Using Torrent Inflation to Efficiently Serve the Long Tail in Peer-assisted Content Delivery Systems

Niklas Carlsson[1], Derek L. Eager[2], and Anirban Mahanti[3]

[1] University of Calgary, Calgary, Canada
[2] University of Saskatchewan, Saskatoon, Canada
[3] NICTA, Alexandria, Australia
niklas.carlsson@cpsc.ucalgary.ca, eager@cs.usask.ca,
anirban.mahanti@nicta.com.au

Abstract. A peer-assisted content delivery system uses the upload bandwidth of its clients to assist in delivery of popular content. In peer-assisted systems using a BitTorrent-like protocol, a content delivery server seeds the offered files, and active torrents form when multiple clients make closely-spaced requests for the same content. Scalability is achieved in the sense of being able to accommodate arbitrarily high request rates for individual files. Scalability with respect to the number of files, however, may be much more difficult to achieve, owing to a "long tail" of lukewarm or cold files for which the server may need to assume most or all of the delivery cost. This paper first addresses the question of how best to allocate server resources among multiple active torrents. We then propose new content delivery policies that use some of the available upload bandwidth from currently downloading clients to "inflate" torrents for files that would otherwise require substantial server bandwidth. Our performance results show that use of torrent inflation can substantially reduce download times, by more than 50% in some cases.

Key words: Peer-assisted, multi-torrent, torrent inflation, long tail

1 Introduction

Peer-assisted content delivery has the key advantage that the bandwidth available for serving client requests scales with the client population, potentially allowing a system to accommodate increasing file request rates with fixed server resources. This has led to considerable research interest in such systems, as well as increasing numbers of deployments by content providers.[4]

We consider here peer-assisted content delivery systems in which BitTorrent-like protocols are used, with files seeded only by a content provider server, and in which peers contribute upload bandwidth to assist in file delivery only when

[4] The clients of such a system are generically termed "peers" in the following, regardless of whether the context is one concerning peer-to-peer, or peer-server interaction.

downloading a file from the service. In addition to making a conservative assumption about peer participation, such an architecture may provide the content provider with greater content control, and require less functionality at peers, than alternative approaches relying on longer-term peer storage or caching of files.

A potential problem with this type of system, however, arises from the fact that file popularity distributions typically have a power law form, with a "long tail" of lukewarm/cold files that has a substantial aggregate request rate (e.g., [1–3]). These files may not be sufficiently popular to enable torrents consisting of multiple concurrent requesters to form, resulting in a server load that grows unsustainably with the number of offered files.

This paper devises new protocols for such systems, with the goal of enhancing their scalability. First, we propose new server scheduling policies for selecting which peers to upload to next, among multiple peers requesting different files. We find that, in comparison to a baseline "random peer" policy, the new policies improve both the overall average download time and the download times seen by the requesters of the lukewarm/cold files. However, these improvements are quite modest, less than 10% in the scenarios considered.

Second, we propose *torrent inflation*, whereby some of the peer upload bandwidth available from actively downloading peers is applied in torrents for files other than those requested. These other files are ones for which the number of concurrently active requesters may be insufficient to enable self-sustaining torrents. (We refer to a torrent as *fully self-sustained* if all peer requests can be served by the peers themselves, with negligible server assistance.) Note that since we consider systems in which there is no seeding of files by peers, use of torrent inflation entails the overhead of delivering to actively downloading peers some minimal number of blocks from files other than those requested, for those peers to then upload during their respective download sessions to other peers. Torrent inflation policies are designed that seek to harvest only peer upload bandwidth that could otherwise go unused within the peer's own torrent.[5] We find that torrent inflation policies provide significant reduction in both the overall average download time and the download times for the lukewarm/cold files (achieving reductions of more than 50% in some cases). In addition, we find that inflation policies can be designed that are remarkably insensitive to the server capacity and therefore can be used to replace server bandwidth capacity in some systems.

Torrent inflation entails utilizing upload bandwidth of actively downloading peers for improving a system-wide performance objective. The content provider can build the desired policies into software required for accessing the service. It may also be possible to devise incentives to directly reward peers for the desired behaviour. Investigation of these options is outside the scope of this work.

The remainder of the paper is organized as follows. Related work is discussed in Section 2. Section 3 describes our system assumptions, simulation model, and baseline policies, and presents simulation results for the baseline "random peer"

[5] As in the peer-to-peer context [3–5], while peers in popular torrents, with high piece diversity, typically are able to fully utilize their upload bandwidth, this is not necessarily the case in lukewarm/cold torrents.

and "random file" server scheduling policies. Alternative server scheduling policies are defined and their performance is compared with baseline policies in Section 4. Section 5 proposes torrent inflation policies that seek to harvest only peer upload bandwidth that could otherwise go unused, and presents performance results for these policies. Conclusions are given in Section 6.

