" with its peers in terms of the degree of completion of its download. For instance, all nodes join the flash crowd at approximately the same time and with none of the blocks already downloaded.

However, there are situations, such as the post-flash-crowd phase, where there may be a greater diversity in the degree of completion of the download across the peers. This in turn would result in a divergence in the download goals of the participating nodes — those that are starting out have a wide choice of blocks that they could download whereas nodes that are nearing the completion of their download are looking for specific blocks that they are missing.

Here we consider two extremes of the divergent goals scenario. In the first case, a small number of new nodes join when the bulk of the existing nodes are nearing the completion of their download. This might reflect a situation where new nodes join in the post-flash-crowd phase. In the second case, a small number of nodes that have already completed the bulk of their download at some point in the past rejoin the system during a subsequent flash crowd to complete their download. The majority of their peers in this case would be nodes that have not downloaded much of the file.

A. *Performance of Nodes in the Post-Flash Crowd Phase*

A post flash-crowd scenario is different from a flash-crowd in that there may be a wide range in the fraction of the download completed by each node. Nodes that have been present in the system longer are typically looking for a more specific set of blocks. Thus, it may be harder for a newcomer to establish a TFT exchange with such older nodes, which could lead to increased download times as well as greater load on the seed. Our goal here is to investigate whether this problem actually happens and how severe it is.

We start with a flash crowd of 1000 nodes joining in the first 10 seconds of the experiment. Then, a batch of 10 nodes is introduced into the system at 1800 seconds, when the flash-crowd nodes have finished downloading approximately 80% of the file-blocks. All nodes have down/up bandwidths of 1500/400 Kbps. We use two settings for seed bandwidth: 800 Kbps (low) and 6000 Kbps (high). The seed node utilizes the *smartseed* fix.

Figure 9 plots the number of *interesting* outgoing connections over time, averaged over all newly joined nodes, *until all the flash-crowd nodes leave*. An outgoing connection is deemed interesting if the node in question has some block that its peer needs. Note that the newcomer would be interested in content from almost all its peers during the first several seconds since it does not have any block to start with. Thus, for every interesting connection, the newcomer can establish a TFT exchange with its peer.

Figure 9 shows that a newcomer is quickly able to gather blocks that are interesting to at least a few of its peers, as

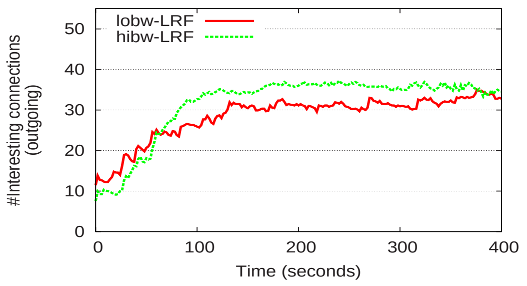


Fig. 9: Number of *interesting* outgoing connections of a randomly sampled post flash-crowd node for (a) low-bandwidth seed, and (b) high-bandwidth seed.

seen from the non-zero count of interesting connections in the figure. The reason that a newcomer is quickly able to establish interesting connections to its peers is as follows: if p is the probability that a downloaded block is interesting to some neighbor, and if this probability is the same and independent for each neighbor, then the probability that a downloaded block is useful to at least one neighbor is $1-(1-p)^d$. This probability increases very quickly with d, even if p is relatively small. Thus, while a large degree, d, may not be necessary for a flash-crowd situation, it is important for ensuring effective TFT exchanges for new nodes that join in the post-flash-crowd phase.

B. *Performance of Pre-seeded Nodes*

We now consider the case where a small number of nodes which have already completed the bulk of their download (i.e., nodes that have been "pre-seeded" with the bulk of the blocks) rejoin the system during a subsequent flash crowd to complete their download. The key question is whether and to what extent such pre-seeded nodes are penalized because they are looking for specific blocks whereas the majority of nodes in the system are interested in most of the blocks (since they have few blocks).

Again, we start with a flash-crowd of 1000 nodes joining in the first 10 seconds. After that, a new node is introduced every 200 seconds into the system. Each new node is pre-seeded with a random selection of k% blocks – this simulates a situation where the node completed k% of its download, disconnected, and then re-joined during a subsequent flash-crowd to finish its download. Ideally, a node that is pre-seeded with k% of the blocks should take approximately $(1-\frac{k}{100})T$ time to download the remaining blocks, where T is the mean time to download the entire file. ($T = 2000$ seconds, for this setting.) However, a pre-seeded node could take longer because the specific blocks that it is looking for may be hard to find, a penalty that we would like to quantify.

Figure 10 plots the ratio of actual download time to the ideal download time for such a "pre-seeded" node that joined 200 seconds into the flash crowd, for different values of k. A ratio close to 1.0 indicates that a pre-seeded node does not have to wait substantially longer than ideal. We use a seed bandwidth of 6000 Kbps in this experiment; thus, the seed has injected at least one copy of each block into the system at approximately

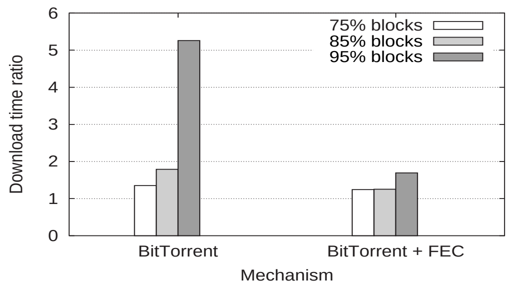


Fig. 10: Download time ratios for a pre-seeded node introduced into the system at 200 seconds into the flash crowd.

135 seconds.

