An Interest-based Per-Community P2P Hierarchical Structure for Short Video Sharing in the YouTube Social Network

Haiying Shen, Yuhua Lin and Harrison Chandler
Department of Electrical and Computer Engineering
Clemson University
Clemson, South Carolina 29634
Email: {shenh, yuhual, hchandl}@clemson.edu

***Abstract*—The past few years have seen an explosion in the popularity of online short-video sharing in YouTube. As the number of users continued to grow, the bandwidth required to maintain acceptable quality of service (QoS) has greatly increased. Peer-to-peer (P2P) architectures have shown promise in reducing the bandwidth costs; however, the previous works build one P2P overlay for each video, which provides limited availability of video providers and produces high overlay maintenance overhead. To handle these problems, in this work, we novelly leverage the existing social network in YouTube, where a user subscribes to another user's channel to track all his uploaded videos. The subscribers of a channel tend to watch the channel's videos and common-interest nodes tend to watch the same videos. Also, the popularity of videos in one channel varies greatly. We study real trace data to confirm these properties. Based on these properties, we propose SocialTube that builds the subscribers of one channel into a P2P overlay and also clusters common-interest nodes in a higher level. It also incorporates a prefetching algorithm that prefetches higher-popularity videos. Extensive trace-driven simulation results and PlanetLab real-world experimental results verify the effectiveness of SocialTube at reducing server load and overlay maintenance overhead and at improving QoS for users.**

I. Introduction

In the past few years, the prevalence and popularity of video-on-demand (VoD) services (e.g., YouTube, Bing Video, Vimeo, Tudou) have grown enormously, fueled by the ability of users to generate video content using affordable digital cameras and the ubiquity of high-speed Internet access. On YouTube, the videos uploaded every day had increased from 135 hours in 2006 to over 50,000 hours in 2011, and the number of videos watched per day had increased from 100 million in 2006 to over 2 billion in 2011 [1, 2]. Currently, YouTube attracts 800 million unique visitors each month [2]. The dramatic success of VoD systems, along with rapid increases in video content and users in VoD systems, however, comes with a serious scalability problem.

Current VoD systems generally use a client-server architecture, in which videos are stored and downloaded solely from dedicated servers and there is no requirement on the upload bandwidth resource on the user side. While the client-server architecture is simple to implement, it produces prohibitive bandwidth costs for server owners, and has a low scalability to handle the rapid increase of users. Back in 2006, YouTube already paid over $1 million a month on bandwidth; with its traffic over 20 times what it once was [3]. In 2009, the cost has grown to $1 million per day [4]. The bandwidth cost has certainly increased substantially since then. While YouTube's finances are kept under wraps, bandwidth cost certainly makes up a huge portion of their expenses. Furthermore, quality of service (QoS) often suffers from massive number of requests to the server during peak usage times. YouTube users face an average service delay of over six seconds, much higher than other sites. It would be advantageous, then, to minimize the bandwidth cost on the server while achieving high QoS for users. Complementing the client-server architecture with a peer-to-peer (P2P) architecture that takes advantage of the extra bandwidth capacity of the huge number of users for P2P video sharing is a solution.

In a P2P architecture, users can download files from other users instead of from the centralized server. Previous works [5–14] have focused on applying the P2P paradigm to VoD by building one overlay for each video (i.e., **per-video** structure). PA-VOD [6] and NetTube [15] are two representative systems. In PA-VOD, when a user requests a video, the server directs the request to several other users currently watching the video. When a user finishes watching a video, it no longer acts as a provider. Since videos on YouTube tend to be short, many videos do not have peer providers so the server must provide the videos instead. NetTube leverages the video social network property that related videos tend to be viewed by a user, i.e., users who watch the same video tend to watch same videos in the future, and builds the viewers of the same video into an overlay to enable users to find other desired videos through their neighbors. Nodes maintain a cache of previously watched videos to boost video availability in the system. When a node requests a video for the first time, it sends its request to the server, which directs it to connect to the providers in the overlay of the video. When a node finishes watching a video, it remains in its overlay (i.e., maintains links to other nodes) to act as a video provider. To find a next video to watch, the node sends a query to its neighbors within two hops; if the video is not found, the user resorts to the server. However, the per-video overlays generate a prohibitive cost of overlay maintenance on nodes. A node that has watched multiple videos must stay in multiple overlays and maintain its links in each of the overlays. Also, two nodes may need to maintain redundant links for different per-video overlays though one link is sufficient. NetTube also provides limited availability of peer video providers as the probability that nodes watched the same video will watch the same videos

Fig. 1: Organization of YouTube videos.

is not necessarily high due to many related videos to a video.

To resolve the aforementioned deficiencies in P2P VoD systems, in this work, we novelly leverage the YouTube social network, where users are linked by subscription relationships. Note that unlike NetTube, we use the actual established social network in YouTube. In YouTube social network, each user has a *channel* that features a distinct webpage with all the user's uploaded videos. The channel in YouTube makes it easy for users to browse all videos of a specific user. If user A wishes to track all videos uploaded by user B, A can subscribe to B's channel, then will receive updates on new videos in B's channel. A registered user can subscribe to the channels, then receive updates on new videos in the channel. Once a new video is uploaded to his/her subscribed channels, a feed of the uploaded video is provided on his/her YouTube homepage. For example, the YouTube channel named ReutersVideo features short video clips relating to recent world news. Whenever ReutersVideo uploads a new video, a feed of the uploaded video is provided on each of its subscribed users' YouTube homepage. Users can find the video directly on ReutersVideo's homepage. YouTube recommends a user's frequently visited channels to the user to subscribe. Note the concept of channel in YouTube is completely different from the channel in P2P live streaming.

As shown in Figure 1, YouTube classifies videos into interest categories (e.g., Gaming, Sports, and Comedy), and organizes each category (interest and category are interchangeable terms in this paper) by individual videos or channels. A user can browse either videos or channels in each category. Therefore, channels are granular classification of categories. For example, a user interested in the category of *Science and Technology* can go to the page of this category to find popular content and the most-subscribed channels within that category. Based on these YouTube social network structural features, for low maintenance cost and high QoS, rather than building a **per-video** structure, we propose SocialTube, which builds an interest-based **per-community (i.e., channel)** hierarchical structure to connect nodes subscribed to the same channel and also connect users interested in the same category in the higher level. The utilization of the real social network in YouTube is the key to the design of SocialTube. First, the subscribers to the same channel tend to view the videos in the channel. Second, nodes with similar interests tend to view similar set of videos [6] and each channel involves a few interests. Third, the popularity of videos in a channel varies.

