Peer-to-Peer Multimedia Streaming Using BitTorrent

Purvi Shah Jehan-François Pâris

Department of Computer Science
University of Houston,
Houston, TX 77204-3010
{purvi, paris}@cs.uh.edu

Abstract

We propose a peer-to-peer multimedia streaming solution based on the BitTorrent content-distribution protocol. Our proposal includes two modifications that allow it to deliver multimedia data on time. First, we replace the rarest-first chunk downloading policy of BitTorrent by a policy requiring peers to download first the chunks that they will watch in the near future. Second, we introduce a new randomized tit-for-tat peer selection policy that gives free tries to a larger number of peers and lets them participate sooner in the media distribution. Our simulations indicate that both changes are required to achieve a good streaming quality.

1. Introduction

Recent technology advances have allowed the successful deployment of multimedia streaming for video-conferencing and surveillance applications over private networks. However streaming high-quality video over the Internet to address the demands of large-scale distance learning, telemedicine and video-on-demand still remains a challenging issue.

In recent years, peer-to-peer (P2P) technology has captured the interest of the research community as well as the industry. By allowing peers to serve each other, P2P solutions overcome many limitations of traditional client-server architectures. They can handle flash crowds (that is, very large and sudden surges of demand) as well as achieve bandwidth scalability (that is, overcome the bandwidth limitations of the server). In addition, P2P solutions do not require any special support from the network, let it be IP multicast or any specific content distribution infrastructure.

As a result, several content-providers over the Internet have adopted P2P technology to reduce the server-workload, minimize the content-distribution costs and improve their distribution times [1, 2]. In addition, there has been an increasing interest in the use of P2P architecture for large-scale, high-quality multimedia streaming.

The paper presents a P2P solution for multimedia streaming based on the BitTorrent (BT) protocol. We first identify the algorithms of BT that require to be modified to make it suitable for multimedia streaming and discuss how these changes affect the performance of our solution.

The rest of the paper is structured as follows: Section 2 presents a quick overview of BT. Section 3 introduces our modifications to the BT protocol to enhance its applicability for multimedia streaming. Sections 4 and 5 describe our experimental setup and present our results. In section 6 we discuss related work in P2P multimedia streaming. Finally Section 7 has our conclusions.

2. The BT protocol

BT [3] is a very scalable P2P protocol for large-scale content-distribution over the Internet. BT works by chopping up the data to be distributed into small *chunks* and delivering the chunks in a non sequential manner.

BT differentiates between two types of peers: *leeches* and *seeds*. Leeches are peers that only have some or none of the data while seeds are peers that have all the data but stay in the system to let other peers download from them. Thus seeds only perform uploading while leeches download chunks that they do not have and upload chunks that they have.

BT implements a set of algorithms that balances the content distribution load among a *swarm* of peers, that is, an overlay mesh network of peers. Each swarm is managed by a centralized process, a *tracker*. The tracker does not host any content but maintains metadata about it.

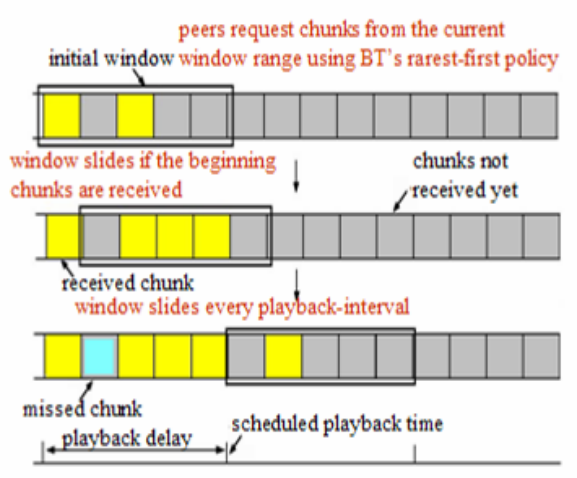

Fig. 1. Sliding window notion using the rarest-first policy for BT.

As leeches enter the swarm they first connect to the tracker. The tracker returns a random list of peers that have the content. Each leech then randomly selects a subset of that list as its *neighbors* and initiates requests to set up bidirectional TCP connections with these neighbors.

Instead of downloading directly from the server, each leech requests chunks from all the peers it is connected to. To increase the overall efficiency of the swarm, BT employs a *rarest-first* policy for selecting chunks to download. Each peer tries to download first the chunks that are least replicated among its neighbors.

