Toward Efficient Short-Video Sharing in the YouTube Social Network

HAIYING SHEN, University of Virginia
HARRISON CHANDLER, University of Michigan
HAOYU WANG, University of Virginia

The past few years have seen an explosion in the popularity of online short-video sharing in YouTube. As the number of users continue 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/her 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, which 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. To enhance the system performance, we further propose the demand/supply-based cache management scheme and reputation-based neighbor management scheme. 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.

CCS Concepts: • **Computer systems organization** → **Distributed architectures**; • **Networks** → *Application layer protocols*;

Additional Key Words and Phrases: Video on demand, P2P networks, social networks, youtube

ACM Reference format:
Haiying Shen, Harrison Chandler, and Haoyu Wang. 2018. Toward Efficient Short-Video Sharing in the Youtube Social Network. *ACM Trans. Internet Technol.* 18, 3, Article 33 (March 2018), 25 pages.
https://doi.org/10.1145/3137569

This research was supported in part by U.S. NSF grants OAC-1724845, ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. We would like to thank the anonymous reviewers for their feedback and Dr. Yuhua Lin for his help on this work.

Authors' addresses: H. Shen, Department of Computer Science, University of Virginia, 85 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904-4740; email: hs6ms@virginia.edu; H. Chandler, Computer Science and Engineering Division, University of Michigan, Bob and Betty Beyster Building, 2260 Hayward Street, Ann Arbor, MI 48109-2121; email: hchandl@umich.edu; H. Wang, Department of Computer Science, University of Virginia, 85 Engineer's Way, office 532, Charlottesville, VA 22904-4740; email: hw8c@virginia.edu.

https://doi.org/10.1145/3137569

1 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 135h in 2006 to over 144,000h in 2014, and the number of videos watched per day had increased from 100 million in 2006 to over 2 billion in 2014 (you 2016a, 2016b). Currently, YouTube attracts more than one billion unique visitors each month (you 2016a). 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. Back in 2006, YouTube already paid over $1 million a month on bandwidth; with its traffic over 20 times what it once was (VOD 2016). In 2009, the cost has grown to $1 million per day (run 2016). 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 the massive number of requests to the server during peak usage times. YouTube users face an average service delay of over 6s, much higher than other sites. Then, it would be advantageous 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 (Venkataraman et al. 2006; Zhu 2012; Ding et al. 2012; Huang et al. 2007; Wang et al. 2008; Tran et al. 2003; Castro et al. 2003; Ho et al. 2010; Mathieu et al. 2010; Wang and Lin 2009; Magharei and Rejaie 2007; Wu et al. 2008) have focused on applying the P2P paradigm to VoD by building one overlay for each video (i.e., **per-video** structure). PA-VOD (Huang et al. 2007) and NetTube (Cheng and Liu 2009) 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 is not necessarily high due to many related videos to a video.

To resolve the aforementioned deficiencies in previously proposed 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 user A will receive updates on new videos in B's channel. A registered user can subscribe to the channels, and 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.

YouTube classifies videos into interest categories (e.g., Gaming, Sports, and Comedy), and organizes each category (interest and category are interchangeable terms in this article) 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 properties in 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 a similar set of videos (Huang et al. 2007), 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. Note that our data analysis observations are not new observations and they are widely recognized facts in practice. Without the trace analysis, we can still design our system based on these widely recognized facts. Based on the 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. Besides, 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. To further enhance the availabilities of videos and stimulate nodes to contribute their upload bandwidth resource, we propose a demand/supply-based cache management scheme and a reputation-based neighbor management scheme. 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 remainder of this article is arranged as follows. Section 2 presents an overview of the related work. Section 3 presents the analysis on our crawled trace data from YouTube. Section 4 presents the details of the design of SocialTube. Section 5 presents the trace-driven experimental results in simulation and PlanetLab testbed. Section 6 concludes this article with remarks on future work.

2 RELATED WORK

Most of today's VoD applications rely on centralized servers to provide video-sharing service. Other popular VoD systems, such as PPLive (PPL 2016), PPStream (PPS 2016), and UUSee (UUS 2016), are based on the tracker service and program source provided by centralized servers and uploading contributions from peers. Huang et al. (2008) provided an insightful design analysis of PPLive for future system design. There have also been significant research efforts on optimizing sharing efficiency (Zhou et al. 2013; Zhang et al. 2005; Liao et al. 2006; Pai et al. 2005; Locher et al. 2006; Venkataraman and Francis 2006), where servers are in charge of both providing and helping users locate video resources, and most videos are shared among peers. SocialTube is similar to these works in using P2P video sharing. However, SocialTube is novel in that it leverages the channel-subscription relationship connected user social network in YouTube to build a P2P structure for efficient video sharing.

There are also several exploits in utilizing P2P video sharing to ameliorate the bandwidth cost for YouTube-like services. An early work called nVoD (Annapureddy et al. 2006) incorporates an unstructured network that can make best use of network resources while providing high QoS. Huang et al. (2007) analyzed a 9-month trace of MSN Video (predecessor of Bing) and proposed a peer-assisted VoD system that localizes P2P traffic within an ISP. GridCast (Cheng et al. 2008) identifies that the single uploading scheme leads to idling in P2P networks and that multiple video caching can better reduce the server load. Yu et al. (2006) revealed a Poisson distribution of user arrival rates and an inverse correlation between video watching time and video popularity. BitTube (Liu et al. 2009) combines the client-server and P2P models and supports the transition across the spectrum for pure client-server mode to BitTorrent mode. Li et al. (2012) studied major features of the P2P VoD overlay networks and compared them with P2P file sharing and live streaming systems. Zhou et al. (2012) proposed a unifying request scheduling model, in which a single request can be served by a number of peers. To reduce the cost of inter-ISP traffic for VoD providers, Magharei et al. (2014) proposed to group the nodes using the same ISP into the same overlay to reduce the volume of inter-ISP traffic. Lehrieder et al. (2010) studied whether P2P users can benefit from locality-awareness. Consideration of inter-domain traffic is crucial for video QoS, e.g., Lehrieder et al. (2010). Adaptive streaming technologies (Seufert et al. 2015; Lin and Shen 2017; Zambelli 2009) can be used to improve QoS. For example, Zambelli (2009) proposed the Microsoft Smooth Streaming method, in which with each video download, the user measures the network bandwidth and runs a Rate Determination Algorithm (RDA) to determine which bit rate to request next. When selecting the next bit rate, the RDA must consider the available bandwidth, CPU processing power, screen size, and the fullness of its buffer. In fog computing, Yi et al. (2015) explored the concept of computing and claimed that "fog computing can provide dynamic customizable optimization based on client devices and local network conditions." Zhu et al. (2013) considered the user's local network conditions and then used the information to leverage the video website performance with lower latency. SocialTube shares the same goal as these works in improving the performance of video sharing but with different a focus. SocialTube focuses on using the channel-subscription relationship connected user social network in YouTube to build a system structure for efficient P2P video sharing. The above works can be adopted to SocialTube to further improve the video sharing performance.

A number of previous works analyzed the characteristics of YouTube and user Quality of Experience (QoE) on YouTube (Cheng et al. 2013; Brodersen et al. 2012; Cha et al. 2007; Mislove et al. 2007; Cheng et al. 2013; Casas et al. 2014; Arantes et al. 2016; Hossfeld et al. 2012; Seufert et al. 2015; Wamser et al. 2016; Nam et al. 2016). NetTube (Cheng and Liu 2009) 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 to reduce the waiting

time before playback. Cheng et al. (2008) 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 (Liu et al. 2008): tree-based (Venkataraman et al. 2006) and mesh-based (Zhu 2012; Ding et al. 2012; Huang et al. 2007; Wang et al. 2008; Tran et al. 2003; Castro et al. 2003; Ho et al. 2010; Mathieu et al. 2010; Wang and Lin 2009; Magharei and Rejaie 2007; Wu et al. 2008). As Cheng and Liu (2009) pointed out, these protocols are only suitable for relatively long videos, typically of 1–2h (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. To address this problem, we propose SocialTube for YouTube-like short videos.

