Watching User Generated Videos with Prefetching

Samamon Khemmarat[†], Renjie Zhou[‡], Lixin Gao[†], Michael Zink[†]

[†] Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, USA
{khemmarat,lgao,zink}@ecs.umass.edu

[‡] College of Computer Science and Technology
Harbin Engineering University, Harbin, China
renjie_zhou@hrbeu.edu.cn

ABSTRACT

Even though user generated video sharing sites are tremendously popular, the experience of the user watching videos is often unsatisfactory. Delays due to buffering before and during a video playback at a client are quite common. In this paper, we present a prefetching approach for user-generated video sharing sites like YouTube. We motivate the need for prefetching by showing that video playbacks of videos on YouTube is often unsatisfactory and introduce a series of prefetching schemes: the conventional caching scheme, the search result-based prefetching scheme, and the recommendation-aware prefetching scheme. We evaluate and compare the proposed schemes using user browsing pattern data collected from network measurement. We find that the recommendation-aware prefetching approach can achieve an overall hit ratio up to 81%, while the hit ratio achieved by the caching scheme can only reach 40%. Thus, the recommendation-aware prefetching approach demonstrates a strong potential for improving the playback quality at the client. We also explore the trade-offs and feasibility of implementing recommendation-aware prefetching.

Categories and Subject Descriptors

C.4 [**Performance of systems**]: Design studies

General Terms

Design, Experimentation, Performance

1. INTRODUCTION

The advent of user-generated video sharing sites such as YouTube, Dailymotion, Metacafe, Tudou, and Daum has provided tremendous opportunities for Internet users to share their personal experiences as well as to conduct business. The astronomical amount of video content uploaded on video sharing sites has made these sites information sources to which Internet users often turn to be informed, entertained, and even educated. For example, YouTube has hundreds of millions of viewers and delivers billions of videos each month. Unlike the traditional video-on-demand (VoD) systems that typically deliver professionally produced content, video sharing sites typically contain short video clips produced for a particular purpose [5]. The short duration of video clips combined with the huge collection of videos makes it possible for users to browse around for content of interest.

Despite the tremendous popularity of user generated video sharing sites, user experience with watching videos from these sites can vary significantly [18]. As we show in this paper, it is common that a user experiences a pause when watching a video online. These interruptions during video playback can be quite annoying and can potentially discourage users from watching more videos or simply turn users off at the very beginning of a video browsing session. Even a small number of pauses can have a very negative impact since the majority of videos on video sharing sites are usually relatively short (on the order of a few minutes) [22, 11]. Clearly, an increase in network bandwidth and scalable solutions on video sharing sites can solve some of these problems. However, the desire for and the increasing availability of high quality videos (such as high quality or high definition videos) might further exacerbate the experience of browsing video sharing sites.

In this paper, we propose to prefetch video content in order to reduce or eliminate the potential of pauses during video playback and decrease the service delay. We introduce a series of prefetching schemes: conventional caching scheme, search result-based prefetching scheme, and recommendation aware prefetching scheme. Our proposed prefetching scheme conserves bandwidth by prefetching only a *prefix* of a video, since a video clip can playback smoothly if a sufficiently large prefix of the video is prefetched [19]. Furthermore, the prefetching scheme can take advantage of many "idle" periods of a video browsing session by prefetching when the current playback does not saturate the available bandwidth or when users read comments between watching videos.

We evaluate our proposed prefetching schemes with user browsing pattern data collected from a university network. We focus on user browsing patterns on YouTube since YouTube is the most popular video sharing web site in North America. Our measurement results show that the recommendation-aware prefetching approach can achieve an overall hit ratio of up to 81%, while the hit ratio achieved by the caching scheme and search result-based prefetching scheme can reach only 40% and 38%, respectively. Therefore, our study demonstrates a strong potential for improving the playback quality

MMSys'11, February 23–25, 2011, San Jose, California, USA.

at the client using recommendation-aware prefetching. Although building an effective recommendation system itself is a challenge [2], it can potentially provide sufficient clues for predicting what users are most likely to watch next. We also explore the trade-offs and feasibility of implementing the recommendation-aware prefetching.

Although our evaluation is presented in the context of a proxy cache architecture, the proposed prefetching scheme can potentially be applied to a peer-to-peer architecture or to the servers in content delivery networks (CDNs). Despite the fact that prefetching has been proposed in the context of web and multimedia delivery, demonstrating its effectiveness has been challenging without user browsing traces. To the best of our knowledge, our work is the first to systematically measure and compare the effectiveness of various prefetching schemes based on *actual user browsing activities* and demonstrate the advantage of exploiting the recommendation system for video delivery.

The rest of the paper is organized as follows. In Section 2, we investigate the user experience on YouTube regarding the pauses users experienced in the video playout. Section 3 describes the prefetching schemes and the algorithms to select videos to prefetch. In Section 4, we describe our datasets and measurement of the usage of video referrers. The evaluation of the proposed prefetching schemes is presented in Section 5, and in Section 6, we discuss the trade-offs and feasibility of prefetching. Related work is presented in Section 7. Finally, Section 8 concludes the paper.

2. INVESTIGATING USER EXPERIENCES WITH WATCHING YOUTUBE VIDEOS

Previous work has shown that service delay on YouTube is longer than on other video sharing websites [18]. To further demonstrate the need for prefetching, we perform an experiment to evaluate user experience in watching YouTube videos. In particular, we measure how likely it is that a user experiences pauses during video playback and how long the pauses are. We describe our data collection methodology and how we emulate the playback. Then, we present our results on estimating the possibility and duration of pauses experienced by a viewer.

2.1 Data Collection

We derived the information of pause frequency automatically by analyzing *video download traces*. A video download trace is a trace of incoming and outgoing network traffic captured while a user is watching a video on YouTube. In our case, we asked volunteers to use Wireshark network protocol analyzer [3] on their computers to capture the traffic. We automated the process of detecting pauses in video playbacks to make the process easy for the volunteers and as precise as possible. Instead of asking the volunteers to watch videos and record the number of pauses, the volunteers only had to start the capturing before clicking on the link to a video and stop the capturing after the video playback ends. We then used the trace data from Wireshark to estimate whether pauses in a video playback occurred.

12 volunteers were asked to capture video download traces from various environments representing different locations and network access technologies as shown in Table 1. We believe that the locations chosen for the experiment present a good variety and represent typical places where users would watch YouTube videos.

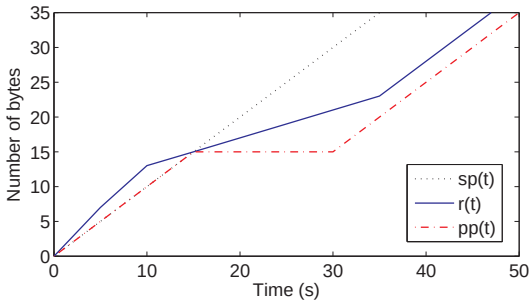


Figure 1: Example plot of $r(t)$, $sp(t)$ and $pp(t)$.

Environment	Location	Network Technology
E1	University 1	Campus WLAN
E2	Company 1	DSL
E3	Home 1	DSL
E4	Apartment 1	Cable Internet
E5	Dormitory 1	Campus LAN
E6	Dormitory 2	Campus LAN
E7	Apartment 2	Cable Internet
E8	Town Library	Wireless Network
E9	Coffee shop	Wireless Network
E10	University 2	Campus WLAN
E11	Home 2	DSL
E12	Hotel	Wireless Network

Table 1: Environment information.

We asked the volunteers to watch 10 videos from our selection and obtained 10 video download traces from each of them. We selected videos that have different levels of quality (standard quality (SD), high quality (HQ), and high definition quality (HD)). The average bit rate for these videos ranges from 162 to 2150 kbps.