2 Related Work

To the best of our knowledge, prior work has not directly addressed the problem of efficient delivery of potentially large numbers (or the so called "long tail") of lukewarm/cold files, in the context of peer-assisted download systems that do not rely on longer-term peer storage or caching of files. There has, however, been related work on multi-torrent systems in which peers seed files and some peers concurrently participate in distributing more than one file (e.g., [6–10]). Of course, any policy presented here would further benefit from additional seed capacity available to the system.

2.1 Multi-torrent Systems

Guo *et al.* [6] use measurements and analytical models to illustrate that torrent lifetimes can be extended if a node that acts as a downloader in one torrent also acts as a seed in another torrent. Yang *et al.* [7] propose a cross-torrent tit-for-tat scheme in which unchoking is done based on the aggregate download rate from each candidate peer across all torrents (rather than just within a single torrent). Other prior work has proposed that inter-swarm exchanges be incentivized through propagation of peer reputation [11], incentive-based token schemes [12], and/or history-based priority rules [13]. These works are complimentary to ours, and could perhaps be used to incentivize peer participation in torrent inflation.

To increase torrent lifetimes and file availabilities, Menasche *et al.* [8] propose that the original seeder (or content provider) "bundle" a number of its files for distribution, rather than offering the files individually. While static bundling is shown to be effective for the purpose of improved file availability, it can easily result in increased download times, especially of otherwise popular content. In contrast, we consider a peer-assisted context (in which file availability is not an issue since files are not seeded by peers, but by a content delivery server) and focus on reducing download times. Other related work has investigated the advantages offered by dynamically inflating swarm sizes by merging small swarms of the same torrent (rather than different torrents) [3].

As in our work, Wu *et al.* [9,10] propose techniques wherein peers contribute to the distribution of content they may not be interested in. The assumed context is live streaming, however, rather than our context of content download, and therefore the proposed techniques are quite different than those considered here.

2.2 Server Scheduling

The importance of server scheduling policies has been discussed in the context of both peer-assisted download [14] and live streaming systems [15]. However, to the best of our knowledge, the only previous work on server scheduling in multi-torrent file download systems is that by Sun *et al.* [16,17]. They use an analytic steady-state fluid-style model to compare the performance of equal server bandwidth allocation among all peers, and equal server bandwidth allocation among all files, with an analytic lower bound [16]. A version of the second of these policies has been implemented in FS2You [17]. We note that these two policies are very similar to the baseline server scheduling policies discussed in Section 3.

In addition, there are some similarities between the problems of allocating server bandwidth among the requesters of multiple files in a peer-assisted system, and server scheduling in a batched service content delivery system [18, 19]. In both types of systems, multiple clients can be served using a single operation by the server. In a peer-assisted system, peers are able to replicate among themselves any piece injected into a torrent by the server. In a batched service system, multiple clients can be served at once using broadcast or multicast. On the other hand, there are also obvious significant differences; for example, at high request rates, torrents can become largely self-sustaining and require minimal allocation of server bandwidth.

3 Baseline Performance

3.1 Baseline Server Scheduling Policies

Our baseline *random peer* and *random file* policies closely correspond to two policies modeled by Sun *et al.* [16], and are motivated by the goals of allocating server bandwidth fairly among active peers and active torrents, respectively.

With random peer, a peer is randomly selected from the set of all peers that the server is not currently uploading to. With random file, the server chooses a file randomly from the group of files that: (1) the server is not currently uploading data from, and (2) has at least one peer with an outstanding piece request. If there are no such files, a file is selected randomly among the files for which there is a piece request from a peer that the server is not currently uploading to, if any. Finally, within the set of peers that are downloading the selected file but that are not currently being uploaded to by the server, a peer is selected at random.[6]

3.2 Multi-Torrent Simulation Model

For policy evaluation, an existing event-based simulator of BitTorrent-like systems [20] was modified to support simulation of systems with multiple files. A

[6] A version of random file is implemented in FS2You [17]. In their implementation, the server maintains a file popularity index for each file (based on the number of references for the file within some time window), and serves peers with a probability which is inversely proportional to the requested file's popularity index.

single server is assumed that stores all N files, and varying numbers of active clients (peers). For the results presented here, it was assumed that peers have connections to all other peers in the same torrent, as well as to the server.[7]

As with BitTorrent, a *rarest-first* piece selection policy is used with *tit-for-tat* peer selection and optimistic unchoke. It is assumed that each peer concurrently uploads to at most six peers, with one of the peer's six upload connections, on average, used for optimistic unchoke. The maximum number of server upload connections is chosen as the total server bandwidth divided by the peer upload rate (an integer value owing to the parameters chosen in our experiments).

