Toward Efficient On-demand Streaming with BitTorrent

Youmna Borghol[1,2], Sebastien Ardon[1], Niklas Carlsson[3], and Anirban Mahanti[1]

[1] NICTA, Locked Bag 9013, Alexandria, NSW 1435, Australia
[2] School of Electrical Engineering and Telecommunications, University of New South Wales, NSW 2030 Sydney, Australia
[3] University of Calgary, Calgary, AB, Canada
{youmna.borghol,sebastien.ardon,anirban.mahanti}@nicta.com.au

Abstract. This paper considers the problem of adapting the BitTorrent protocol for on-demand streaming. BitTorrent is a popular peer-to-peer file sharing protocol that efficiently accommodates a large number of requests for file downloads. Two components of the protocol, namely the rarest-first piece selection policy and the tit-for-tat algorithm for peer selection, are acknowledged to contribute toward the protocol's efficiency with respect to time to download files and its resilience to free riders. Rarest-first piece selection, however, is not suitable for on-demand streaming. In this paper, we present a new adaptive window-based piece selection policy that balances the need for piece diversity, which is provided by the rarest-first algorithm, with the necessity of in-order piece retrieval. We also show that this simple modification to the piece selection policy allows the system to be efficient with respect to utilization of available upload capacity of participating peers, and does not break the tit-for-tat incentive scheme which provides resilience to free riders.

Key words: Peer-to-Peer, Video-on-Demand, BitTorrent

1 Introduction

BitTorrent [5], probably the most popular peer-to-peer (P2P) file sharing system, is a loosely coupled distributed system that allows peers to opportunistically exchange file pieces. One important aspect of BitTorrent is the *rarest-first* piece selection algorithm used by each peer to select which piece to download from another peer. This algorithm provides a simple, fully decentralized, mechanism for system-wide replication of pieces such that file download times are minimized. Another important component is the *tit-for-tat* algorithm. Using rate-based reciprocation, this algorithm provides peers with upload incentives and provides resilience to free riders. Note that the symmetrical nature of tit-for-tat results in the average aggregate download rate (in systems with limited server or seed resources) being capped by the average peer upload capacity. BitTorrent can therefore be seen as a system allowing upload capacity sharing. It has been shown that these two components are fundamental to the scalability and performance of BitTorrent [9] [12].

We consider the problem of adapting BitTorrent to support on-demand streaming and more specifically, a view-as-you-download service. From a user experience point-of-view, we desire that the system provides low latency to begin playback as well as

jitter-free playback. Networked streaming systems attempt to attain these properties by encoding the media at a bitrate lower than the average available download bandwidth, and using a playout buffer to hide variation in available bandwidth.

Designing a system that satisfies the desired user experience, while retaining the salient features of BitTorrent, namely scalability with respect to swarm size and resilience to non-cooperative peers, is challenging. BitTorrent's file download efficiency is, in part, due to the use of a rarest-first piece selection policy; this policy increases piece diversity in the swarm, and therefore allows efficient use of available upload capacity of peers. Rarest-first piece selection cannot satisfy the user requirements mentioned above. Replacing the rarest-first piece selection policy with a strict in-order policy, however, makes the system sluggish as piece diversity is significantly reduced and tit-for-tat becomes largely irrelevant [11].

This paper presents a new adaptive window-based piece selection policy that achieves a balance between the system scalability provided by the rarest-first algorithm and the necessity of in-order pieces for seamless media playback. In particular, we propose that each peer maintains a sliding window wherein the window size is adapted based on the amount of in-order data available in the peer's buffer and its current playback position. Within the window, a variant of the rarest-first piece selection policy is used. Thus, when the window is small, near in-order piece retrieval occurs, whereas when the window is large, near rarest-first piece retrieval occurs.

Our simulations show that the simple adaptive window-based piece selection policy allows the system to be efficient with respect to utilization of available upload capacity, and also does not break the tit-for-tat incentive scheme. A swarm-based on-demand streaming protocol needs to be resilient to non-cooperative peers (free riders); in particular, the performance of the cooperating peers should not degrade because of the presence of free riders. Note that non-cooperating peers are not necessarily the result of malicious activity: a misconfigured firewall, NAT router, or other middle boxes can be the source of such problem as these may block the upload traffic from a peer.

The remainder of the paper is structured as follows. Section 2 discusses related work. The simulation tool used and our experiment methodology are discussed in Section 3. The adaptive window-based piece selection policy is presented in Section 4, along with a comparaison with prior work on fixed-sized window-based piece retrieval policies. Section 5 presents a detailed performance evaluation. A proposal that can substantially reduce the latency to begin playback is outlined in Section 6, followed by conclusions in Section 7.