2.2 Modeling a Video Player

The main requirement for a smooth video playback is that each byte of the video arrives at the client before the time it is required to be played. More formally, let $sp(d)$ be the number of bytes needed to play the first d seconds of a video, $r(t)$ be the number of all bytes received at the client at time t, D be the video length in seconds, and t_s be the time the video starts playing. To get a smooth playback, the following condition needs to be satisfied: $r(t) >= sp(t - t_s)$ where $t_s <= t <= t_s + D$.

In the example shown in Figure 1, the video starts playing at $t = 0$. During the first 15 seconds, $r(t) > sp(t)$, thus the video can be played smoothly. However, just after $t =$ 15 seconds, the number of bytes received is less than the number of bytes required. At that point the video playback cannot be continued.

To deal with the buffer depletion, video players, including YouTube's video player, pauses to perform buffering whenever there is insufficient data at the client to render the next frame. The video playback is resumed when the player's buffer fills up to a certain level. Based on the data rate at which the video is received at the client, this may lead to one or more pauses during the playback of the video.

To model the video player, we define the function $pp(t)$ as the number of bytes required by a player at time t. The value of $pp(t)$ depends on the length of the video that has

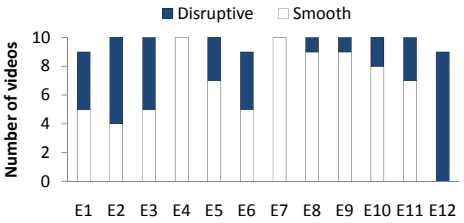


Figure 2: Video playback quality.

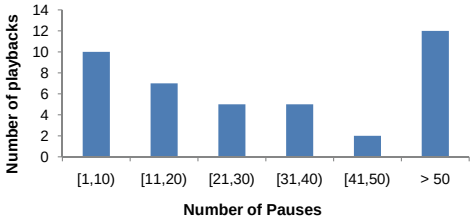


Figure 3: Histogram of number of pauses in disruptive playbacks.

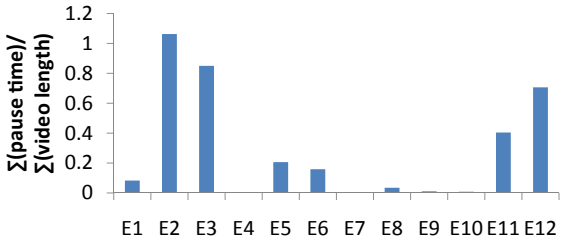


Figure 4: Fraction of time spent in waiting for the videos.

been played at time t. If the player has played up to d seconds of the video at time t, then we have $pp(t) = sp(d)$. In Figure 1, after $t = 15$ seconds, which is the point when the buffer depletion starts, $pp(t)$ remains steady for some period. This period corresponds to a pause in the playback. $pp(t)$ continues to increase after $t = 30$ seconds, corresponding with the player resuming the playback after it has filled up enough data in its buffer.

Based on the previous functions, the video player works as follows. At any time t, the player's state is either 'play' or 'pause'. Let B be the minimum amount of data required to be in the buffer for the playback to resume from pausing. The player changes its state in these two cases:

- 'play' to 'pause': when there is insufficient data to play the video or $pp(t) > r(t)$
- 'pause' to 'play': when the data in the buffer reaches the resume threshold value or $r(t) - pp(t) >= B$, or when the player has received the full video file

2.3 Emulating Video Playback from Video Download Trace

The function $sp(d)$ and $r(t)$ are essential in emulating the video player. In this section, we describe how we derive these two functions from a video download trace.

To derive $r(t)$, we examine the receive time and the TCP sequence number of the packets that contain the video file to get the number of contiguous bytes of the video file we have at each point in time.

To derive $sp(t)$, we analyze the video file which we reassembled from the payload of the packets. Video encoding divides video data into segments. Each segment has its own play timestamp which specifies the time that the segment should be rendered relative to the first segment. From this analysis, we can determine how many bytes are required to render each frame of the video without any delay to allow for an uninterrupted playback.

In addition to the two functions, we need to determine the value of B, the amount of data required to resume from pausing. Since YouTube does not disclose its video player's specification, we let B equal to the amount of data needed to play 2 seconds of a video based on our observation. This means the required buffer size varies for different videos. If we use larger B, the number of pauses in our results will be fewer, but each pause period will also be longer. Thus, although the value of B used in our experiment are not exactly the same as YouTube's video player, we believe our results can reflect the user experience on YouTube well.

2.4 User Experience on YouTube

Using the described model, we emulate video playback from video download traces. First, we determine whether the video was played at the client with a pause or not. Figure 2 shows the number of smooth playbacks and disruptive playbacks for each dataset. Some datasets contain 9 playbacks due to packet capture error. The results show that 10 out of 12 environments contain playbacks with pauses. In addition, 41 of 117 playbacks (35%) contain pauses.

Next, we estimate the number of pauses in the interrupted playbacks. Figure 3 shows the estimated number of pauses in all 41 disruptive playbacks. We find that 31 playbacks, which are 75.6% of disruptive playbacks, contain more than 10 pauses. (Even when we increase B to 5 seconds of videos, 57% of disruptive playbacks contain more than 10 pauses.) Considering that the duration of the videos in our datasets ranges between 3 to 10 minutes, pausing as much as 10 times or more would be extremely unpleasant to users.

Finally, we compute the time that a user had to spend waiting for the videos. We note that since the users's download rate in our datasets is relatively stable, the accumulated pause period is not significantly affected by the size of B. In Figure 4, we show the ratio between accumulated pause time and accumulated video length from all the playbacks in each environment. The user experience varies across different locations. In some locations, like E2, E3, E11 and E12, the time spent waiting for the videos (when the videos were paused), is longer than 40% of the total duration of the videos. In E2, which is the worst sceanario, the time spent waiting is even longer than the total video length.

Our results lead to our conclusion that YouTube users indeed experience disruptive playbacks on YouTube, especially when they watch videos with higher quality. Although a user can choose to wait for a video to buffer before she starts watching, it is undesirable. We expect that this problem will become even more common as high definition videos become increasingly popular on YouTube. The results of this experiment motivated us to devise a video prefetching approach that has the potential to reduce or even eliminate pauses during video playback. The approach is described in the next section.

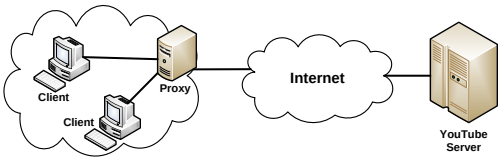


Figure 5: The architecture of prefetching proxy system.

3. VIDEO PREFETCHING SCHEME

The main principle of prefetching is to retrieve content from the source before it is requested by a user and store it in a location that can be accessed by a user conveniently and fast. This is fundamentally different from caching where content is only stored locally if it has already been requested by a client. Prefetching can be applied to various architectures and in different ways. In this section, we describe the settings of the proposed prefetching scheme, followed by the algorithms used to select videos to prefetch.

3.1 Prefetching Agent

Consider a typical network as shown in Figure 5, there are two apparent places where we can implement the prefetch functionality, at the client and the proxy. We call the module that performs prefetching the *prefetching agent* (PA). In this paper, we consider two settings of the prefetching scheme: in the first one, the PA is located at the client (PF-Client), and in the second one, the PA is located at a proxy server (PF-Proxy).

A PA is a module responsible for prefetching. It has a storage to store prefetched prefixes of videos. It determines the videos to be prefetched, retrieves their prefixes from YouTube, and stores them. In addition, the PA can perform caching, i.e., it stores either a whole or a prefix of videos that are requested by clients. Caching YouTube videos at the network edge has been evaluated by Zink et al. and shown to be useful in reducing network traffic and providing faster video access [22].