There are works investigating social networks in popular VoD sites (Cha et al. 2007; Mislove et al. 2007; Gill et al. 2007; Cheng et al. 2008). Complementary approaches like Social Video Distribution over Cloud Content Delivery Network (CDN) are investigated by Hu et al. (2016). Chang and Wu (2015) considered a social feature-based P2P system. The social network behind YouTube is investigated by Wattenhofer et al. (2012) and Afrasiabi Rad and Benyoucef (2014). The Impact of social network structure on content propagation is investigated by Yoganarasimhan (2012). Shen et al. (2014) proposed a P2P-assisted video-sharing system in online social networks based on the video-sharing patterns in Facebook. The domain of information-centric networking (ICN) also considers content popularity or social structures (Ioannou and Weber 2015; Thar et al. 2015; Nicolas et al. 2013). Hu et al. (2016) proposed to use social video distribution over a Cloud CDN. Wang et al. (2013) proposed a social aware video prefetching mechanism for mobile users. This method defines three different prefetching levels: "all, parts, little," for three different social content sharing ways: "direct recommendation, subscription, public sharing." All of these works show the relationship between the social networks and the content distribution or indicate the advantage of leveraging social networks in content distribution. Along this line of research, SocialTube also leverages the social network feature for improving the performance of video content delivery. Different from the previous works, SocialTube uniquely uses the channel-subscription relationship connected user social network in YouTube and focuses on the video-sharing structure design.

Also, many works focus on video popularity study and caching. Efficient approaches to cache video-streaming contents taking into account user behaviour and popularity of contents are proposed by Krishnappa et al. (2015) and Burger et al. (2015) while Broxton et al. (2013) considered the dynamics of popularity in caching. Nicolas et al. (2013) investigated the benefits of peer-assisted caching of YouTube content from an ISP's point of view. Rossi and Rossini (2012) studies the caching performance of Content Centric Networking (CCN), with special emphasis on the size of individual CCN router caches. Ioannou and Weber (2015) evaluated four dynamic approaches for identifying content popularity. Thar et al. (2015) proposed efficient forwarding and popularity-based caching for content centric network. In ICN, Jacobson et al. (2009) proposed the Named Data Network (NDN), which has the in-network caching methods. A user requests a content by broadcasting its interest to the network and then any content holder hearing the interest can respond with the requested data. The data is retrieved only in the response to one interest. SocialTube is similar as the previous works that consider video popularity in caching. However, as SocialTube is for P2P video sharing, it is different from theses works in that it additionally considers the peer upload bandwidth when making decision on video caching to improve the efficiency of sharing of cached videos. SocialTube focuses on the caching on individual peers, and caching strategies in the system level (e.g., in routers or from an ISP's point of view) can be adopted directly to further improve the efficiency of video sharing in SocialTube.

In summary, 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

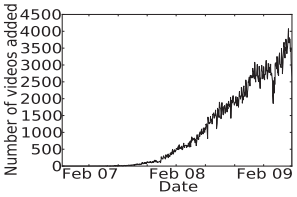


Fig. 1. # of videos added over time.

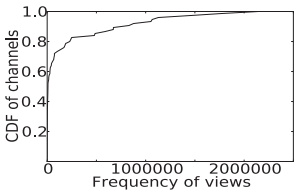


Fig. 2. View frequency of videos in different channels.

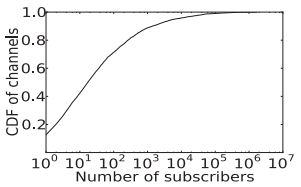


Fig. 3. # of subscribers to different channels.

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.

3 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 (Mislove et al. 2007). 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 January 2006–17 September 2010 were crawled.

3.1 Scalability

Figure 1 shows the number of videos added over time in YouTube, taken from a single crawl of YouTube by Cheng and Liu (2009). The figure shows an obvious increase in the number of videos posted over a period of two years. If the growth rate continues to increase, then the amount of bandwidth required merely for uploading videos could make operating such a site unprofitable.

Observation(O) 1. *As usage of VoD service increases, VoD providers will face rapidly increasing demand for server bandwidth.*

3.2 Channel Popularity

First, we examine the popularity of channels in YouTube. Figure 2 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 the popularity of

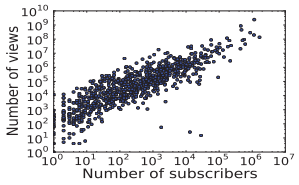


Fig. 4. Channel views vs. subscriptions.

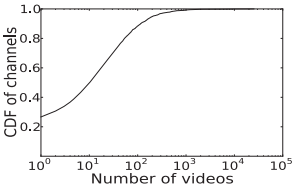


Fig. 5. # of videos per channel.

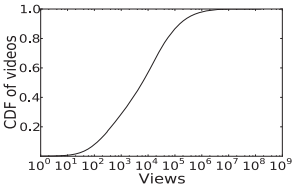


Fig. 6. # of views per video.

channels varies widely, so 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 3 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 4 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 5 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.

Observation(O) 2. *Building a P2P structure based on channels would produce reduced server load and enable efficient video retrieval for users.*

3.3 Video Popularity

We then examine the distribution of video popularity, which is measured as the number of views. Figure 6 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 7 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. (2010) 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 6, Figure 7 indicates that a small set of videos receive most of the

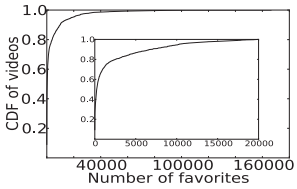


Fig. 7. # of times videos are marked as favorites.

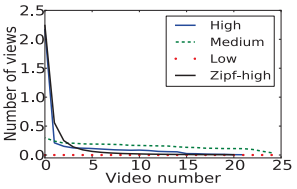


Fig. 8. Video popularity variation within channels.

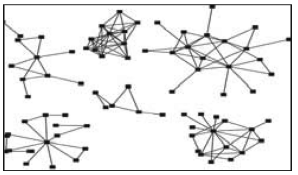


Fig. 9. A graph of channels connected by shared users.

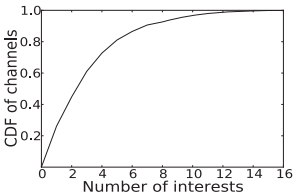


Fig. 10. # of interests in each channel.

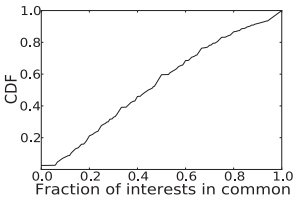


Fig. 11. Similarity between user interests and subscribed channels' interests.

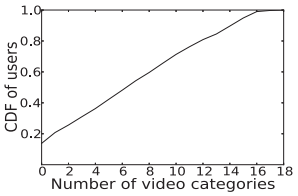


Fig. 12. Number of favorite. video interests in each channel.

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. To enhance this technique, a metric is needed to select videos for prefetching to increase the probability that the prefetched videos will be watched. Figures 6 and 7 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 8 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.

Observation(O) 3. *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.*

3.4 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 to several channels in this category. Figure 9 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 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.