BT employs a *tit-for-tat* policy to deter free riding (that is, when the peers behave selfishly and use the swarm only to download chunks without making any contribution to the swarm). Peers upload to typically k peers that recently provided it with the best downloading rate, even though it may have received requests from more than k leeches. This process of temporary refusal to upload to some peers is called *choking*. It lets each peer attempt to maximize its own interest by downloading as much as it can. BT also includes a mechanism, called *optimistic unchoking*, that lets peers reserve a share of their available bandwidth for uploading chunks to randomly selected peers. Among other things, optimistic unchoking gives newcoming peers a chance to join the swarm. The decision to choke/unchoke is performed at regular *rechoking* intervals.

In addition to having a low control overhead, a BT swarm is scalable, efficient, cost-effective, self-improving and easy to deploy. These interesting properties make BT an attractive choice for P2P multimedia streaming. In addition, BT swarms bear strong similarities to the functionality required in P2P multimedia streaming networks. In BT as in multimedia streaming, the content to be distributed is divided into multiple chunks or *segments* to be downloaded from the server.

3. Proposed modifications

Although multimedia streaming networks have similarities with BT swarms, BT in itself is not suitable for multimedia streaming since it does not account for the real-time needs of streaming applications. First, peers do not download chunks in sequence, which leaves them unusable until the download is complete. Second, the tit-for-tat policy forces too many peers to wait for too long before joining the swarm.

We propose two modifications to the BT algorithm that address these issues.

3.1. Chunk selection policy

Maintaining a window of chunks of the video at the application level is a common approach in multimedia streaming to smooth out the playback jitters. This introduces a *playback delay* at the client.

We introduce a *sliding window* which is illustrated in Fig. 1. This window contains the next w chunks to be consumed by a client. Observe first that we can safely drop chunks that arrive after their scheduled playback deadline as they are useless. Conversely, downloading chunks located ahead of the current window is not desirable either since it would take away download time from chunks within the window. Hence the best approach is to prevent peers from requesting chunks located outside their current window as this window contains the most urgently needed chunks. While this additional constraint on chunk selection may decrease overall efficiency of the BT protocol, it will also improve the quality of the delivered stream.

A decision to be made by each peer is which chunks within its sliding window it should download first. Adopting a sequential policy would require peers to download first the chunks at the beginning of the window as these chunks will be needed first. This policy would have the major drawback of not taking into account the rarity of the chunks. We decided instead to keep the original rarest-first policy of the BT protocol. As we will see later, we found out that this policy allowed a larger variety of chunks to be downloaded from the seed and delayed the download of most common chunks until the end of the playback delay.

We want the size of the sliding window to correspond to the playback delay d in a way that would allow a peer to use the playback delay to download all the chunks necessary to play back the first d time units of the stream, then use that time to download the chunks necessary to play back the next d time units of the stream and so on. To achieve that goal, the size w of the window, expressed in number of chunks should satisfy the relation

$$w = \frac{db}{c} \quad (1)$$

where d is the playback delay, b is the video consumption rate and c is the chunk size. The assumption here is that the observed download rate at each peer must be at least equal to the streaming rate of the video in order to obtain a good streaming quality.

3.2. Neighbor selection policy

In BT, peers select other peers according to their observed behaviors. Each seed selects its peers based on their observed download rates, that is, the rates at which these peers can download chunks from it. It will always prefer the faster peers in order to speed up the chunk propagation in the BT swarm.

We could thus see a subset of peers with high-speed connections consuming a large portion of the seed's entire upload bandwidth. These advantaged peers will in turn favor each other, effectively denying slower peers a chance to get chunks.

As a result, a significant number of BT peers suffer from slow start. This is because new neighbor discovery is done only by a single optimistic unchoke that occurs at a longer interval. Hence a slower peer in the swarm will have to wait until it has been optimistically selected by an advantaged peer before it can download chunks from that advantaged peer.

To use BT for real-time multimedia streaming it is essential to speed up this bootstrapping process. We use a randomized policy in which at the beginning of every playback each peer selects neighbors at random for the *randomized choking interval* from the list of peers that it got from the tracker. This policy gives more free tries to a larger number of peers in the swarm to download chunks which they can use to share later.

Once they have chunks to exchange peers then use tit-for-tat policy until the end of the playback duration to deter free riding. As more peers become able to actively participate during each playback interval, the efficiency of the system improves and more peers will improve the quality of their streaming.

