Dynamic Swarm Management for Improved BitTorrent Performance [*]

György Dán
ACCESS Linnaeus Centre
KTH, Royal Institute of Technology
Stockholm, Sweden
gyuri@ee.kth.se

Niklas Carlsson
University of Calgary
Calgary, Canada
niklas.carlsson@cpsc.ucalgary.ca

Abstract

BitTorrent is a very scalable file sharing protocol that utilizes the upload bandwidth of peers to offload the original content source. With BitTorrent, each file is split into many small pieces, each of which may be downloaded from different peers. While BitTorrent allows peers to effectively share pieces in systems with sufficient participating peers, the performance can degrade if participation decreases. Using measurements of over 700 trackers, which collectively maintain state information of a combined total of 2.8 million unique torrents, we identify many torrents for which the system performance can be significantly improved by re-allocating peers among the trackers. We propose a light-weight distributed swarm management algorithm that manages the peer torrents while ensuring load fairness among the trackers. The algorithm achieves much of its performance improvements by identifying and merging small swarms, for which the performance is more sensitive to fluctuations in the peer participation, and allows load sharing for large torrents.

1 Introduction

BitTorrent is a popular peer-to-peer file-sharing protocol that has been shown to scale well to very large peer populations [9]. With BitTorrent, content (e.g., a set of files) is split into many small pieces, each of which may be downloaded from different peers. The content and the set of peers distributing it is usually called a *torrent*. A peer that only uploads content is called a *seed*, while a peer that uploads and downloads at the same time is called a *leecher*. The connected set of peers participating in the piece exchanges of a torrent is referred to as a *swarm*.

A client that wants to download a file can learn about other peers that share the same content by contacting a *tracker* at its *announce URL*. Each tracker maintains a list with state information of known peers that currently are downloading and/or uploading pieces of the file, and

provides a peer with a subset of the known peers upon request. Upon requesting the list of peers, the peer has to provide the tracker with information about its download progress. Additionally, the peers must inform the tracker about changes in their status (i.e., when they join, leave, or finish downloading the file). To avoid overloading trackers, the BitTorrent protocol only allows a peer to associate with one tracker per file that it is downloading (unless the tracker is no longer available and a new tracker must be contacted). Naturally, torrents may therefore have multiple parallel swarms.

While BitTorrent allows peers to effectively share pieces of popular torrents with many peers in a swarm, the performance of small torrents and swarms is sensitive to fluctuations in peer participation. Measurement data suggests that peers in small swarms achieve lower throughput on average (e.g., Figure 9 in [9]). Most swarms are unfortunately small; several sources confirm that the popularity distribution of p2p content follows a power-law form, with a "long tail" of moderately popular files (see Figure 1(a) and, e.g., [1, 3, 4, 5]). At the same time, the measurement data we present in this paper shows that many torrents consist of several swarms (see Figure 1(b)). Potentially, if one could dynamically re-allocate peers among the trackers such that multiple small swarms of a torrent are merged into a single swarm, then one could improve the file sharing performance of the peers belonging to these torrents.

Motivated by these observations, the goal of our work is to evaluate the feasibility and the potential gains of dynamic swarm management for BitTorrent trackers. To support our evaluation, we performed measurements of 721 trackers, which collectively maintain state information of a combined total of 2.8 million unique torrents. We propose a light-weight distributed swarm management algorithm, called *DISM*, that also ensures load fairness among the trackers. Based on our measurement data and using *DISM* we argue that dynamic swarm balancing could lead to a significant performance improvement in

terms of peer throughput. We also briefly discuss alternatives for swarm management that could lead to similar performance improvements.

The remainder of the paper is organized as follows. Section 2 describes our design objectives and a lightweight distributed swarm management algorithm. Section 3 presents validation and performance results. Related work is discussed in Section 4. Finally, Section 5 concludes the paper.

2 Distributed Swarm Management

Ignoring the seed-to-leecher ratio, small swarms typically perform worse than larger swarms. However, for load sharing and reliability purposes it may be advantageous to split the responsibility of maintaining per-peer state information across multiple trackers, i.e., to allow several swarms to coexist.

Consequently, dynamic swarm management should make it possible to (i) merge swarms belonging to a torrent if they become too "small", and to (ii) split a swarm or to re-balance swarms if the swarms become sufficiently "large". In general, it is hard to define when a swarm should be considered "small" or "large", but for simplicity, we assume that a swarm can be considered "large" if it has at least $\tilde{x}$ participating peers, for some threshold $\tilde{x}$. "Large" swarms are likely to have high piece diversity and are more resilient to fluctuations in peer participation.