We first conduct an extensive analysis on YouTube data to verify the key social network properties of short-video sharing on YouTube. Based on these properties, we design a two-level hierarchical overlay; subscribers of the same channel are built into one lower-level cluster, and users watching channels in the same category are built into another higher-level cluster. Each node has limited links in both overlays. This design reduces overlay maintenance overhead by limiting the number of overlays a user joins, and enables users to locate a peer video provider through a limited number of hops. In addition, as the popularity of videos in one channel varies greatly, SocialTube lets nodes prefetch higher-popularity videos in the channel to improve the availability of videos. This is the first work that leverages the YouTube social network to develop an enhanced P2P video sharing system. Trace-driven simulation and PlanetLab experimental results show the efficiency and scalability of SocialTube in comparison with PA-VOD and NetTube.

The contributions of this paper are summarized as follows: (1) We analyze extensive YouTube trace data to verify the features of the YouTube social network. (2) Based on the features, we introduce the design of our system, and (3) We provide trace-driven simulation and PlanetLab experimental results to show the efficiency and scalability of our system design.

The remainder of this paper is arranged as follows. Section II presents an overview of the related work. Section III presents the analysis on our crawled trace data from YouTube. Section IV presents the details of the design of SocialTube. Section V presents the trace-driven experimental results in simulation and PlanetLab testbed. Section VI concludes this paper with remarks on future work.

II. Related work

Most of today's VoD applications rely on centralized servers to provide video sharing service, which is not scalable as the amount of video content and the number of clients have been dramatically increasing. Other popular VoD systems, such as PPLive [16], PPStream [17] and UUSee [18], are based on the tracker service and program source provided by centralized servers and uploading contributions from peers. Huang *et al.* [19] provided an insightful design analysis of PPLive for future system design. There have also been significant research efforts on optimizing sharing efficiency [20–24], where servers are in charge of both providing and helping users locate video resources, and most videos are shared among peers.

There are also several exploits in utilizing P2P video sharing to ameliorate the bandwidth cost for YouTube-like services. An early work called nVoD [25] incorporates an unstructured network that can make best use of network resources while providing high QoS. Huang *et al.* [6] analyzed a nine-month trace of MSN Video (predecessor of Bing) and proposed a peer-assisted VoD system that localizes P2P traffic within an ISP. GridCast [26] identifies that the single uploading scheme leads to idling in P2P networks and that multiple video caching can better reduce the server load. Chatzopoulou *et al.* [27] revealed a Poisson distribution of user arrival rates and an inverse correlation between video watching time and video popularity. BitTube [28] combines the client-server and P2P models, and supports the transition across the spectrum for

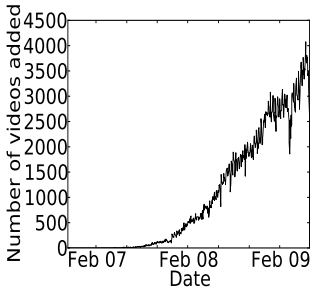


Fig. 2: # of videos added over time.

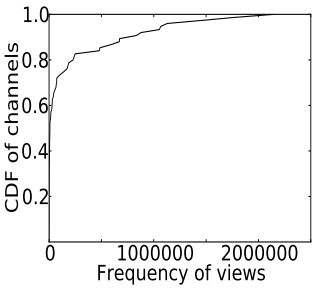


Fig. 3: View frequency of videos in different channels.

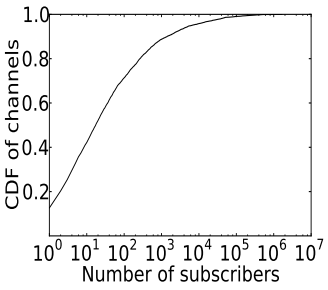


Fig. 4: # of subscribers to different channels.

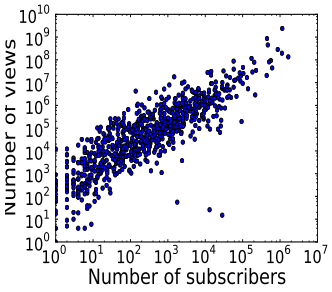


Fig. 5: Channel views vs. subscriptions.

pure client-server mode to BitTorrent mode. Li *et al.* [29] studied major features of the P2P VoD overlay networks and compared them with P2P file sharing and live streaming systems. Zhou *et al.* [30] proposed a unifying request scheduling model, in which a single request can be served by a number of peers.

A number of previous works analyzed the characteristics of YouTube [31, 32]. NetTube [15] attempts to identify users that watch the same video to group them into the same overlay for P2P video sharing. It also utilizes the existing related video list in YouTube to help nodes prefetch certain videos in order to reduce the waiting time before playback. Cheng *et al.* [33] provided an extensive analysis of the YouTube's video social structure to study video correlation. They showed that the videos in YouTube exhibit clear small-world characteristics. P2P VoD overlays can be broadly classified into two categories [34]: tree-based [5] and mesh-based [6–14]. As Cheng and Liu [15] pointed out, these protocols are only suitable for relatively long videos, typically of 1-2 hours (for movies) or even longer (for continuous TV broadcast). Also, a video is served by a separate overlay, with little interaction among overlays. YouTube-like short videos need new solutions for P2P video sharing.

SocialTube is different from previous works in that it is the first that uses the channel-subscription relationship connected user social network in YouTube for efficient P2P video sharing. Based on the structural features of YouTube that enable users to view videos based on either videos or channels within each category, SocialTube builds an interest-based per-community hierarchical structure (in contrast to per-video structure), which leads to low maintenance overhead and high QoS in P2P video sharing.

III. Trace Analysis

It is obvious to see the properties of the YouTube social network below based on normal user behaviors on YouTube. The subscribed users of a channel tend to watch the videos in the channel. The video popularity varies greatly. Also, each channel focuses on a certain number of categories and users subscribe to channels within their interests. These properties of user behavior and interests are normally true and conform user daily life behavior. We are still interested in studying and verifying these properties through analyzing a sample of YouTube data.