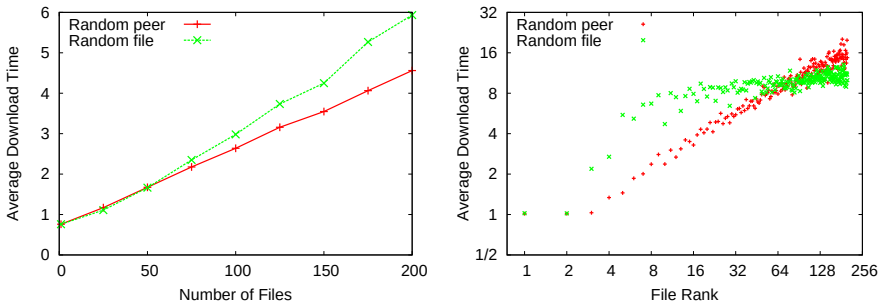
3.3 Performance Comparisons of Baseline Policies

The simulation results that we present in this paper are for a relatively simple workload scenario. Peers arrive according to a constant-rate Poisson process, and request files according to a Zipf popularity distribution (i.e., the request rate λ_i of the i^{th} most popular file is proportional to $\frac{1}{i^\alpha}$). Zipf and Zipf-like popularity distributions have been observed in many contexts such as Web servers, media-on-demand servers, and video sharing services (often, however, with deviations at the head and/or tail) [1–3]. Each peer requests a single file upon its arrival to the system, and leaves the system once its download is complete. Each simulation run is for 40,000 requests, with the initial 10,000 and the last 2,000 requests removed from the measurements.

For simplicity, peers are assumed to be homogenous, and motivated by the observed asymmetry in upload-download bandwidth (e.g., [21]), it is assumed that peers have a download-upload bandwidth capacity ratio (D/U) equal to 3. Files are assumed to be of identical size with 256 pieces each. Without loss of generality, the file size L and upload bandwidth capacity U are fixed at one. With these normalized units, the volume of data transferred is measured in units of the file size and the download time is measured in units of the minimum time it takes for a peer to fully upload a file.

Other parameters, together with their default values, are as follows: server bandwidth $B = 10$, hottest file request rate $\lambda_1 = 40$, and Zipf parameter $\alpha = 1$. For a system with $N = 200$ files, these choices imply a total request rate of roughly 235 (i.e., on average about 235 requests arrive during the time it takes a peer to upload a file at its upload capacity rate), and that the server can upload at ten times the rate of a peer, but at most can satisfy less than 5% of the total

[7] Note that the default parameters in recent versions of the mainline BitTorrent client allow peers to be connected to up to 80 other peers, which is often achieved in practice [5], and that typically many fewer peers than this are participating in cold/lukewarm torrents. Furthermore, peers not satisfied with their performance are able to request additional peers from the tracker, or directly from other peers using the peer exchange protocol (PEX). For simulating the transmission rates of piece transfers, it is assumed that connection bottlenecks are located at the end points (i.e., the transmission rate is limited either by the upload bandwidth at the sender or by the download rate at the receiver) and the network operates using max-min fair bandwidth sharing (using TCP, for example).



(a) Average download time (b) Per-file download time

Fig. 1. Baseline policies ($B = 10$, $\lambda_i = 40/i$, $L = 1$, $U = 1$, $D/U = 3$, $K = 256$, $N = 200$)

demand without peer assistance. For $N = 1$ and $N = 25$, the server can satisfy at most 25% and 7% of the total demand, respectively.

Figure 1 shows performance with the baseline server scheduling policies. Figure 1(a) shows the average download time for systems with varying numbers of files, while Figures 1(b) shows the average download time for each file individually (ordered accorded to the file popularity rank), for $N = 200$ files.

Note that for both policies, the overall average download time increases approximately linearly with increasing numbers of offered files. From Figure 1(b), it can be seen that the high overall average download time in systems with large numbers of files is caused by high download times for relatively unpopular files. The most popular files, in contrast, have largely self-sustaining torrents for these parameter values, and thus the download times for these files are only minimally affected when the number of offered files increases.

Although the random peer policy achieves a better overall average download time than the random file policy, for these parameter values, it is less fair in the sense that it yields greater differences in download times between files. For example, the average download times for the least popular files are about 1.5 to 2 times as big with the random peer policy than with random file.

4 Prioritized Server Scheduling

This section develops prioritized server scheduling policies that use additional state information to make their scheduling decisions. Our primary goal in this section is to find policies that achieve an overall average download delay similar to, or better than, that achieved with random peer, and fairness across files similar to or better than that achieved with random file.

Towards this goal, we investigate policies similar to ones previously developed for scheduling in batching systems [18, 19]. Similar to the random file policy, we use a probabilistic two-step approach in which a file is first selected at random (but now with weighted probabilities), and then a peer is selected at random

(with uniform probability) from within the set of peers requesting pieces for this file. For file selection we consider three different weighting functions.

While we only present results for probabilistic policies, we have also evaluated several deterministic policies including deterministic versions of the policies defined in this section. However, these deterministic policies were significantly outperformed by their probabilistic counterparts (in most cases).