Every YouTube request from a client is directed to the PA. If the request is a video request, the PA checks if the prefix of the video exists in its storage. If so, it serves the client with the prefix of the video, and at the same time retrieves the remaining part of the video from YouTube and sends it to the client. Note that the video prefix and the remaining part are sent to the client simultaneously. This further helps to decrease the chance of buffer depletion at the client. If the prefix is not in its storage, the PA retrieves the whole video from YouTube and sends it to the client. If the PA also performs caching, it stores the retrieved videos in its storage. Based on the requests received, the PA selects the videos to be prefetched (which have not been requested by any users yet), retrieves their prefixes from YouTube, and stores them in its local storage.

The difference between PF-Client and PF-Proxy is the location of the PA. In PF-Client, every client is connected to its own PA, thus each PA receives requests from only one client. In PF-Proxy, the PA resides in a proxy which is situated between clients and YouTube servers, close to the clients as shown in Figure 5. For example, the proxy may be located at the gateway of a campus network or at the local aggregation point of an ISP network. In this setting, the PA receives requests from all clients in the local network.

The next section describes how the PA selects the videos to prefetch based on the requests it receives.

3.2 Video Selection for Prefetching

In order to perform prefetching, the PA needs to determine the set of videos to be prefetched. Given YouTube requests from clients, the PA needs to predict a set of videos that are likely to be requested in the future. Here, we describe two algorithms that the PA can use to select videos to prefetch.

The first algorithm is based on users' search results. YouTube provides a search box in which a user can enter a query phrase to search for videos of interest. After the search query submission, a list of videos that match the query phrase, or a *Search Result* list is shown in a *search result page*.

To implement this algorithm, the PA detects search result pages sent from YouTube which are the responses to clients' search queries. Then, it extracts the list of videos to determine which videos to prefetch. A search result page can contain up to 20 videos in one page. Prefetching all of those videos may or may not be practical depending on the available bandwidth and storage space at the PA. Therefore, the PA may prefetch only the top N videos of the Search Result list based on their positions on the list. We call this algorithm SR-N.

The second algorithm is based on the YouTube recommendation system. Each YouTube video has its own web page, which we call a *video page*. Each video page contains a *Related Video* list which is a list of videos that have similar content recommended by the YouTube recommendation system. As shown in Section 4.2, besides Search Result lists, a large number of video views originates from Relate Video lists. Thus, videos in Related video lists are also good candidates for prefetching.

A Related Video list contains up to 25 videos. Similar to SR-N, we may prefetch only the top N videos on the Related Video list according to the order they are shown in the list. We call this algorithm RV-N. The PA implements the RV-N algorithm by detecting all the videos pages from the responses it receives from YouTube and parsing the video pages to obtain the Related Video lists.

The advantage of both algorithms, SR-N and RV-N, is that they are simple and not computationally expensive. The PA can obtain the lists of videos to prefetch without requesting or storing any additional data. In the next section, we present the datasets we used to evaluate the two settings of prefetching scheme and video selection algorithms we have described.

4. DATA COLLECTION

In this section, we describe the data collection process and datasets we use to evaluate the prefetching schemes.

4.1 Datasets

Our data collection consists of two phases. In the first phase, we monitored and recorded data traffic between a campus network and YouTube servers. Due to the campus privacy policy, we only recorded fix-length headers of the data packets, so we cannot obtain the Related Video lists and Search Result lists which are essential for our experiments from the traces. In the second phase, we retrieved the two lists from YouTube using YouTube Data API [4].

The details of the two phrases are described in the following subsections.

4.1.1 Network Traces

We obtained three network traces from monitoring YouTube traffic entering and leaving a campus network. The monitoring device is a PC with a Data Acquisition and Generation (DAG) card [1], which can capture Ethernet frames. The device is located at a campus network gateway, which allows it to see all traffic to and from the campus network. It was configured to capture a fixed length header of all HTTP packets going to and coming from YouTube domain.

The monitoring periods are 1 day, 3 days, and 7 days (T1, T2 and T3 respectively). The general statistics of the traces are shown in Table 2. Since T2 was obtained during the winter break, it has fewer video requests than T1 although the capture period is longer. T3 has the most video requests because it was taken when class was in session and it has the longest capture period.

Trace File	T1	T2	T3
Duration	1 day	3 days	7 days
Start Date	20-Oct-09	8-Jan-10	28-Jan-10
# Request	71,282	7,562	257,098
# Unique Clients	7,914	607	10,511
# Unique Videos	48,978	5,887	154,363

Table 2: Statistics from network traces collected at the campus network gateway.

4.1.2 Search Result Lists and Related Video Lists

In addition to the network traces, to validate the prefetching approach, we need the Search Result lists for every video search query in the traces and the Related Video lists for every requested video. These lists are used by the prefetching agent to determine the set of videos to be prefetched. We retrieved the Search Result lists and the Related Video lists via YouTube Data API.

To retrieve the Search Result lists, we started from identifying all the video search queries in the traces using URI pattern matching. A URI of a video search query on YouTube starts with `results?search_query=`, followed by the query phrase and other parameters. After identifying the search queries in the network traces, we retrieved the Search Result list for each query by sending the same search query to YouTube via YouTube Data API. We retrieved at most 25 videos for each Search result list.

Similarly, to retrieve the Related Video lists, we first extracted video page requests from the traces. A video page request's URI starts with `watch?v=`, followed by a video's ID, which is a 11-character string. With the set of video requests, we then proceeded to fetch the Related Video list for each video through YouTube Data API. We retrieved at most 25 videos for each Related Video lists.

4.2 Usage of Search Result Lists and Related Video Lists

In this section, we present our measurement results on the usage of different view referrers. A referrer of a video is the source that refers a user to the video, for example, a Related Video list of another video, YouTube featured video page, and links on other web sites. The result from this study shows that the two most frequently used referrers are Search Result lists and Related Video lists, which prompted us to use the two lists in the proposed video selection algorithms.

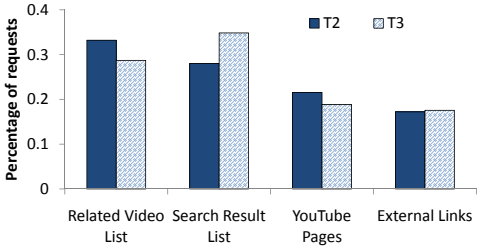


Figure 6: Fraction of requests from each referrer type.

We perform the study by analyzing the referrer of each video request in our traces. The HTTP referrer fields of the video requests are not contained in our traces due to the limited length of the captured packets. Hence, we employ another method to identify the referrers. The requests coming from certain referrers contain the referrer types explicitly in their URIs. This includes requests generated from users clicking on Related Video lists, which contain the tag `feature=related` in their URIs. Therefore, we can extract the referrers of these video requests from their URIs. However, there are also video requests that contain no referrers information in their URIs, including the requests from Search Results lists and most links from external websites. We use an additional heuristic to infer the referrers of these video requests by analyzing YouTube user sessions. A user's YouTube session is a series of requests sent to YouTube by a user in one visit [14]. We consider that a session ends when a user is idle for 40 minutes, which is the threshold timeout used in [12]. A referrer of a video request without tags is inferred from the previous pages visited in the same session before the request is made. Referrers are then grouped into 4 types: Related Video lists, Search Results lists, other YouTube pages, and external links. The external links category are referrers that are outside YouTube such as video links on blogs and social network sites.

We perform the analysis on trace T2 and T3 because they were captured with longer packet length, so we have complete tags from the URIs. In Figure 6, we show the requests from each referrer type as a percentage of all requests. The results show that the Search Result lists and Related Video lists are major view referrers. There are 28% and 35% of the video requests with the Search Result lists as their referrers (in T2 and T3, respectively), and there are 33% and 29% of the requests with the Related Video lists as their referrers.