4. Experimental setup

In this section we discuss some details of our simulation settings. We collected results for a P2P network consisting of one hundred peers. We assumed that each multimedia streaming session consisted of a single initial seed streaming the video to all peers. We further assumed that we had a well behaved BT swarm where each peers would continue uploading until it views the complete video.

We considered a homogeneous setting where all leeches within a sub-network have the same link bandwidth. Unless otherwise specified we use the following default settings in our simulations:

Video size = S = 150 MB
Chunk size = 256 KB
Number of initial-seeds/servers = 1
Link bandwidth = 10 Mbps
Maximum number of concurrent upload transfers = 5
Rechoking interval = 5 s
Optimistic unchoking interval = 15 s
Number of random peers returned by the tracker = 50
Number of neighbors of each peer = 10

Assuming that the original streaming rate b is *3* Mbps, the total duration D of the video is

$$D = \frac{S}{b} = 400\text{s}$$

We modeled the network transmission and queuing delays but assumed that the network-propagation delays could be neglected since they are relevant only for small sized control packets, such as the packets used by peers to requests chunks from their neighbors, while the multimedia streaming time is dominated by the chunk exchange traffic.

To keep our model simple, we ignored the complexity of the dynamics of TCP connections. We assumed the idealized performance of TCP and relied on the long-term fairness of TCP (that connections traversing a link share the link bandwidth equally with the portion of each connection fluctuating as the number of connections vary).

Like previous simulation studies [4, 5] we assumed that bandwidth bottlenecks only occurred at the edge and did not model shared bottleneck links in the interior of the swarm.

We used the following two metrics to measure the performance of BT and the improvements brought by our proposed modifications:

TABLE 1. Success ratio for different chunk selection policies for video consumption rate 4 Mbps and a playback delay of 60 s.

Chunk Selection Policy	*Success Ratio (percent)*
Rarest-first policy (original BT)	23.6%
Sequential policy	8.1%
Sliding window and rarest-first policy	83.3%

- *Success ratio*: This metric represents the playback continuity defined as the number of chunks that arrive before a scheduled playback deadline over the total number of chunks in the video. We use the success ratio to quantify the performance of the P2P multimedia streaming network. The quality is different from throughput and captures performance parameters such as the playback delay, chunk loss and jitter. We use the average of success ratio of all the peers in the network to define the quality of the P2P streaming network.
- *Normalized network throughput*: This metric is the ratio of the total number of bytes uploaded by all the peers up until that time to the network capacity (i.e. total network bandwidth).

5. Experimental results

We performed a series of experiments to validate our proposals to modify the chunk selection policy and the neighbor selection policy.

5.1. Chunk selection policy

We investigated first the impact of the chunk selection policy on the quality of the streaming.

Table I summarizes the impact of the policy on the fraction of chunks that arrive before scheduled playback deadline. As we can see, the original BT protocol can only deliver 23.6 percent of the chunks before that deadline, thus showing that it is not suited to streaming applications. Adopting a sequential policy that downloads first the chunks at the beginning of a sliding window would have a very negative impact on the effectiveness of the policy as only 8.1 percent of the chunks would arrive on time. The best solution is to introduce a sliding window while using a rarest-first policy to select the chunks to download within that window as this solution allows the BT protocol to deliver 83.6 of the chunks on time. While this is not yet enough to ensure a satisfactory streaming quality, it is already 250 percent better than the original BT protocol.

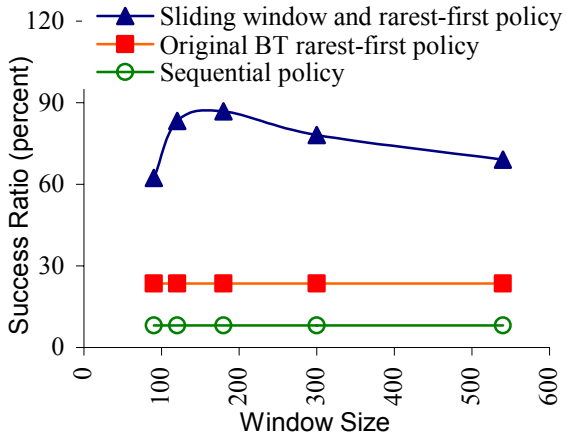

Fig. 2. Effectiveness of the sliding window concept.