Apart from these two properties, one would like to minimize the effect of swarm balancing on the traffic load of the trackers by (iii) avoiding a significant increase in the number of peers for any individual tracker (load conservation), and by (iv) minimizing the number of peers that are shifted from one tracker to another (minimum overhead, especially shifts associated with torrents being re-balanced).

The distributed swarm management (DISM) algorithm we describe in the following was designed with these four properties in mind. Our algorithm allows swarms to be merged and to be split: swarms are merged whenever a swarm has less than $\tilde{x}$ participating peers, and a swarm is split (or re-balanced) over multiple trackers only if splitting the peers does not cause any swarm to drop below $\tilde{x}$ peers. The algorithm ensures that no tracker will see an increase of more than $\tilde{x}$ peers in total (across all torrents) and typically much less, and it does not balance swarms that have at least $\tilde{x}$ peers each.

The DISM algorithm is composed of two algorithms. It relies on a distributed negotiation algorithm to determine the order in which trackers should perform pairwise load balancing. On a pairwise basis, trackers then exchange information about the number of active peers associated with each torrent they have in common (e.g.,

Table 1: Notation

Parameter	Definition
$\mathcal{R}$	Set of trackers
$\mathcal{T}$	Set of torrents
$\mathcal{R}(t)$	Set of trackers that track torrent t
$\mathcal{T}(r)$	Set of torrents tracked by tracker r
M_r	Number of peers tracked by tracker r
x_t	Number of peers in torrent t
$x_{t,r}$	Number of peers of torrent t that are tracked by tracker r
$\tilde{x}$	Threshold parameter

by performing a scrape), and determine which tracker should be responsible for which peers.

2.1 System Model

Table 1 defines our notation. In the following, we consider a system with a set of trackers $\mathcal{R}$, with a combined set of torrents $\mathcal{T}$. We will denote by $\mathcal{R}(t)$ the set of trackers that track torrent $t \in \mathcal{T}$, and by $\mathcal{T}(r)$ the set of torrents that are tracked by tracker $r \in \mathcal{R}$. Every torrent is tracked by at least one tracker (i.e., $\mathcal{T} = \bigcup_{r \in \mathcal{R}} \mathcal{T}(r)$), but the subsets $\mathcal{T}(r)$ are not necessarily pairwise disjoint. Finally, let us denote the number of peers tracked by tracker r for torrent t by $x_{t,r}$, the total number of peers tracked by tracker r by M_r, and the total number of peers associated with torrent t by x_t.

2.2 Distributed Negotiation

The distributed negotiation algorithm assumes that each tracker r knows the set of torrents $\mathcal{T}(r,r') = \mathcal{T}(r) \cap \mathcal{T}(r')$ that r has in common with each other tracker $r' \in \mathcal{R}$ for which the trackers' torrents are not disjoint (i.e., for which $\mathcal{T}(r,r') \neq 0$). Note that this information should be available through the torrent file, which should have been uploaded to the tracker when the torrent was registered with the tracker.

The algorithm works as follows. Tracker r invites for pairwise balancing the trackers r' for which the overlap in tracked torrents, $\mathcal{T}(r,r')$ is maximal among the trackers with which it has not yet performed the pairwise balancing. A tracker r' accepts the invitation if its overlap with tracker r is maximal. Otherwise, tracker r' asks tracker r to wait until their overlap becomes maximal for r' as well. The distributed algorithm guarantees that all pairs of trackers with an overlap in torrents will perform a pairwise balancing once and only once during the execution of the algorithm. If tracker r' accepts the invitation, tracker r queries $x_{t,r'}$ for $t \in \mathcal{T}(r,r')$ from r', executes the pairwise balancing algorithm described below, and finally tells the results to tracker r'.

2.3 Pairwise Approximation Algorithm

Rather than applying optimization techniques, we propose a much simpler three-step greedy-style algorithm. First, peers are tentatively shifted based only on information local to each individual torrent. For all torrents t that require merging (i.e., for which $x_{t,r} + x_{t,r'} < 2\tilde{x}$), all peers are tentatively shifted to the tracker that already maintains information about more peers for that torrent. For all torrents that should be re-balanced (i.e., for which $\min[x_{t,r}, x_{t,r'}] < \tilde{x}$ and $x_{t,r} + x_{t,r'} \geq 2\tilde{x}$), the minimum number of peers ($\tilde{x} - \min[x_{t,r}, x_{t,r'}]$) needed to ensure that both trackers have at least $\tilde{x}$ peers are tentatively shifted to the tracker with fewer peers.