2 Related Work

There has been much work on the design of peer-assisted video-on demand systems [8] using BitTorrent-like protocols. Accounting for the real-time needs of the streaming application has been the key challenge. In order to tackle the problem, modifications have been mainly proposed to BitTorrent's piece and peer selection policies.

Related studies on the piece selection algorithms have concentrated on finding a policy that can achieve a good trade-off between meeting the sequential requirement of playback and maintaining a high level of piece diversity in the system. Prior research

can be broadly classified into probabilistic [3, 6], segment-based [1], and window-based [13–16] approaches. With probabilistic piece policies, pieces are selected based on some probability distribution, which biases the selected pieces toward earlier pieces. With window-based approaches, a sliding window is typically used from within which pieces are given preference. Naturally, a smaller window ensures close to sequential piece retrieval, whereas a larger window, within which rarest-first can be used, for example, can ensure higher piece diversity. While all the above approaches provide a reasonable tradeoff between downloading pieces in roughly sequential order and ensuring sufficient diversity, we note that none of these approaches adaptively increases the use of rarest-first when possible.

While we are not aware of any adaptive window-based piece selection policies, we note that Carlsson *et al.* [4] have studied how a Zipf-based probabilistic piece selection policy can be enhanced to take current buffer conditions into account. They show that there are significant advantages to quickly provide newly arrived peers with rare pieces that can be shared with many users, as well as prioritize pieces that have urgent deadlines. Based on these insights they propose new policies that better utilize the server bandwidth to quickly bootstrap the tit-for-tat behavior of newly arrived peers. In contrast, we propose an adaptive-window approach which is used for both peer-to-peer or peer-to-server communication, and most importantly allows peers to dynamically help increase the piece diversity when conditions are favourable, making it more and more tit-for-tat compatible the higher download rates are available to the peers.

Another proposal has assumed in-order piece selection, and proposed performance-based peer selection policies in which peers are selected based on the urgency of the pieces they request (i.e., how soon they need the requested pieces for uninterrupted playback) and the availability of serving peers [7]. One key drawback of such design is its exposure to selfish peers as no incentive mechanism is provided for contribution. Parvez *et al.* [11] showed that upload utilization (and system throughput) can be significantly improved in systems using in-order piece selection, by giving upload preference to peers with similar playback points. This is particularly important for older peers which typically have less potential uploaders. In this paper, we show that a window-based approach in combination with rate-based tit-for-tat can achieve much of the same attractive clustering characteristics, in which peers with similar playback points (or overlapping windows) share pieces with each other.

To discourage free-riding, mechanisms have been suggested to favour peers who are verified to be good uploaders through some feedback mechanism [10]. As such techniques rely on externally gathered information about peer contributions, these systems potentially are more vulnerable to malicious peers than tit-for-tat schemes (in which the peers themselves can easily measure the rate at which pieces are exchanged). Due to the sequential nature of streaming there has, however, been some debate regarding whether or not tit-for-tat should be used. For example, it has been argued that peers will be downloading from older peers and will not have much to give in return. This is the case provided the piece selection policy is in-order or a close variant. Naturally, the effectiveness of tit-for-tat in such systems is (at best) limited. In this paper, we develop a policy that supports on-demand streaming while retaining the tit-for-tat component of BitTorrent.

3 Methodology

3.1 Simulation environment

We use the Microsoft Research BitTorrent simulator [2]. The simulator simulates both uplink and downlink peer capacities without incurring the overhead of a packet-level simulator, and has been widely used.

All simulations have a single seed with uplink capacity of 6Mbps. The file size is 300MB with a video playback bitrate of 800Kbps (approximately 50 minute long). The file consists of 1200 pieces, each of size 256KB. Unless stated otherwise, all peers have an upload capacity of 1Mbps, and a download capacity of 2Mbps. In our simulations, peers leave the system as soon as they finish downloading the file. We use BitTorrent's default settings for the remaining parameters.

Two typical scenarios, as discussed below, are used in the experiments reported in this paper.

- Flash crowd scenario: In this scenario, we assume that all peers join the swarm nearly simultaneously (uniformly within 30s). The flash crowd scenario is often seen as a benchmark to evaluate swarm-based streaming systems, and provides a measuring point with regards to how the system handles extremely bursty request loads. With all peers typically progressing together in the download the seed may be the only peer that has pieces. In this scenario, the simulation ends when the last peer completes the download. Unless noted otherwise, results are shown for experiments with a population of 200 peers.
- Poisson scenario: In this scenario, peers arrive to the torrent according to a Poisson arrival process with rate λ. The Poisson scenario captures the performance of a system in which peers arrive to a swarm where there are peers that have been in the system for different time durations, and such peers already may have accumulated some number of pieces, depending on how long they themselves have been in the system. The arrival rate λ reflects the current file popularity. For these scenarios, the simulation time is set to five times the movie duration, and statistics from the second last duration were presented. To verify that we had reached steady state these results where compared with those reached in the second duration. For evaluation, we used peer arrival rates of $0.01, 0.1, 1$ peers per second. Assuming that peers download at roughly the play rate and leave immediately after completing their downloads, these arrival rates translate to swarm size of 30, 300, and 3000 peers, respectively.