Fig. 2 shows how the window size w affects the quality of the streaming. We let the window size vary from *90* to *540* chunks and assume a streaming rate of *4* Mbps and a playback delay of *60* s. Hence the optimal window size w, as obtained from Equation (1), should be 120 chunks. In fact, we can observe that the maximum of the curve occurs for values of w around 150 chunks, that is, 20 percent higher than the reasonable expectation.

When the window size is smaller than that, the success ratio goes down because a smaller window puts undue restrictions on the chunk selection process, thus reducing the effectiveness of the swarm.

As the window size increases, the number of chunks than can be exchanged also increases. Having more chunks per window increases the effectiveness of the process and thus the probability that a given chunk will arrive on time.

Increasing the window size beyond a certain point slowly degrades the streaming quality as peers end up downloading chunks the rarest chunks within their neighborhood while considering less and less the scheduled playback deadline.

5.2. Neighbor selection policy

In this subsection we evaluate our modified neighbor selection policy—which we will call *BT-randomized-tit-for-tat*—and compare it with BT's tit-for-tat policy—which we will refer to as *BT-tit-for-tat*.

Define t as the average video distribution time, that is, the average time required by a peer to obtain the whole content of the object being distributed. We can then use this value to compute the average observed download rate r at which the peers receive the content.

TABLE II. Average video distribution time and average observed download rate using BT.

Average video distribution time t (s)	Average observed download rate r (Mbps)
296.418	4.048

$$r = \frac{S}{t} \quad (2)$$

The value of r for the network plays an important role in the analysis of the streaming quality in each region. Table II displays that value for our simulated network.

Comparing this rate r to the original streaming rate b let us identify three types of regions:

1. *Resource-rich*: This is an over-capacity region when r is greater than the original streaming rate b. Here there is more than enough bandwidth available to stream the video across the network. Higher streaming quality in terms of shorter playback delays and higher success ratio can be achieved in this region.
2. *Resource-critical*: This is when r is approximately equal to the original streaming rate b. Achieving a high streaming quality is difficult in this region.
3. *Resource-starved*: This is an under-capacity region when r is less than the original streaming rate b. In this region the peers are starved for bandwidth resources hence not all peers in the network will be able to achieve an acceptable streaming quality.

Let us consider first how our new neighbor selection policy would affect the streaming performance of the BT protocol in the resource-rich region. Fig. 3 shows how the playback delay affects the streaming qualities achieved by the original BT tit-for-tat policy with our proposed modification in resource-rich regions. We assumed that both policies were combined with our new chunk section policy and calculated the size of the sliding window using Equation (1). In addition, we set the randomized choking period to one-third of the playback interval.

As we can see, the success ratio of our randomized-tit-for-tat policy is 100 percent for all playback delays. This property will hold as long as the streaming rate remains below or around 3Mbps. In contrast, the original BT tit-for-tat policy performs rather unsatisfactorily, especially for the smaller playback delays.

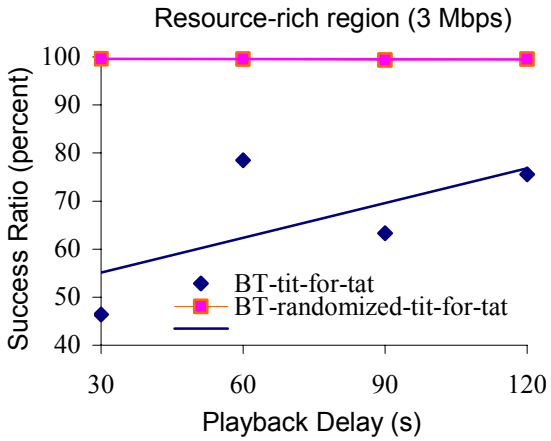

Fig. 3. Comparing BT original tit-for-tat policy with our new randomized-tit-for-tat policy for resource-rich region in terms of success ratio: We have an overall good streaming solution.

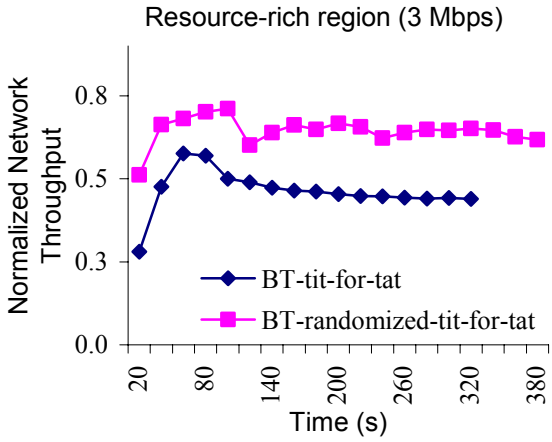

Fig. 4. Comparing BT original tit-for-tat policy with our new randomized-tit-for-tat policy for resource-rich region in terms of normalized network throughput.