Observation(O) 4. *Channels have strong clustering features that can be exploited to efficiently find providers across channels in P2P short-video sharing.*

Figure 10 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

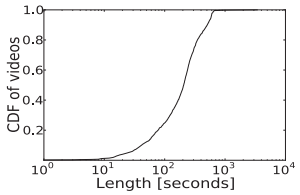


Fig. 13. Distribution of video lengths.

personal interests (denoted by I_u) by examining the categories of the user's favorite videos. We use I_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 $\frac{|I_u \cap I_c|}{|I_u|}$. Figure 11 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 12 shows the CDF of 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.

Observation(O) 5. *Channels tend to focus on a small number of video categories; users tend to subscribe to channels that match their interests.*

Figure 13 shows the length of videos on YouTube. Almost 50% of the videos are less than 200s, a very short time for users to be in an overlay. We conclude that single-video-based overlays without caching, such as PA-VoD, are not effective for short-video sharing.

4 SYSTEM DESIGN

SocialTube is designed based on the properties of the YouTube social network presented in Section 3. 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. Offline users are not part of the P2P overlay network in SocialTube. 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 to minimize startup delay and improve video availability.
- **Demand/supply-based cache management.** For each cached video, a node calculates the ratio of the total bandwidth demand from user requests to the total bandwidth supply from video holders. When a node's cache is full, it swaps out the video that has the least demand/supply ratio to maximally increase the peer bandwidth contribution and reduce the server load.

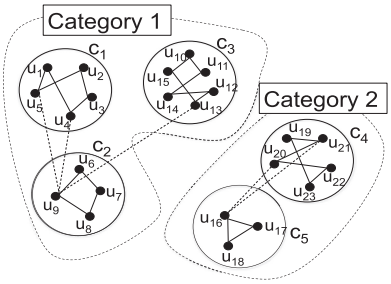


Fig. 14. A diagram of the network structure of SocialTube.

—**Reputation-based neighbor management.** It helps nodes choose peer video providers who can provide videos, encourage peer contribution and avoids "free-riding." It is advantageous in that it can prevent collusion and also prevent nodes from selectively choosing neighbors to provide videos. Specifically, it jointly considers two factors in reputation evaluation: the contributed upload bandwidth and the number of served video requesters. A node chooses nodes with high reputation scores as video providers. When a node receives an excessive number of video requests, it rejects the requests from peers with small reputation scores.

4.1 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., videos have strong correlations with each other) (Cheng et al. 2008). 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, then 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 (Time To Live) hops. Specifically, it first asks its inner-neighbor(s), u_6, along with a TTL. If u_6 does not have the video, then 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), then 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 of its inner-links 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, then 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, then 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, then 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. 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, as an existing node, 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, then they forward the request to their neighbors in c_2. If the video cannot be located in the channel overlay after TTL, then 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, then u_9 sends the request to the 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, then 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 it finds that its neighbors have left the system abruptly or have failed, then 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 its neighbors, which will update the links to the departing node.

4.2 Channel-Facilitated Prefetching

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 to reduce the playback delay of future video watching. This technique has been widely used in previous VoD systems (Cheng and Liu 2009; Huang et al. 2007). The authors of PA-VoD discussed the system states, in which prefetching can benefit system performance (Huang et al. 2007). In NetTube (Cheng and Liu 2009), a node randomly chooses the videos its neighbors (i.e., nodes that watched the same video as the node) 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. 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 rank 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 with rank k in a channel is watched next is $p_k^c = \frac{V_k^c}{V^c}$, where V_k^c is the number of views on a video of rank k and V^c is the total number of views in a channel. From Figure 8, we know that the number of views in a channel tends to follow Zipf's distribution with the characteristic exponent $s = 1$. Then, $V_k^c = (\frac{1/k}{\sum_{n=1}^{N_c} 1/n})V^c$, where N_c is the number of videos in the channel and $p_k^c = \frac{1/k}{\sum_{n=1}^{N_c} 1/n}$ is simply the standard Zipf's distribution function. For a channel with 25 videos, the probability that a single prefetch is accurate (p_k^c) equals 26.2% ($p_1^c = \frac{1/1}{\sum_{n=1}^{25} 1/n} = 0.262$). Since users are often able to prefetch 3-4 videos during a single video playback, the prefetch accuracy rises to 54.6% ($p_1^c + p_2^c + p_3^c + p_4^c = \frac{1/1}{\sum_{n=1}^{25} 1/n} + \frac{1/2}{\sum_{n=1}^{25} 1/n} + \frac{1/3}{\sum_{n=1}^{25} 1/n} + \frac{1/4}{\sum_{n=1}^{25} 1/n} = 0.546$).

4.3 Demand/supply-based Cache Management

Nodes store their previously watched videos on their caches to boost video availability in the system. For video v_k, when its demand of download bandwidth from all viewers (denoted by B_k^d) is greater than the accumulated upload bandwidth from all video holders (denoted by B_k^u), part of the video requests should be served by the video server. In a P2P VoD system, the goal of adopting P2P techniques is to reduce the server load by leveraging the peer upload bandwidth to support users' video playback. To reduce the server load, we hope to keep the B_k^d of each video no more than its B_k^u. However, this requirement is difficult to meet due to the limited size of node cache. Thus, we consider the cached video replacement scheme to meet this requirement for most demanded videos.

The cache replacement policy selects a cached video to be replaced with the current playing video when the cache is full. Common cache replacement strategies include the Least Recently Used (LRU) scheme (Podlipnig and Böszörmenyi 2003) and the First-in-First-out (FIFO) scheme (Krishnappa et al. 2011). However, both schemes fail to maximally reduce the server load, since popular videos are likely to be replaced in a node's cache, and then the node's upload bandwidth is wasted as unpopular cached videos are rarely requested by other users. However, simply replacing a less popular video also cannot handle the problem effectively. Because a more popular

video usually has more viewers and hence more video holders at a certain time after its creation, its download bandwidth demand from all requests may be less than the aggregated upload bandwidth from video holders. For a popular cached video, if its B_k^d is greater than B_k^u, replacing this video in a holder's cache will further decrease the video's availability and upload bandwidth from video holders, and thus increases the server load. To effectively maximize the peer bandwidth contribution (i.e., minimize the server load), we then try to replace the video whose B_k^d is less than B_k^u and estimate B_k^d based on the video popularity.

We propose a demand/supply-based cache management scheme that considers both video popularity and peer upload bandwidth. The server ranks all videos according to their popularity in descending order. As stated in Yu et al. (2006) and Zhou and Xu (2002), the number of video views in the VoD system tends to follow the Zipf's distribution. Given characteristic exponent $s = 1$, we can derive the probability of attracting a new viewer for video v_k: $p_k^s = \frac{1/k}{\sum_{n=1}^{N_s} 1/n}$, where N_s is the total number of videos and k is the popularity rank of v_k in the system. We use U_a to denote the number of active users in the system. The total number of new views for video v_k (V_k^s) can be estimated by: $V_k^s = U_a \times p_k^s$. Suppose the highest video bitrate for video v_k is b_k, to provide a fluent playback of video v_k for all potential viewers, the lower bound of accumulated download bandwidth from both peers and server B_k^d is: $B_k^d = b_k \times V_k^s$. The upload bandwidth of v_k from all peers is $B_k^u = \sum_{i \in \Phi_k} b_i^u$, where Φ_k is the set of peers that have store video v_k in their caches, and b_i^u is the available upload bandwidth of peer u_i.