The first policy, called *weight by number of excess-wait peers* (NEWP), weights a file according to the number of requesting peers whose time-in-system exceeds that for a self-sustaining torrent. This time-in-system threshold is equal to the expected download time if each peer is downloading at its own upload rate (i.e., L/U). In contrast, the second policy, called *weight by maximum excess wait* (EW), weights files not by the *number* of "excess-wait" requesting peers, but according to the *longest* excess wait time (i.e., time-in-system beyond the expected download time for a self-sustaining torrent), among such peers, with weight zero given if there is no such peer.[8]

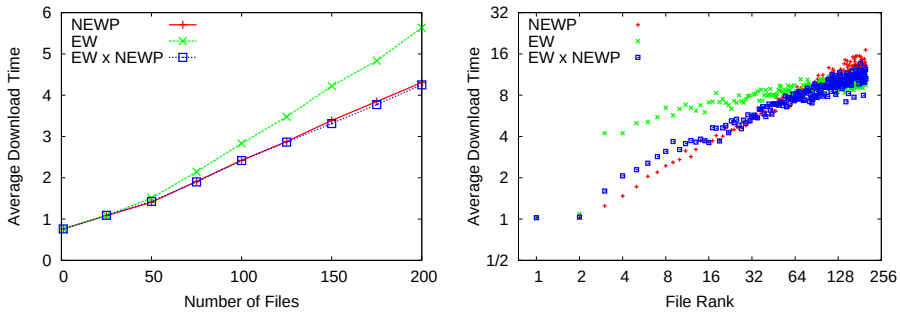
Finally, we define a policy called *weight by product of the maximum excess wait and number of excess-wait peers* (EW x NEWP) that uses the product of the two preceding metrics to determine its weights. Similar to RxW [19], this policy attempts to strike a balance between considerations of time in system, and number of requesters, in determining its weights. In contrast to RxW, however, this policy is probabilistic rather than deterministic, and considers only requesting peers whose time-in-system exceeds that for a self-sustaining torrent.[9]

Figure 2 shows results for the above prioritized server scheduling policies, using the same workload scenarios as in Figure 1. Compared to the other prioritized policies as well as to the baseline policies, EW x NEWP achieves the best overall average download time. Also, EW x NEWP achieves similar fairness with respect to the per-file download times as that achieved by random file. (This is exemplified by the similarity in average download times of the least popular files, for the two policies.) Therefore, EW x NEWP achieves our objective with respect to improving on the baseline server scheduling policies.

Figure 3 shows the average server bandwidth usage for each file individually, for $N = 200$ files. Note from Figure 3(a) that the random peer policy wastes significant server resources on the two most popular files, relative to random file; random file achieves the same performance for these files (as shown in Figure 1(b)) with considerably lower server bandwidth usage. Comparing Figure 3(a) and Figure 3(b), it can be seen that the prioritized server scheduling policies further substantially reduce the server bandwidth used for the most popular files. This bandwidth can be allocated to torrents in greater need of server assistance. Of course, dynamic server bandwidth allocation is able to make sub-

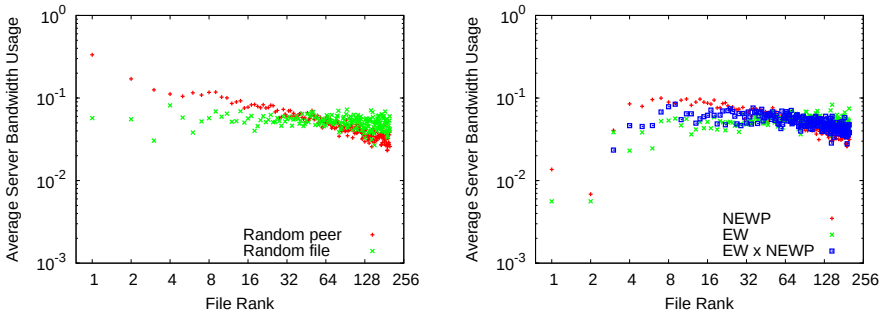
[8] Denoting the set of peers currently downloading file i by P_i, and the time-in-system of peer p by w_p, the weight given to file i is $|\{p \in P_i | w_p > L/U\}|$ using NEWP and $\max[\max_{p \in P_i}[w_p] - L/U, 0]$ using EW.

[9] Other policies attempting to achieve a compromise of similar form were considered, such as a policy that used the total number of requesting peers in place of the NEWP component, but did not perform as well as this particular version.



(a) Average download time (b) Per-file download time

Fig. 2. Prioritized server scheduling policies ($B = 10$, $\lambda_i = 40/i$, $L = 1$, $U = 1$, $D/U = 3$, $K = 256$, $N = 200$)



(a) Baseline policies (b) Prioritized policies

Fig. 3. Server bandwidth usage ($B = 10$, $\lambda_i = 40/i$, $L = 1$, $U = 1$, $D/U = 3$, $K = 256$, $N = 200$)

stantially better use of the server resources than policies in which server bandwidth is statically allocated among files. In this paper we focus on the relative performance of dynamic allocation policies, and do not quantify the performance differences relative to static bandwidth allocation.