From the result, we decided to base the video selection algorithms on the two lists, Search Result list and Related Video list. We note here that although this result might leave the impression that the prefetching approach using the Search Result lists or Related Video lists can only achieve a hit ratio around the same level as the usage rate of the lists, as we show in Section 5, this is not the case since our evaluation of the prefetching approach based on the Related Video lists results in hit ratios up to 81%.

5. EVALUATION

In this section, we present our evaluation of the video prefetching approaches. We compare the performance of the

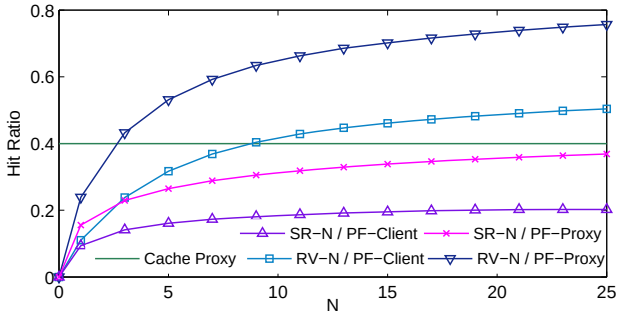


Figure 7: Hit ratios of SR-N and RV-N for T3.

two video selection algorithms and the two settings proposed in Section 3.

5.1 Methodology and Evaluation Metrics

Our evaluation for the prefetching schemes is based on real user usage patterns. This is achieved by performing a trace-driven simulation using the traces captured from a campus network as presented in Section 4.1.1. In the simulation, video requests are issued based on the network traces, which means the videos that are requested and the order of the requests are exactly the same as in the traces. The simulated PA determines the videos to be prefetched based on the requests received and keeps track of the set of video prefixes that are in its storage. Thus, it can determine whether a requested video has been prefetched or not. With this method, we can determine the proportion of video requests from the traces that could have been served faster from the PA if the prefetching system was implemented at the time the traces were captured.

To study the characteristics and compare the performance of different prefetching schemes, we first perform experiments in the cases when the PA always has sufficient storage space, from Section 5.2 to 5.4. Then, in Section 5.6, we explore the case when there is limited storage space. For simplicity, the storage space size is defined by the number of slots, where each slot can hold a prefix of a video. Based on the measurement result in [6], the average video size on YouTube is 8.4 MB. Suppose the prefix size is 30% of a video, then each slot corresponds to about 2.5 MB.

In this study, two metrics are used to evaluate the prefetching schemes. The first metric is the hit ratio, defined as a fraction of the number of requests for a video that can be served from the prefetching storage (called hit requests): $hit\_ratio = hit\_requests/all\_requests$. A higher hit ratio means we can serve more requests from the prefetching agent's storage, resulting in better user experience. The second metric is the precision, which reflects the accuracy of the video selection algorithm. The precision is defined as a number of prefetched videos that are actually requested by users (called the hit videos) over the total number of prefetched videos: $precision = hit\_videos/all\_prefetched\_videos$.

5.2 Performance of Prefetching Using Search Result Lists (SR-N)

We first present the performance of the prefetching scheme which prefetches based on the top N videos on Search Result lists (SR-N). Figure 7 shows the hit ratio of the prefetching scheme using the SR-N algorithm when there is always sufficient space at the PA. We also show the hit ratio of the cache proxy, which caches all videos that users have requests, as a baseline. From the figure, the maximum hit ratio at $N = 25$ is equal to 20.62% in PF-Client and 36.86% in PF-Proxy. It may be unexpected that the maximum hit ratio achieved in the PF-Client setting is lower than the inferred percentage of video requests from users clicking Search Result lists. This may be attributed to two reasons. The first reason is that a user may click on a video contained in a playlist in a search result, which we cannot retrieved via the API. The second reason is that a user may click on a search result in the position lower than 25. The result here shows that the hit ratio we obtain using the Search Result lists cannot surpass hit ratio of the caching scheme which is 39.96% despite the fact that Search Result lists are one of the the major sources of video views.

5.3 Performance of Prefetching Using Related Video Lists (RV-N)

We now proceed to evaluate the performance of the prefetching scheme that relies on the YouTube recommendation system, or the Related Video lists (RV-N). The hit ratio of the prefetching scheme using the RV-N algorithm is shown in Figure 7. At $N = 25$, the RV-N algorithm results in the maximum hit ratio of 50.38% and 75.68% in the PF-Client and the PF-Proxy setting, respectively. These maximum hit ratios are higher than the hit ratio achieved by the cache proxy. In fact, the PF-Proxy setting can outperform the cache proxy with the value of N as low as 3. As for the PF-Client setting, we need to prefetch at least 9 videos to surpass the cache proxy. From the results, we also observe that as N increases, the increasing rate of the hit ratio is smaller. This suggests that the top videos in the Related Video lists are better predictions of users' future views.

So far, we observe that PF-Proxy yields much higher hit ratio than PF-Client. This suggests that users in the same local network share similar interests, and thus videos from a Related Video list or a Search Result list of a user are also watched by other users in the same local network. PF-Proxy benefits from this fact and achieves about 50% to 100% improvement in the hit ratio compared to PF-Client.

Up to this point, the RV-N algorithm, which is based on the Related Video lists, in combination with the PF-Proxy setting gives us the best hit ratio of up to 75.68%. Consequently, we will focus on the particular prefetching scheme - the combination of the RV-N algorithm and the PF-Proxy setting.

5.4 Analyzing the High Hit Ratios

One interesting observation from previous results is that the maximum hit ratio achieved with the RV-N algorithm is much higher than how often users click on Related Video lists, which is around 30% as analyzed in Section 4.2. This means that the hit requests are not only the requests that come from users clicking on Related Video lists, but also the requests from other referrers. To gain further insight, we conduct an analysis to see how many requests from other referrers are hit requests in the RV-N prefetching scheme.

In Section 4.2, we have identified the referrer of each video request in the traces. Therefore, we can determine how many requests from each referrer are hit when we prefetch using Related Video lists. Figures 8 and 9 show the referrers of the hit requests for the prefetching schemes with the PF-Client setting and the PF-Proxy setting. In PF-Client, only

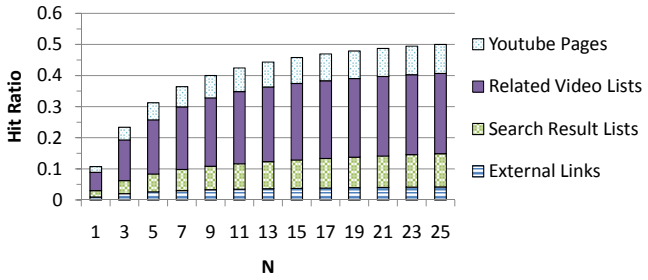


Figure 8: Referrers of hit requests (PF-Client).

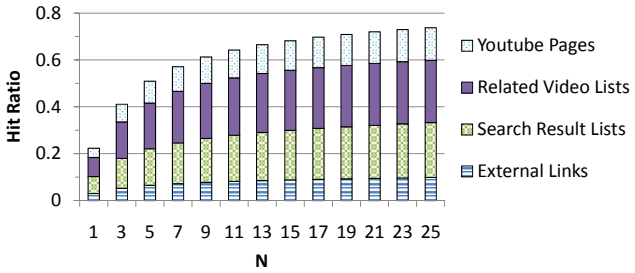


Figure 9: Referrers of hit requests (PF-Proxy).

about 55% of the hit requests are from the users clicking on Related Video lists. The remaining 45% are the requests caused by users clicking on the Search Result lists and other referrers. This means that, for many videos in the Related video lists, a user may not click on them from the lists, but she eventually watches them through other referrers.

