# MANTIS: Time-Shifted Prefetching of YouTube Videos to Reduce Peak-time Cellular Data Usage

Shruti Lall
Georgia Institute of Technology,
Atlanta, GA, USA
slall@ece.gatech.edu

Uma Parthavi Moravapalle*
Georgia Institute of Technology,
Atlanta, GA, USA
parthavi@ece.gatech.edu

Raghupathy Sivakumar
Georgia Institute of Technology,
Atlanta, GA, USA
siva@ece.gatech.edu

## ABSTRACT

The load on wireless cellular networks is not uniformly distributed through the day, and is significantly higher during peak periods. In this context, we present *MANTIS*, a time-shifted prefetching solution that prefetches content during off-peak periods of network connectivity. We specifically focus on YouTube given that it represents a significant portion of overall wireless data-usage. We make the following contributions: first, we collect and analyze a real-life dataset of YouTube watch history from 206 users comprised of over 1.8 million videos spanning over a 1-year period and present insights on a typical user's viewing behavior; second, we develop an accurate prediction algorithm using a K-nearest neighbor classifier approach; third, we evaluate the prefetching algorithm on two different datasets and show that *MANTIS* is able to reduce the traffic during peak periods by 34%; and finally, we develop a proof-of-concept prototype for *MANTIS* and perform a user study.

## CCS CONCEPTS

• **Networks** → Mobile networks; • **Computing methodologies** → Classification and regression trees;

## KEYWORDS

Edge caching; KNN classification; Prefetching

*Presently at Google, Mountain View, CA



*MMSys'20, June 8–11, 2020, Istanbul, Turkey*



ACM ISBN 978-1-4503-6845-2/20/06…$15.00
https://doi.org/10.1145/3339825.3391864

**ACM Reference Format:**
Shruti Lall, Uma Parthavi Moravapalle, and Raghupathy Sivakumar. 2020. MANTIS: Time-Shifted Prefetching of YouTube Videos to Reduce Peak-time Cellular Data Usage. In *11th ACM Multimedia Systems Conference (MMSys'20), June 8–11, 2020, Istanbul, Turkey*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3339825.3391864

## 1 INTRODUCTION

Wireless spectrum is expensive. The federal communications commission's AWS-3 auction (in 1700MHz and 2100MHz blocks) netted approximately $45B for 65MHz of spectrum (at $2.71 per MHz-POP[1]), with AT&T being the highest bidder at $18.2B followed by Verizon at $10.2B. Wireless service providers upgrade their infrastructure and add spectrum in reaction to load characteristics on their networks. It is typical for upgrades to be triggered when there is a reasonably sustained peak usage that exceeds 80% of capacity [1].

Several strategies can be used to address the peak load conditions and hence defer consequent upgrades. Examples of these strategies include reducing the load using compression and deduplication algorithms, improving the efficiency of the communication through protocol optimization, and disincentivizing users from imposing such peak loads by enforcing penalties [2–4]. In this paper we consider the strategy of *time-shifted prefetching*. Specifically, we explore the problem of prefetching content during off-peak periods of the cellular network[2] even when such periods are substantially separated from the actual usage-time. Prefetching is not a new strategy, and has been extensively considered in prior related work [5–7]. What is unique about the focus of this work is the substantially time-shifted nature of the prefetching done with the specific goal of shifting peak load to off-peak periods.

We restrict the focus of this paper to a specific application - *YouTube*, and explore the time-shifted prefetching of videos to the mobile device so that the videos do not have to be

[1]MHz passing one person.
[2]While our contributions can be extended to the scenario of prefetching over "cheaper" WiFi networks, we only focus on cellular networks in this paper.

fetched when watched during peak periods. YouTube videos reportedly account for 38% of a mobile user's cellular data usage [8]. This represents the largest share of the cellular bandwidth usage among all applications on the mobile device. Given such a dominant portion of wireless bandwidth usage, strategies to prefetch YouTube videos during off-peak periods can have a meaningful impact on the overall peak usage of the cellular network. At the same time, focusing on a single application allows for specifically tailored efficient solutions to be developed as we show in the rest of the paper.