From the bars labeled "BitTorrent" in Figure 10, we see that as the number of blocks required by the pre-seeded node decreases, the likelihood of the node taking longer than ideal to finish increases.[3]

There are two reasons for this behavior: first, each block takes a non-trivial amount of time to spread out from the seed to every node in the system. The maximum possible fanout of this distribution tree is bounded by $u = 5$ (refer Section V-B). Furthermore, the degree d of the pre-seeded node determines how quickly it can "intercept" this distribution tree. The second reason is that a pre-seeded node is looking for specific blocks, and would like these blocks to be replicated quickly. However, BitTorrent's LRF policy dictates that all blocks get replicated equally so that none remains rare. This "resource-sharing" across blocks decreases the distribution rate of the specific blocks desired by the pre-seeded node, resulting in larger download times.

Notice that pre-seeded nodes are delayed basically because they are looking for *specific* blocks. If the source were to employ Forward Error Correction (FEC) [20] [4] and inject a large number of *equivalent* coded blocks into the system, pre-seeded nodes would have a greater choice of blocks to download and hence should be able to reduce the download time penalty. Note that injecting additional blocks does not waste seed bandwidth since every unique block uploaded by the seed is equally useful. We repeated the above experiment with the source introducing 100% additional FEC-encoded blocks. As shown in the bars labeled "BitTorrent+FEC" in Figure 10, the download time ratio with FEC are substantially lower. The download time ratio is close to 1.0 for $k = 75\%$ and 85%, and well under 2.0 even when $k = 95\%$.

Summary

Our experiments with the divergent goals scenarios indicates that BitTorrent tends to "equalize" the performance of newly joined nodes that have fewer or more blocks than the average node. The ones that have fewer blocks are "pulled up" since the LRF mechanism is able to ensure that the new nodes quickly become effective in TFT exchanges. The ones that have a larger number of blocks get "pulled down" (even if the penalty is not much in terms of absolute time) because the LRF policy does not preferentially replicate the specific blocks that such nodes are looking for. A simple application of source-based FEC can significantly reduce the severity of this problem.

[3]Note that this increase is in the *ratio* of the actual to ideal download times, not in the absolute difference between these times.

[4]With FEC, the source blocks are augmented with coded blocks, such that the possession of a threshold number of blocks, whether source or coded, is sufficient to recover the original content.

IX. Summary and Conclusion

In this paper, we have described a series of experiments aimed at analyzing and understanding the performance of BitTorrent in a range of scenarios. We focused our attention on two main metrics: utilization of the upload capacity of nodes, and (un)fairness in terms of the volume of data served by nodes. Our emphasis on fairness stems from the belief that any systematic unfairness in a P2P network is quickly exploited, reducing the effectiveness of the overall network.

Our findings are summarized as follows: *(a)* BitTorrent's rate-based Tit-For-Tat (TFT) policy fails to prevent unfairness across nodes in terms of volume of content served. This unfairness arises principally in heterogeneous settings when high bandwidth peers connect to low bandwidth ones. *(b)* The combination of pairwise block-level TFT (Section VII-B) and the bandwidth matching tracker (Section VII-C) almost eliminates the unfairness of BitTorrent with a modest decrease in utilization. *(c)* It is critical to conserve seed bandwidth, especially when it is scarce; it is important that the seed node serve unique blocks at first (which it alone can do) to ensure diversity in the network, rather than serve duplicate blocks (a function that can be performed equally well by the leechers). *(d)* The Local Rarest First (LRF) policy is critical in eliminating the "last block" problem and ensuring that new leechers quickly have something to offer to other nodes.

Acknowledgments

We thank Phil Chou, Kamal Jain, Pablo Rodriguez, and Aditya Ramamoorthy for participating in discussions and for their insightful suggestions. We thank Ernst Biersack for providing us the Redhat tracker log, and Sharad Agarwal for his comments an earlier draft of this paper. Finally, we thank the anonymous reviewers whose comments improved the paper.

References

[1] "BitTorrent," `http://bittorrent.com`.

[2] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed Content Delivery Across Adaptive Overlay Networks," *SIGCOMM*, Aug. 2002.

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

[4] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips, "A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System," Technical Report PDS-2004-003, Delft University of Technology, The Netherlands, April 2004.

[5] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks," *SIGCOMM*, Sep. 2004.

[6] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data ," *SIGCOMM*, Sep. 1998.

[7] A. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving BitTorrent Performance," Tech. Rep. MSR-TR-2005-03, Microsoft Research, February 2005.

[8] Bram Cohen, "Incentives Build Robustness in BitTorrent," 2003, `http://bittorrent.com/bittorrentecon.pdf`.

[9] C. Gkantsidis and P. Rodriguez, "Network Coding for Large Scale Content Distribution," in *IEEE INFOCOM*, 2005.

[10] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Trans on Info Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[11] R. Sherwood and R. Braud and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol," in *INFOCOM*, Mar 2004.

[12] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," *Proc. ACM Symposium on Operating Systems Principles*, Oct. 2003.

[13] Dejan Kostic and Ryan Braud and Charles Killian and Erik Vandekieft and James W. Anderson, "Maintaining High Bandwidth under Dynamic Network Conditions," in *USENIX Annual Technical Conference*, Apr 2005.

[14] Bharambe, A. and Herley, C. and Padmanabhan, V. N., "Microsoft Research Simulator for the BitTorrent Protocol," `http://www.research.microsoft.com/projects/btsim`.

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

[16] Stefan Saroiu and P. Krishna Gummadi and Steven D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, Jan 2002.

[17] "BitTornado," `http://bittornado.com`.

[18] Jacob Strauss, Dina Katabi, and Frans Kaashoek, "A measurement study of available bandwidth estimation tools," in *IMC*, 2003.

[19] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley and Sons, 1991.

[20] R. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, MA, 1983.