For PF-Proxy, the fraction of the hit requests that are from users clicking on the Related Video lists becomes even smaller, while the requests that come from users clicking the Search Result lists become a significant portion of the hit requests. This means that the additional hit requests in the PF-Proxy setting, which are those requests of a client that can be served by a video prefetched based on another client's request, are mostly the requests that come from a user clicking on Search Result lists.

From both figures, we learn that there is overlap between the videos shown in the Related Video lists, which we prefetch, and video requests that users request through other referrers. This overlap contributes to the high hit ratios when we use the RV-N algorithm. Next, we investigate how large this overlap is for the video requests from each referrer type.

Figure 10 and 11 shows the hit ratios computed separately for the requests from each referrer type. In PF-Client, we can see that the requests from Search Result lists, external links, and YouTube pages, have the hit ratio of up to to 20-50%. Note that the hit ratio for the requests from the Related Video lists is 90%, less than 100% due to the small changes in the Related Video lists when we retrieved them. In PF-Proxy, the hit ratios for these referrers are significantly improved, which makes the aggregated hit ratio in PF-Proxy much higher than PF-Client. Especially, the improvement of the hit ratio of requests from the Search Result lists, from up to 30% in PF-Client to up to 65% in PF-Proxy, is the major contributor because a large number of requests are from Search Result lists.

In sum, the property of the videos in the Related Video list that they largely overlap with the video requests generated from a user clicking on the Search Result lists, which are the large fraction of all requests, and also from other referrers makes them very effective choices for prefetching. The videos in the Search Result lists, on the other hand, do not have this property, and thus the SR-N algorithm does not give as high hit ratio as the RV-N algorithm although the frequency that users use the Related Video list and Search Result list are about the same.

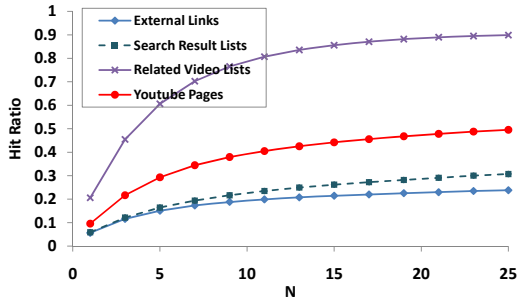


Figure 10: Hit ratios of each request category (PF-Client).

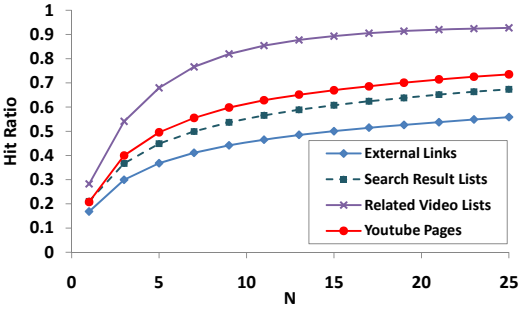
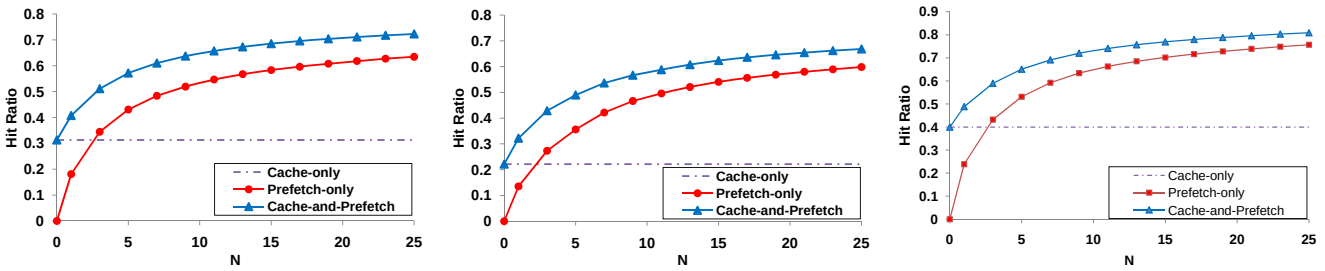


Figure 11: Hit ratios for each request category (PF-Proxy).

5.5 Combining Caching and Prefetching

Because of the difference in their underlying principals, the prefetching scheme and caching scheme conceptually captures different sets of videos, although there may be some overlapping. Thus, combining these two schemes can potentially result in a higher hit ratio. Figure 12 shows the hit ratio improvement resulting from the combination of caching and prefetching called the cache-and-prefetch mode. The combination of the two schemes increases the hit ratio by 5-20% compared to the prefetch-only mode. The maximum hit ratios we obtain at $N = 25$ increase from 63.47%, 59.85% and 75.68% to 72.30%, 66.83% and 80.88% for trace T1, T2 and T3, respectively. Note that the hit ratio of the cache-and-prefetch mode is not the sum of the cache-only and prefetch-only mode. This is because there is an overlap between the set of cached videos and the set of prefetched videos. As shown in Figure 12, the improvement of the hit ratio induced by the cache-and-prefetch mode becomes smaller as N increases. This means that as we prefetch more videos from the Related Video lists, the overlap between the set of prefetched videos and the set of videos that users have watched becomes larger. Thus, with regards to the hit ratio, the addition of caching functionality is more helpful when we prefetches a small number of videos. In Section 6.4.2, we show another advantage of combining caching and prefetching when we discuss the traffic overhead introduced by the prefetching scheme.



(a) T1 (b) T2 (c) T3

Figure 12: Hit ratio improvement from combining caching and prefetching.

5.6 Storage Requirement

So far, the prefetching scheme using the RV-N algorithm and the PF-Proxy setting has yielded the best hit ratio of up to 80% in cache-and-prefetch mode. The presented results are based on the assumption that there is always sufficient storage space to store the prefetched and cached videos. This gives us the highest hit ratio that can be reached. In Figure 13, we show the storage space that is actually required for the case of sufficient storage space. The storage size in the figure is converted from the number of slots to gigabytes where each slot is equal to 2.5 MB, as explained in Section 5.1. The storage space required in cache-only mode is also shown as a baseline.

As shown in Figure 13, when N is larger, which means more videos are prefetched, the required space increases. The required spaces for the three traces are different because of the difference in the number of requests in the traces. The more requests there are, the more space we need. Although prefetch-only mode requires much more space than cache-only mode, the actual space it needs is merely 4.69 TB where it can reach 75.68% hit ratio (in T3). For cache-and-prefetch mode, the storage requirement are very close to prefetch-only mode, while it improves the hit ratio on the order of 5-20%. The maximum space needed is 4.76 TB, which results in a 80.88% hit ratio.

Although the storage requirement given here are specific to our traces with different duration and request volumes, it demonstrates that the storage required to achieve the highest hit ratio with prefetching for a campus-size network is within a feasible range. Later, in Section 6.1, we consider the cases when storage space is insufficient and study how the storage size impacts the performance of the prefetching scheme.

6. DISCUSSION

In this section, we further explore the trade-offs when using the prefetching scheme with the RV-N algorithm and the PF-Proxy setting. We also study certain aspects of the feasibility of prefetching.

6.1 Impact of Storage Space

In reality, the always-sufficient space is not realistic since there are always new video requests and more prefixes to store as the PA continues running. The storage space is fixed, while the storage requirement continues to increase. To investigate the impact of limited storage space on the performance of the prefetching scheme, we ran the simulation with limited storage sizes of 1k, 3k, 5k, 10k, 25k, 50k, and 400k slots. As mentioned in Section 5.1, each slot is about

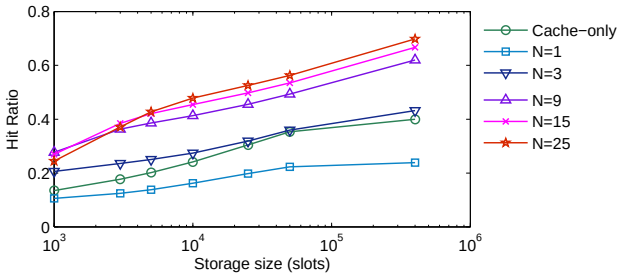


Figure 14: Performance vs. storage size for prefetch-only mode (T3).