Second, towards achieving load conservation of M_r, the peer responsibility of some torrents may have to be adjusted. Using a greedy algorithm, the tentative allocations are shifted from the tracker that saw an increase in peer responsibility (if any) towards the tracker that saw a decrease in overall peer responsibility. To avoid increasing the number of peers associated with partial shifts, priority is given to shifting responsibilities of the torrents that are being merged. Among these torrents, the algorithm selects the torrent that results in the largest load adjustment and that does not cause the imbalance in overall load to shift to the other tracker. By sorting torrents based on their relative shift, this step can be completed in $O(|\mathcal{T}(r,r')| \log |\mathcal{T}(r,r')|)$ steps.

Finally, if overall load conservation is not yet fully achieved, additional load adjustments can be achieved by flipping the peer responsibilities of the pair of torrents that (if flipped) would result in the load split closest to achieving perfect load conservation (across all torrents), with ties broken in favor of choices that minimize the total shift of peers. Of course, considering all possible combinations can scale as $O(|\mathcal{T}(r,r')|^2)$. However, by noticing that only the torrents with the smallest shift for each load shift are candidate solutions, many combinations can be pruned. By sorting the torrents appropriately, our current implementation achieves $O(|\mathcal{T}(r,r')| \log |\mathcal{T}(r,r')|)$ whenever $\tilde{x}$ is finite.

We have also considered an alternative version in which priority (in step two) is given to allocations of the torrents that are being re-balanced. For these torrents, we (greedily) balance the load of the torrent with the largest imbalance first, until load conservation has been achieved or all such torrents have reached their balancing point. Results using this algorithm are very similar to those of our baseline algorithm and are hence omitted.

2.4 Implementation and Overhead

The proposed DISM algorithm could be implemented with a minor extension to the BitTorrent protocol. The only new protocol message required is a *tracker_redirect* message that could be used by a tracker to signal to a peer that the peer should contact an alternative tracker for the torrent. The message would be used by a tracker r for a torrent t for which $x_{t,r}$ decreases due to the execution of DISM. Peers that recieve the message should contact another tracker they know about from the tracker file.

The communication overhead of DISM is dominated by the exchange of torrent popularity information between the trackers and by the redirection messages sent to the peers. Distributed negotiatation involves one tracker scrape before every pairwise balancing, and the corresponding exchange of the results of the balancing. The amount of data exchanged between trackers r and r' is hence $O(\mathcal{T}(r,r'))$. The amount of redirection messages is proportional to the number of peers shifted between swarms, and is bounded by $\sum_{t \in \mathcal{T}(r,r')} x_t$.

3 Protocol Validation

3.1 Empirical Data Set

We used two kinds of measurements to obtain our data set. First, we performed a screen-scrape of the torrent search engine *www.mininova.org*. In addition to claiming to be the largest torrent search engine, *mininova* was the most popular torrent search engine according to *www.alexa.com* during our measurement period (Alexa-rank of 75, August 1, 2008). From the screen-scrapes we obtained the sizes of about $330,000$ files shared using BitTorrent, and the addresses of $1,690$ trackers.

Second, we scraped all the $1,690$ trackers for peer and download information of all the torrents they maintain. (Apart from interacting and helping peers, a tracker also answers *scrape* requests at its *scrape URL*.) For the tracker-scrapes we developed a Java application that scrapes the scrape URL of each tracker. By not specifying any infohash, the tracker returns the scrape information for all torrents that it tracks. This allowed us to efficiently obtain the number of leechers, seeds, and completed downloads as seen by all trackers that we determined via the screen-scrape of *mininova*.

We performed the tracker-scrapes daily from October 10, 2008, to October 17, 2008. All scrapes were performed at 8pm GMT. We removed redundant tracker information for trackers that share information about the same swarms of peers, and identified 721 independent trackers. Table 2 summarizes the data set obtained on October 10, 2008.