Thus, the key question we answer in this paper is the following: *For a given wireless user, can YouTube videos be prefetched during off-peak periods so that the actual cost of fetching videos during the peak periods is reduced?* The key contributions made in the paper are summarized as:

- User dataset analysis: We collect a dataset of YouTube watch history from 206 users. The dataset comprises of 1,798,132 videos spanning a 1-year period. We use the dataset to study whether YouTube watch behavior is predictable, especially up to 24 hours ahead. We show that a significant percentage of the watch behavior (> 40%) is indeed predictable by relying on the past watch history of that user. To overcome any biases in our collected dataset, we also verify this conclusion using an independent dataset collected by Park *et al.* [9].
- Prefetching algorithm: We design and develop a machine learning algorithm for prefetching that is trained on a user's watch history and predicts the user's likely video watch behavior over the next 24 hours. We show that the algorithm performs well for four different metrics: prefetch accuracy (how many of the predictable videos does the algorithm successfully select), prefetch efficiency (how many of the videos prefetching are actually watched by the user), prefetch selectivity (what is the ratio of the number of videos selected for prefetching to the number of videos in the candidate set), and overall accuracy (how well the classification algorithm is able to classify videos).
- Prototype and user study: Finally, we implement the proposed prefetching algorithm using a simple strategy of a control application for the YouTube app on mobile devices. The implementation relies on YouTube's offline mode and stores prefetched videos directly into YouTube's offline folder. We recruit 10 volunteers and evaluate the results of *MANTIS* using the prototype over a 2-week period; we show the overall performance of *MANTIS* and the subsequent reduction in peak-time YouTube traffic across the users. We also show that the implementation is lightweight in terms of CPU, memory, and network consumption.

The rest of the paper is organized as follows: In Section 2 we provide a primer on YouTube and its usage, show the network load imbalance present throughout the day, discuss the related work on prefetching, and present the problem definition we focus on. In Section 3, we describe the YouTube watch history dataset that we collect, associated statistics, and the key motivational results that show that a significant portion of YouTube's watch behavior is indeed predictable. In Section 4, we present the design and details of the prefetching algorithm for YouTube and also discuss the system design for the implementation. In Section 5, we evaluate both the algorithm and the implementation. Finally, in Section 6, we discuss some open research issues not addressed by this work and in Section 7 we present our conclusions.

## 2 BACKGROUND & MOTIVATION

### 2.1 Peak vs. off-peak performance

Traffic load on mobile networks is significantly higher during peak periods. To exemplify this, we performed a bandwidth probe with a Google Pixel smartphone (with Android Pie) and measured the available bandwidth (BW) over a T-Mobile cellular network at different times during a day. The probe was done by running a speedtest that downloads a small file from a web server to the mobile device, and using the download time to estimate the throughput. The speedtest was conducted every 30 minutes on the Android device for 5 consecutive days, while the device was connected to a cellular network; Fig. 1 shows the average of the measurements across 5 days. We observe an increase in the available BW between 2 AM to 5 AM , and a subsequent decrease from 6 AM to 8:30 PM and a gradual increase from 8:30 PM. This indicates that the traffic load varies through the course of the day i.e. low available BW correspond to high traffic, and vice-versa. Similar trends have also been shown in other cellular traffic distribution studies [10, 11]. Using Fig. 1 as a reference, the *off-peak period* is defined as 2am to 5am, and the *peak period* is defined as 5am to 12am, and 12am to 2am. There is thus potential to utilize the available bandwidth during off-peak periods for prefetching video content.

### 2.2 On YouTube usage

YouTube content currently dominates mobile data traffic and is reported to account for 38% of all mobile traffic [8]. Furthermore, YouTube's data traffic usage is the highest among all other mobile apps. As reported by Cisco, the average mobile data traffic consumed per smartphone per month is 2.3 GB, and the average usage for PC/tablets is 3.3 GB per month [12]. The dominance of Youtube's data traffic extends to wireline platforms with Youtube accounting for 58% of the overall downstream traffic. YouTube makes up the largest traffic share in Europe, the Middle East, and Africa.

