

BitTube: Case Study of a Web-based Peer-Assisted Video-on-Demand System

Bo Liu

Yi Cui

Bin Chang

Ben Gotow

Yuan Xue*

Abstract

Recent theoretical and simulation-based studies have confirmed the tremendous benefit of peer-to-peer (P2P) communication at reducing the cost of running a VoD service. To date, very limited effort has been paid to validate the concept of peer-assisted VoD service, especially in terms of system implementation and service deployment. In this paper, we present the case study of a peer-assisted video-on-demand (VoD) system. We designed and developed BitTube, a BitTorrent-compliant VoD system. By combining client/server and P2P downloading, it supports seamless transition across the spectrum from pure client-server mode to BitTorrent mode. Within this framework, we experiment with a series of piece picking policies to enhance BitTube's support to video streaming and promote locality-aware P2P downloading. We evaluate our system over PlanetLab, which hosts the user-side component of the BitTube system and emulates the global-scale user requests to the VoD service.

1 Introduction

Peer-to-peer (P2P) communication has been proven to be an extremely powerful paradigm to a diverse family of Internet applications. A few examples include bulk content distribution [1], voice over IP [5], and broadcasting of TV-quality program [4, 6], all of which have been proved by the commercial deployment of planet-scale systems serving tens of millions of users.

A new member joining this family is the video-on-demand (VoD) application. Through theoretical analysis [23, 21] and trace-driven simulation study [13, 19], recent research results have confirmed the tremendous benefit of P2P at reducing the cost of running a VoD service. An

important insight gained through these studies is that P2P is unable to replace the conventional client-server mode. Instead, a more appropriate role for P2P is to *assist* the delivery of VoD service.

To date, very limited effort has been paid to validate the concept of peer-assisted VoD service, especially in terms of system implementation and service deployment. In this paper, we argue the necessity for such a system study: (1) real system deployment and field test could provide an realistic estimation of the server cost reduction and how it will be affected by the user request patterns. (2) Besides server cost, which is the central focus of the previous studies, the field study based on a functioning system could provide us an opportunity to examine several other important factors of a VoD service, such as streaming experience at each peer and load imposed on the Internet.

In this paper, we conduct a systematic field study of a peer-assisted VoD service *BitTube*, which is designed and developed by our research group at Vanderbilt University. BitTube is a BitTorrent-compliant VoD system. By combining client/server (e.g., HTTP) and P2P downloading, it supports seamless transition across the spectrum from pure client-server mode to P2P mode. The target application of BitTube is the VoD system serving user-generated content (UGC), represented by YouTube. The rationale of this choice is two-fold. First, recent study [19] shows the ephemeral and polarizing popularity of the video content in such applications, which naturally aligns with the hybrid nature of peer-assisted VoD. Second, as the popularity of UGC services rapidly grows, they are forced to keep up by expanding the system capacity (e.g., the alleged one million dollar bandwidth cost for YouTube [8]), which provides strong economic justification to our research. In order to fully evaluate BitTube, we integrate BitTube into VandyVideo [7], a web-based UGC service developed and hosted at Vanderbilt University. We evaluate our system over PlanetLab, which hosts the user-side component of the BitTube system and emulates the global-scale user requests to the UGC VoD service.

We claim the following contributions. First, given the existing studies on BitTorrent-compliant streaming systems [9, 11, 22, 2], this paper is the first work which designs and implements a system which integrates P2P downloading

*Bo Liu is from the School of Computer Science and Technology at Huazhong University of Science and Technology, his email address is newpoo@smail.hust.edu.cn; Yi Cui, Ben Gotow, and Yuan Xue are from the Department of Electrical Engineering and Computer Science at Vanderbilt University, their email addresses are {yi.cui,ben.gotow,yuan.xue}@vanderbilt.edu; Bin Chang is from YouTube Inc., his email address is binchang@google.com.

with the web-based VoD service. Our system is intentionally designed to bring minimum disruption to the existing service both in terms of web infrastructure and user viewing experience. Second, we propose a series of novel piece picking policies with built-in locality awareness. Finally, by observing peers' downloading behaviors at the piece-level detail, we study the relationship of several design objectives and derive some key insights, such as the conflict between achieving "viewing-while-downloading" and fairness in peer contribution.

The rest of this paper is organized as follows. First, we discuss the related work in Section 2 to prepare background information to the introduction of BitTube. Section 3 gives the architectural overview of the BitTube system. Section 4 presents a series of new piece picking policies to enhance BitTube's performance at supporting peer-assisted VoD. We present our system experiment results¹ in Section 5 and conclude the paper in Section 6.