Previous research has shown that sampling a graph by ending a breadth-first search before its completion tends to overestimate node degree and underestimate the level of symmetry between nodes, but other metrics remain true to the entire graph [32]. Therefore, we crawled a sample of the graph using a breadth-first search. The process of crawling YouTube for user subscriptions and video upload data was completed as follows. A random user was added to a queue of users to crawl; information on all of the videos the user has uploaded was collected, including the video id, the total views of the video, the upload date, and the video length. The user's subscriptions were collected using the API and added to the queue; then, the user was deleted from the queue. This process continued until the queue was empty. All the information was collected using the YouTube Data API. In total, 2,301 users and 261,110 videos with upload dates ranging from 18 Jan. 2006 - 17 Sept. 2010 were crawled.

A. Scalability

Does a VoD system face a problem of scalability? Figure 2 shows the number of videos added over time in YouTube, taken from a single crawl of YouTube by Cheng *et al.* [15]. The figure shows an obvious increase in the number of videos posted over a period of two years. If the the growth rate continues to increase, the amount of bandwidth required merely for uploading videos could make operating such a site unprofitable. Further, with such an increasing number of video uploads, the QoS for watching videos will decline as fewer resources will be dedicated to serving videos.

O1: As usage of VoD service increases, VoD providers will face rapidly increasing demand for server bandwidth, reducing QoS for users.

B. Channel Popularity

First, we examine the popularity of channels in YouTube. Figure 3 shows the cumulative distribution function (CDF) of average video view frequency per channel, where video view frequency is the number of total views divided by the number of days the video has been online. Around 20% of channels receive less than 390 views per days; 80% receive less than 233,285 views per day; and the top 10% receive more than 783,240 views per day. This result implies that

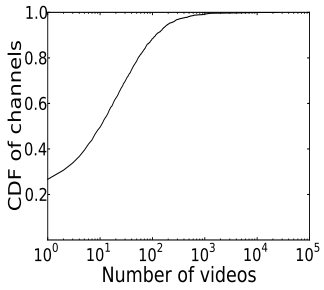


Fig. 6: # of videos per channel.

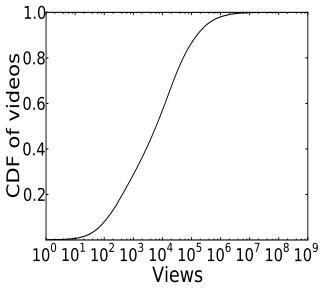


Fig. 7: # of views per video.

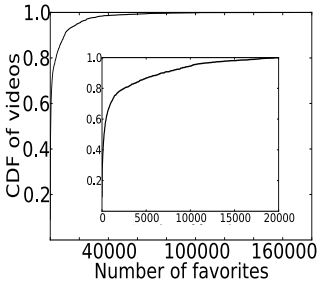


Fig. 8: # of times videos are marked as favorites.

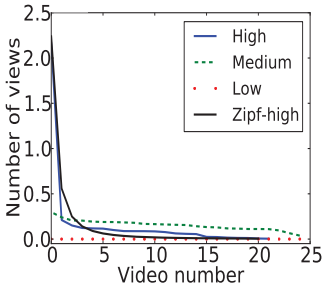


Fig. 9: Video popularity variation within channels.

the popularity of channels varies widely, enabling users that frequently visit the videos in the same channel to share videos among each other can relieve the server of delivering videos in high-popularity channels with high video view frequencies. Thus, a channel-based P2P structure can greatly reduce the bandwidth load of the server.

Figure 4 shows the distribution of the number of subscribers to channels. The number of subscribers is another way to measure the popularity of a channel. The bottom 25% of channels have less than 10 subscribers, while the top 25% have over 1,390 subscribers. This figure is a further evidence that a channel-based P2P structure would be useful in a YouTube-like application. Such a structure would enable subscribers of the same channel to share their frequently visited videos between each other, reducing the bandwidth load on the server and improving the video retrieval efficiency.

Figure 5 shows the relationship between a channel's number of subscriptions and its total number of views. The distribution of the points clearly indicates a strong, positive correlation between the number of subscriptions and the total number of views. This figure confirms that users are driven to select videos based on their subscription. Again, this indicates the desirability of a channel-based P2P structure to enable these users with shared channel subscriptions to efficiently retrieve videos without the use of a server. Figure 6 shows the number of videos in each channel. 50% of channels have 9 or fewer videos; however, the top 25% of channels have over 36 videos, and the top 10% of channels have over 116 videos.

O2: Building a P2P structure based on channels would produce reduced server load and enable efficient video retrieval for users.

C. Video Popularity

We then examine the distribution of video popularity, which is measured as the number of views. Figure 7 shows the distribution of views. We see that 50% of videos have 5,517 views or less and 10% of videos have more than 385,000 views. This figure shows that a large portion of videos draw a small number of views, and about 10% of videos get a large number of views. This viewing behavior is suitable for implementing a P2P architecture for video sharing. In the P2P architecture, providers for very popular videos can be readily found, providers for moderately popular videos can also be found from peers, and the server is needed to complement the P2P structure in locating the providers for unpopular videos.

Figure 8 shows the number of times each video has been marked as a favorite by users. The bottom 20% of videos have been marked as favorites less than 5 times, 75% of videos have been marked as favorites less than 2,115 times, and the top 10% of videos have been marked as favorites more than 9,865 times. Chatzopoulou *et al.* [35] showed that the Pearson Correlation Coefficient between the number of times a video has been marked as a favorite and its number of views is more than 0.8. Similar to Figure 7, Figure 8 indicates that a small set of videos receive most of the attention from users, so these videos can be readily shared among peers in an overlay without server dependence, while the sever is needed for locating unpopular videos.

One common strategy to reduce delay between video playback is to enable a user to prefetch chunks of videos that the user is likely to watch. In order to enhance this technique, a metric is needed to select videos for prefetching in order to increase the probability that the prefetched videos will be watched. Figures 7 and 8 show that the popularity of videos varies, which indicates that nodes can prefetch the highly popular videos to reduce video startup delay and improve QoS. Figure 9 shows the number of views on videos within a very popular channel, a moderately popular channel, and an unpopular channel, with popularity based on the total number of views for the channel. We observe that the actual number of views in the most popular channel roughly follows the Zipf distribution. The figure verifies that not all videos in a channel are equal with respect to popularity regardless of the popularity of the channel; then, prefetching popular videos in a channel would improve performance for most users in the channel.

O3: A channel-based VoD system can use video view counts as a metric to determine video prefetching, thus minimizing the delay between successive video views.