3.2 Evaluation Metrics

For performance evaluation and policy comparisons, we use three primary metrics:

- *Upload Capacity Utilization*: The upload capacity utilization is defined as the fraction of the peers' upload capacity that is utilized during their download duration. Upload utilization is an important metric as higher utilization means higher overall system efficiency, and higher resilience to changing network available bandwidth.

– *Success Ratio*: The success ratio is defined as the ratio of pieces that were downloaded before their playback deadline, over the total number of pieces. This metric aims at measuring the playback continuity. This metric is also referred to as the continuity index [15].
– *Initial Buffering Time*: We assume that each peer must have received all pieces in its initial buffer, of size B, before playback can commence. We refer to the duration of this process as the initial buffering time.

Unless stated otherwise, results presented in Section 5 and 6 represent performance metrics using whiskers-plot diagrams which show the median, the 25 and 75 percentiles, as well as the minimum and maximum values, across all peers.

4 Adaptive Window for On-Demand Streaming

Prior work has proposed using a *fixed-size* sliding window to limit the set of pieces which can be requested by a peer [13–15]. Within the window, rarest-first or a variant has been used, with the objective of balancing the need of in-order downloads for streaming and the need of piece diversity for maintaining system efficiency. With these approaches, the window has to be kept small relative to the file size to support on-demand streaming with relatively small startup delay. A by-product of this requirement, however, is that the number of pieces available for exchange is limited, and therefore, peer's upload bandwidth utilization cannot be maximized [13].

We propose an *adaptive* window strategy where each peer dynamically computes its window size, depending on how well the peer's download is progressing. In particular, the adaptive window grows or shrinks depending upon how much additional data is available in the playback buffer, with respect to the playback position of the peer. A peer that has a large amount of in-order data available in its playback buffer will increase its window, and therefore, contribute to the system by increasing piece diversity. A peer close to stalling with a near-empty playout buffer, will shrink its window and request pieces closest to its playback position. As we demonstrate in this paper, this adaptive window mechanism is able to maximize utilization of the peers' upload capacities, and is also able to retain the tit-for-tat behavior of BitTorrent.

Fig. 1 illustrates the adaptive window mechanism. Each peer computes its window w at each piece selection event using:

$$w = \max\left[k(d - p - \theta), 0\right] + w_{min}, \tag{1}$$

where w is the effective window size, w_{min} is a pre-defined minimum window size, k is a scaling coefficient, d is the peer download position (i.e., the last contiguous piece available in the playout buffer), p is playback position (i.e., the piece that is currently being played back), and θ is a threshold that places a lower bound on the number of contiguous pieces required in the playout buffer before the window can grow. When a peer joins the swarm, its window size is initialized to w_{min}; this value needs to be sufficiently large to introduce enough diversity without playback interruption and yet ensure small buffering time. (Note that the larger the window, the more time it takes to obtain a set of contiguous pieces starting with the first piece of the file because of

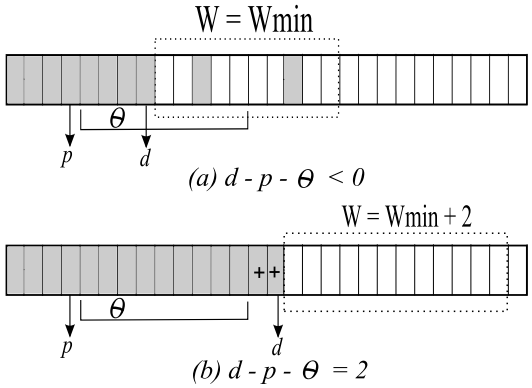


Fig. 1. Illustration of the adaptive window mechanism.