2 Related Work

Given the rich literatures on streaming system study and user behavior analysis, only a few have started examining P2P as the candidate solution to VoD. The work by Huang et al.[13] is the first comprehensive study targeted on this topic. By analyzing video traces from MSN VoD service, it delivers several positive messages in favor of peer-assisted solutions. Cha et al. [19] studies traces collected from YouTube and other UGC services, and reveals a series of unique characteristics about this special type of VoD system. In particular, it shows that although the popularity of the content differs dramatically, few of the most popular still leave room for tremendous server load reduction using P2P, e.g., more than 90% saving if peers cache the content for 24 hours. Another study on VoD is by Yu et al.[16], which presents an in-depth analysis of user access patterns in a centralized VoD system.

An extended set of works have proposed peer-assisted solutions to address the problem of asynchronous user request, which is unique to VoD. oStream[23] utilizes application-layer multicast and peer buffering to support VoD. A cache-and-relay architecture is proposed in [18]. In [15], Xu et al. perform a theoretical analysis of a hybrid CDN-P2P system. While above proposals rely on caching at peer side, some works seek support via advanced error-correction coding schemes. Li et al. propose PeerStreaming[17], a P2P streaming system utilizing erasure resilient coding. By the same token, rStream[14] utilizes rateless coding for resilient P2P streaming. Network coding is employed by a few systems, such as Avalanche[12], to support content distribution network.

¹Due to space constraint, we only report part of our results. The full presentation can be found in our technical report[10].

The success of BitTorrent has highly influenced the design of following P2P systems including streaming systems[4, 6]. Many commonly-employed techniques, such as receiver-driven streaming and the mesh-based downloading structure resonate with the BitTorrent design. Given BitTorrent's status as the de facto P2P downloading protocol and a high-quality open-source software, there have been several proposals[20, 11, 9, 22] to modify BitTorrent's non-sequential downloading mechanism to support the "viewing-while-downloading" feature. The basic idea of these works is to restrain the piece picking action within a moving window along with the playback, which is also adopted by our system. These previous works primarily focus on server load reduction by the aid of P2P downloading, and proved that significant saving is achievable via simulation and system deployment. In our work, we also explore other performance-enhancement dimensions such as locality-aware P2P downloading. Perhaps the closest solution to ours is BitTorrent DNA[2], a content distribution solution recently promoted by BitTorrent Inc., which claims to support "viewing-while-downloading" as well. However, its technical details and whether it cooperates with client-server downloading methods such as HTTP, remain unknown at the moment of writing.

3 BitTube System

The design of BitTube centers around the principle that the impact brought to the existing VoD infrastructure should be controlled to its minimum. First, the web-based platform of current UGC service should stay largely uninterrupted, since any change to its massive video archive pages will undermine the likelihood of adopting peer-assisted solutions. Second, we favor the P2P technique which is open and commonly accepted, to encourage participation. This promotes our choice of BitTorrent, the de facto standard in P2P downloading. Third, with the addition of P2P component, BitTube retains the easy usability of current UGC service, where a user can view the video with a single click.

In what follows, we first present the architecture and interface design of BitTube, followed by the introduction of an integrative scheme of HTTP and P2P downloading to achieve our goal to build a peer-assisted VoD service. Finally, we discuss the possible extensions of BitTube.

3.1 System Architecture and Interface

Fig. 1 shows the basic architecture of our system, which can be divided into server side and peer side.

On the server side, there are two entities: VoD server and BitTorrent tracker. In the context of UGC service, the VoD server is a web server, which acts as the information portal, from which users can browse, submit and search

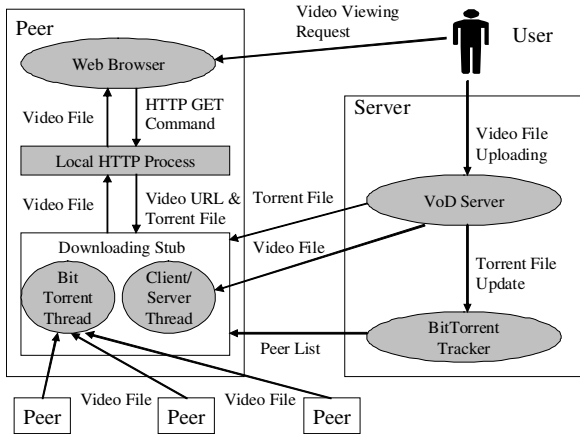


Figure 1. BitTube Architecture

video files. The VoD server manages all torrent files. Each time a new video file is uploaded to the VoD server, it creates the corresponding torrent file. On the peer side, each peer machine runs a P2P downloading stub program, which is a BitTorrent-compliant module responsible to coordinate with the BitTorrent tracker and organize the downloading of video content from other peers. In addition, it supports traditional client-server downloading from the VoD server via HTTP protocol, which is automatically triggered when there are not enough peers to support the video streaming.