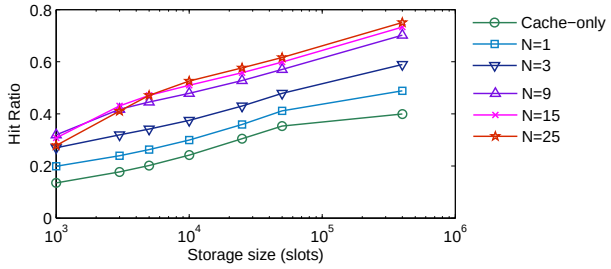


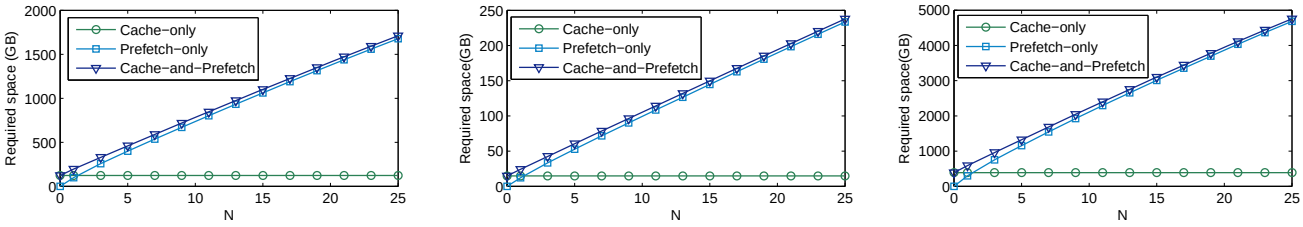
Figure 15: Performance vs. storage size for cache-and-prefetch mode (T3).

2.5 MB. Thus, the maximum slot size of 400k roughly translates to 1 TB. The Least-Recently-Used (LRU) replacement policy is used in our simulation. Figure 14 and 15 show the hit ratio of the prefetch-only and prefetch-and-cache modes with different storage sizes for T3. The results for T1 and T2 are similar but omitted due to space limitation.

The results indicate a correlation between the performance of the prefetching scheme and the storage space size. The hit ratio decreases with the storage space. However, even with a smaller storage space like 125 GB (50k slots), which is less than 3% of space required in the sufficient case, we can still achieve high hit ratios up to 52.59%, 59.36% and 56.26% (for T1, T2 and T3, respectively) with prefetch-only mode and 59.84%, 66.26%, and 61.62% for the cache-and-prefetch mode. In comparison to the caching scheme, the two prefetching schemes can achieve a much better hit ratio using the same storage size. We believe that the hit ratios could even be further improved by applying a smarter cache replacement policy than the LRU policy.

6.2 How large should N be?

In the RV-N algorithm, N is the number of videos we prefetch from each Related Video list. The value of N directly affects the performance of prefetching. To investigate



(a) T1 (b) T2 (c) T3

Figure 13: The sufficient storage size for the RV-N prefetching scheme with the PF-Proxy setting.

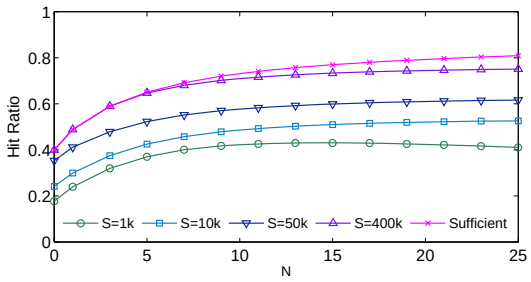


Figure 16: Hit ratio of cache-and-prefetch mode with different storage sizes S and different N (T3).

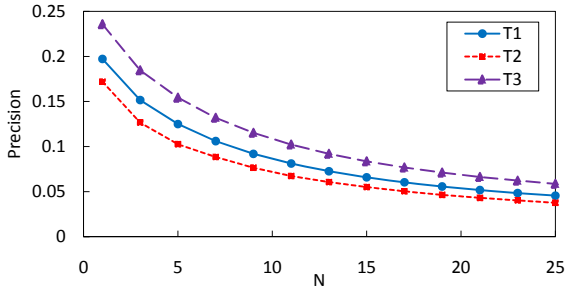


Figure 17: Precision of the RV-N algorithm vs N.

the impact of N, we plot the hit ratio versus N for each storage size for the prefetching scheme with cache-and-prefetch mode, shown in Figure 16. From the figure, with sufficiently large storage space, increasing N always results in a higher hit ratio. However, using small N like 5, we can still achieve a hit ratio up to 65%.

On the other hand, with limited storage size, as N increases, the hit ratio improves up to a certain point and then begins to decline. This effect can also be seen in Figure 14 and 15. When the storage size is small (1k slots), using large N like $N = 25$ results in a lower hit ratio than $N = 9, 15$. In order to explain the cause of this effect, we show the precision of the prefetching scheme for each value of N in Figure 17. From the figure, the precision decreases when N is larger. This means that with larger N the fraction of prefetched videos that are never requested by clients is higher. With limited space, using a too large value of N results in those unused videos taking up the space of the popular videos, giving us lower hit ratios. From Figure 16, a value of N between 5 and 11 seems to be an appropriate range, yielding hit ratios between 65-74% using 1 TB storage space (400k slots). We conclude that when the prefetching scheme is implemented, the value of N should be chosen carefully to avoid the adverse effect when there is limited space and to balance the trade-off between the hit ratio and the additional bandwidth requirement.

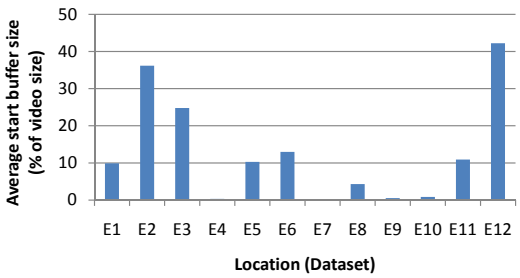


Figure 18: Average minimum start buffer size for smooth playout.

6.3 How much should we prefetch?

In the proposed prefetching scheme, only prefixes of videos are prefetched to save storage space and bandwidth. First, we would like to show that we do not need to prefetch the whole video in order to deliver a smooth video playout, thus prefetching only a prefix of a video is sufficient. To demonstrate this, we compute the minimum size of video data that should be buffered before a video starts playing to give a smooth video playout, or minimum start buffer size b_{min}, for each video playout in the datasets from Section 2.

For a video playout with the function $r(t)$ and $sp(d)$ (as described in Section 2), suppose we have video data of size b in the buffer when a video playout starts at $t = t_s$. The total data we have at time t becomes $r(t)+b$. Thus, to get a smooth playout, the start buffer size b must satisfy the condition $\forall t : r(t)+b >= sp(t-t_s)$. The minimum start buffer size b_{min} is then given by $b_{min} = \max\{\max_t(sp(t-t_s) - r(t)), 0\}$.

Using the derived function $sp(t)$ and $r(t)$ from each video download trace in Section 2, we compute b_{min} for every playout. Figure 18 shows the average value of b_{min} as a percentage of the full video size for each dataset collected at different locations. The result shows that b_{min} is much smaller than the full file size; therefore, we do not need to prefetch the whole video file to deliver a smooth playout. The average minimum start buffer size of each location varies from close to 0% to 42% due to the different network condition at each location.