5 Torrent Inflation

In the results shown in Figures 1 and 2, requests for the lukewarm/cold files suffer relatively high download times, even though requests for such files receive a greater average per-request share of the server bandwidth than the hot files. This is because self-sustaining torrents are not formed for these files, and the upload bandwidth of the requesting peers often can not be utilized. In this section, new *torrent inflation* policies are described that harvest peer upload bandwidth to *inflate* the number of active peers in a torrent, thus making the torrent more self-sustaining. Torrent inflation is achieved by assigning each active peer an "inflation file". Some number of pieces from the inflation file are downloaded

in parallel with the requested file. These pieces may then be uploaded to other peers, providing torrent inflation. Torrent inflation requires a policy for assigning inflation files, as well as a policy for peers to choose among piece uploads from their requested file, versus from their inflation file, and the peer to upload to.

5.1 Inflation File Selection Policies

This section defines three alternative inflation file selection policies. As with server scheduling policies, probabilistic policies were found to outperform deterministic policies, and therefore, only probabilistic policies are considered in this section. With each of these policies, the decision of which inflation file is assigned to each peer is made by the server at the time of the peer's original file request. The information needed to make these decisions would be available to any BitTorrent-like system with tracker functionality.

The simplest policy considered here, *random active torrent* (AT), randomly (with uniform probability) selects one of the files with an "active" torrent (i.e., such that there is at least one currently active peer for which the file is either the peer's requested file, or inflation file, and the peer has acquired at least one piece). Motivated by the success of the server scheduling policy EW x NEWP, the same policy is also considered for inflation file selection. Finally, we define a policy called *conditional weight by number of peers* (CNP) that makes a weighted random selection among those files with at least one requesting peer whose time-in-system exceeds that for a self-sustaining torrent. The weights are assigned according to the current number of active requesters. In our simulations of the EW x NEWP and CNP selection policies, we default to the AT policy if there are no peers whose time-in-system exceeds that for a self-sustaining torrent.

Finally, with torrent inflation, we increase the time-in-system threshold value (previously chosen as L/U) that is used when identifying a torrent as non-self-sustaining, as needed in server scheduling and the EW x NEWP and CNP inflation file selection policies, by a factor f (chosen as 1.2 in all our experiments). This change is motivated by the potential benefit of being somewhat more cautious when identifying a torrent as non-self-sustaining, when using torrent inflation.[10]

5.2 Upload Prioritization

We now describe the policy used to choose among the various options that may exist when a peer wishes to initiate a new upload operation. In the upload policy proposed here, highest priority is given to uploads in which the piece is from the common requested file of both uploader and downloader. Uploads of pieces from the requested file of the downloader, when this file is the inflation file of the uploading peer, are given the next highest priority. These two rules ensure that

[10] The particular choice of 1.2 has been found to provide a reasonable tradeoff between responsiveness to high download delays, and minimizing provision of service to torrents that are largely self-sustaining. We note that this choice roughly corresponds to peers being able to use one out of six upload connections to help other torrents.

Table 1. Upload priority levels with inflation.

	File type at downloader		
Uploader file	Requested	Inflation (case 1*)	Inflation (case 2)
Requested	1^{st}	3^{rd}	5^{th}
Inflation	2^{nd}	4^{th}	6^{th}

* Case 1 holds if at least one of the downloader's six upload connections is not currently in use, even though the peer has at least one downloaded piece; otherwise, case 2 holds. Note that in case 1 the downloader has upload capacity that may otherwise go unused.

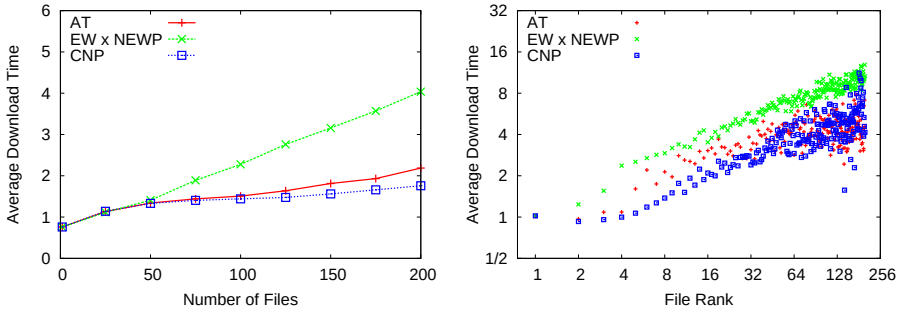
the peer upload bandwidth is used as much as possible for uploads of pieces wanted by the downloader.