In traditional web-based UGC service such as YouTube, user's viewing request is initiated as a HTTP GET command from the web browser to the VoD server. To redirect this request to the P2P downloading stub, we introduce a local HTTP process on each peer machine. Listening at a local port, it intercepts the HTTP GET command, downloads the torrent file of the requested video, then hands it to the downloading stub, which contains a BitTorrent thread and a client/server thread. Consequently, the BitTorrent thread contacts the BitTorrent tracker and starts downloading. The client/server thread downloads from the VoD server. The coordination of these two threads will be discussed in Sec. 3.2. Finally, the downloading stub returns the downloaded content to the local HTTP process, which further delivers it to the web browser by replying to the HTTP GET command it intercepts previously.

BitTube design introduces minimum disruption to both the existing architecture of web-based UGC service and its users' viewing experience.

From the perspective of the UGC service provider, with the introduction of BitTube, its massive web infrastructure and archival content remain intact. As shown in Fig. 2, the only change needed is to modify the embedded URL of its video files to 127.0.0.1, the address of the local machine².

²Despite this modification, the local HTTP process still needs to provide the downloading stub two pieces of information: (1) the original URL

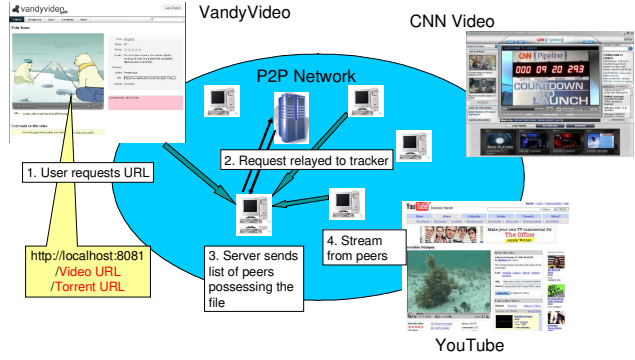


Figure 2. BitTube Service Federation

This design also enables BitTube to form a service federation, where multiple UGC services can join the same P2P network managed by BitTube.

From the perspective of users, for any UGC service supported by BitTube, its interface is identical to a normal web-based UGC service. This is clearly shown in Fig. 3, the screen shot of a proof-of-concept UGC service we developed to prototype BitTube. In addition, a user needs to run on his/her own machine a program including the local HTTP process and the downloading stub, which we introduced in Fig. 1.

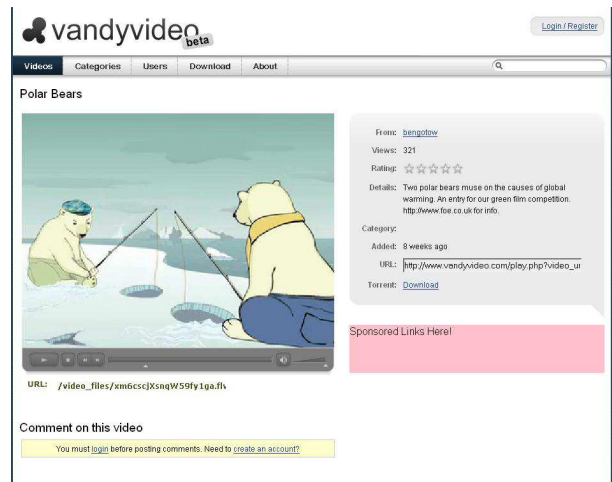


Figure 3. Screen Shot of VandyVideo

3.2 Peer-assisted VoD: Integrating Client-Server and P2P Downloading

So far, our discussion builds on a hidden assumption that there must be at least one seed for each video file, of the video file, and (2) the URL of the corresponding torrent file. In our current implementation, we embed them into the HTTP GET command as parameter strings.

which is neither necessary nor realistic. The video popularity of UGC service follows heavy-tail distribution, where majority of the video files remain unpopular and dormant, denying the remote chance that an online peer possessing such a file would exist. Besides bookkeeping concerns, the stringent performance requirement of VoD applications demands the downloading to be no slower than the playback rate of the media. These challenging issues motivate us to design a scheme offering the seamless integration of client-server and P2P downloading, which offers smooth transition across the entire spectrum from client-server mode to P2P mode.