To further validate the data usage associated with YouTube, we perform a separate study of YouTube cellular data usage, using Amazon Mechanical Turk (a crowd-sourcing platform which allows users to complete tasks for a fee). This effort

collected data from 90 users across the world and was specifically focused on understanding what percentage of cellular usage was attributable to YouTube. Users in the study were required to send in screenshots of their YouTube mobile app usage, for 1 full month, for both cellular and WiFi data. A box-plot showing the percentage of YouTube's data consumption while connected to a cellular network, and also while connected to a WiFi network is shown in Fig. 2. We found that on average, users watched YouTube videos while connected to a cellular network *38%* of the time. For these users, the YouTube mobile app consumed 10.2 GB of data in a month.

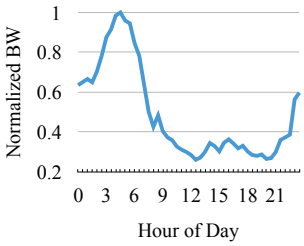


**Figure 1: Normalized available BW across the day**

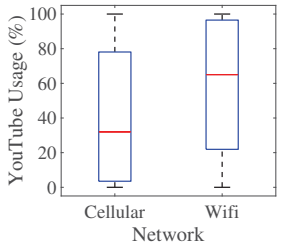


**Figure 2: Usage over Cellular and WiFi networks for 90 users**

## 2.3 Related Works

Prefetching content has extensively been used to reduce user-perceived latency when loading web pages across the internet [13, 14]. These strategies anticipate the content a user is likely to consume, download the content ahead of time, and make the content available at the time of consumption. Time-shifted prefetching, as opposed to just-in-time prefetching, requires that content be prefetched well in advance (potentially a few hours). There have been several solutions that focus on prefetching suitable content based on user's prior interactions with web pages [5–7]. In this paper, we focus on the time-shifted prefetching of YouTube videos.

**Edge prefetching**: To deal with heavy traffic inside core networks, several works prefetch content at edge devices, such as edge servers, routers and mobile devices [15]. [16] utilizes edge servers to prefetch content for image recognition applications. [17] proposes a selective data object prefetching strategy for mobile apps, and [18] performs prefetching for social media multimedia content to mobile devices.

The motivation for prefetching video stems from one of two reasons: 1) to reduce network usage during peak times, and 2) to enable high video viewing QoE by prefetching content to avoid unstable network connections.

**Prefetching for load shaping:** A related work with the goal of reducing peak traffic, was presented in [19] where the authors propose a YouTube video prefetching scheme based only on user channel preferences. While we propose a prefetching scheme that also considers the user channel preferences, it is only one of the several factors affecting prefetching. [20] proposes prefetching episodes from on-demand television series to WiFi access points with the goal of shifting traffic away from peak periods. While this simple scheme works for sequential videos like television series, the problem of prefetching shorter video content (such as YouTube videos) is still unsolved.

**Prefetching for QoE:** [21] tackles the second problem of maintaining high QoE during unstable network conditions by proposing two prefetching algorithms; one that is based on what the users search for, and the other on recommended videos. Both these schemes perform just-in-time prefetching, and consider only the current session. [22] developed CPSys, a mobile video prefetching system that determines other similar users and prefetches content that these users are watching. CPSys requires a central predictor to form a user similarity graph. However, it does not consider any of the user's preferences in the prediction.

## 2.4 Problem definition and goals

The problem we address in this paper is how to shift cellular network load, specifically *YouTube* videos from peak to off-peak periods. The following performance metrics, which are mathematically defined in section 5, are used to evaluate the proposed solution: (i) *Prefetch Accuracy* (PA) is the fraction of watched videos that have been prefetched, over the total number of watched videos; (ii) *Prefetch Efficiency* (PE) is the fraction of watched videos among the prefetched videos; (iii) *Prefetch Selectivity* (PS) is the fraction of prefetched videos among the candidate set of videos; and (iv) *Overall accuracy* (OA) is the fraction of correctly classified videos among the entire candidate set. The goals of the proposed prefetching solution are as follows:

- Decrease the peak-period mobile data traffic consumed by the end-user by ensuring the prediction algorithm has a high PA and PE, with low PS.
- Ensure that the user's real-time video viewing experience is not negatively impacted.
- Be light-weight, and not burden the resource-constrained mobile device's power and storage consumption.