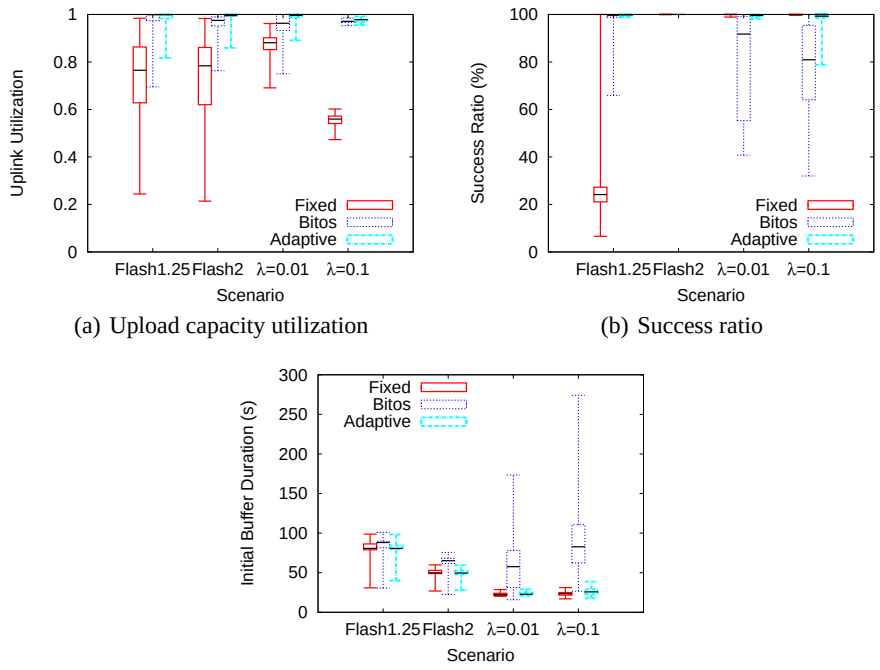
the use of rarest-first piece selection policy or its variants, as discussed below.) Once playback starts, and if a peer enjoys a faster download rate than its media playback rate, for every extra piece beyond threshold θ the window is incremented by a scale factor k. Following extensive experiments, we found k = 1, w_{min} = 20, and θ = 2.5 w_{min} to give the best performance, and we use these values in the remainder of the paper.

For the purpose of defining the window boundaries, we use two definitions: one in the initial buffering state and another once playback starts. In the initial buffering state, to define the boundaries of the window, we count the arrived and non-arrived pieces, as in [14]. In the playback state, we count only the non-arrived pieces, as in [15].

Another design consideration is the piece selection policy inside in the window. We experimented with a number of alternative piece selection policies [3]. We found that rarest-first (within a window) performs reasonably well; it attains very high success ratios (e.g., in excess of 99%), with piece misses distributed nearly uniformly over the file. The piece misses tend to happen at the beginning of the window as the pieces toward the end of the window appear to be rarer in the system. Our experiments suggest that a variant of the Portion policy [3], where each peer independently applies, within the window, rarest-first selection with probability q and in-order selection with probability $(1 - q)$, achieves similar upload utilization as rarest-first but with fewer misses. In all applications of the adaptive window policy in this paper, Portion with $q = 0.1$ is used.

Fig. 2 presents illustrative results for the proposed *adaptive window* approach along with results for two recently proposed fixed-window approaches, namely *fixed-size window* [14] and *Bitos* [15]. The window size for the fixed-size window protocol is set to 20 pieces. We report results from two flash crowd experiments; one with the ratio of upload bandwidth (U) over playback rate (PBR) equal to 1.25, and another with U/PBR = 2. We also report results from two Poisson scenarios with U/PBR = 1.25 and arrival rates λ of 0.01 and 0.1.

From Fig. 2, we observe that the adaptive-window policy is generally more successful for streaming across both flash crowd and Poisson scenarios. In the flash crowd scenario, the fixed window policy fails to provide good upload bandwidth utilization, and as a result fails to provide a viable streaming experience when available upload



(a) Upload capacity utilization (b) Success ratio

(c) Initial buffering time

Fig. 2. Comparison of adaptive and fixed window policies.

bandwidth is not abundant (e.g., illustrated by the low median success ratio in the U/PBR=1.25 experiment). The main reason for the lower system throughput is the conservative definition of the window boundaries. In the Poisson scenario, the Bitos policy typically results in poor success ratios, with wide variations across peers (e.g., for $\lambda = 0.01$, 25% of peers have as low as 50% success ratio). Bitos typically achieves reasonable upload utilization. As Bitos clients always download 20% of pieces outside the window, their rate of in-order pieces retrieved is affected. Meanwhile, the adaptive-window policy always takes the current buffer conditions into account when adapting the rate at which non-in-order pieces is being retrieved. Overall, the adaptive-window policy consistently achieves high upload capacity utilization, low initial buffering times, and success ratios close to 100% in nearly all cases, with very little variations across peers.

From Fig. 2(c), we can also see that the buffer time in the flash crowd scenario, were all peer are synchronized, is approximately the same for all three policies. In the Poisson scenarios, the buffer time is much lower with the fixed-size and adaptive-window policies. This is because the fixed-size window policy (also used for the initial buffering in the adaptive-window policy), does not allow peers to move the boundary of the window until the first piece is retrieved, ensuring small buffering times.