The integrated downloading scheme of BitTube functions as follows. As illustrated in Fig. 1, we maintain two threads, one for BitTorrent downloading and the other for HTTP downloading. Both threads are allowed to request pieces from the current playback window³. However, BitTorrent thread enjoys higher priority, in that HTTP thread only requests pieces when the downloading speed of the BitTorrent thread is insufficient. The HTTP thread achieves this by periodically checking the download speed of the BitTorrent, then requesting from the VoD server the unrequested pieces, whose total number is determined by the video streaming rate and number of pieces already requested by BitTorrent. If the downloading speed of BitTorrent exceeds the streaming rate, the client/server thread will remain idle.

We note that the streaming rate should be learned before the downloading initiates. The streaming rate, as a basic metadata field, can be found in most, if not all, streaming file formats.

4 BitTube Piece Picking Policies

In the previous section, we presented the architecture and functionalities of BitTube system. Our discussion now focuses on the key performance enhancement technique we develop in BitTube to make it serve the purpose of peer-assisted VoD.

In BitTorrent-compliant systems, a file is split into pieces, each of which is requested and distributed among peers. Therefore, piece picking policy, which decides which piece to download from which peer, is crucially important to decide the pattern of the P2P downloading. In what follows, we first review the piece picking policy of the original BitTorrent system, then discuss a series of new piece picking policies we introduce in BitTube system.

³For HTTP thread, the HTTP/1.1 content-range entity-header feature enables it to request a particular piece by specifying the name of the file, and the starting and ending byte of the piece.

4.1 Overview

In BitTorrent, peers interested in the same file form a *swarm*. In a swarm, each peer sends other peers the *have* messages, which advertise the availability information of the pieces it already owns. Based on the *have* message collected from others, each peer maintains an array of *interest* values, where each entry is the number of peers owning the corresponding piece.

During its downloading process, a peer maintains connections with a set of other peers in the same swarm. In each connection, the peer collects, one by one, new pieces from the peer at the other side, until their collections become the same. During this process, the piece picker policy plays an important role: to decide the sequence of the downloading, i.e., which piece to download first.

Traditional BitTorrent system employs the rarest-first policy, in which the piece with the minimum interest value is chosen. If multiple pieces have the same minimum interest value, such as 1, then the tie is broken by randomly choosing one. The greatest benefit of this policy is that it helps promote the piece diversity of the entire swarm by help distributing the rare pieces. A diverse swarm can ensure concurrent downloading over multiple connections, thus increasing the aggregated downloading speed.

However, the rarest-first policy ignores the locality issue on both the temporal and geographical dimensions. On the temporal dimension, since the pieces are downloaded regardless their position in the video playback, the “viewing-while-downloading” feature cannot be supported. In the worst case, the first piece might not be downloaded until all other pieces are, which makes the video unable to play until the whole file is fetched. On the geographical dimension, the distance between peers is not considered by the policy, which often ends up downloading from far-away peers when we can download the same piece from a nearby peer. This can introduce unnecessary strain on the network, or inter-ISP traffic.

4.2 Window-based Piece Picking Policies

In light of above concerns, we propose a series of new piece picking policies to address them. As shown in Fig. 4, a common feature is that they all constrain the piece picking action within a window synchronized with the video playback. We now discuss them one by one.

4.2.1 Rarest-first Policy

As shown in Fig. 4 (a), this policy behaves the same as BitTorrent’s rarest-first policy, only restrained within the playback window. We note that it is the client/server thread (presented in Sec. 3), not the BitTorrent thread, controlling the advancing speed of the playback window. In the