# 3 QUANTITATIVE ANALYSIS OF YOUTUBE USAGE

## 3.1 Methodology

*3.1.1* **Dataset collection:** To collect the dataset, we rely on Amazon Mechanical Turk (mTurk) to gather anonymized watch history from the users [23]. The mTurk platform allows a task to be posted for a fee, which in turn can be completed by users known as mTurkers. Previous studies have shown that mTurk samples can be accurate when studying technology use in the broader population [24]. The task we posted required mTurkers to navigate to Google's *Takeout*

**Table 1: User statistics**

| Attribute | Mean | Std Dev. | Min | Max |
|---|---|---|---|---|
| Videos/day | 15.01 | 6.24 | 0 | 48 |
| Categories | 4.2 | 0.7 | 3 | 13 |
| Playlists | 1.4 | 5.8 | 0 | 24 |
| Subscriptions | 10.9 | 12.8 | 0 | 57 |

**Table 2: Videos statistics**

| Attribute | Mean | Std Dev. | Min | Max |
|---|---|---|---|---|
| Duration (min.) | 7.8 | 45.2 | 0.02 | 222 |
| Views ($\times 10^6$) | 3.2 | 26.9 | 3 | 560 |
| Likes ($\times 10^3$) | 20.6 | 124.3 | 0 | 30,079 |
| Dislikes ($\times 10^3$) | 1.4 | 16.9 | 0 | 9,518 |
| Comments ($\times 10^3$) | 1.9 | 13.2 | 0 | 52,639 |

page and download their YouTube related data. The mTurker would then extract the archive file and select the files related to their watch-history, playlists and subscriptions data; these files were then anonymously uploaded via a dropbox link [3]. The archived file that is uploaded contained the following files: watch-history.html, a JSON file for each playlist created by the user, and subscriptions.json. The watch-history.html file contains a list of all video titles, where the title of the video is a hyperlink to the video URL, viewed by the mTurker, and the associated time it was viewed. This data was collected from 206 mTurkers.

*3.1.2* **Independently collected dataset:** To further overcome any biases in the mTurk dataset, we also show performance results for an independently collected dataset used by Park *et al.* [9]. The authors in this paper performed a data-driven study of the view duration of YouTube videos by collecting data from 158 users over several weeks (by monitoring their YouTube activity via a plugin that needed to be installed by the user). The video IDs, along with the watch-date, appear in the dataset for each user.

## 3.2 Data Highlights

*3.2.1* **Users dataset:** A high-level overview of the statistics of the per-user watch-history data is presented in Table 1. In the collected dataset, there are 1,798,132 videos watched by 206 users. The videos watched per user per day (videos/day), the number of categories the user has watched videos from (categories), the number of playlists the user has created (playlists), and the number of channels the user has subscribed to (subscriptions), is reported in the table.

*3.2.2* **Videos dataset:** The total number of *unique* videos watched by the users is 1,116,271 videos. Table 2 summarizes the metrics associated with the videos watched by the users. Further insights were obtained by analyzing the day of week and time of day that videos were viewed across the 206 viewers for their entire watch-history. The percentage of videos watched is shown across the days of the week in Fig. 3 and across hour of day in Fig. 4. We observe that there is a slight increase in vieweing activity from Friday to Sunday. The plot showing the level of viewing activity across hour of day indicates that there is a lull period from 2 AM to 8 AM. There is a fairly constant level of viewing for the rest of the day (as can be seen from 12 PM to 11 PM).

[3]We were advised by the IRB that IRB approval was not required as no private or personally identifiable information was collected.