D. Channel Clustering and User Interest

As YouTube organizes channels in the same interest together under the interest category, a user that has this interest may subscribe several channels in this category. Figure 10 shows the top channels for different categories in YouTube as vertices, with links representing shared subscribers; a threshold value of 5 shared subscribers was used to filter out excessive

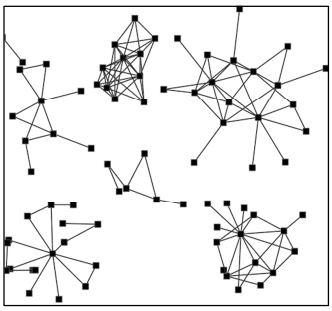


Fig. 10: A graph of channels connected by shared users.

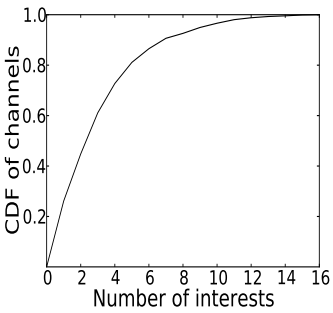


Fig. 11: # of interests in each channel.

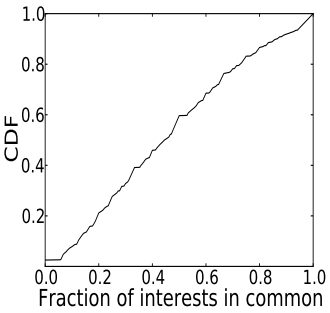


Fig. 12: Similarity between user interests and subscribed channels' interests.

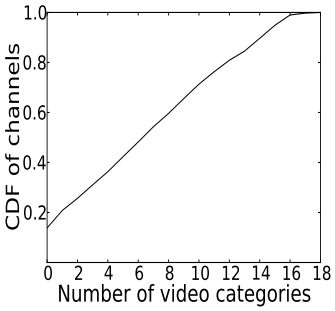


Fig. 13: Number of favorite video interests in each channel.

edges in the graph. In the figure, groups of channels form distinct clusters, indicating a clear tendency for users to subscribe to channels based on interests.

O4: Channels have strong clustering features that can be exploited to efficiently find providers across channels in P2P short video sharing.

Figure 11 shows the number of video categories each channel contains. This figure verifies that channels are generally focused on a small number of video categories. We determined each user's personal interests (denoted by C_u) by examining the categories of the user's favorite videos. We use C_c to denote the categories of the videos in the channels to which a user has subscribed. Then, the similarity of a user is computed as $\frac{|C_u \cap C_c|}{|C_u|}$. Figure 12 plots the CDF of the similarity metric. The similarities range from [0, 1], with 25th, 50th, and 75th percentiles of 0.21, 0.40, and 0.63, respectively. This result confirms that users tend to subscribe to channels that match their interests. Figure 13 shows the number of personal interests each user has. Around 60% of users have less than 10 interests; the highest number of interests for a user is 18. This result confirms that most users are interested in a limited number of categories of videos.

The results show that building a higher-level overlay for clustered channels in a category (i.e., connecting users with the same interest) can help users to find providers for their desired videos across channels without resorting to the server. Also, each user only needs to maintain a limited number of overlay links due to his/her limited number of interests.

O5: Channels tend to focus on a small number of video categories; users tend to subscribe to channels that match their interests.

IV. System Design

SocialTube is designed based on the properties of the YouTube social network presented in Section III. Like NetTube, SocialTube requires users to maintain a cache of all videos watched during the period of time between logging in and logging off (termed a *session*) to increase video availability; since videos are generally small, this does not unduly burden users. The design of SocialTube is summarized as follows, and the details of which are presented in the subsequent sections.

- **Hierarchical per-community structure.** In this structure, same-channel subscribers form an overlay in the lower level and same-interest nodes (channels) form a cluster in the higher level. Thus, subscribers to the same channel and nodes sharing the same interest can share videos between each other.
- **Channel-facilitated prefetching.** Since the videos in one channel have different popularity, nodes prefetch popular videos in their channels in order to minimize startup delay and improve video availability.

A. Hierarchical Per-Community Structure

We consider three goals in designing the P2P structure.

- *Server load minimization*. Requests made to the central server should be minimized; distributing requests to nodes reduces the overhead and bandwidth cost on the server, and video downloading delays due to server overload.
- *System maintenance overhead*. Each node should only need to maintain a limited number of connections to peers; otherwise it must afford a high maintenance cost caused by churn.
- *Peer video provider availability*. As the videos are searched in a distributed manner, quickly finding peer video providers is important to reducing startup delay and improving QoS.

Figure 14 shows an example of the SocialTube network structure. In the lower level, nodes in each channel (i.e., c_1-c_5) are formed into an overlay represented by a small circle. The channels in the same interest category (i.e., Category 1, 2) are formed into a higher-level cluster, where nodes in each channel are connected across the channels. Because a subscriber always watches videos in its subscribed channel, clustering the subscribers in the same channel helps improve peer video provider availability and minimize server load. As users also tend to view videos in their interests, they may want to watch videos in the channels they did not subscribe. The higher-level overlay clustering all users of channels within one interest category helps users to find peer video providers.

Since the subscribers of a channel tend to watch the videos in the channel, if they are randomly connected, a node can still find a video owner within a limited number of hops according to the video social network small world property (i.e.,

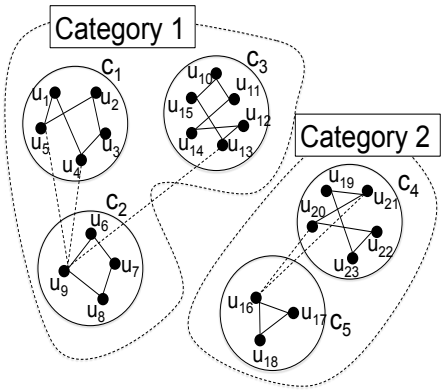


Fig. 14: A diagram of the network structure of SocialTube.

videos have strong correlations with each other) [33]. Thus, SocialTube limits the number of a node's links to a threshold N_l in its lower-level channel overlay, and limits the number of a node's links to a threshold N_h in its higher-level channel cluster. We call a node's links in its lower-level channel overlay *inner-links* and a node's links in its higher-level channel cluster *inter-links*; the neighbors connected by the links are called *inner-neighbors* and *inter-neighbors*, accordingly.