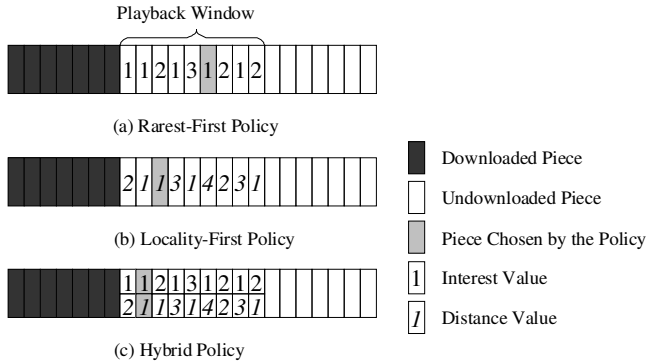


Figure 4. Window-based Piece Picking Policies

BitTorrent thread, the window is automatically pushed forward whenever its leftmost piece is downloaded. Therefore, the window might advance ahead of the playback if the BitTorrent downloading speed is faster than the playback rate, or behind the playback otherwise. In the latter case, the client/server thread will keep pushing the window by downloading the leftmost pieces until it catches up with the playback again. In this way, we keep the BitTorrent thread generic by making it unaware of many video-specific factors such as playback rate. Instead, the time-keeping job is shipped to the client/server thread.

Finally, we note that the name of this policy can bring great confusion with BitTorrent’s rarest-first policy. To avoid such confusion, in the remaining part of this paper, we will refer to the latter simply as BitTorrent policy.

4.2.2 Locality-first Policy

This policy is proposed to encourage a peer to download from its nearby neighbors, if possible. As shown in Fig. 4 (b), we introduce a *distance* value to each piece, which is the mean value of the peer-to-peer distances of all peers possessing this piece. For example, if a piece is owned by three peers (i.e., its *interest* value is 3), and the distances from these peers to the downloading peer are 1, 2, 3, respectively, then the *distance* value associated with this piece should be 2. The locality-first policy chooses the piece with the smallest *distance* value.

The peer-to-peer distance could be represented in many ways, such as hop count, or delay in certain time unit such as seconds. It could also be any artificial metric defined to promote ISP-friendliness. For example, we can define the distance between two peers as 1 if they belong to the same ISP, 2 if they are located within the same city or state, 3 if within the same country or continent, and 4 otherwise.

The distance between two peers seldomly changes when it is based on geographic location. Therefore, an existing

peer can quickly accumulate such knowledge as it contacts more peers. However, a newly-joined peer needs to learn its distance to other peers as early as possible in order for the locality-first policy to function. While a peer can learn on its own the distance to other peers by probing them, the P2P tracker is an ideal identity to collect and disseminate such information. Already a publish-subscribe service to help peers within the same swarm to know each other, the tracker can distribute the peer-to-peer distance values in an add-on field to the peer list it returns to each inquiring peer. The tracker can learn the distance values through any commercial IP-to-location package[3].

4.2.3 Hybrid Policy

This policy combines the purposes of the rarest-first and locality-first policies. As shown in Fig. 4 (c), it gives priority to the rarest-first policy to first come down to a short list of rarest pieces, i.e., those with the smallest *interest* values. If the list contains only one piece, then it is selected. Otherwise, the tie is broken by choosing the piece with the smallest *distance* value.

This policy is expected to perform well when there exists considerable homogeneous piece rarity in the swarm. During the initial stage, when the *interest* value of the rarest piece is 1, it falls back to the rarest-first policy since it has no choice but choosing to download from the peer possessing it. When the rarest piece is already owned by multiple peers, the locality-first policy will start to take effect by choosing the one with the smallest *distance* value.

4.3 Discussions

We compare the locality-first and hybrid policies with two other policies, BitTorrent policy which uses the original rarest-first policy without window constraint, and the sequential policy which strictly downloads pieces in an orderly fashion. In other words, these two policies represent the two extreme cases of the window-based policy, i.e., when the window size is expanded to the whole file, and when it is reduced to one piece.

We note that when the window size is reduced to one, all previously-mentioned window-based policies will behave the same way. This is because, as discussed in Sec. 4, the piece picking policy is executed on a per-connection basis, i.e., the choosing of the best piece is restrained within the only set of pieces owned by the peer at the other end of the connection, instead of all peers in the swarm. Therefore, when theoretically one can determine not only the best piece to download, but also the best peer (such as the nearest one) to download it from, such global optimality is replaced by the fact that each connection races to grab the best piece. BitTorrent further endorses such racing behavior by requir-

ing other connections to give way to the connection which claims a certain piece first.

The fundamental reason for BitTorrent’s choice is to keep each connection fully occupied. If we hold one connection idle to wait for another connection to claim certain piece, valuable bandwidth would be wasted. In other words, the need to fully utilize bandwidth overrides the need for optimal peer selection. We keep this feature intact since fully utilizing P2P downloading capacity is the top priority of BitTube.