We use R_k to denote the ratio of bandwidth demand to bandwidth supply: $R_k = B_k^d / B_k^u$. R_k is an indicator of whether current peer contribution is sufficient to reduce the server load. When $R_k > 1$, the bandwidth demand from new watchers surpasses the potential upload capacity from peers, and the excessive bandwidth demand will be provided by the server. The video with the least $R_k > 1$ in a cache has the most extra bandwidth supply than the bandwidth demand. Thus, when a node's cache is full, it swaps out the video with the least R_k value to maximize the probability of using peer upload bandwidth and minimize the server load. For each video, the server calculates the bandwidth demands from all video requests and accumulated bandwidth contribution from all video holders, and calculates the R_k value. The R_k value for each video is then broadcasted to all nodes in the system.

4.4 Reputation-based Neighbor Management

Existing works on VoD (Cheng and Liu 2009; Huang et al. 2007; Wang et al. 2008) generally assume that peers are altruistic and willing to contribute their bandwidth resources to support other peers' video playback activities. However, it is a common case that in P2P systems, some selfish nodes download much more video packets than the video packets they upload, which leads to "free-riding." Selfish peers may deliberately limit their upload bandwidth or reject video requests. Also, considering node capacity heterogeneity, a node may not able to handle all requests it received due to its actually limited upload bandwidth capacity. A scheme is needed for nodes to choose peer video providers who can provide videos and also encourage peer contribution to reduce the server load. Thus, in this article, we propose a reputation-based neighbor management scheme. In the scheme, nodes evaluate the video provision performance of their neighbors and based on the evaluations, each node gains a reputation score, which represents the capacity and willingness of the node in providing requested videos, for video provider selection.

We explain the method to calculate the reputation scores below. A simple method is to set a node's reputation proportional to its total contributed upload bandwidth. However, this method cannot prevent collusion in which a collective of nodes falsely report high contribution for each other. Also, it cannot prevent nodes from selectively choosing neighbors (e.g., close friends) to

provide videos. To avoid the problems, we jointly consider two factors in reputation evaluation: the contributed upload bandwidth and the number of served video requesters.

If node u_k has downloaded more video packets from node u_i than that from node u_j within a time period (e.g., a week), then node u_k regards node u_i as a more reliable contributing neighbor and tends to ask u_i instead of u_j for videos. In this case, we say that node u_k prefers node u_i over node u_j. If node u_k requested a video from node u_i but was rejected by node u_i, and node u_k never requested videos from node u_j, then node u_k prefers node u_j than node u_i. Suppose there are totally U_s nodes in the system. Each node u_k maintains a preference vector $\mathcal{V}_k^p = \{u^1, u^2, u^3, \ldots u^{U_s-1}\}$, in which u^i means node u in rank i, that orders other nodes it has interacted with in the descending order of the number of packets it has received from them (i.e., node u_k's preferences over them). That is, node u^1 (ranked 1) is u_k's most preferable node, and node u^2 (ranked 2) is u_k's second preferable node. We use $\sigma_k(i)$ to represent the rank of node u_i in node u_k's preference vector $\mathcal{V}_k^p$; $\sigma_k(i) < \sigma_k(j)$ means that node u_k prefers node u_i over node u_j. For example, if node u_k has downloaded 10MB, 8MB, and 6MB from nodes u_i, u_j, and u_l, u_k's preference vector is $\mathcal{V}_k^p = \{u_i, u_j, u_l\}$, and the ranks of each node in $\mathcal{V}_k^p$ are: $\sigma_k(i) = 1$, $\sigma_k(j) = 2$, and $\sigma_k(l) = 3$.

Each node periodically reports its preference vector to the server. The server then creates a $U_s \times U_s$ matrix, and each element in this matrix N_{ij}^p indicates the total number of nodes that prefer node i over j. It is calculated by:$N_{ij}^p = \sum_{\mathcal{V}_k^p \in S} I\{\sigma_k(i) < \sigma_k(j)\}, i \neq j$, where S is the set including all received preference vectors, and $I\{\cdot\}$ is an indicator function. When $\sigma_k(i) < \sigma_k(j)$, $I\{\sigma_k(i) < \sigma_k(j)\} = 1$.

As we have the collective information of the preference between each pair of nodes, the next step is to estimate the reputation of each node in the system. The principle is that the nodes that are preferred by more other nodes should have higher reputations. Specifically, the reputation of node u_i, denoted by r_i is calculated by: $r_i = \frac{1}{U_s-1} \sum_{i \neq j} N_{ij}^p$. The server then broadcasts the reputation scores to all nodes in the system. Each node can support a limited number of peers (denoted by N_i^r) concurrently, that is, node u_i can send video packets to up to N_i^r peers using its upload bandwidth capacity. When the number of video requests received by node u_i is larger than N_i^r, node u_i will reject the video request from a peer with the smallest reputation. The proposed reputation-based neighbor management scheme is resilient against sybil attack and collusion, as a small number of malicious nodes cannot greatly boost a node's reputation by giving it high reputation scores. It helps create a cooperative environment for P2P video sharing, and nodes will be encouraged to not only provide high upload bandwidth but also serve as more nodes as possible. To avoid low QoS provided by peers, when a node chooses video providers from peers, it will choose peers with high reputation values. If it cannot find peers with reputation values larger than a certain threshold, then it resorts to the server for the video contents.

4.5 Advantages and Limitations of SocialTube

As we indicated, SocialTube saves the maintenance overhead of NetTube. NetTube forms 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 an 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.

However, SocialTube still has limitations that need to be addressed in the future work. First, its maintenance overhead is still relatively high. Each node needs to maintain links to inner-neighbors and inter-neighbors. Second, the performance in churn needs to improve. If all of a node's links fail, then the node cannot take advantage of SocialTube's service. Third, adaptive streaming can be adopted to further improve the performance. Fourth, SocialTube assumes that a certain number of nodes stay in the system for a relatively long time so the videos can be shared between nodes. The situation that the majority nodes do not stay in the system long needs to be handled. Fifth, the video provision performance from peers cannot be guaranteed and additional strategy is needed for such a guarantee.

5 PERFORMANCE EVALUATION

To measure the performance of SocialTube in comparison with PA-VoD (Huang et al. 2007) and NetTube (Cheng and Liu 2009), we conducted trace-driven experiments using PeerSim (Pee 2016), a well-documented event-driven simulator, and PlanetLab (pla 2016), a real-world testbed for distributed networks. The simulator 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 3. We used the video length distribution, video popularity distribution, and video category distribution directly in our simulations. The download and upload bandwidths of nodes follow the data in Cheng and Liu (2009). NetTube (Cheng and Liu 2009) 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, we let 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 by a Poisson distribution (Yu et al. 2006) 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 10min for overlay maintenance. Other default experimental settings are shown in Table 1.

In the 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 20min. 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:

Table 1. 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	320kbps
Server bandwidth	500mbps

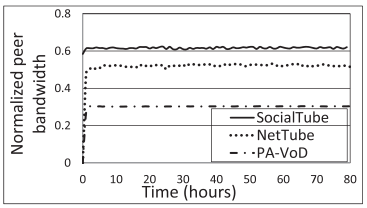


Fig. 15. Bandwidth contribution of peers.

— *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.

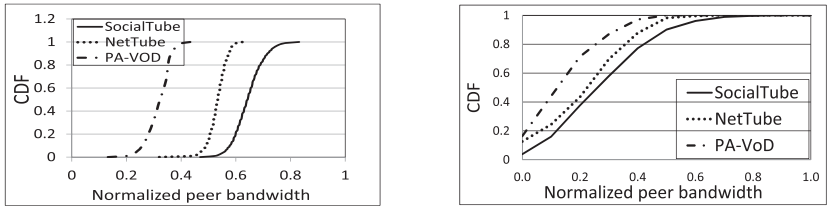
The metric that the prefetching strategy affects is only startup delay, which was measured with and without the prefetching strategy in our experiments. Prefetched chunks of short videos are very small in size (about 150KB), so the prefetching cost (including download bandwidth and storage) can be negligible.