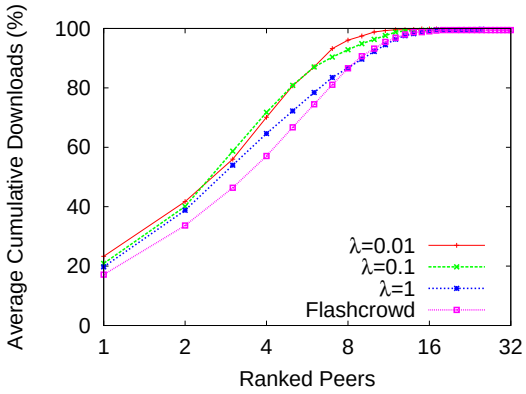


Fig. 3. Fraction of file data from peers ranked by their contribution (U/PBR = 3).

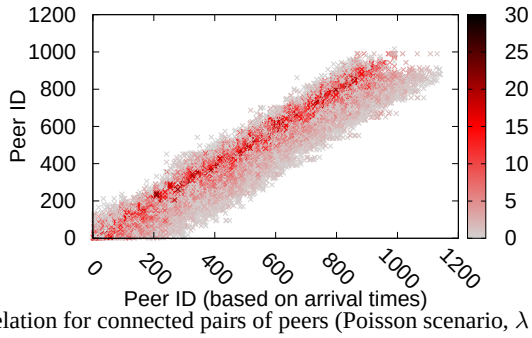


Fig. 4. Time correlation for connected pairs of peers (Poisson scenario, $\lambda = 0.1$, U/PBR = 3).

5 System Evaluation

BitTorrent's rate-based tit-for-tat algorithm selects which peer to serve next at each choking interval [5]. The four peers that provide the best download rates are selected, while a fifth peer is selected at random. This rate-based peer selection algorithm acts as an incentive mechanism for peers to share their upload bandwidth. Legout *et al.* [9] have shown that tit-for-tat ensures reciprocation among peers, and results in relatively stable connections among peers. In this section, we evaluate these system-level properties in the context of our adaptive window-based approach for on-demand streaming.

5.1 Stability of Connections

In general, infrequent chocking of peers leads to higher upload bandwidth utilization as less time is wasted waiting for unchoke messages. Thus, if peer connections are stable for long time periods, we expect a large fraction of the file to be obtained from a small set of peers relative to the swarm size.

We present example results in Fig. 3. For each peer i, we calculate the amount downloaded from any other peer j in the swarm, d_{ij}, and maintain a rank-ordered list

of d_{ij}'s where the highest download is assigned rank one. Fig. 3 shows, on average, the fraction of the total file retrieved from peers in the swarm as a function of peer rank. The figure shows that in the simulations considered, on average, 50-65% of the video file was consistently downloaded from four peers. This confirms that connections using our approach are relatively stable.

We also analysed which peers connect with each other, and more importantly, which peers exchange pieces with each other. Fig. 4 shows the correlation in-time between the data transferred between connected peers in an experiment with Poisson arrivals at rate λ = 0.1. Peer IDs are assigned in ascending order based on a peer's arrival time to the swarm. For every peer i, the scatter plot shows the percentage of the file downloaded from any other peer j. We observe a darker line in the middle of the graph, suggesting that most of the data is retrieved from a few peers that joined the swarm close together in-time (i.e peers with close playback points).

5.2 Reciprocation

We define the Reciprocation Ratio, R_i, for each peer i as $R_i = \frac{\sum_j \min[d_{ij}, u_{ij}]}{\sum_j d_{ij}}$, where u_{ij} is the amount of bytes i uploaded to j, and d_{ij} is the amount of bytes i downloaded from j. Fig. 5 shows the cumulative distribution of the Reciprocation Ratio in different scenarios. Overall, we observe good reciprocation among peers, especially for the flash crowd experiments, and the Poisson experiments with arrival rate $\lambda \geq 0.1$. Note that for lower arrival rates there typically are very few peers in the system.

Our results confirm that the reciprocation enforced by tit-for-tat in the original BitTorrent is maintained in the adaptive window algorithm. We show that peers tend to maintain stable connections with few peers close to their download position, as they have overlapping windows and thus pieces to exchange with each other. The swarm topology is, therefore, constrained by the slack allowed by the window. This is an interesting side-effect. Our adaptation policy naturally tends to group peers based on their playback positions, in contrast to other approaches that create similar 'daisy chains' structures using more complicated peer selection policies [11].

5.3 Heterogeneous Peer Upload Capabilities

Next, we consider the impact of peer heterogeneity on the system. In particular, we are interested in understanding the performance degradation in the presence of peers that have upload capacity less than the media playback rate. Fig. 6 shows the performance of capable peers (U/PBR = 1.25 in these simulations) as a function of the percentage of slow peers in the swarm. Results are shown for cases in which the slow peers have U/PBR = 0.25 and 0.5. We observe that even with 30% slow peers, capable peers miss only between 2-3% of the data, while still achieving reasonable buffering times.