5 Performance Evaluation

5.1 Experiment Setup

We evaluate our system over the PlanetLab testbed. We deploy the BitTube system over 396 PlanetLab nodes. We use two PC machines at Vanderbilt University to be the VoD server and BitTorrent tracker. Both machines run Red-Hat Linux, one with a Intel Xeon(TM) 2.80GHz CPU (VoD server) and the other with a Intel Pentium 4 2.80GHz CPU (tracker server).

The test file is a flash video file downloaded from a web-based UGC service, which lasts 28 minutes 28 seconds and is sized 61889761 bytes. For the purpose of simplicity, we determine its streaming rate to be the ratio of its size over playback length. The request rate is 1.5 requests per second. We schedule each PlanetLab node to initiate a request based on this speed. All runs of our experiment follows the same request sequence. Our experiment has 8 runs, 1 for BitTorrent, 1 for sequential, and 2 for each of the three window-based policies. For each policy, we try the window size of 20 pieces and 60 pieces, which translate to time lengths of 2 minutes 25 seconds and 7 minutes 15 seconds. you have ,

Our locality-first and hybrid policies require the distance values between each pair of peers. We define the distance between two peers as 1 if they belong to the same ISP, 2 if they are located within the same city or state, 3 if within the same country or continent, and 4 otherwise. The ISP and location information of all PlanetLab nodes are obtained from a commercial package[3].

In each run of the experiment, each PlanetLab node streams the file from beginning to end. The experiment ends when all peers have the file. In all experiments, the VoD server is the only node with the entire file at the beginning. Due to various reasons such as machine failure or testbed administration, only 188 PlanetLab nodes have successfully participated all 8 runs of our experiment. Therefore, we only analyze and exhibit the results obtained from these 188 nodes.

5.2 Results

In Fig. 5, we show the load experienced at the VoD server along the time. The picture shows that the hybrid and rarest-first policies give the best performance, which consistently constrain the server load to be within 250000 bytes/second. Locality-first policy has the worst performance, followed by BitTorrent policy. The difference between these two groups of policies is that the first group (hybrid and rarest-first) promotes piece diversity within the playback window. BitTorrent policy ignores the real-time streaming requirement, thus constantly activate the client/server thread to download from the server. Locality-first policy, although restrained within the window, does not try to maximize the P2P downloading, which also results in constant client/server downloading to keep up with the playback. We summarize the server load reduction in Tab. 5.2.

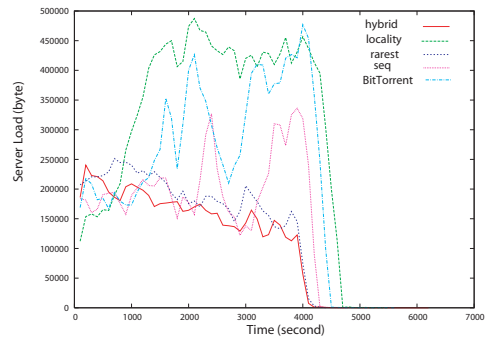


Figure 5. Server Load Along Time

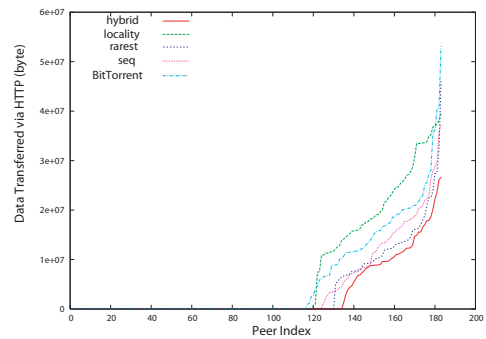


Figure 6. Server Load Imposed per Peer

Fig. 6 reveals the load that each peer imposes on the VoD server⁴. For all policies, around 120 out of 188 peers are able to support themselves through P2P downloading. Also the performance of each individual policy is consistent with its performance in Fig. 5, i.e., the more peers it is to support

⁴In this and all following figures, the curves are sorted independently for different piece picking policies

BitTorrent	Sequential	Rarest(w=20)	Rarest(w=60)	Locality(w=20)	Locality(w=60)	Hybrid(w=20)	Hybrid(w=60)
91.8%	93.9%	94.5%	94.4%	89.6%	95.3%	95.6%	94.8%

Table 1. Server Load Reduction

entirely through P2P downloading, the less server load it incurs.