Figure 1 summarizes some of the main characteristics of the world of trackers and torrents captured by our measurements. Figure 1(a) shows the size of torrents and swarms against their rank. Similar to previous content popularity studies (e.g., [1, 3, 4, 5]), we find that
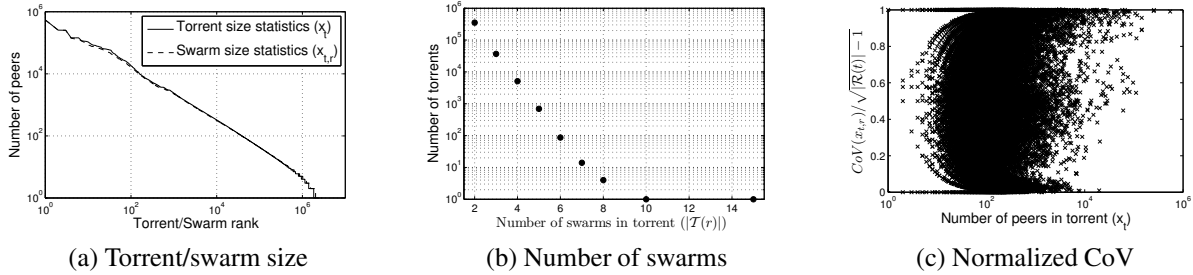
(a) Torrent/swarm size   (b) Number of swarms   (c) Normalized CoV

Figure 1: Basic properties of the multi-torrent, multi-swarm system.

(a) Number of peers   (b) Number of leechers   (c) Number of seeds
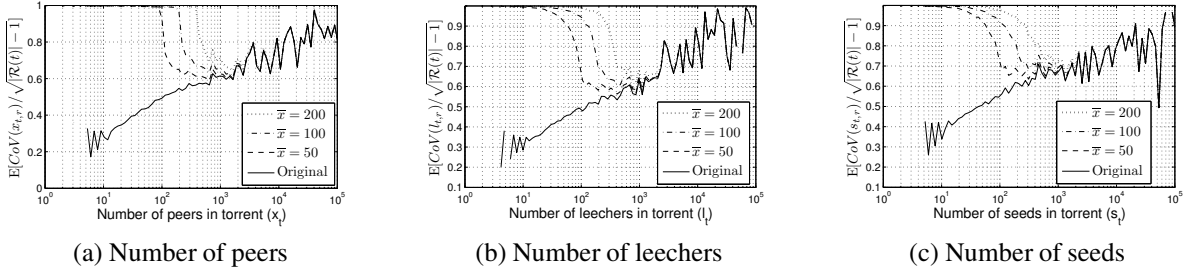
Figure 2: The average coefficient of variation (CoV) as a function of (a) peers, (b) leechers, and (c) seeds.

Table 2: Summary of a snapshot (Oct. 10, 2008).

Item	Value
Total trackers	1,690
Unique trackers	721
Unique torrents	2,864,073
Unique swarms	3,309,874
Total leechers	21,088,533
Total seeds	16,985,796
Total downloads	$4.52 \cdot 10^9$

the torrent rank distribution is heavy tailed, with Zipf-like characteristics, and a substantial number of torrents have moderate popularity. Based on our screen-scrapes, we found no correlation between content size and torrent popularity (the correlation coefficient is approximately 0.01).

Figure 1(b) shows the number of torrents with a given number of unique swarms (after removing duplicates). Clearly, there are a substantial number of torrents that are served independently by multiple trackers. To assess the fraction of small swarms that would benefit from being merged, Figure 1(c) shows the normalized coefficient of variation (CoV) of the swarm sizes versus the torrent size, of each observed torrent. A value of 0 of the normalized CoV shows that swarm sizes are equal; a value of 1 shows that all swarms are empty except for one. For small torrents (e.g., smaller than 200 peers) we would like the normalized CoV to be close to one. While the normalized CoV only can take a finite number of values (hence the distinguishable patterns), this figures show that there are many small torrents that could benefit from being merged.

3.2 Protocol Convergence/Complexity

Before analyzing the performance benefits of dynamic swarm management, we note that DISM executes quickly, with low overhead for the trackers. Using the data obtained on October 10, 2008, there are 550 trackers that have to participate in the algorithm (the rest do not have overlap in torrents). The average overlap between these 550 trackers is $1,800$ torrents. In total, $1,255$ pairwise balancings were performed; i.e., slightly more than two per tracker on average. We argue that this overhead is negligible compared to the load due to scraping and announce traffic caused by the peers.