5.4 Free-riding Peers

BitTorrent's tit-for-tat mechanism manages free riding peers (i.e., peers that only download but do not contribute any upload bandwidth) by penalising them. Here we investigate the effectiveness of tit-for-tat in the context of our proposed adaptive window.

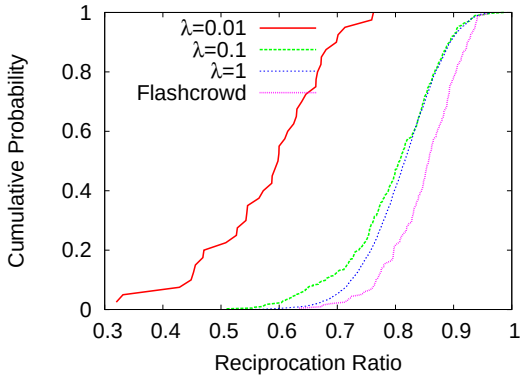
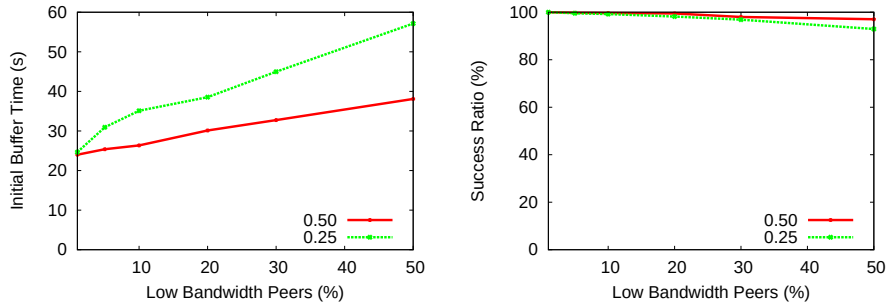
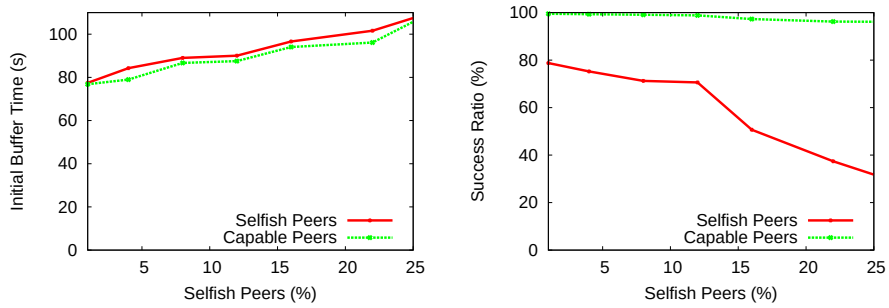


Fig. 5. Reciprocation among peers (Poisson and flash crowd scenarios, U/PBR =3).



(a) Initial buffering time (b) Success ratio

Fig. 6. Impact of network heterogeneity (Poisson scenario, $\lambda = 0.1$, U/PBR =1.25).



(a) Initial buffering time (b) Success ratio

Fig. 7. Resilience to free-riding peers (Flash crowd scenario, U/PBR = 1.25).

Figs. 7(a) and (b) show the initial buffering time and success ratios, respectively, attained by both the free riders and the contributing peers. As desired, our results show that free riders are punished while the contributing peers are not substantially affected.

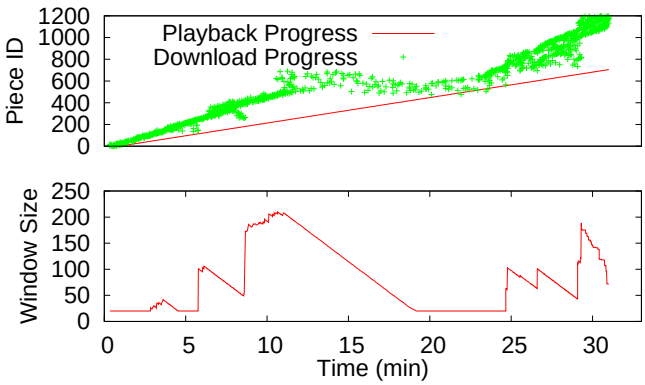


Fig. 8. Responsiveness under changing network conditions.

These results demonstrate the ability of the adaptive window policy to retain the effectiveness of tit-for-tat, which is regarded as a fundamental component of BitTorrent.

5.5 Changing Network Conditions

Finally, we consider an example scenario in which one peer has time varying upload capacity. We use the default flash crowd scenario with U/PBR=2 and 200 peers. During the course of the simulations, one peer's upload rate is reduced to 200Kbps at 700s after its arrival to the system, and remains reduced for the next 700s (i.e., from roughly 12 to 23 minutes). Fig. 8 shows the window progress and piece retrievals for this peer. We note that the policy is responsive to the changing network conditions, and reduces the window size sufficiently quickly such that almost no pieces are retrieved late. (This particular peer had a success ratio of 99.8%.) We also note that the reduced download rate as the peer's upload bandwidth changes, provides clear evidence that tit-for-tat is working.