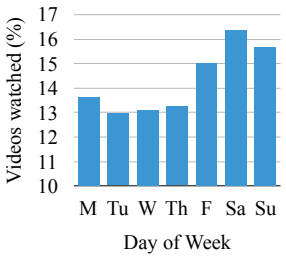


**Figure 3: Percentage of videos watched by day across all users**

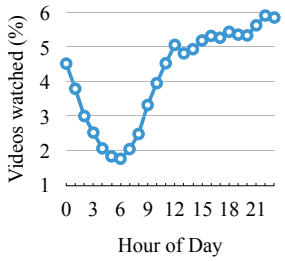


**Figure 4: Percentage of videos watched by hour across all users**

*3.2.3* **User preferences:** Additionally, with regard to user-preferences, on average users watch 95% of their videos from 3 of their most preferred categories. As for channel preferences, on average, 38% of all videos watched by a user are uploaded by their 10 most preferred channels, and 63% are watched from their 30 most preferred channels. The preference of a user's channel and category is indicated through the number of videos that are watched belonging to the particular category or channel i.e. the most number of videos watched by a user belonging to a certain channel, over their watch-history, will be deemed as their most preferred channel; it is similarly computed for their category preference. For playlists preference, we computed on average what percentage of videos are viewed by a user that belongs in user-created playlists; we observed that less than 3% of videos were watched from the user-created playlists. Similarly, for user subscribed channels, it was found that around 10% of videos watched were watched from subscribed channels.

## 3.3 Prefetching Strategies and Potential

Successful prefetching of YouTube videos requires the ability to predict what videos the user is likely to watch in the future. In order to study the feasibility of prefetching, we perform an analysis on a dataset comprised of YouTube usage history collected from 206 users. *Note that the intent of this section is only to show that successful prefetching has potential. We focus on how the prefetching should be done in the next section.*

*3.3.1* **Simple history:** A simple approach for prefetching would be to prefetch videos from the user's watch-history (similar to typical web-caching; YouTube currently does not employ such a cache on the mobile device). Such an approach would work if there is repetition in the user's watch behavior. In other words, *how often does a user watch the*

*same video again?* To explore the feasibility of this approach,

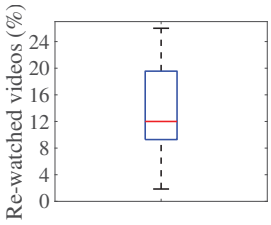


**Figure 5: Percentage of re-watched videos across all users**

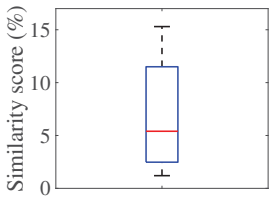


**Figure 6: Watch-profile similarity scores across all users**

for each user in the dataset, we compute the percentage of videos that are watched more than once by the user. The summary of the repeated viewings is shown as a box-plot in Fig. 5. We also calculate the time difference between subsequent watches of the same video. We find that, on average, 13% of videos are re-watched by the user, and the time difference between subsequent watches is on average 3.2 months, ranging from 1 minute to 0.8 years. Based on this preliminary analysis, the following can be concluded:

*The potential for effective prefetching of videos simply based on what a user has previously watched is low, with only 13% of videos being re-watched on average.*

*3.3.2* **Collaborative filtering:** Another approach for time-shifted pre-fetching is to use collaborative filtering in a time-shifted manner. In other words, we explore if there are patterns in the watch-behavior of different users. If users have similar watch patterns, what one user watches could be used to inform the prefetching decision of the other *matching* users. We call these users that have similar watch-patterns as *user clones*. If two users who are *clones* are in different time-zones (eg. GMT and PST) we can predict what the user in the PST zone will watch based on what has been watched by the user in the GMT zone. To find these user clones, we compute a *watch-profile similarity score* across the 206 users, where for every user, this score is computed as the percentage of its videos the user has in common with every other user. Across the 206 users, we found that the highest similarity score was 26.2% between any two users; the median similarity score across all users was only 7.6%. This is shown in Fig. 6.