Two additional rules govern selection of upload targets in the event the data uploaded is for the inflation file of the recipient. Such uploads are not directly productive, and are useful only to the extent that the downloader can later pass the data to a peer that needs it. Our first rule is to favor uploads of this type only to peers that are not fully utilizing their upload connections, as this suggests that such peers are unable to fully utilize their upload bandwidth in their respective requested file torrent. (As noted earlier and discussed elsewhere in the peer-to-peer context [3,4], the upload bandwidth of peers in lukewarm/cold torrents may not be utilized as effectively as that of peers in larger torrents.) Our second rule is used as a tiebreaker for the first rule, and is to favor uploads of pieces of a file that at least one of the two peers has requested (in this case the uploader).

The above prioritizations result in the upload policy with the priority levels enumerated as in Table 1. Comparisons and preliminary experiments using alternative priority rules suggest that these priority levels work well together.

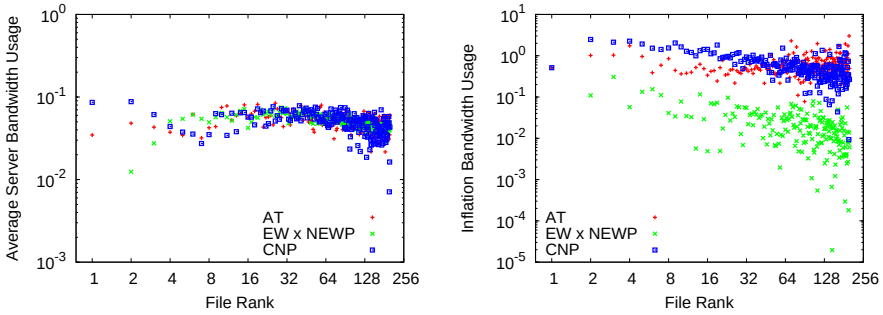
5.3 Principal Performance Comparisons

Figure 4 shows results for the inflation policies used in combination with the EW x NEWP server scheduling policy. We find that both AT and CNP, when used in conjunction with the peer upload policy described in Section 5.2, are able to provide major improvements in both the overall average download time and the download times for cold files (achieving reductions of more than 50% in some cases). While both the average delays and fairness across files with the EW x NEWP inflation file selection policy are slightly better than without inflation (Figures 1 and 2), they are not as good as with AT and CNP. This is likely due to the adverse effects of strongly correlating the choice of inflation file with the choice of which file the server will deliver data from next. For example, such correlations may result in undesirable scenarios where the peers that have requested a particular file complete their downloads fairly soon after that file becomes highly ranked as an inflation file choice, causing ineffective torrent inflation. This is seen in the results of Figure 5(b), which shows the average rate at which data from each file is uploaded by the peers for which that file was their inflation file (i.e., the "inflation bandwidth usage" for that file). Note that with EW x NEWP, these bandwidth usage values are considerably lower than with the other policies.



(a) Average download time (b) Per-file download time

Fig. 4. Inflation policies ($B = 10$, $\lambda_i = 40/i$, $L = 1$, $U = 1$, $D/U = 3$, $K = 256$, $N = 200$)

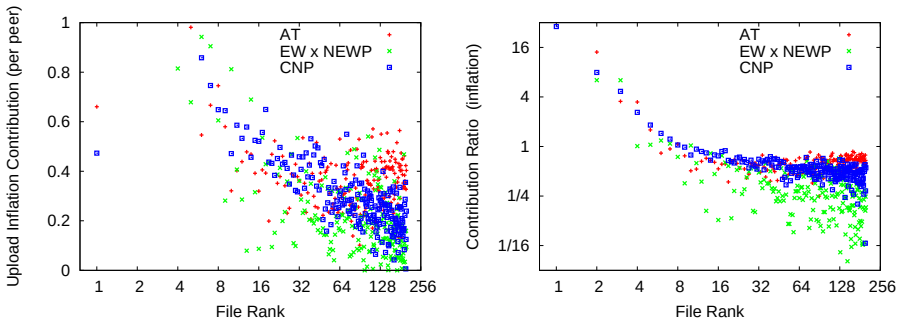


(a) Server bandwidth usage (b) Inflation bandwidth usage

Fig. 5. Bandwidth usage for inflation ($N = 200$, $B = 10$, $\lambda_i = 40/i$, $L = 1$, $U = 1$, $D/U = 3$, $K = 256$)

Figure 6(a) shows, for each file, the average amount of data uploaded by each peer selecting that file as its inflation file. Not surprisingly, the average amount uploaded is generally higher for more popular files. This is since more popular files are likely to have a greater number of concurrently active peers that have requested the file, and that can be uploaded to. Note that the case of the most popular file is an exception. This file is likely sufficiently hot that pieces are only rarely uploaded to peers with that file as their inflation file, according to our upload policy, and therefore such peers only rarely obtain any inflation file data to upload to others. Overall, the amount of peer upload bandwidth used for inflation file uploads is relatively modest, and yet, as seen, when judiciously applied can yield substantial benefits with respect to download times and fairness.