This is because in BT the peers selected by the seed are at advantage and have more chunks to upload due to a quick bootstrap. These peers are given a higher priority and suffer less chunk loss. The original BT tit-for-tat policy motivates these peers to pair up among themselves. In contrast, other peers achieve lower download rates and hence worse streaming quality.

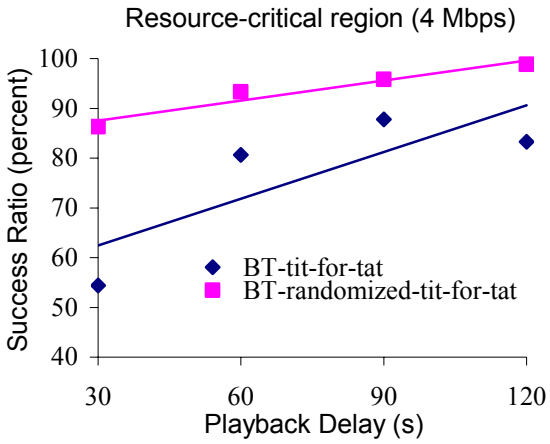

Fig. 5. Comparing BT original tit-for-tat policy with our new randomized-tit-for-tat policy for resource-critical region in terms of success ratio: Larger playback delays lead to a good streaming solution.

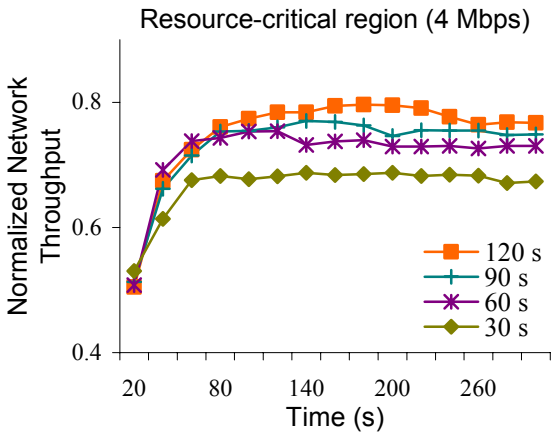

Fig. 6. Compares normalized network throughput for different playback delays.

This phenomenon would indeed worsen in the case of a heterogeneous environment consisting of a mix of peers with different link bandwidths. Here the peers with high bandwidth would pair up among themselves and hence receive higher quality compared to other low bandwidth peers.

In contrast, our policy makes the bootstrapping quicker for all the peers in the system. As soon as these peers have chunks to share, they actively participate in the swarm. The resulting increase in total network utilization can be seen in Fig. 4, which shows how the normalized network throughput evolves over time

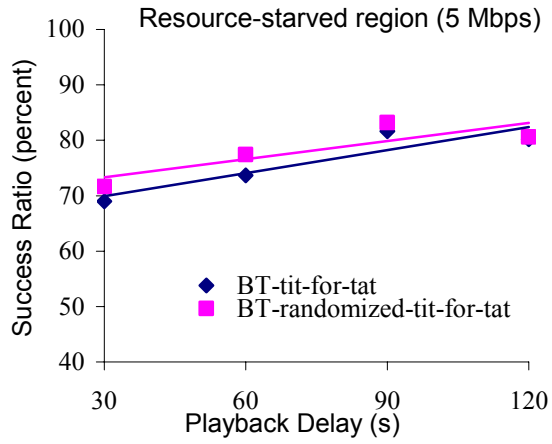

Fig. 7. Comparing BT original tit-for-tat policy with our new randomized-tit-for-tat policy for resource-starved region in terms of success ratio.

assuming a playback delay of 60 s and a streaming rate of 3 Mbps.

Our randomized-tit-for-tat policy still performs much better than the BT tit-for-tat policy in the --- consumption rate is around 4 Mbps. As we can see on Fig. 5, the performances of both protocols improve as the playback delay increases.