We now evaluate the impact of different streaming schemes on user experience, which is measured as the *aggregated interruption time*. We define this metric as follows. For each peer, during its streaming, we monitor the pieces received along the playback time. During the playback, if any piece is missing, the peer enters the “interruption” stage where the playback is starved. It will exit this stage when all missing pieces are received. The aggregated interruption time is the summation of time lengths of all interruptions experienced by the peer. This monitoring procedure starts from the point where the first piece is received. In Fig. 7, the advantage of the client/server downloading thread clearly shows. For all policies, only less than 20 peers experience interruptions. Although the download volume of the client/server thread is quite diverse depending on the piece picking policy (shown in Fig. 5), Fig. 7 shows that such difference is largely masked to the end users.

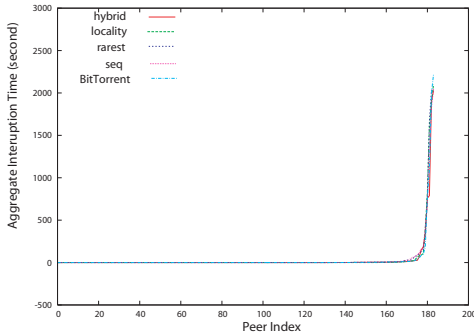


Figure 7. Aggregated Interruption Length

Next, we examine the amount of data contributed by each peer. Fig. 8 shows the result. We have the following common observations. First, all peers are able to contribute. Second, the contribution is highly asymmetric. Hybrid and rarest-first policies, the two best policies in terms of server load reduction, have the most skewed curve.

We then examine the impact of our streaming solutions on Internet traffic. We propose *weighted hop counts*, which is the weighted average hop count across all downloading paths of a peer. Specifically, it is the ratio between the aggregated hop counts experienced by all its downloaded pieces and the number of downloaded pieces. In Fig. 9, we find out that the hybrid policy has the biggest weighted hop count, whereas the locality-first policy has the smallest weighted hop count due to its design purpose. These two

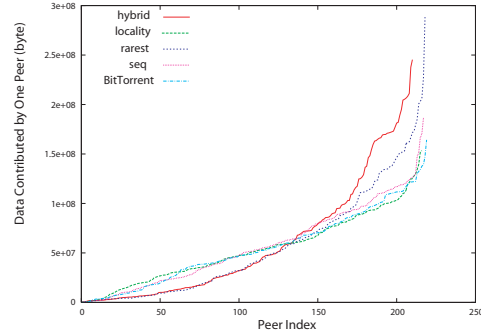


Figure 8. Peer Contribution

policies happen to achieve respectively the best and worst server load reduction in Fig. 5. Such an inverse correlation apply for other policies as well.

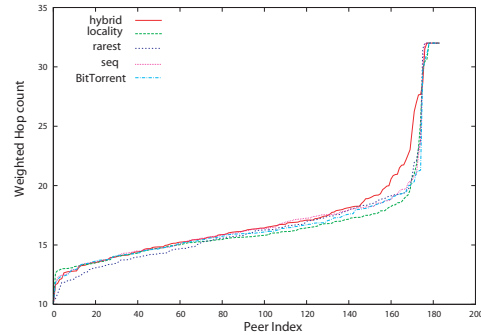


Figure 9. Weighted Hop Counts

Finally, we study the diversity of traffic exchange among peers. In Fig. 10, we plot the traffic difference between each pair of peers, which is defined as follows. If peer A sends peer B 10 bytes, and B sends A 4 bytes, then the traffic difference between peer A and B is 6 bytes. Here, the policies with the least number of peer pairs showing traffic difference, namely hybrid and rarest-first policies, actually have the highest aggregate volume of traffic difference, which is consistent with their highly-skewed peer contribution curves observed in Fig. 8.

5.3 Summary

Besides the significant server load reduction achieved by all policies, our experiment shows the following observation consistent in all runs. A piece picking policy able

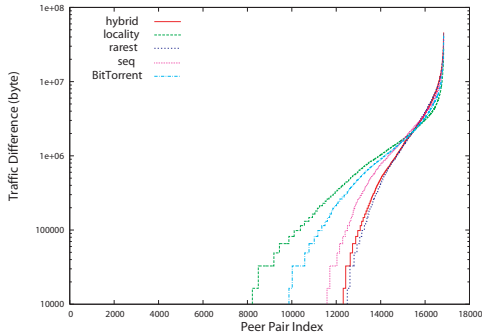


Figure 10. Traffic Exchange Difference per Peer Pair in Log Scale

to achieve more server load reduction exhibits more uneven peer contribution, higher weighted and normalized hop count, and higher aggregated volume of traffic difference. Collectively, these evidences suggest that in the VoD setting, the favored policy, in terms of server load reduction, is likely to promote uneven peer contribution, leave little improvement room for locality-awareness, and discourage traffic exchange among peers. A data distribution structure with matching features is a tree, or a mesh with clear direction of data flow from the source to leaf nodes.