Figure 6(b) shows, for each file, the ratio of the average amount of data from the file that is uploaded by each peer selecting that file as its inflation file, to the average amount that is downloaded by such a peer. Higher ratios indicate greater efficiency of torrent inflation. In particular, ratios greater than one indicate that each downloaded piece tends to be uploaded to multiple other peers. Note that



(a) Inflation uploads (b) Upload/download ratio

Fig. 6. Contribution statistics for inflation ($N = 200$, $B = 10$, $\lambda_i = 40/i$, $L = 1$, $U = 1$, $D/U = 3$, $K = 256$)

the cases in Figure 6(b) where the ratio is substantially less than 1, correspond to cases where not much data is uploaded (as seen from Figure 6(a)).

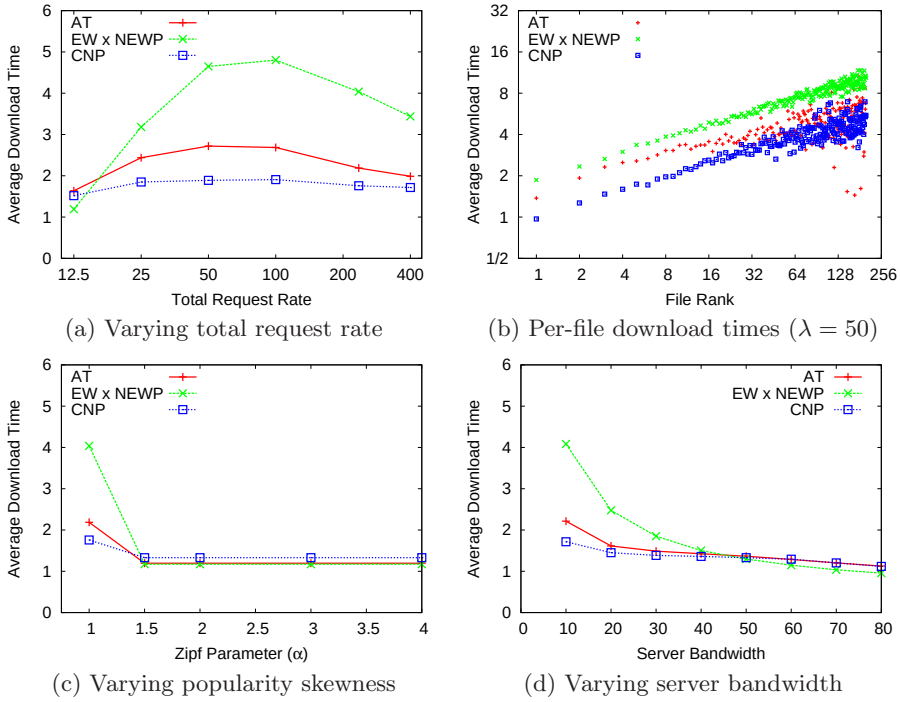
5.4 Impact of Workload and Server Parameters

Figure 7(a) shows results for different values of the total request arrival rate λ. As in the other figures, the parameters not being varied are set to their default values. Figure 7(b) shows per-file results for a lower total request arrival rate ($\lambda = 50$) than used in previous experiments.

As seen in Figure 7(a), the average download time is largest for intermediate request rates. With lower request rates, the server is more able to serve requests itself, without peer assistance. For example, with an arrival rate $\lambda = 12.5$ the server has the capacity to satisfy as much as 80% of the demand itself. With higher request rates, for which the server can only satisfy a small fraction of the demand without peer assistance (e.g., at most 2.5% when $\lambda = 400$), self-sustaining torrents develop for a larger fraction of the files. As observed by comparing Figures 7(b) and 4(b), with higher request rates, a larger fraction of the files are able to develop self-sustaining torrents.

Figures 7(c) and 7(d) show results for different values of the Zipf parameter α, and server bandwidth B, respectively. Figure 7(c) shows that the average download time quickly decreases towards that with self-sustaining torrents, as α increases beyond 1. This is since greater skewness in the popularity distribution results in a larger fraction of the requests being directed towards a few highly popular files. Figure 7(d) shows that a system using torrent inflation with the CNP policy is remarkably insensitive to server capacity. Evidently, torrent inflation in such a system can effectively replace server bandwidth capacity.

In summary, Figures 4 through 7 show that the CNP inflation file selection policy typically outperforms the AT and EW-NEWP policies with respect to the average download time. However, the AT policy yields quite competitive performance, and as seen in Figure 4(b) may yield somewhat lower variability among the download times for popular versus unpopular files.