The remaining question is how large should the prefix be. One simple solution is to let the prefix size be a sufficiently large constant percentage of the full video size, but this approach is inefficient since the minimum start buffer size of each playout is actually different. If a prefix is too large, we unnecessarily waste storage space and bandwidth. On the other hand, if a prefix is too small, prefetching would not be useful. For a better solution, the prefetching proxy should choose the prefix size dynamically for each video. The ideal solution is to let the prefix size be equal to b_{min}, which is

different for each playout. This solution will give a smooth playout, while using as least storage and bandwidth resource for prefetching as possible. Unfortunately, the computation of b_{min} can only be done after a video is played. Therefore, we propose a mechanism to determine the size of the prefix dynamically.

Our experiment in Section 2 shows that although a video's bit rate is not constant, it usually does not vary significantly. Thus, we assume that a video's bit rate is constant and equal to the average bit rate. We also observe that a video's download rate is usually stable over some short period of time, so we assume that a video download rate is constant as well. These two assumptions greatly simplifies the computation of b_{min}. Let b be a video bit rate, d be a video duration and r be a video download rate. The prefix size is given by $b_{min} = d(b - r)$.

From the equation, we still need to determine the value of b, d and r before we actually download a video. For the future download rate r, the PA can conservatively use the lowest download rate it has seen in some time window as the worst case estimate. In addition, the PA can determine the average video bit rate, b, and video duration, d, from the header of a video file containing video metadata. Therefore, not long after the PA starts prefetching the video, it can determine the value of d, b and r, so it can compute the appropriate prefix size and stop prefetching accordingly. In this manner, the prefix size are adapted according to the network condition and the property of each video.

6.4 Feasibility of Prefetching

One concern about prefetching scheme may be that it will worsen the situation because it requires additional network bandwidth, while interrupted video playouts implies that the network bandwidth is insufficient. Our arguments are as follows. First, users are not watching videos all the time. For example, after watching a video, a user may read or write comments, browse through a list of videos, or replay the video. This provides "idle" time to perform prefetching. Second, since each video's bandwidth requirement is different, we may not have enough bandwidth to accommodate higher bit rate videos, but for lower bit rate videos, we have more than sufficient bandwidth. As shown in our experiments in Section 2, we found both types of playouts, with and without pauses, in the same environment. Thus, we can take advantage of the period where the bandwidth is sufficient to prefetch the videos. Third, by combining caching and prefetching, the bandwidth consumption reduced by caching can compensate the additional bandwidth requirement from prefetching. In the following subsections, we further discuss some aspects about the feasibility of prefetching.

6.4.1 Time to prefetch

In practice, a time gap between video requests may sometimes be short, and thus we may not be able to prefetch some prefixes in time before they are requested. Here we measure the time available to perform prefetching to estimate how this issue will effect the performance of the prefetching scheme. Figure 19 shows the CDF of the time gap between the time that the PA decided to prefetch a video and the time the video was actually requested for every hit requests in trace T3 when N is 5 and 25. When N is larger, the distribution of time to prefetch shifts to a higher value. This means we have longer time to prefetch videos in the lower ranks of Related Video lists. Comparing the PF-Client to PF-Proxy setting, PF-Proxy has longer time to prefetch. This demonstrates the benefit of sharing prefetched videos in PF-Proxy. Some videos prefetched based on one client's request are requested by another client some time later, and the time gap between the two events is large, allowing more time to prefetch.

Using the result shown in Figure 19, we can estimate the number of hits that are not feasible in practice, i.e., videos that may not be prefetched in time before clients request them. For example, suppose the available bandwidth is 100 KB/s. To download 25 prefixes of video, each with the size of 2.5 MB, we need 11 minutes. Assuming we use a naive scheme where all the prefixes are downloaded in parallel, then, from the CDF, 26% of the hit requests are not feasible, and the hit ratio in the prefetch-only case with N=25 will decrease from 75.68% to 56.00%, yet it is still in a satisfactory level. In practice, the PA can download the prefixes sequentially and employ a smarter scheme, e.g., prefetching the top ranked videos first, to achieve higher hit ratios.

6.4.2 Network traffic overhead of prefetching

To address the concern about additional network bandwidth required for prefetching, we perform a simple calculation of an example case to show how much prefetching will increase the network load. In this example, we prefetch the prefixes of the top 11 videos from Related Video lists using the prefix size equal to 15% of the video size. For cache-only and cache-and-prefetch modes, we assume that videos are cached in full size. Using the hit ratio from T3 to compute the overhead, we show the results of the calculation in Table 3.

Scheme	Hit Ratio	Normalized load
No scheme	0%	1.00
Cache-only	40%	0.60
Prefetch-only	66%	1.44
Cache-and-Prefetch	74%	1.02

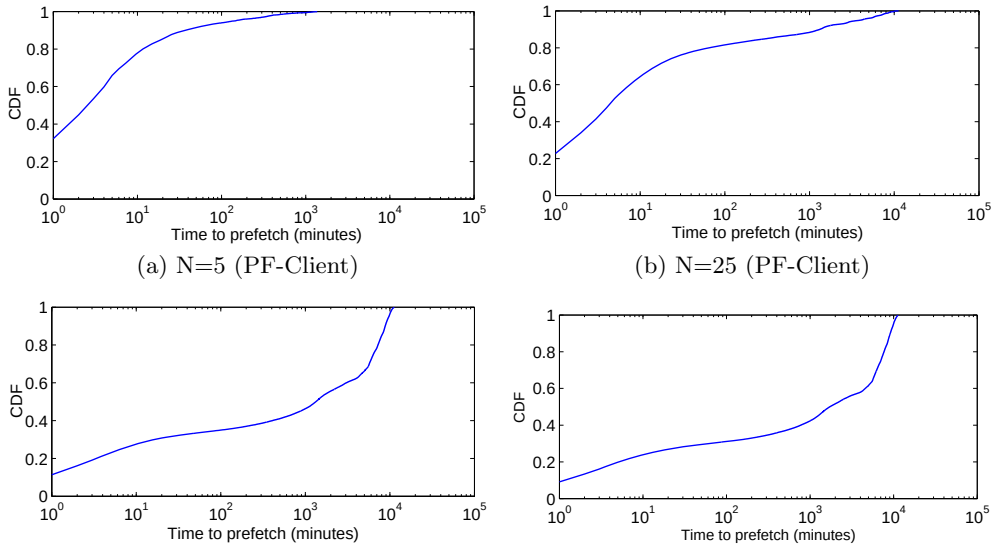
Table 3: Normalized traffic load of prefetching schemes.

Table 3 shows the traffic load for prefetching schemes compared with caching scheme, normalized by the case in which no scheme is implemented. Caching has no traffic overhead and helps reduce the traffic. On the other hand, we gain higher hit ratio with prefetching-only mode, but traffic load is also increased by 44%. These extra work comes from prefetching unused prefixes. Finally, cache-and prefetch mode yields the highest hit ratio and introduces only 2% increase in the network load. Using the combination of caching and prefetching, the extra overhead from prefetching is compensated by the benefit of caching, while we can still maintain the high hit ratio we get from prefetching. In addition, the PA can employ the technique like using TCP Nice [20] to perform prefetching without affecting the peak bandwidth of the network.

7. RELATED WORK

The prefetching technique has its origins in the area of computer architecture. The use of prefetching has been widely studied for web content delivery in early days.

Padmanabhan and Mogul were among the first who applied prefetching within the context of web delivery by proposing the WWW prefetching scheme to reduce latency [15]. In



(a) N=5 (PF-Client) (b) N=25 (PF-Client)

(c) N=5 (PF-Proxy) (d) N=25 (PF-Proxy)

Figure 19: CDF of the time to prefetch.

this scheme, a server makes predictions of the links that are likely to be requested next by a client based on past observations. Clients use these predictions to prefetch web documents. In [9], Cunha and Jaccoud proposed two models based on random walk and digital signal processing to model user web access pattern for prefetching. In [10], Fan et al. proposed the proxy-initiated prefetching for web documents. Their approach addresses the low-bandwidth limitation between clients and a proxy by prefetching the cached documents from the proxy to the clients. In this approach, prefetching does not happen between the proxy and the web servers. The use of proxies in VoD systems has been intensively studied in earlier work. We mention the ones that are closest to our approach in the following.