6 Bootstrapping and Buffering

In this section, we take a closer look at the factors contributing to the initial buffering time and propose a simple mechanism that significantly reduces the initial buffering time.

6.1 Buffering Time Analysis

As previously defined, in Section 3.2, the *initial buffering time* is the time a peer requires to receive all pieces in its initial buffer, before starting playback. One key component of this delay is the time to obtain the first piece, which the peer then can use to upload to others; we refer to this time as the *bootstrap time*. Furthermore, the amount of data buffered during the initial buffering time can directly impact the success ratio. Clearly, the larger the initial buffer size B is, the more chance of maintaining seamless playback.

However, as B increases, so does the initial buffering time. Through experimentation with different swarm sizes, scenarios, network capacities, and media bitrate values, we have found that using B equal to ten pieces provides a good compromise. In particular, this choice typically allows us to achieve a success ratio above 99%. Of course, more conservative choices or more advanced startup rules (e.g., as used in [3]) could easily be augmented to further improve the success ratios.

Fig. 9(a) shows the ratio of the initial buffering time over the bootstrap time in our baseline flash crowd scenario, for different upload capacities. Although the bootstrap time only requires a single piece to be downloaded, we note that the bootstrap time is roughly equal to 50% of the buffering time (i.e., the time to receive the first ten pieces). Clearly, the bootstrap time is the primary cause to the startup delay.

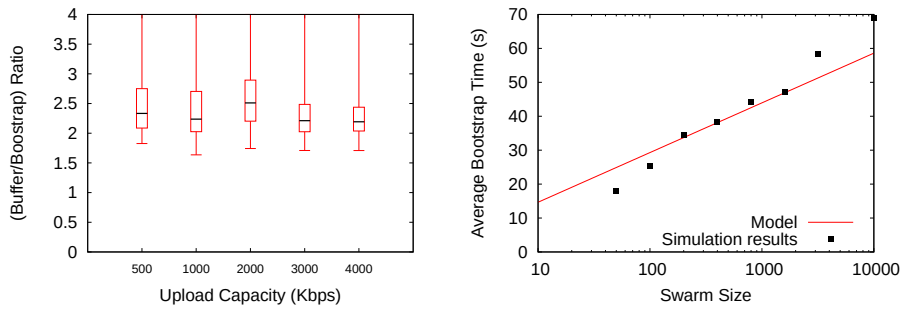
We explain this high bootstrap time by observing that in the flashcrowd scenario, peers initially have no piece to exchange with each other, and the seed is the only initial source. The probability of parallel download is therefore very low, as every peer is trying to get its first piece. Because the total number of concurrent uploads per peer is limited in BitTorrent, peers during this phase effectively form an *k-ary tree*, where k is the maximum number of upload connections per peer. Peers arriving early in the flash crowd are connected closer to the seed, and later peers gradually are forced to connect to other peers further down the tree. The further down a tree a peer is, the longer it will take for it to obtain the first piece as it first has to travel down every other peer in the tree. From this observation, and the fact that the average node depth in an *k-ary tree* is approximately $\log_k N$, we can estimate the average time to obtain the first piece to be:

$$\log_k(N)\frac{kC}{U}, \tag{2}$$

where N is the swarm size, C is the piece size, k is the maximum number of concurrent uploads per peer, and U is the average upload capacity. An interesting observation is that the bootstrap time is proportional to logarithm of the swarm size. To confirm this analysis, Fig. 9(b) shows the evolution of the bootstrap duration as a function of the swarm size. We note that the above approximation (2), provides a reasonable fit, and more importantly that the bootstrap time scales logarithmically with the swarm size, as indicated by the straight-line appearance of the graph with the swarm size on a logarithmic scale. The results are from our flash crowd experiments with varying sizes of the flash crowd.

6.2 Reducing Startup Time through Pre-fetching

In order to reduce the initial buffering time, we propose that each peer should be given a randomly chosen piece from within the initial window (of size w_{min}). From a practical point, a server could quickly upload this piece to each newly arriving peer, (or the tracker, in the case such nodes would be willing to store and serve a small subset of pieces, by piggybacking the piece in the initial reply). Alternatively, the server (or voluntary peers) could push random pieces from media files of interest to each user. For some applications, e.g. video-on-demand systems, it could be practical to drive this pre-fetching from a content recommendation engine, and allow the pre-fetching operation to take place at time of low system utilization. Another option would be to implement



(a) Ratio of buffering time over bootstrap time (b) Bootstrap time vs. swarm size

Fig. 9. Bootstrap time analysis in flash crowds

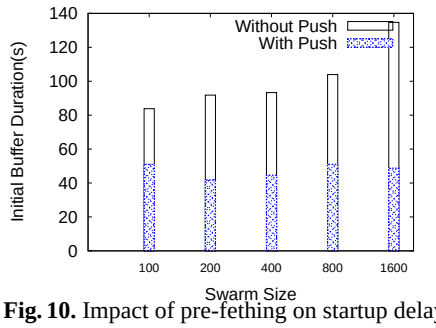


Fig. 10. Impact of pre-fethiing on startup delay

the pre-fetching as part of the content browser; in this case, one 256KB piece could be pushed along with the content thumbnail.