6 Conclusions

In this paper, we present BitTube, a BitTorrent-compliant VoD streaming system. To our knowledge, this is the first work which designs and implements a system which integrates P2P downloading with the web-based VoD service. We propose a series of novel piece picking policies. Through PlanetLab evaluation, we derive a few key insights, such as the potential conflict between server load reduction and even peer contribution. We conjecture that this phenomenon might be shaped by the temporal sequence of peer requests and the real-time streaming requirement, all of which are salient features of VoD applications. Investigation of this conjecture is a topic of future study. It will help us gain a deeper understanding on the relationship between different P2P paradigms such as BitTorrent downloading and P2P streaming, which eventually leads to better design of peer-assisted VoD solutions.

References

[1] BitTorrent. <http://bittorrent.com>.
 [2] BitTorrent DNA. <http://www.bittorrent.com/dna/>.
 [3] IP2Location. <http://www.ip2location.com>.
 [4] PPLive. <http://pplive.com>.
 [5] Skype. <http://skype.com>.

[6] UUSEE. <http://uusee.com>.
 [7] VandyVideo. <http://www.vandyvideo.com/>.
 [8] YouTube. <http://youtube.com>.
 [9] A. Vlavianos and M. Iliofotou and M. Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. In *IEEE INFOCOM*, 2006.
 [10] B. Liu and Y. Cui and B. Chang and B. Gotow and Yuan Xue. BitTube: Case Study of a Web-based Peer-Assisted Video-on-Demand System. In <http://vanets.vuse.vanderbilt.edu/ism2008-report.pdf>, 2008.
 [11] C. Dana and D. Li and D. Harrison and C.N. Chuah. BASS: BitTorrent Assisted Streaming System for Video-on-Demand. In *IEEE MMSP*, 2005.
 [12] C. Gkantsidis and J. Miller and P. Rodriguez. Anatomy of a P2P Content Distribution System with Network Coding. In *IPTPS*, 2006.
 [13] C. Huang and J. Li and K. Ross. Can Internet Video-on-Demand Be Profitable? In *ACM SIGCOMM*, 2007.
 [14] C. Wu and Baochun Li. rStream: Resilient and Optimal Peer-to-Peer Streaming with Rateless Codes. *IEEE Transactions on Parallel and Distributed Systems*, 2007.
 [15] D. Xu and S. Kulkarni and C. Rosenberg and H.-K. Chai. Analysis of a CDN-P2P Hybrid Architecture for Cost-Effective Streaming Distribution. *ACM/Springer Multimedia Systems Journal*, 11(4), 2006.
 [16] H. Yu and D. Zheng and B. Zhao and W. Zheng. Understanding user behavior in large scale video-on-demand systems. In *EuroSys*, 2006.
 [17] J. Li and Y. Cui and B. Chang. PeerStreaming: Design and Implementation of an On-Demand Distributed Streaming System with DRM Capabilities. *ACM/Springer Multimedia Systems Journal*, 2007.
 [18] S. Jin and A. Bestavros. Cache-and-Relay Streaming Media Delivery for Asynchronous Clients. In *Proc. of International Workshop on Networked Group Communication (NGC)*, 2002.
 [19] 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. *ACM IMC*, 2007.
 [20] P. Shah and J. Paris. Peer-to-Peer Multimedia Streaming Using BitTorrent. In *IEEE IPCCC*, 2007.
 [21] S. Tewari and L. Kleinrock. Analytical Model for BitTorrent-based Live Video Streaming. In *IEEE NIME Workshop*, 2007.
 [22] Y. Choe and D. Schuff and J. Dyaberi and V. Pai. Improving VoD Server Efficiency with BitTorrent. In *ACM Multimedia*, 2007.
 [23] Y. Cui, B. Li and K. Nahrstedt. oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks. *IEEE Journal on Selected Areas of Communications, Special Issue on Recent Advances in Service Overlay Networks*, 22, 2004.