In [19], Sen et al. proposed a prefix proxy caching scheme in which the proxy is used to hide latency, packet loss, and jitter between the local network and server sites. Their proxy caches only a prefix of a video to avoid using a large cache space. They focused on using prefix caching to smooth out the network bandwidth requirement from a proxy to a client and used streaming traces of two videos to demonstrate the benefits of the approach. However, they did not address the issue of how to select videos to prefetch. In [5] and [22], trace-driven simulations were performed to investigate the effectiveness of caching for YouTube videos. Although the traces for both studies were different, the results showed that caching can reduce server and network load significantly. Both studies did not consider prefetching. In addition, there are many existing studies on the use of proxy to improve the quality of media streaming, e.g., [16, 13, 17, 21].

Additional studies have been performed to understand YouTube's characteristics. Cha et al. performed an extensive study of video view statistics on Youtube [5]. Gill et al. studied the usage patterns and video characteristics on YouTube using data from the network edges in [11]. In [12], they further studied user sessions on YouTube. Results from this analysis motivated us to investigate the prefetching approach since they were the first ones who showed that users spend extended periods of time on YouTube, often watching more than one videos. In [18], Saxena et al. analyzed the service delay for YouTube and other user-generated video sharing sites. Their measurement-based analysis showed that the service delay for YouTube is high, which can lead to a poor playback experience. How often the playback is actually interrupted is not the focus of their study.

Cheng et al. [8] measured the YouTube video graph created by related video links and found that the graph has a large clustering co-efficient and exhibits the small world property. A simulation-based evaluation of a P2P video sharing systems showed that if users use the Related Video list to browse videos, the percentage of source peers that have the requested video in their cache is high.

Cheng and Liu [7] also proposed a P2P video sharing system to reduce YouTube server load and suggested using prefetching based on YouTube's Related Video list at the clients of a P2P system to provide smooth transition between videos. Their evaluation was based on emulated user browsing pattern. The evaluation of their approach showed that it performs significantly better (55% hit ratio) comparing with a random prefetching approach (nearly 0% hit rate).

In contrast to these previous work, we propose and compare various prefix prefetching schemes. Our focus is on user-generated video sharing sites, which are inherently different from VoD systems. We demonstrate the benefit of the prefetching schemes using real user browsing patterns collected from university network traffic. Our study demonstrates that in addition to views from users clicking Related Video lists benefiting from recommendation aware prefetching, other views such as clicking on search results can also benefit from recommendation aware prefetching. Also, the shared interests among network users leads to 50%-100% increase in hit ratios when prefetching is performed at a proxy server at the network edge. This suggests that recommendation aware prefetching is a good heuristic for predicting the interest of viewers both individually and across a community.

8. CONCLUSION

In this paper, we show that currently the user experience in watching videos on video sharing web sites like YouTube is often dissatisfying. In particular, our experiment indicates that many users experience pauses during a video playout. This motivated us to propose a prefetching technique which improves the playback quality and the delay of videos requested from YouTube.

Our proposed prefetching technique works by predicting a set of videos that are likely to be watched in the near future and then fetching the prefixes of those videos before they are requested. If a video has been prefetched, a user can access it faster and the prefetched portion in the buffer can compensate for any insufficient bandwidth and absorb network delay, resulting in a smooth playout of the video.

Our evaluation of the prefetching approach is based on actual network traces which capture real user access patterns. We compare the performance of various prefetching schemes to the traditional caching scheme. We find that applying prefetching at a proxy while using the Related Video lists to select videos to prefetch is the most effective prefetching scheme. Even with limited storage space, prefetching at the proxy results in a hit ratio twice as high as the caching proxy. In addition, the combination of caching and prefetching can further enhance the hit ratio up to 81%.

Finally, we discuss factors that affect the performance of our prefetching scheme including the storage size and the number of videos to prefetch and explore the trade-offs regarding those factors. We also analyze the additional overhead in network traffic that is introduced by prefetching and show that, in the case of caching and prefetching, it is an almost negligible increase of 2%.

9. REFERENCES

[1] Endace DAG Network Monitoring Interface. `http://www.endace.com/`.

[2] Netflix. `http://www.netflixprize.com/`.

[3] Wireshark. `http://www.wireshark.org/`.

[4] YouTube Data API. `http://code.google.com/apis/youtube/overview.html`.

[5] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proceedings of ACM Internet measurement Conference(IMC), San Diego, CA, USA*, Oct. 2007.

[6] X. Cheng, C. Dale, and J. Liu. Statistics and social network of youtube videos. In H. van den Berg and G. Karlsson, editors, *IWQoS*, pages 229–238. IEEE, 2008.

[7] X. Cheng and J. Liu. Nettube: Exploring social networks for peer-to-peer short video sharing. In *Proceedings of IEEE INFOCOM*, 2009.

[8] X. Cheng, J. Liu, and H. Wang. Accelerating youtube with video correlation. In *WSM '09: Proceedings of the first SIGMM workshop on Social media*, pages 49–56, New York, NY, USA, 2009. ACM.

[9] C. R. Cunha and C. F. B. Jaccoud. Determining www user's next access and its application to pre-fetching. In *Proceedings of ISCC'97: The second IEEE Symposium on Computers and Communications*, pages 6–11, 1997.

[10] L. Fan, Q. Jacobson, P. Cao, and W. Lin. Web prefetching between low-bandwidth clients and proxies: Potential and performance. In *Proceedings of the SIGMETRICS '99 Conference*, May 1999.

[11] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View From the Edge. In *Proceedings of ACM Internet measurement Conference(IMC), San Diego, CA, USA*, Oct. 2007.

[12] P. Gill, Z. Li, M. Arlitt, and A. Mahanti. Characterizing Users Sessions on YouTube. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN), Santa Clara, USA*, Jan. 2008.

[13] C.-M. Huang, T.-H. Hsu, and C.-K. Chang. A proxy-based adaptive flow control scheme for media streaming. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 750–754, New York, NY, USA, 2002. ACM.

[14] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes. A methodology for workload characterization of e-commerce sites.

[15] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve World-Wide Web latency. In *Proceedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, July 1996.

[16] R. Rejaie and J. Kangasharju. Mocha: a quality adaptive multimedia proxy cache for internet streaming. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 3–10, New York, NY, USA, 2001. ACM.

[17] R. Rejaie, H. Yu, M. Handley, and D. Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet. In *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications (INFOCOM-00)*, pages 980–989, Los Alamitos, Mar. 26–30 2000. IEEE.

[18] M. Saxena, U. Sharang, and S. Fahmy. Analyzing video services in web 2.0: A global perspective. In *Proceedings of NOSSDAV 2008*, 2008.

[19] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of IEEE Infocom*, 1999.

[20] A. Venkataramani, R. Kokku, and M. Dahlin. Tcp nice: a mechanism for background transfers. *SIGOPS Oper. Syst. Rev.*, 36(SI):329–343, 2002.

[21] K.-L. Wu, P. S. Yu, and J. L. Wolf. Segment-based proxy caching of multimedia streams. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 36–44, New York, NY, USA, 2001. ACM.

[22] M. Zink, K. Suh, Y. Gu, and J. Kurose. Watch Global, Cache Local: YouTube Network Traffic at a Campus Network - Measurements and Implications. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN), Santa Clara, USA*, Jan. 2008.