We explain how a node joins in the system and searches videos in SocialTube below. As shown in Figure 14, when node u_9 initially requests a video in a channel, it contacts the server. If the node has subscribed to the channel and there are node(s) existing in the channel overlay, the server randomly chooses a node in the channel overlay, say u_6, for u_9 to connect to. The server also randomly chooses a node in each channel in this channel's higher-level overlay, say u_4 and u_{13}, for u_9 to connect to. If there is no node existing in the channel overlay or u_9 has not subscribed to the channel, the server randomly chooses a node in each channel overlay (including a node with the video) in the higher-level overlay of the video's interest and recommends them to u_9. In the former case, u_9 also becomes the first node in the channel overlay. Thus, for non-channel-subscribers, SocialTube still helps them to locate peer video providers by using the high-level interest-based overlay. They can easily find videos from their common-interest peers.

After joining in the system, u_9 starts searching its desired video. It first searches its channel overlay and then searches its higher-level interest overlay within TTL hops. Specifically, it first asks its inner neighbor(s), u_6, along with a TTL. If u_6 does not have the video, it forwards the request to its inner-neighbors (u_7) in the channel overlay, and each neighbor decrements TTL and forwards the request to its neighbors if it does not have the video and TTL$\neq 0$. If a request receiver has the video (say u_8), it provides the video directly to u_9. Then, u_9 connects to the video provider and ignores other responses. In this way, newly joined node u_9 builds its links to other nodes in the lower-level channel overlay until the number reaches N_l. Thus, u_9 connects to nodes that tend to watch the same videos as u_9 later on. If u_9 cannot find its desired video within TTL along inner-links in its channel overlay, it sends its request to its inter-neighbors (u_4 and u_{13}). Within each channel overlay, the request is forwarded along TTL hops. If a video provider is found, say u_5, it provides the video to u_9, and u_9 connects to u_5 if the number of its inter-links is less than N_h. If a video cannot be found in the inter-channel querying, u_9 resorts to the server for the video. The TTL is used to limit the forwarding hops of messages and hence the video searching overhead. Later on, node u_9 searches videos using the same method as presented previously. In order to ensure that the server is able to accurately assist new users in joining the system, users should report their changes of subscribed channels. However, the server is required to keep track of much less information in SocialTube than in NetTube, where users need to report the changes of videos they watch.

In figure 14, user u_9 is currently watching videos in channel c_2, so it maintains links to users u_6 and u_8 in c_2. It also maintains links to users u_4 and u_5 in channel c_1, and u_{13} in channel c_3. u_9 maintains no links to users outside of his/her channel or category. When u_9 wants to watch v_1, it sends a query to u_6 and u_8. If neither peer has the video, they forward the request to their neighbors in c_2. If the video cannot be located in the channel overlay after TTL, u_9 sends a query to u_4, u_5 and u_{13} in c_1 and c_3, who forward the query to their channel peers until TTL$= 0$. If the video has still not been located, u_9 sends the request to the server. This process is detailed in Algorithm 1.

Algorithm 1: Pseudo-code of the SocialTube process executed by node u_i.

```
//C_i: channel peers of u_i; K_i: category peers of u_i
Begin: u_i selects v_i to watch
if C_i is empty {
  u_i requests peers from the server
  if no peers exist in the channel overlay of v_i
    server sends video v_i to u_i
  else {
    C_i ← list of peers in channel
    K_i ← list of peers in category
    REQUEST(C_i,K_i)
  }
}
else REQUEST(C_i,K_i)

REQUEST(C_i,K_i) {
u_i sends query with TTL to C_i
if query is successful
  return
else u_i sends query with TTL to K_i
  if query is successful
    return
  else
    u_i requests video from server
}
```

A node leaves all of its overlays in SocialTube when it logs off the system (*i.e.* close the YouTube webpages). The next time when the node logs in, it first tries to connect to its previous neighbors. If none of the inner-neighbors exist in

the lower-level channel overlay or none of the inter-neighbors exist in the higher-level overlay, the node contacts the server to build links as if it is first joining in the SocialTube system. SocialTube adopts the methods used in P2P networks for structure maintenance. That is, each node periodically probes its neighbors. If a node finds that its neighbors have left the system abruptly or have failed, it removes its links to these neighbors and adds more neighbors as described previously. For graceful departures, before a node leaves the system, it notifies all of its neighbors, which will update the links to the departing node.

B. Channel-Facilitated Prefetching

The average bitrate of a YouTube video is 330 kbps [33]; most Internet users have typical download bandwidths of at least twice that bitrate, and are able to fully download a video before finishing playback [15]. During the time in which a user is watching a fully downloaded video, the first chunk from his/her possible future requested videos can be downloaded in order to reduce the playback delay of future videos. This technique has been widely used in previous VoD systems [15, 6]. The authors of PA-VoD discussed the system states, in which prefetching can benefit system performance [6]. In NetTube, a node randomly chooses the videos its neighbors have watched to prefetch. This is based on the rationale that a user tends to watch related videos. However, there are many related videos for a given video; thus, these randomly prefetched videos do not have a high probability of being watched later on. A challenge here is to use limited cache space to store the videos that are most likely to be watched. To handle this challenge, SocialTube implements a simple, channel-based prefetching decision algorithm. We know that videos within a channel have greatly varying popularities and users tend to select a next video in the same channel. In order to exploit these facts, in SocialTube, a node prefetches the first chunks of highly popular videos in its subscribed channels, which enhances the probability that a prefetched video is watched by the node. YouTube website displays the most viewed videos and most subscribed channels, which means that the centralized server in YouTube keeps track of the visit rate of each video and each channel. Thus, the server provides the popularities of videos in each channel to its subscribers periodically. Then, the node prefetches the first chunks of M videos at the top of the ordered list based on video popularity. The value of M is determined by each node's cache size.

In SocialTube's prefetching algorithm, the accuracy of the prefetch decision is correlated to the relative view count of the prefetched video. Assume the probability that a video is watched next is $p_k = \frac{v_k}{v_t}$, where v_k is the number of views on a video of rank k and v_t is the total number of views in a channel. From Figure 9, we know that views tend to follow Zipf's distribution with the characteristic exponent $s = 1$. Then, $v_k = (\frac{1/k}{\sum_{n=1}^{N} 1/n})v_t$, where N is the number of videos in the channel and p_k is simply the standard Zipf's distribution function. For a channel with 25 videos, the probability that a single prefetch is accurate (p_k) equals 26.2%. Since users are often able to prefetch 3-4 videos during a single video playback, the prefetch accuracy rises to 54.6%.