5.1 Server Bandwidth Reduction

First, we examine the effect of SocialTube at reducing server load. Figure 15 shows the normalized bandwidth cocontribution of peers over time on PeerSim. We see that SocialTube, NetTube, and PA-VoD reduce server bandwidth by 44%, 36%, and 18%, respectively. Since the 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.

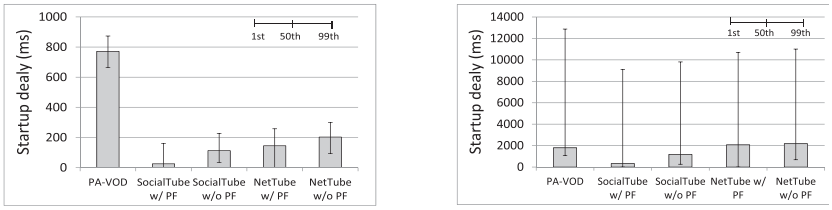
SocialTube forms same-channel users into a lower-level overlay and further forms common-interest users into a higher-level overlay, so users have a high probability to find chunk providers in the overlays. NetTube clusters nodes based on each video they watch. Though nodes watching a video are likely 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.

Figure 16(a) and Figure 16(b) show the CDF of the normalized bandwidth contribution by peers on PeerSim and PlanetLab, respectively. We see from Figure 16(a) that 50% of nodes in SocialTube,



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

Fig. 16. The CDF of bandwidth contribution of peers.



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

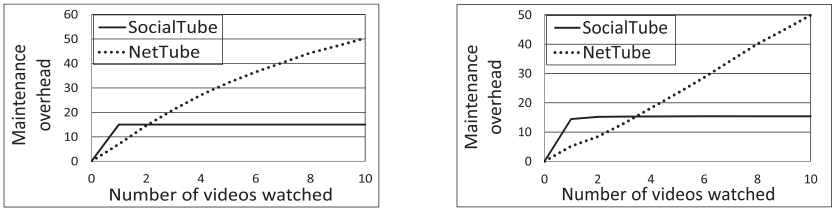
Fig. 17. Startup delay and effectiveness of prefetching.

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. The results are caused by the same reasons as explained in Figure 15. We obtain the same observations as those in Figure 16(b) due to the same reasons. The results in both figures demonstrate the success of SocialTube in reducing the dependency on the server for video retrieval, and confirm that SocialTube outperforms NetTube and PA-VoD in reducing server bandwidth.

5.2 Startup Delay

We then examine the effect of SocialTube at improving QoS for users. Figures 17(a) and 17(b) show the startup delay of NetTube and SocialTube with and without prefetching (PF), and of PA-VoD on PeerSim and PlanetLab. The figures show the 1st, 50th, and 99th percentiles of the startup delays. The size of a chunk was set to 100KB (Cheng and Liu 2009). For NetTube, users prefetch the first chunks of three videos randomly from their neighbors' watched videos. For SocialTube, users prefetch the first chunks of three top popular videos within the channel it currently is watching. PA-VoD does not have 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 relies 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



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

Fig. 18. Overlay maintenance overhead.

NetTube's. This is because SocialTube's channel-based prefetching targets the videos that users are most likely to watch next; NetTube's strategy randomly prefetches from neighbors' watched videos. The experimental results confirm the effectiveness of SocialTube's channel-facilitated popularity-based prefetching strategy.

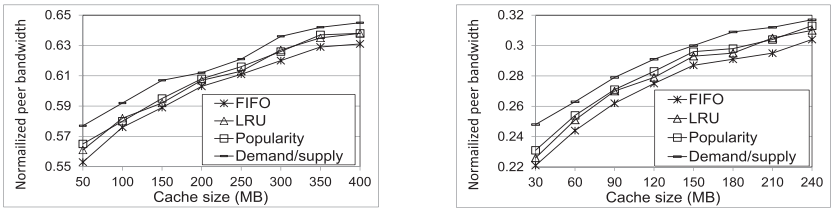
We use simulation results to show the relative performance of different methods. Due to our simulation setting including high available bandwidth of nodes, the startup delay is within 1 second. In the PlatnetLab real-world testbed, the 99th startup delay is 13s.

5.3 Overlay Maintenance Overhead

Figures 18(a) and 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, NetTube has 35 more links per user than SocialTube. This result confirms that the 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) confirm that SocialTube demands significantly lower maintenance overhead than NetTube.

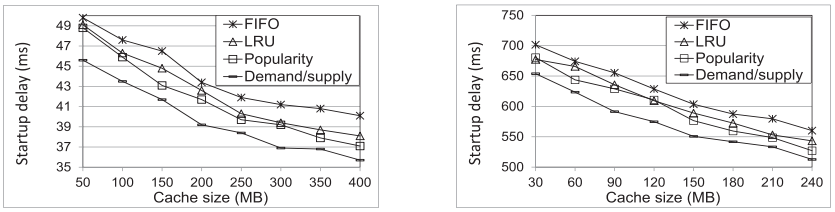
5.4 Effectiveness of the Demand/supply-based Cache Management

In this experiment, we varied the cache size of each node from 50MB to 400MB with an increase step of 50MB on PeerSim, and from 30MB to 240MB with an increase step of 30M on PlanetLab. We compared our demand/supply-based cache management scheme (denoted by *Demand/supply*) with the *FIFO* (Krishnappa et al. 2011), *LRU* (Podlipnig and Böszörmenyi 2003), and *Popularity* cache management schemes on SocialTube. When a node's cache is full, *FIFO* selects the video that is first downloaded and replaced it with the new video; *LRU* replaces the video that is least recently visited; and *Popularity* replaces the video that is least popular. Figures 19(a) and 19(b) show the average peer bandwidth contribution of all video requests versus the cache size of each node on PeerSim and PlanetLab, respectively. We see that the peer bandwidth contribution of all three methods increases as the cache size increases. This is because a node can store more videos with a larger cache, which increases the availability of videos from peers and reduces the traffic demand on the server. We also see that the peer bandwidth contribution follows *Demand/supply*>*Popularity*>*LRU*>*FIFO*. In *Demand/supply*, videos with small ratios of bandwidth demand to bandwidth supply are replaced by new videos when the cache is full, so the videos that have high upload bandwidth demand and low bandwidth supply from peers are kept in node caches. Then, peers have a higher probability to serve video requests and contribute their bandwidth capacity. In *FIFO*, the first downloaded



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

Fig. 19. The bandwidth contribution of peers with different cache management schemes.

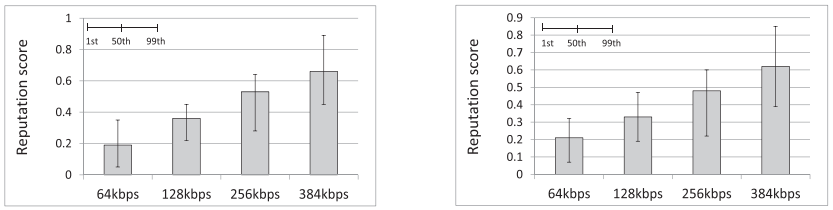


(a) The PeerSim simulator. (b) PlanetLab real-world testbed.

Fig. 20. Startup delay with different cache management schemes.