(a) Varying total request rate

(b) Per-file download times ($\lambda = 50$)

(c) Varying popularity skewness

(d) Varying server bandwidth

Fig. 7. Impact of Workload and Server Parameters (Default scenario: $B = 10$, $N = 200$, $\lambda_i = 40/i$ ($\lambda \approx 235$), $L = 1$, $U = 1$, $D/U = 3$, $K = 256$)

6 Conclusions

This paper has considered peer-assisted delivery using BitTorrent-like protocols, with files seeded only by a content provider server. A fundamental problem in such systems is that of how to efficiently support delivery of the "long tail" of lukewarm/cold files for which there are insufficient concurrent requesters for active torrents to form. To address this problem, "torrent inflation" was proposed, in which some of the available upload bandwidth from currently downloading peers is used to "inflate" torrents for other files, making them more self-sustaining. Current work is investigating analytic modelling approaches that would allow us to evaluate larger systems than we can evaluate with simulation.

7 Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada, the Informatics Circle of Research Excellence (iCORE) in the Province of Alberta, and the National Information and Communications Technology Australia (NICTA).

References

1. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and Zipf-like distributions: Evidence and implications. In: Proc. IEEE INFOCOM, New York, NY (Mar. 1999)
2. Cha, M., Kwak, H., Rodriguez, P., Ahn, Y., Moon, S.: I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In: Proc. ACM IMC, San Deigo, CA (Oct. 2007)
3. Dan, G., Carlsson, N.: Dynamic swarm management for improved BitTorrent performance. In: Proc. IPTPS, Boston, MA (Apr. 2009)
4. Yang, X., de Veciana, G.: Service capacity of peer-to-peer networks. In: Proc. IEEE INFOCOM, Hong Kong, China (Mar. 2004)
5. Legout, A., Urvoy-Keller, G., Michiardi, P.: Rarest first and choke algorithms are enough. In: Proc. ACM IMC, Rio de Janeiro, Brazil (Oct. 2006)
6. Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X., Zhang, X.: Measurement, analysis, and modeling of BitTorrent-like systems. In: Proc. ACM IMC, Berkley, CA (2005)
7. Yang, Y., Chow, A.L.H., Golubchik, L.: Multi-torrent: A performance study. In: Proc. MASCOTS, Baltimore, MD (Sept. 2008)
8. Menasche, D.S., Rocha, A.A.A., Li, B., Towsley, D., Venkataramani, A.: Content availability and bundling in swarming systems. In: ACM CoNEXT, Rome, Italy (Dec. 2009)
9. Wu, D., Liu, Y., Ross, K.W.: Queuing network models for multi-channel p2p live streaming systems. In: Proc. IEEE INFOCOM, Rio de Janeiro, Brazil (Apr. 2009)
10. Wu, D., Liang, C., Liu, Y., Ross, K.W.: View-upload decoupling: A redesign of multi-channel p2p video systems. In: Proc. IEEE INFOCOM Mini-conference, Rio de Janeiro, Brazil (Apr. 2009)
11. Piatek, M., Isdal, T., Krishnamurthy, A., Anderson, T.: One hop reputations for peer to peer file sharing workloads. In: Proc. NSDI, San Francisco, CA (2008)
12. Ramachandran, A., das Sarma, A., Feamster, N.: Bitstore: An incentive compatible solution for blocked downloads in BitTorrent. In: Proc. NetEcon, San Diego, CA (2007)
13. Carlsson, N., Eager, D.L.: Modeling priority-based incentive policies for peer-assisted content delivery systems. In: Proc. IFIP Networking, Singapore (2008)
14. Das, S., Tewari, S., Kleinrock, L.: The case for servers in a peer-to-peer world. In: Proc. IEEE ICC, Istanbul, Turkey (June 2006)
15. Wu, C., Li, B., Zhao, S.: Multi-channel live p2p streaming: Refocusing on servers. In: Proc. IEEE INFOCOM, Phoenix, AZ (Apr. 2008)
16. Sun, Y., Liu, F., Li, B., Li, B.: Peer-assisted online storage and distribution: Modeling and server strategies. In: Proc. NOSSDAV, Williamsburg, VA (2009)
17. Sun, Y., Liu, F., Li, B., Li, B., Zhang, X.: Fs2you: Peer-assisted semi-persistent online storage at a large scale. In: Proc. IEEE INFOCOM, Rio de Janeiro, Brazil (Apr. 2009)
18. Aggarwal, C.C., Wolf, J.L., Yu, P.S.: On optimal batching policies for video-on-demand storage servers. In: Proc. IEEE ICMCS, Hiroshima, Japan (June 1996)
19. Aksoy, D., Franklin, M.: RxW: A scheduling approach for large-scale on-demand data broadcast. IEEE/ACM Trans. on Networking **6**(7) (Dec. 1999) 846–860
20. Carlsson, N., Eager, D.L.: Peer-assisted on-demand streaming of stored media using BitTorrent-like protocols. In: Proc. IFIP Networking, Atlanta, GA (2007)
21. Saroiu, S., Gummadi, K.P., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proc. MMCN, San Jose, CA (Jan. 2002)