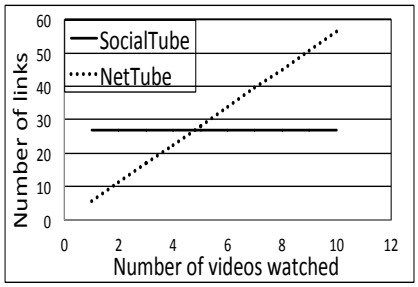


Fig. 15: Overlay maintenance overhead.

C. Comparing SocialTube with NetTube

NetTube [15] builds users watching the same video into an overlay (i.e., per-video overlay). While this P2P structure increases video availability in peers, it comes at the cost of high maintenance overhead. In addition, two nodes may be connected by redundant links; each link corresponds to one video overlay, generating unnecessary maintenance cost. Also, the probability that two same-video viewers will watch the same video later on is not necessarily high. Rather than building per-video overlays, SocialTube builds per-community overlays that form subscribers of the same channel into a overlay and groups common-interest channels into a higher-level cluster. In SocialTube, a node maintains a limited number of connections to other nodes in the same channel, and to other nodes in different channels that share the same interest with the channel. Unlike NetTube, SocialTube builds one link between two nodes that always watch the same multiple videos (in a channel), thus avoiding redundant links. As two nodes in the same channel are more likely to watch the same videos and common-interest nodes tend to watch the same videos, SocialTube improves the availability of peer video providers of NetTube. Thus, SocialTube significantly reduces the overlay maintenance overhead and improves QoS of NetTube.

We then provide a brief analysis of the maintenance overhead of SocialTube compared with NetTube, which is measured as the number of connections a user must maintain. Let u_c be the number of users in a channel and u_t be the total number of users within all channels in an interest. In the unstructured P2P structure with random links between nodes, to achieve an optimal tradeoff between the number of lookup hops and the number of links a node maintains, we can set $N_l = log(u_c)$ and $N_h = log(u_t)$. Then, the maintenance overhead is $log(u_c) + log(u_t)$. For comparison, we consider the overhead of NetTube. Let m be the number of videos a user watches from different overlays in a session, and u be the number of users watching a video. Assume a node connects to $log(u)$ in an overlay for a video. Then, the number of connections a user must maintain in NetTube is $mlog(u)$.

Figure 15 compares the estimated overhead for different values of m, with values for u, u_c, and u_t arbitrarily chosen to be 50, 500, and 250,000, respectively. Clearly, for small values of m, NetTube has very low overhead. As m

increases, however, the overhead of NetTube increases linearly while the overhead of SocialTube stays constant. The figure demonstrates the high scalability and low overlay maintenance overhead of SocialTube.

In conclusion, the design of SocialTube achieves the aforementioned three goals. 1) Users have a high probability of finding next videos in the channel overlay or in the higher-level interest-based overlay without going to the server, minimizing server load and increasing peer video provider availability, and 2) overhead is reduced since each user only maintains a limited number of neighbor links in per-community overlays (in contrast to the per-video overlays in NetTube that accumulates excessive links as users watch more videos).

V. Performance Evaluation

To measure the performance of SocialTube in comparison with PA-VoD [6] and NetTube [15], we conducted trace-driven experiments using PeerSim [36], a well-documented event-driven simulator, and PlanetLab [37], a real-world testbed for distributed networks. The simulation can test a large-scale network while PlanetLab can provide a real-world testing environment (e.g., real video transmission delay).

Simulation settings were derived from our trace of YouTube channels as presented in Section III. We used the video length distribution, video popularity distribution, and video category distribution directly in our simulations. NetTube [15] assumes that a node always views the videos in its interests in experiments, which is not true. A user may view videos not in her/his interests sometimes in real life. To more accurately simulate realistic viewing behavior, users perform the following video selection mechanism. When a node chooses a video to view, it has a 75% chance of selecting a video in the same channel, a 15% chance of selecting a video in the same category, and a 10% chance of selecting a video in a different category. Other percent values keeping the same magnitude relationship will not change the relative performance differences between the three methods. Each node is assumed to watch ten videos in one session. One experiment consists of 25 sessions for each user. Each node leaves the system after each session and joins in the system for the next session; the off time periods for a user's sessions are determined using a Poisson distribution [27] with mean of 50s. Thus, all experimental results presented below are for the environment with churn. Nodes store their cached videos for their next session. The number of inner-links and inter-links per node were set to 5 and 10, respectively. The TTL was set to 2. Nodes probe their neighbors every 10 minutes for overlay maintenance. Other default experimental settings are shown in Table I.

In PlanetLab experiment, we use 250 globally distributed nodes. As this number of users is much smaller than in the simulation, experimental settings were scaled down accordingly. The number of categories was set to 6, with each category having 10 channels and each channel having 40 videos, for a total of 2,400 videos. The number of inner-links and inter-links per node were set to 5 and 10, respectively. One experiment consists of 5 sessions for each user, and the off time periods for a user's sessions are determined using the Poisson distribution with mean of 20 minutes. The node at 130.127.39.152 was chosen to be the server. All other settings are the same as those in the simulation. In our experiments, we are interested in the following metrics:

- *Startup delay*. This is the time period a user must wait after (s)he selects a video before the video playback starts, including the time it takes to query peers or the server.
- *Normalized peer bandwidth*. This is the percent of video chunks provided by peers out of the total video chunks provided. It measures the effectiveness of a P2P video sharing system at reducing server bandwidth.
- *Maintenance overhead*. This is the number of links a node must maintain in the overlays. This metric measures the maintenance overhead of the P2P overlays.

The metric that the prefetching strategy affects is only startup delay, which was measured with and without the prefetching strategy in our experiments. Prefetched chunks of short videos are very small in size (about 150 KB), so the prefetching cost can be negligible.

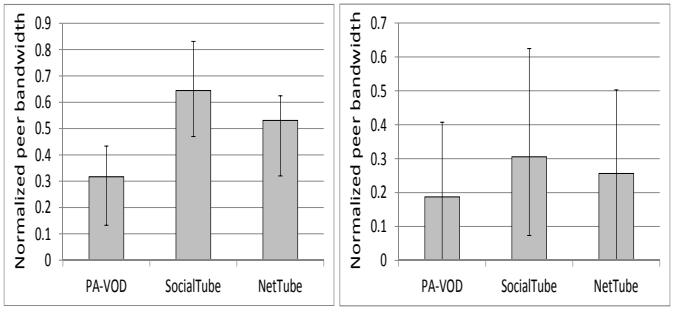
TABLE I: Experiment default parameters.