This is because BT's efficiency depends on the number of chunks to exchange. As Equation (1) shows, the window size w is proportional to the playback delay d. As a result any increase in that delay will also increase the window size. A larger window will contain more chunks, allow more parallel downloading and utilize the network more efficiently. In particular, Fig. 6 shows how increasing the playback delay from 30 s to 120 s will increase the normalized network throughput by at least 10 percent.

There is little to say about the performance of the two policies in the resource-starved region, that is when the streaming rate is *5* Mbps. As Fig. 7 indicates, neither of the two policies can achieve a satisfactory success rate. Even setting the window size to the video size and the playback delay to the duration of the video would not suffice to achieve an average download rate equal to the video consumption rate.

An admission control policy should be used so that only a number of peers that can be satisfied are allowed to participate in the swarm while the other peers are put on hold. Peers could be selected based on their priority or at random depending on the application needs.

Even in this resource-starved region our BT-randomized-tit-for-tat policy never performs worse and

sometimes performs slightly better than the original BT tit-for-tat policy.

6. Related work

Several studies have explored the P2P multimedia streaming from different aspects. Narada [5] focuses on multi-sender multi-receiver streaming applications and maintains a mesh among the peers and establishes a tree whenever a sender wants to stream the video to a set of receivers. Due to heavy control overhead (because of intensive interactions between peers) Narada does not scale well to large P2P networks. Xu et al. [6] were among the first to propose the concept of P2P multimedia streaming. It mainly focuses on the analysis of the capacity of P2P networks for multimedia streaming.

Nice [7], DirectStream [8] and ZIGZAG [9] try to construct an overlay tree such that they minimize the end-to-end delay and maximize the utilization of the bandwidth of the peers. Banerjee *et al.* [10] the authors suggest that any traditional overlay tree scheme can be made resilient by duplicating chunks along a small number of randomly chosen additional overlay links. This way they utilize all the peers in the network. The major issue of such single overlay tree protocols is to build a scalable overlay tree with high efficiency.

In contrast multi-tree approaches, such as SplitStream [11], CoopNet [12] and P^2Cast [13], advocate the use of multiple distribution trees and assume the presence of uninterested nodes to forward traffic. The multi-tree overlay protocols cannot provide backup streaming services in case of leaving or crashing of the peers in the upper layers of the trees.

To improve the reliability and increase the resource utilization mesh-based P2P protocols have been proposed [14-16]. Gnustream [14] is built upon the Gnutella [17] P2P content-distribution protocol. Unlike most P2P protocols such as Napster [18], Gnutella, Kazaa [19] and eDonkey [20], BT focuses on fast and efficient replication to distribute files. Most traditional protocols were more specifically designed to share MP3 or image files a few mega-bytes in size where the search time is more crucial than the distribution time.

Unlike our tracker based solution, CoolStreaming [15] has each peer periodically exchange the availability information of the media stream with different neighbors. The construction and maintenance of the swarms requires much higher overhead which may result in peers experiencing longer playback delays. In addition these solutions [14-16] use very different internal policies to deal with the real-time requirements of multimedia streaming.

There have been several analyzes of performance of BT for content-distribution. Previous work on BT has focused on measurements and theoretical analysis. Wu and Chiueh [21] have suggested that BT-style mechanisms may be adapted for streaming purposes.

Vlavianos et al. have presented BiToS, a BT-based protocol that can support P2P streaming [22]. These authors claim that the chunk selection policy is the only feature of the original BT that has to be modified in order to support streaming applications. In contrast to them, we believe that modifications to the chunk selection policy together with the neighbor selection policy are both required to achieve that goal.

Furthermore, in the chunk selection process the peers in BiToS choose with some probability p to download a chunk from the current *high-priority set* or from the *remaining set*. Thus a cumbersome process of dynamically adapting the value of p as well as the size of *high-priority set* is required in BiToS to adjust to different flash-crowd scenarios.

In contrast, we believe in large BT swarms periodically selecting neighbors randomly for a short duration is a more feasible approach to make sure that peers would have enough chunks to exchange and consequently not be choked by other peers. Using tit-for-tat neighbor selection policy for the remaining duration ensures fairness and deters free riding.