3.3 Peer Allocation using DISM

To analyze the performance benefits of dynamic swarm management, we used the DISM algorithm to re-allocate the peers between swarms belonging to the same torrent. Figure 2 shows the average normalized CoV as a function of the number of peers for October 10, 2008. Figure 2(a) shows results in which the full torrent size is considered. Figures 2(b) and (c) show the same results, but for leechers and seeds, respectively. All figures show results for both the original peer allocation, as well as for the distributed algorithm using three different threshold values (i.e., $\tilde{x} = 50, 100, 200$). We note that the distributed algorithm pushes the CoV for torrents smaller than the threshold values (and even a bit beyond) to roughly one. As previously discussed, this is exactly what we want.

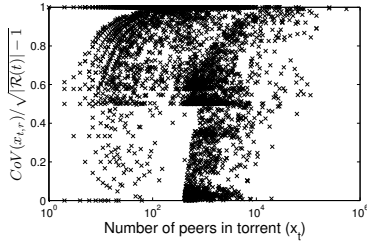Finally, to further analyze the re-balancing achieved of

Figure 3: Normalized CoV after applying DISM.

torrents, Figure 3 shows the normalized CoV for all torrents. Comparing Figure 3 with Figure 1(c) we note that the distributed algorithm significantly reduces the number of torrents operating in the lower left corner. While there still are some torrents that do, we note that Figure 2 suggests that this number is very small.

3.4 Throughput Analysis

Thus far our analysis has been based on the assumption that small torrents are more likely to perform poorly. To validate this assumption and allow us to obtain estimates of the potential performance gains small torrents can obtain using our distributed algorithm we need to estimate the throughput of different sized swarms.

To estimate the throughput of any particular swarm we measured the number of seeds, leechers, and cumulative number of downloads for two consecutive days. Using these values we estimated the throughput per leecher as the file size L divided by estimated download (service) time S, which using Little's law can be expressed as $S = l/\lambda$, where l is the average number of leechers currently being served in the system and λ the current arrival rate, which due to flow conservation can be estimated as the overall throughput (equal to the number of download completions D during that day, times the file size L, and divided by the time T between two consecutive measurements). To summarize, we have an estimated throughput of $\frac{LD}{lT}$. Finally, throughput estimates for any particular swarm type were obtained by taking the average over all swarms of that particular size.

Figure 4 shows our throughput estimation results. Here, swarms are classified based on the total number of peers in the swarm ($s + l$) and the seed-to-leecher ratio $\frac{s}{l}$; 60 bins are used for the swarm sizes and three curves are used for the different seed-to-leecher ratios. We note that the results for swarms up to just over $1,000$ peers are consistent with intuition and previous measurement results [9]. For larger swarm sizes we believe that the above estimation is less accurate. Estimation errors may be due to inaccurate estimation of the average number of leechers l over the measurement period, for example. Furthermore, we note that the popularity of these type of workloads typically follows a diurnal pattern and

our measurements are done at peak hours. Therefore, the throughput estimations are pessimistic.

Using these throughput estimations we can estimate the speedup achieved by the distributed algorithm. Figure 5 shows the relative throughput improvement for leechers as a function of the torrent size. The results show a good match with our objectives: the smallest torrents experience throughput gains up to 70 percent on average, while torrents above approximately 200 peers are not affected by our algorithm.

Figure 6 shows the relative estimated throughput improvement for leechers in torrents smaller than 300 peers over a week. The figure shows the estimated increase of the throughput per leecher averaged over the torrents for three different values of the threshold $\tilde{x}$. The curves marked with $\times$ show results weighted with the torrent size; the curves without markers show the non-weighted gain. The throughput gains are rather steady in both cases, and show that dynamic swarm managenemt could improve peer throughput significantly.

4 Related Work

Much research has been done to understand BitTorrent and BitTorrent-like [6, 11] systems. While most of these efforts have been towards understanding the performance of single torrent systems, there have been some recent papers that consider multi-torrent environments [7, 10]. Guo *et al.* [7] provide measurements results that show that more than 85% of torrent users simultaneously participate in multiple torrents. The authors also illustrate that the "life-time" of torrents 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.* [10] propose incentives that motivate users to act as seeds in such systems. In particular, Yang *et al.* propose a cross-torrent tit-for-tat scheme in which unchocking is done based on the aggregate download rates of a peer across all torrents (instead of only based on the download rate for a single torrent).