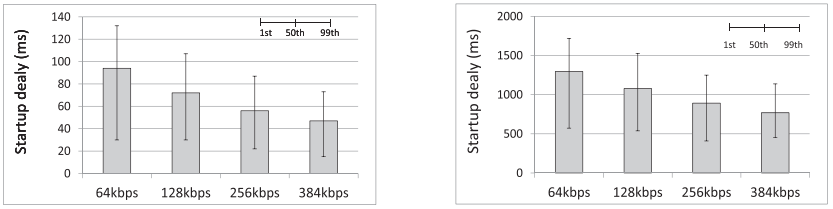
videos are replaced by the new videos when the cache is full, and then a node may need to visit many peers to find a video provider. It increases the probability of resorting to the server after several failures of peer video provider seeking. Thus, *FIFO* leads to the smallest peer bandwidth contribution. As observed from Figure 8, users tend to watch popular videos, so the least recently visited video in a node's cache is likely to be a less popular video. Thus, *LRU* is able to maintain the availabilities of popular videos and outperforms *FIFO* with increased peer bandwidth contribution. *Popularity* generates higher peer bandwidth contribution than *LRU* as it explicitly replaces the least popular video and thus maintains the availability of popular videos in peers. However, as *LRU* and *Popularity* do not consider the user upload bandwidth of a video, replacing a video whose cumulated user upload bandwidth is smaller than its bandwidth demand from all requests may decrease the video's availability from peers. Thus, they generate smaller peer bandwidth contribution than *Demand/supply*.

Figures 20(a) and 20(b) show the startup delay of all video requests versus the cache size of each node on PeerSim and PlanetLab, respectively. We see that as the cache size increases, the startup delay decreases due to the reason that the requested videos have a high probability to be stored in the caches of peers. We also see that the startup delay follows *Demand/supply*<*Popularity*<*LRU*<*FIFO* due to the same reasons in Figures 19(a) and 19(b). *Demand/supply* generates smaller startup delay than other methods, because it tries to keep the videos that have high upload bandwidth demand and low bandwidth supply in node caches, so nodes are likely to contribute upload bandwidth to video requests. These experimental results confirm the effectiveness of the demand/supply-based cache management scheme in reducing server load and increasing peer contribution. The difference between simulator and testbed is caused by the reason we explained in the above. Also, though the order of magnitude of the startup delay can be ignored from an end user's point of view, here we intend to show the relative performance between the different methods. In practice, there is now a 20–30s startup delay in YouTube, which means there is still a need to reduce the startup delay.



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

Fig. 21. Reputation scores for different group of nodes.



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

Fig. 22. Startup delay for different group of nodes.

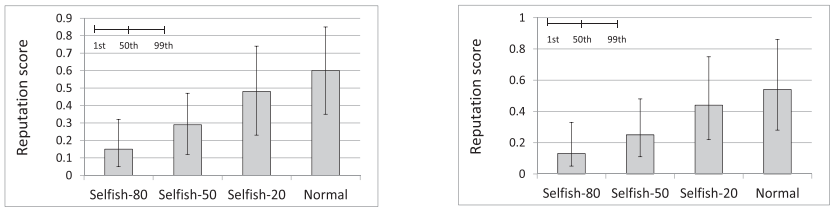
5.5 Effectiveness of the Reputation-based Neighbor Management

In this experiment, we varied the maximum upload bandwidth capacities of all nodes in the system, which are specified by the users themselves to indicate the upload bandwidth they would like to contribute in real applications. Specifically, 25% of nodes have upload capacity of 64kbps, 25% of nodes have 128kbps, 25% of nodes have 256kbps, and 25% have 384kbps on both PeerSim and PlanetLab (Guo et al. 2008). Each node can send video packets to up to 4 peers concurrently. We had a warmup period of 10 sessions to accumulate reputation scores for nodes. Each node reports the cumulated size of video packets it has downloaded from every other node to the server after each session, and the serve calculates and broadcasts the reputation score of each node to all nodes at the beginning of the next session.

Figure 21(a) and Figure 21(b) show the 1st, 50th, and 99th percentiles of the reputations scores of peers at the end of the experiments for each group of nodes with the same maximum upload capacity on PeerSim and PlanetLab, respectively. We see that the nodes with higher upload capacity generally get higher reputation scores, because a node is more likely to be selected as a favorite node if it can send more video packets to its neighbors. These figures show the effectiveness of the reputation scores in reflecting the bandwidth contributions of nodes.

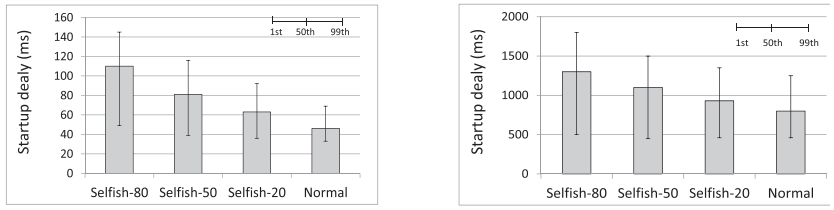
Figures 22(a) and 22(b) show the 1st, 50th, and 99th percentiles of the startup delay of peers on PeerSim and PlanetLab, respectively. We see that the group of nodes with low upload bandwidth capacity have high startup delay. This is because these nodes are not regarded as contributing nodes by others and have relatively low reputation scores. Then, their video requests are likely to be rejected by peers and they need to visit more nodes in seeking video providers. As a result, to reduce the video startup delay, nodes are encouraged to increase their upload bandwidth capacities.

Next, we set a maximum upload bandwidth capacity of 384kbps to all nodes on both PeerSim and PlanetLab. We randomly chose 10% of all nodes to form each of the groups denoted by *selfish-20*, *selfish-50*, and *selfish-80*, where *selfish-x* means nodes in this group have x% probability to reject video requests when they still have capacity to serve. We use *normal nodes* to denote the remaining 70% of nodes that serve all video requests using their maximum bandwidth capacity. We had



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

Fig. 23. Reputation scores for different group of nodes.



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

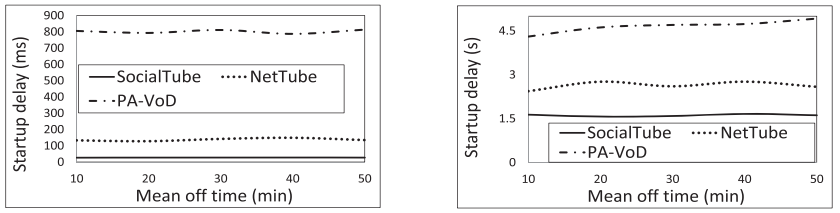
Fig. 24. Startup delay for different group of nodes.

a warmup period of 10 sessions to accumulate reputation scores for nodes. Figures 23(a) and 23(b) show the 1st, 50th, and 99th percentiles of the reputations scores of peers at the end of the experiment for each group of nodes on PeerSim and PlanetLab, respectively. We see that the reputations scores follow *Normal*>*Selfish-20*>*Selfish-50*>*Selfish-80*. Normal nodes have higher reputation scores as they serve more requests and send more video packets to its neighbors. *Selfish-80* nodes get lowest reputation scores due to the reason that they are more likely to reject video requests. These figures show that the reputation scores can reflect node selfish degree in contributing its bandwidth resource.

Figures 24(a) and 24(b) show the 1st, 50th, and 99th percentiles of the startup delay of peers on PeerSim and PlanetLab, respectively. We see that the startup delay follows *Selfish-80*>*Selfish-50*>*Selfish-20*>*Normal*. That is, the group of nodes that are more likely to reject video requests have high startup delay. This is because that nodes with a higher probability to reject video requests have relatively lower reputation scores, and their video requests are more likely to be rejected by other peers. Thus, the selfish nodes who deliberately reject video requests are punished by experiencing a relatively long video startup delay.