7. Conclusion

Despite its numerous advantages, the BitTorrent protocol is poorly suited to multimedia streaming applications. We have presented two modifications correcting this limitation. First, we replace the rarest-first chunk downloading policy of BitTorrent by a policy preventing peers from requesting chunks located outside of a sliding window containing the chunks that they will watch in the near future. Second, we introduce a new randomized tit-for-tat peer selection policy that gives free tries to a larger number of peers and lets them participate sooner in the media distribution. Our simulations indicate that both improvements significantly improve the number of chunks that streaming clients will receive on time. Combining them ensures the on-time delivery of more than 90 percent of the chunks as long as the average peer download rate remains greater than or equal to the multimedia consumption rate.

Future work includes developing a control mechanism capable of striking a balance between the

playback delay and Quality of Service (QoS) requirements.

More work is also needed to improve the applicability of our solution for video-on-demand applications because in Internet P2P networks, peers are free to join at any time. A P2P multimedia streaming network therefore must incorporate this peer dynamicity and employ an admission control mechanism to ensure that the resources required by a new request do not affect the QoS requirements of streams already being serviced.

References

[1] *http://www.internetnews.com/dev-news/article.php/3321911.*

[2] *http://news.bbc.co.uk/1/hi/business /4753435.stm.*

[3] B. Cohen, "Incentives build robustness in BitTorrent," In Proc. of *First Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[4] A. Bharambe, C. Herley and V. Padmanabhan, "Analyzing and improving a BitTorrent network's performance mechanisms," In Proc. of *25th IEEE INFOCOM Conference,* Barcelona, Spain, Apr. 2006.

[5] Y. Chu, S. Rao and H. Zhang, "A case for end system multicast," *Proc. ACM SIGMETRICS Conference,* Santa Clara, CA, June 2000.

[6] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," *Proc. 22nd International Conference on Distributed Computing Systems* (ICDCS 2003), Wien, Austria, July 2002.

[7] S. Banerjee, B. Bhattacharjee and C. Kommareddy, "Scalable application layer multicast," *Proc. ACM SIGCOMM 2002 Conference* , Pittsburgh, PA, Aug. 2002.

[8] Y. Guo, K. Suh, J. Kurose and D. Towsley "A Peer-to-Peer on-demand streaming service and its performance evaluation," *Proc. 2003 IEEE International Conference on Multimedia & Expo* (ICME 2003*)*, Baltimore, MD, July 2003.

[9] D. Tran, K. Hua and T. Do, "Zigzag: an efficient peer-to-peer scheme for media streaming," *Proc. 22nd IEEE INFOCOM Conference*, San Francisco, CA, 2003.

[10] S. Banerjee, S. Lee, B. Bhattacharjee and A. Srinivasan, "Resilient multicast using overlays," *Proc. 2003 ACM SIGMETRICS Conference*, San Diego, CA, June 2003.

[11] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: high-bandwidth multicast in a cooperative environment," *Proc. 19th ACM Symposium on Operating Systems Principles* (SSOP 2003), Bolton Landing, NY, Oct. 2003.

[12] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," *Proc. ACM NOSSDAV*, Miami Beach, FL, May 2002.

[13] Y. Guo, K. Suh, J. Kurose and D. Towsley, "P2Cast: Peer-to-peer Patching Scheme for VoD Service," *Proc. World Wide Web Conference (WWW)*, Budapest, Hungary, May, 2003.

[14] X. Jiang, Y. Dong, D. Xu, and B. Bhargava, "Gnustream: a p2p media streaming prototype," *Proc. of 2003 IEEE International Conference on Multimedia & Expo* (ICME 2003*)*, Baltimore, MD, July 2003.

[15] X. Zhang, J. Liu, and B. Li, "On large-scale peer-to-peer live video distribution: CoolStreaming and its preliminary experimental results," *Proc. of IEEE Multimedia Signal Processing Workshop (MMSP 2005)*, Shanghai, China, Oct. 2005.

[16] J. Li, "PeerStreaming: a practical receiver-driven peer-to-peer media streaming system," MSR-TR-2004-101, Sept. 2004.

[17] *http://www.gnutella.com*.

[18] *http://www.napster.com.*

[19] *http://www.kazaa.com.*

[20] *http://www.eDonkey.com*

[21] G. Wu and T. Chiueh, "How efficient is BitTorrent?" *Proc. of 2006 SPIE Multimedia Computing and Networking Conference* (MMCN 2006), San Jose, CA, 2006.

[22] A. Vlavianos, M. Iliofotou and M. Faloutsos, "BiToS: enhancing BitTorrent for supporting streaming applications," *Proc. 9th IEEE Global Internet Symposium*, Barcelona, Spain, Apr. 2006.