Rather than increasing seed capacity through cooperation among peers downloading different files, this paper focuses on how multiple swarms of the *same* file can be merged to improve the performance of small swarms. To the best of our knowledge, no prior work has considered the performance benefits from adaptively merging and/or re-distributing peer information among trackers.

Other related works have considered the effectiveness of BitTorrent's tit-for-tat and rarest-first mechanism [2, 8]. BitTorrent-like systems have also been considered for streaming [11]. Finally, we note that long-tailed popularity distributions have been observed in many other contexts, including Web workloads [1, 3] and user generated content [4, 5].
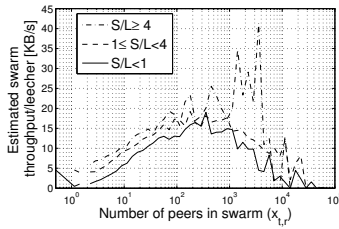
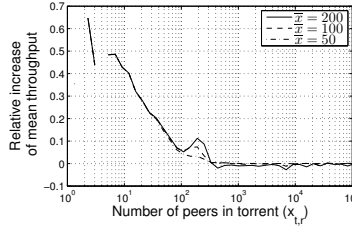Figure 4: Throughput estimation for three classes of swarms
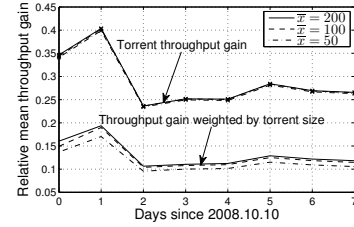

Figure 5: Estimated speedup vs. torrent size

Figure 6: Estimated speedup for torrents smaller than 300 peers vs. time

5 Conclusions

Using an extensive measurement study we have observed that there exist many moderately popular torrents that could significantly benefit from the re-balancing of peer information available on the trackers. In this paper, we consider a light-weight distributed swarm management algorithm that manages the peer information at the trackers while ensuring load conservation among the trackers. The algorithm is relatively simple and achieves much of its performance improvements by identifying and merging peer information of small swarms. Apart from improving the average throughput, merging small swarms also facilitates locality-aware peer selection.

Finally, we note that there are at least two alternatives to DISM to avoid forming small swarms. One alternative is for peers to perform a scrape of all responsible trackers before joining a torrent, and to pick the largest swarm. The disadvantage of this solution is that it does not allow load balancing for large torrents. The other alternative is based on random tracker hopping and the peer exchange protocol (PEX). A peer would change tracker (i.e., swarm) with a probability proportional to $1/x_{t,r}$, and would use PEX to distribute the addresses of the peers it knows about from the previous swarm. The disadvantage of this alternative is that not all BitTorrent clients support PEX.

While DISM is not the only way to improve the throughput of small swarms, the potential benefits of the other solutions would be similar to those of DISM, discussed in this paper. Future work will compare alternative approaches.

6 Acknowledgments

References

[1] M. Arlitt and C. Williamson. Internet Web Servers: Workload Characterization and Performance Implications. *IEEE/ACM Trans. on Networking*, 5(5):631–645, Oct. 1997.

[2] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving a BitTorrent Networks Performance Mechanisms. In *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006.

[3] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proc. IEEE INFOCOM*, New York, NY, Mar. 1999.

[4] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proc. ACM IMC*, San Diego, CA, Oct. 2007.

[5] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View from the Edge. In *Proc. ACM IMC*, San Deigo, CA, Oct. 2007.

[6] C. Gkantsidis and P. R. Rodriguez. Network Coding for Large Scale Content Distribution. In *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005.

[7] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurement, Analysis, and Modeling of BitTorrent-like Systems. In *Proc. ACM IMC*, Berkley, CA, Oct. 2005.

[8] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest First and Choke Algorithms are Enough. In *Proc. ACM IMC*, Rio de Janeiro, Brazil, Oct. 2006.

[9] X. Yang and G. de Veciana. Service Capacity of Peer-to-Peer Networks. In *Proc. IEEE INFOCOM*, Hong Kong, China, Mar. 2004.

[10] Y. Yang, A. L. H. Chow, and L. Golubchik. Multi-Torrent: A Performance Study. In *Proc. MASCOTS*, Baltimore, MD, Sept. 2008.

[11] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: A Datadriven Overlay Network for Peer-to-Peer Live Media Streaming. In *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005.