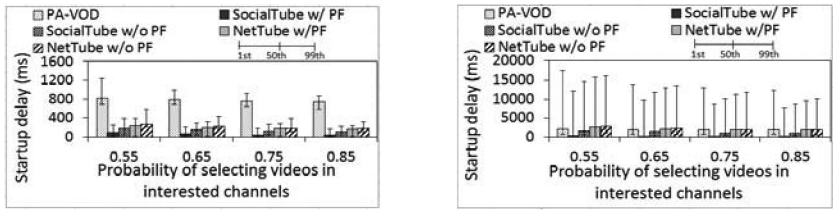
5.6 Performance in Various Churns

Recall that user off time is the time period between two successive joins and it follows Poisson distribution in our experiments. By varying the mean user off time periods from 10 to 50min, we simulated scenarios where users join and leave the system at different arrival and departure rates. A shorter off time period means a higher churn rate. Figures 25(a) and 25(b) show the average video startup delay with different off time on PeerSim and PlanetLab, respectively. The startup delay for PA-VoD increases when the mean off time period increases, while that of SocialTube and NetTube does not change much. This is because longer off time leads to fewer concurrent nodes in the system and fewer available nodes as video providers, so peers need to download more videos from the server, which overloads the server and incurs longer latency. As nodes share cached videos, SocialTube and NetTube maintain relatively stable peer contribution, so their video startup delays



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

Fig. 25. Video startup delay in various churns.



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

Fig. 26. Video startup delay with different probabilities of choosing videos in interested channels.

keep nearly constant. We see that the startup delay follows SocialTube<NetTube<PA-VoD due to the same reasons as in Figure 17(b).

5.7 Performance with Different Probabilities of Choosing Videos in Interested Channels

Figures 26(a) and 26(b) show the 1st, 50th, and 99th percentiles of the startup delay with different probabilities of choosing videos in interested channels of different methods. We see that the relative performance between different methods follow that in Figures 17(a) and 17(b) due to the same reasons. We also see that as the probability of choosing videos in interested channels increases, the startup delay of SocialTube decreases. This is because SocialTube is designed based on the observation that users tend to watch videos in their interested channels. As the probability of choosing videos in interested channels increases, the startup delay of NetTube and PA-VOD also decreases. In PA-VOD, when a user requests a video, the server directs the request to several other users currently watching the video. NetTube forms users watching the same video into an overlay (i.e., per-video overlay). Therefore, if a user has a higher probability of choosing videos in its interested channels, it has a higher probability to find the video it wants to watch from its connected neighbors in both PA-VOD and NetTube. As a result, the startup delay decreases.

6 CONCLUSIONS

This work leverages the YouTube social network to develop an enhanced short-video sharing system. In YouTube social network, users subscribed to a channel tend to view the videos in the channel. Based on the properties of YouTube social network, 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 it reduces overlay maintenance costs over previous per-video overlays. SocialTube also incorporates several

strategies to further improve performance. Trace-driven simulations and the experiments on the PlanetLab real-world testbed have proven the superior performance of SocialTube over previous systems, as well as the effectiveness of its incorporated schemes. In our future work, we will study the impact of the different number of links per node on the video-sharing performance.

REFERENCES

PeerSim: A Peer-to-Peer Simulator. 2016. PeerSim P2P Simulator. Retrieved from http://peersim.sourceforge.net/.

Planet Lab. 2016. Planetlab: An open platform for developing, deploying, and accessing planetary-scale services. Retrieved from http://www.planet-lab.org/.

PPLive. 2016. PP Video. Retrieved from http://www.pplive.com.

PPStream. 2016. PPS Video. Retrieved from http://www.ppstream.com.

UUSee. 2016. UUSee Website. Retrieved from http://www.uusee.com.

Your Tube, Whose Dime? 2016. Forbes Welcome. Retrieved from http://www.forbes.com/2006/04/27/video-youtube-myspace_cx_df_0428video.html.

YouTube costs Google $2 million per day. 2016. YouTube costs Google $2 million per day. Retrieved from http://www.inquisitr.com/24740/youtube-costs-google-2-million-per-day/.

YouTube Press Statistics. 2016a. Press-Youtube. Retrieved from http://www.youtube.com/t/press_statistics.

YouTube Press Timeline. 2016b. Press-Youtube. Retrieved from http://www.youtube.com/t/press_timeline.

A. Afrasiabi Rad and M. Benyoucef. 2014. Similarity and ties in social networks a study of the youtube social network. *J. Info. Syst. Appl. Res.* 7, 4 (2014), 14.

S. Annapureddy, C. Gkantsidis, P. R. Rodriguez, and L. Massoulie. 2006. Providing video-on-demand using peer-to-peer networks. In *Proceedings of the IPTV Workshop in WWW*.

M. Arantes, F. Figueiredo, and J. M. Almeida. 2016. Understanding video-ad consumption on youtube: A measurement study on user behavior, popularity, and content properties. In *Proceedings of the ACM Conference on Web Science*.

A. Brodersen, S. Scellato, and M. Wattenhofer. 2012. Youtube around the world: Geographic popularity of videos. In *Proceedings of the WWW*.

T. Broxton, Y. Interian, J. Vaver, and M. Wattenhofer. 2013. Catching a viral video. *Journal of Intelligent Information Systems* 40, 2 (2013), 241–259.

V. Burger, G. Darzanos, I. Papafili, and M. Seufert. 2015. Trade-off between QoE and operational cost in edge resource supported video streaming. In *Proceedings of the 3PGCIC*.

P. Casas, P. Fiadino, A. Bar, A. D'Alconzo, A. Finamore, and M. Mellia. 2014. YouTube all around: Characterizing youtube from mobile and fixed-line network vantage points. In *Proceedings of the EuCNC*.

M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. 2003. SplitStream: High-bandwidth multicast in cooperative environments. In *Proceedings of the SOSP*.

M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. 2007. I tube, you tube, everybody tubes: Analyzing the worlds largest user generated content video system. In *Proceedings of the IMC*.

W. Chang and J. Wu. 2015. Social VoD: A social feature-based P2P system. In *Proceedings of the ICPP*.

G. Chatzopoulou, C. Sheng, and M. Faloutsos. 2010. A first step towards understanding popularity in youtube. In *Proceedings of the INFOCOM*.

B. Cheng, L. Stein, H. Jin, X. Liao, and Z. Zhang. 2008. GridCast: Improving peer sharing for P2P VoD. *ACM TMCCA* 4, 4 (2008), 26.

X. Cheng, C. Dale, and J. Liu. 2008. Statistics and social network of youtube videos. In *Proceedings of the IWQoS*. 229–238.

X. Cheng and J. Liu. 2009. NetTube: Exploring social networks for peer-to-peer short-video sharing. In *Proceedings of the INFOCOM*.

X. Cheng, J. Liu, and C. Dale. 2013. Understanding the characteristics of internet short-video sharing: A YouTube-based measurement study. *TMM* 15, 5 (2013), 1184–1194.

Y. Ding, Y. Yang, and L. Xiao. 2012. Multisource video on-demand streaming in wireless mesh networks. *TON* 20, 6 (2012), 1800–1813.

P. Gill, M. Arlitt, Z. Li, and A. Mahanti. 2007. YouTube traffic characterization: A view from the edge. In *Proceedings of the ACM IMC*.

Y. Guo, C. Liang, and Y. Liu. 2008. AQCS: Adaptive queue-based chunk scheduling for P2P live streaming. In *Proceedings of the NETWORKING*. 433–444.

C. Ho, S. Lee, and J. Yu. 2010. Cluster-based replication For P2P-based video-on-demand service. In *Proceedings of the ICEIE*.

T. Hossfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen. 2012. Initial delay vs. interruptions: Between the devil and the deep blue sea. In *Proceedings of the QoMEX Workshop*.