While quantifying the overhead of this pre-fetching is difficult and application and/or implementation-specific, we note that this approach can reduce the average bootstrap time to nearly zero, as peers immediately have blocks to exchange. Equation (2) suggests this reduction should be proportional to the logarithm of the swarm size. We implemented this pre-fetching scheme in the simulator. We ran a number of simulations with varying flash crowd sizes, both with and without prefetching. Our simulation parameters are as discussed earlier in Section 3. Fig. 10 shows the overall difference with and without pre-fetching. The figure shows that a large reduction in the initial buffering time is possible with prefetching a single piece, and further suggests that removing the bootstrap time contribution makes the initial buffering time roughly independent of the swarm size.

7 Conclusion

This paper presented a new adaptive window-based policy that effectively balances the need for piece diversity with the necessity of in-order piece retrieval. The policy makes effective use of the available upload capacity to improve piece diversity, and increase the overall system efficiency. Compared with previously proposed window-based approaches, the policy consistently achieves high upload capacity utilization, low

initial buffering times, and high success ratio. We showed that our adaptive window-based policy ensures effective use of tit-for-tat, is robust in the presence of free riders, and is resilient to variations in network available bandwidth. Finally, we showed that the latency to begin playback can be be reduced by 50%, on average, by pre-fetching a single random piece from within the initial window.

8 Acknowledgements

This work was supported by the National Information and Communications Technology Australia (NICTA) and the Informatics Circle of Research Excellence (iCORE) in the Province of Alberta, Canada. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. S. Annapureddy and D. Gunawardena. Is high-quality vod feasible using p2p swarming. In *Proc. WWW*, Banff, Canada, May 2007.
2. A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent network's performance mechanisms. In *Proc. IEEE INFOCOM*, Barcelona, Spain, 2006.
3. N. Carlsson and D. L. Eager. Peer-assisted on-demand streaming of stored media using bittorrent-like protocols. In *Proc. IFIP Networking*, Atlanta, GA, May 2007.
4. N. Carlsson, D. L. Eager, and A. Mahanti. Peer-assisted on-demand video streaming with selfish peers. In *Proc. IFIP Networking*, Aachen, Germany, May 2009.
5. B. Cohen. Incentives build robustness in bittorrent. In *Proc. Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, 2003.
6. P. Garbacki, D. H. J. Epema, and J. Pouwelse. Offloading servers with collaborative video on demand. In *Proc. IPTPS*, Tampa Bay, FL, February 2008.
7. Y. Guo, S. Mathur, K. Ramaswamy, S. Yuy, and B. Patel. Ponder: Performance aware p2p video-on-demand service. In *Proc. IEEE GLOBECOM*, Washingston, DC, November 2007.
8. Y. Huang, T. Z. J. Fu, D. M. Chiu, J. C. S. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p-vod system. In *Proc. ACM SIGCOMM*, August 2008.
9. A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *Proc. ACM IMC*, Rio de Janeiro, Brazil, October 2006.
10. J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, , and H. Sips. Give-to-get: free-riding resilient video-on-demand in p2p systems. In *Proc. MMCN*, San Jose, CA, January 2008.
11. K. Parvez, C. Williamson, A. Mahanti, and N. Carlsson. Analysis of bittorrent-like protocols for on-demand stored media streaming. In *Proc. ACM SIGMETRICS*, Annapolis, MD, 2008.
12. D. Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proc. SIGCOMM*, 2004.
13. P. Savolainen, N. Raatikainen, and S. Tarkoma. Windowing bittorrent for video-on-demand: Not all is lost with tit-for-tat. In *Proc. IEEE GLOBECOM*, New Orleans, LA, 2008.
14. P. Shah and J.-F. Pris. Peer-to-peer multimedia streaming using bittorrent. In *Proc. IEEE IPCCC*, New Orleans, LA, April 2007.
15. A. Vlavianos, M. Iliofotou, and M. Faloutsos. Bitos: enhancing bittorrent for supporting streaming applications. In *Proc. IEEE Global Internet Symposium*, Barcelona, Spain, 2006.
16. Y. Zhou, D. M. Chiu, and J. C. Lui. A simple model for analyzing p2p streaming protocols. In *Proc. IEEE ICNP*, Beijing, 2007.