Parameter	Default value
Simulation duration	3 days
Number of nodes	10,000
Number of videos	10,121
Number of channels	545
Video size	YouTube video size distr.
Number of chunks per video	20
Video bitrate	320 kbps
Server bandwidth	500 mbps

A. Server Bandwidth Reduction

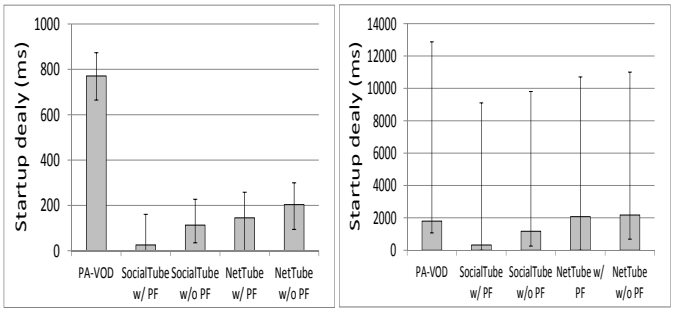
First, we examine the effect of SocialTube at reducing server load. Figure 16(a) and Figure 16(b) show the 1^{st}, 50^{th}, and 99^{th} percentiles of the normalized bandwidth contribution by peers on PeerSim and PlanetLab, respectively. We see from Figure 16(a) that 50% of nodes in SocialTube, NetTube, and PA-VoD receive more than 0.63, 0.53 and 0.31 normalized peer bandwidth, respectively; 99% of nodes in SocialTube, NetTube, and PA-VoD receive more than 0.46, 0.32 and 0.14 normalized peer bandwidth, respectively. Since users in PA-VoD do not maintain a cache of watched videos, video availability from peers is restricted. In NetTube and SocialTube, users keep a cache of previously watched videos to share with others, so the likelihood that a video can be found in a peer is much greater.

SocialTube forms same-channel users into a lower-level overlay and further forms common-interest users into a higher-level overlay, so users have a high probability to find chunk providers in the overlays. NetTube clusters nodes based on each video they watch. Though nodes watching a video are likely to watch the same video subsequently, this probability is not always high, so nodes have relatively lower probability to find chunk providers in the same overlay. Note that in Figure 16(b), the 1^{st} percentiles peer bandwidth contribution



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

Fig. 16: The bandwidth contribution of peers.



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

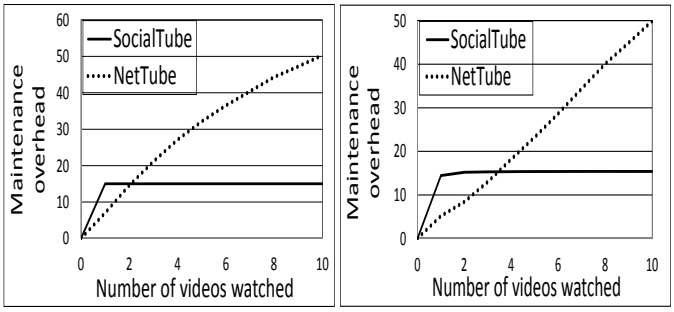
Fig. 17: Startup delay and effectiveness of prefetching.

for NetTube, and PA-VoD have reached 0, partly due to the unstable network environment on PlanetLab (e.g., connection failure and network congestion). However, the 1^{st} percentiles peer bandwidth contribution of SocialTube is around 0.07. The results in both figures demonstrate the success of SocialTube in reducing the dependency on the server for video retrieval. Figure 16(b) confirms Figure 16(a) that SocialTube outperforms NetTube and PA-VoD in reducing server bandwidth.

B. Startup Delay

We then examine the effect of SocialTube at improving QoS for users. Figure 17(a) and Figure 17(b) show the startup delay of NetTube and SocialTube with and without prefetching, and of PA-VoD on PeerSim and PlanetLab. For NetTube, users prefetch the first chunks of 3 videos randomly from their friends' watched videos. For SocialTube, users prefetch the first chunks of 3 top popular videos within the channel it currently is watching. PA-VoD does not implement a prefetching scheme. We see that PA-VoD generates a long startup delay. Also, SocialTube generates shorter startup delay than NetTube with and without their own prefetching strategy, respectively. PA-VoD has the worst performance because it frequently fails to find peers to serve videos and must instead rely on the server; when this begins to overload the server, videos are delayed. In NetTube, queries travel two hops, and if a video is not found, the server is contacted. In SocialTube, a query for a video is forwarded with TTL=2 in the channel overlay; if the video is not found, the query is then forwarded with TTL=2 in the higher-level category overlay. Though a query is forwarded for more hops in the P2P overlay before being sent to the server, SocialTube reduces more requests sent to the server, avoiding overloading the server. As a result, it produces shorter startup delay than NetTube.

The figures also shows that the individual prefetching strategies of SocialTube and NetTube help reduce startup delay. SocialTube's prefetching strategy reduces greater startup delays than NetTube's. This is because SocialTube's channel-based prefetching targets the videos that users are most likely to watch next; NetTube's strategy prefetches randomly from neighbors' watched videos. The experimental results confirm the effectiveness of SocialTube's channel-facilitated popularity-based prefetching strategy.



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

Fig. 18: Overlay maintenance overhead.

C. Overlay Maintenance Overhead

Figure 18(a) and Figure 18(b)show the average maintenance overhead at different intervals during a video-watching session. Figure 18(a) shows that SocialTube users maintain 15 links at all times through their sessions after the initial phase; NetTube users, on the other hand, start out with few links but rapidly accumulate more as their sessions continue. At the end of the session, the average NetTube user has 35 more links than the average SocialTube user. This result confirms the validity of Figure 15; namely, users in NetTube initially have a small overhead, but it increases rapidly as the user session continues. From these results, we see that NetTube fails to constrain the overhead of maintaining the P2P system, while SocialTube reduces server bandwidth with much less overhead incurred to the users. Experimental results on PlanetLab in Figure 18(b) confirms that SocialTube demands significantly lower maintenance overhead than NetTube.