H. Hu, Y. Wen, T. Chua, J. Huang, W. Zhu, and X. Li. 2016. Joint content replication and request routing for social video distribution over cloud CDN: A community clustering method. *IEEE Trans. Circ. Syst. Video Technol.* 26, 7 (2016), 1320–1333.

C. Huang, J. Li, and K. W. Ross. 2007. Can internet video-on-demand be profitable? In *Proceedings of the SIGCOMM*.

Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang. 2008. Challenges, design and analysis of a large-scale P2P VoD system. In *Proceedings of the SIGCOMM*.

A. Ioannou and S. Weber. 2015. Exploring content popularity in information-centric networks. *J. China Commun.* 12, 7 (2015), 12–22.

V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. 2009. Networking named content. In *Proceedings of the Conference on Emerging Networking Experiments and Technologies*.

D. Krishnappa, S. Khemmarat, L. Gao, and M. Zink. 2011. On the feasibility of prefetching and caching for online TV services: A measurement study on hulu. In *Passive and Active Measurement*. 72–80.

D. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen. 2015. Cache-centric video recommendation: An approach to improve the efficiency of YouTube caches. *ACM Trans. Multimedia Comput. Commun. Appl. (TOMM)* 11, 4 (2015), 48.

F. Lehrieder, S. Oechsner, T. Hoßfeld, Z. Despotovic, W. Kellerer, and M. Michel. 2010. Can p2p-users benefit from locality-awareness?. In *Proceedings of the P2P*.

B. Li, M. Ma, Z. Jin, and D. Zhao. 2012. Investigation of a large-scale P2P VoD overlay network by measurements. *Peer-to-Peer Network. Appl.* 5, 4 (2012), 398–411.

X. Liao, H. Jin, Y. Liu, L. Ni, and D. Deng. 2006. AnySee: Peer-to-peer live streaming. In *Proceedings of the INFOCOM*.

Y. Lin and H. Shen. 2017. CloudFog: Leveraging fog to extend cloud gaming for thin-client MMOG with high quality of experience. *IEEE Trans. Parall. Distrib. Syst. (TPDS)* 28, 2 (2017), 431–445.

B. Liu, Y. Cui, B. Chang, B. Gotow, and Y. Xue. 2009. BitTube: Case study of a web-based peer-assisted video-on-demand system. In *Proceedings of the ISM*. 242–249.

J. Liu, S. G. Rao, B. Li, and H. Zhang. 2008. Opportunities and challenges of peer-to-peer internet video broadcast. In *Proceedings of the IEEE*.

T. Locher, S. Schmid, and R. Wattenhofer. 2006. eQuus: A provably robust and locality-aware peer-to-peer system. In *Proceedings of the P2P*.

N. Magharei and R. Rejaie. 2007. PRIME: Peer-to-peer receiver-drIven MEsh-based streaming. In *Proceedings of the INFOCOM*.

N. Magharei, R. Rejaie, I. Rimac, V. Hilt, and M. Hofmann. 2014. ISP-friendly live P2P streaming. *TON* 22, 1 (2014), 244–256.

B. Mathieu, P. Paris, G. Guelvouit, and S. Rouibia. 2010. A secure and legal network-aware P2P VoD system. In *Proceedings of the ICIW*.

A. Mislove, M. Marcon, K. Gummadi, P. Dreschel, and B. Bhattacharjee. 2007. Measurement and analysis of online social networks. In *Proceedings of the IMC*.

H. Nam, K. Kim, and H. Schulzrinne. 2016. QoE matters more than QoS: Why people stop watching cat videos. In *Proceedings of the INFOCOM*.

Y. Nicolas, D. Wolff, D. Rossi, and A. Finamore. 2013. I tube, youtube, P2Ptube: Assessing ISP benefits of peer-assisted caching of YouTube content. In *Proceedings of the P2P*.

V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. 2005. Chainsaw: Eliminating trees from overlay multicast. In *Proceedings of the IPTPS*.

S. Podlipnig and L. Böszörmenyi. 2003. A survey of web cache replacement strategies. *CSUR* 35, 4 (2003), 374–398.

D. Rossi and G. Rossini. 2012. On sizing CCN content stores by exploiting topological information. In *Proceedings of the INFOCOM Workshop*.

M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld, and P. Tran-Gia. 2015. A survey on quality of experience of HTTP adaptive streaming. *IEEE Commun. Surveys Tutor.* 17, 1 (2015), 469–492.

H. Shen, Z. Li, Y. Lin, and J. Li. 2014. SocialTube: P2P-assisted video sharing in online social networks. *IEEE Trans. Parall. Distrib. Syst. (TPDS)* 25, 9 (2014), 2428–2440.

K. Thar, T. Z. Oo, C. Pham, S. Ullah, D. H. Lee, and C. S. Hong. 2015. Efficient forwarding and popularity-based caching for content centric network. In *Proceedings of the International Conference on ICOIN*.

D. A. Tran, K. A. Hua, and T. Do. 2003. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proceedings of the INFOCOM*.

J. Venkataraman and P. Francis. 2006. Chunkyspread: Multi-tree unstructured peer-to-peer multicast. In *Proceedings of the IPTPS*.

V. Venkataraman, K. Yoshida, and P. Francis. 2006. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *Proceedings of the ICNP*.

F. Wamser, P. Casas, M. Seufert, C. Moldovan, P. Tran-Gia, and T. Hossfeld. 2016. Modeling the youtube stack: From packets to quality of experience. *J. Comput. Networks* 109 (2016), 211–224.

J. Wang, C. Huang, and J. Li. 2008. On ISP-friendly rate allocation for peer-assisted VoD. In *Proceedings of the ACM Multimedia*.

K. Wang and C. Lin. 2009. Insight into the P2P-VoD system: Performance modeling and analysis. In *Proceedings of the ICCCN*.

X. Wang, T. Kwon, Y. Choi, H. Wang, and J. Liu. 2013. Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users. *IEEE Wireless Communications* 20, 3 (2013), 72–79.

M. Wattenhofer, R. Wattenhofer, and Z. Zhu. 2012. The youtube social network. In *Proceedings of the ICWSM*.

C. Wu, B. Li, and S. Zhao. 2008. Multi-channel live P2P streaming: Refocusing on servers. In *Proceedings of the INFOCOM*.

S. Yi, C. Li, and Q. Li. 2015. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the Conference on Mobile Big Data*.

H. Yoganarasimhan. 2012. Impact of social network structure on content propagation: A study using YouTube data. *Quantitative Marketing and Economics* 10, 1 (2012), 111–150.

H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng. 2006. Understanding user behavior in large-scale video-on-demand systems. *SIGOPS Oper. Syst. Rev.* 40, 4 (2006), 333–344.

A. Zambelli. 2009. IIS smooth streaming technical overview. *Microsoft Corporation* 3 (2009), 40.

X. Zhang, J. Liu, B. Li, and T. Yum. 2005. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of the INFOCOM*.

X. Zhou and C. Xu. 2002. Optimal video replication and placement on a cluster of video-on-demand servers. In *Proceedings of the ICPP*.

Y. Zhou, T. Fu, and D. Chiu. 2012. A unifying model and analysis of P2P VoD replication and scheduling. In *Proceedings of the INFOCOM*. 1530–1538.

Y. Zhou, T. Fu, and D. Chiu. 2013. On replication algorithm in P2P VoD. *TON* 21, 1 (2013), 233–243.

J. Zhu, D. Chan, M. S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi. 2013. Improving web sites performance using edge servers in fog computing architecture. In *Proceedings of the SOSE*.

Y. Zhu. 2012. Evaluating mesh-based P2P video-on-demand systems. In *Proceedings of the IPDPS*.

Received March 2016; revised July 2017; accepted August 2017