VI. Conclusions

With the incredible growth in demand for short video sharing online, challenges of scalability and cost with the client-server architecture have prompted the design of P2P architectures to provide high quality VoD service to users. This work is the first that leverages the YouTube social network to develop an enhanced P2P video sharing system. In YouTube social network, users subscribed to a channel tend to view the videos in the channel; users with similar interests tend to view similar set of videos and each channel involves a few

interests; and the popularity of videos in a channel varies greatly. We confirmed these properties through analyzing a sample of YouTube data. Based on these properties, we designed SocialTube, an interest-based per-community P2P system for short video sharing. SocialTube has a two-level overlay structure: subscribed users of the same channel form into one lower-level cluster, and users watching channels in the same interest form into another higher-level cluster. This enables users to efficiently find videos within their channel and interest clusters without resorting to the server, and reduces overlay maintenance costs over previous per-video overlays. SocialTube also has a channel-facilitated popularity-based prefetching strategy to minimize video startup delay. Trace-driven simulations and the experiments on the PlanetLab real-world testbed have proven the superior performance of SocialTube over previous systems. In our future work, we will study the impact of the different number of links per node on the video sharing performance and explore the value that can achieve an optimal tradeoff between the system maintenance overhead and availability of peer video providers.

ACKNOWLEDGEMENTS

This research was supported in part by U.S. NSF grants IIS-1354123, CNS-1254006, CNS-1249603, CNS-1049947, CNS-0917056 and CNS-1025652, Microsoft Research Faculty Fellowship 8300751.

REFERENCES

[1] YouTube Press Timeline. http://www.youtube.com/t/press_timeline.

[2] YouTube Press Statistics. http://www.youtube.com/t/press_statistics.

[3] Your tube, whose dime? http://www.forbes.com/2006/04/27/video-youtube-myspace_cx_df_0428video.html.

[4] YouTube costs Google $2 million per day. http://www.inquisitr.com/24740/youtube-costs-google-2-million-per-day/.

[5] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-To-Peer Multicast. In *Proc. of ICNP*, 2006.

[6] C. Huang, J. Li, and K. W. Ross. Can Internet Video-On-Demand Be Profitable? In *Proc. of SIGCOMM*, 2007.

[7] J. Wang, C. Huang, and J. Li. On ISP-friendly Rate Allocation For Peer-Assisted VoD. In *Proc. of ACM Multimedia*, 2008.

[8] D. A. Tran, K. A. Hua, and T. Do. ZIGZAG: an Efficient Peer-To-Peer Scheme For Media Streaming. In *Proc. of INFOCOM*, 2003.

[9] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proc. of SOSP*, 2003.

[10] C. Ho, S. Lee, and J. Yu. Cluster-based Replication For P2P-Based Video-on-Demand Service. In *Proc. of ICEIE*, 2010.

[11] B. Mathieu, P. Paris, G. Guelvouit, and S. Rouibia. A Secure and Legal Network-Aware P2P VoD System. In *Proc. of ICIW*, 2010.

[12] K. Wang and C. Lin. Insight into the P2P-VoD System: Performance Modeling and Analysis. In *Proc. of ICCCN*, 2009.

[13] N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-drIven MEsh-Based Streaming. In *Proc. of the INFOCOM*, 2007.

[14] C. Wu, B. Li, and S. Zhao. Multi-Channel Live P2P Streaming: Refocusing on Servers. In *Proc. of the INFOCOM*, 2008.

[15] X. Cheng and J. Liu. NetTube: Exploring Social Networks for Peer-to-Peer Short Video Sharing. In *Proc. of INFOCOM*, 2009.

[16] PPLive. http://www.pplive.com.

[17] PPStream. http://www.ppstream.com.

[18] UUSee. http://www.uusee.com.

[19] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P VoD System. In *Proc. SIGCOMM*, 2008.

[20] X. Zhang, J. Liu, B. Li, and T. Yum. CoolStreaming/DONet: A Data-Driven Overlay Network For Peer-To-Peer Live Media Streaming. In *Proc. of INFOCOM*, 2005.

[21] X. Liao, H. Jin, Y. Liu, L. Ni, and D. Deng. AnySee: Peer-to-Peer Live Streaming. In *Proc. of INFOCOM*, 2006.

[22] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating Trees From Overlay Multicast. In *Proc. of IPTPS*, 2005.

[23] T. Locher, S. Schmid, and R. Wattenhofer. eQuus: A Provably Robust And Locality-Aware Peer-To-Peer System. In *Proc. of P2P*, 2006.

[24] J. Venkataraman and P. Francis. Chunkyspread: Multi-Tree Unstructured Peer-To-Peer Multicast. In *Proc. of IPTPS*, 2006.

[25] S. Annapureddy, C. Gkantsidis, P.R. Rodriguez, and L. Massoulie. Providing Video-On-Demand Using Peer-To-Peer Networks. In *Proc. of IPTV Workshop in WWW*, 2006.

[26] B. Cheng, L. Stein, H. Jin, X. Liao, and Z. Zhang. GridCast: Improving Peer Sharing for P2P VoD. *ACM TMCCA*, 2008.

[27] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng. Understanding User Behavior in Large-scale Video-on-Demand Systems. *SIGOPS Oper. Syst. Rev.*, 2006.

[28] B. Liu, Y. Cui, B. Chang, B. Gotow, and Y. Xue. BitTube: Case Study of a Web-Based Peer-Assisted Video-On-Demand System. In *Proc. of ISM*, pages 242–249, 2009.

[29] B. Li, M. Ma, Z. Jin, and D. Zhao. Investigation of a Large-scale P2P VoD Overlay Network by Measurements. *Peer-to-Peer Networking and Applications*, 5(4):398–411, 2012.

[30] Y. Zhou, T. Fu, and D. Chiu. A Unifying Model and Analysis of P2P VoD Replication and Scheduling. In *Proc. of INFOCOM*, pages 1530–1538, 2012.

[31] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the Worlds Largest User Generated Content Video System. In *Proc. of IMC*, 2007.

[32] A. Mislove, M. Marcon, K. Gummadi, P. Dreschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proc. of IMC*, 2007.

[33] X. Cheng, C. Dale, and J. Liu. Statistics and Social Network of Youtube Videos. In *Proc. of IWQoS*, pages 229–238, 2008.

[34] J. Liu, S. G. Rao, B. Li, and H. Zhang. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. In *Proc. of the IEEE*, 2008.

[35] G. Chatzopoulou, C. Sheng, and M. Faloutsos. A First Step Towards Understanding Popularity in YouTube. In *Proc. of INFOCOM*, 2010.

[36] Peersim: A peer-to-peer simulator. http://peersim.sourceforge.net/.

[37] Planet lab. http://www.planet-lab.org/.