*Due to the low similarity of videos watched across users, there is minimal scope for utilizing the watch-behavior of other users for prediction in a time-shifted manner.*

*3.3.3* **Recommended Videos:** YouTube's recommendation engine uses sophisticated algorithms to understand user preferences and suggest videos that the user is likely to watch. YouTube's recommendation engine consists of two neural networks that are the candidate generation network and the ranking network. The candidate generation network takes into account the users' watch-history and applies collaborative filtering to obtain videos, and then the ranking network prioritizes and suggests these videos using live A/B testing, to the user. Due to the recommendation engine being dependent on the user's live actions and the prioritization of fresh and popular content, there is no simple approach to obtain the recommended videos set for a user.

To emulate the recommendation engine's behavior for a user, we create a test YouTube account (account that had no prior watch-history) and programmatically re-played the user's watch-history between June 2016 and May 2017 (for the full video length). Using this account populated with the user's history for the past year, we compute, for every video watched between June 2017 and May 2018, the fraction of future videos watched by the user that was in the set of recommended videos shown on the right pane of the video that is currently being watched on the YouTube website (39 recommended videos were scraped for each video that was being replayed during the testing). It is important to note that these recommended videos will not be an exact match of what the user would have seen since the recommendation engine will include videos that have been uploaded to YouTube recently but were not available when the user watched the videos in the history. However, despite these differences, we see that across 10 randomly selected users, a relatively large fraction, 67%, of their future video watches have previously appeared as a recommendation. This is consistent with other reports that approximately 70% of YouTube views are driven by YouTube's recommendation system [25].

*Using recommended videos as the candidate set from which to prefetch videos is a promising solution. Approximately 70% of a user's watch behavior is represented by the videos suggested in the recommended set of videos.*

# 4 THE MANTIS PREFETCHING SOLUTION

## 4.1 Overview

In the previous section, we established that the recommended videos based on the user's watch-history is a promising candidate set for a prefetching algorithm to operate on. That is, videos that are likely to be watched by the user in the future, can be determined from the recommended videos of videos the user has previously seen. However, several fundamental challenges need to be addressed by the prefetching algorithm. One of the key challenges is that even with the focus on recommended videos, the size of the candidate set is likely to be large enough to prohibit prefetching the entire set. *With this core insight, we propose an intelligent prefetching algorithm,* MANTIS, *that accurately predicts videos a user will watch from her candidate videos set, while ensuring an acceptable prefetching efficiency.* In the proceeding sections, we

present *MANTIS* in three stages: 1) generating the candidate set, 2) selecting features for the algorithm and 3) designing the classifier. We also present the system design for *MANTIS*.

## 4.2 Candidate Set Generation

When the candidate set of videos is populated by all the recommended videos of videos the user has previously seen (over the past year), we found that 67% of their future watch behavior is predictable. For the average user that watches 15 videos per day, the size of the candidate set increases by 585 videos for each day in the past we use to populate the candidate set; it is thus infeasible and inefficient to prefetch the *full* candidate set. We thus need to be able to intelligently select videos to prefetch, and for this, being able to accurately capture the user's viewing patterns and preferences regarding the videos they have watched from the candidate set is imperative. As YouTube's recommendation engine cannot exactly be emulated (owing largely to live A/B testing), we cannot obtain precise results regarding which of the recommended videos were watched by the user. As a result, a major caveat with utilizing the recommended videos set is that the classifier cannot operate until the training period is complete i.e. we cannot use the user's past watch-history to understand their interaction with the recommendation engine, but would instead need to monitor their behavior for the entire training period. To address the issue, we explore an alternative set of videos for the candidate set, which can act as a close proxy for the recommended videos set.

*4.2.1* **Related Videos as the candidate set:** YouTube algorithmically determines videos that are related to one another using the video's meta-data, and also collaborative filtering methods- these videos are used as input to YouTube's recommendation engine's candidate generation network. The related videos are independent of the particular user and their watch history. We thus explore utilizing *related videos* as the set of videos from which to prefetch, and use YouTube API's *relatedToVideoId* endpoint to retrieve a list of videos which is related to a particular video.

For a particular user, we fetch 50 related videos of every video that has been watched by the user, and then see if any of the related videos were watched later. We perform this analysis for all the users in our collected data set for their 1 year of watch-history, and found that 59% (with a standard deviation of 16%) of all videos watched by the user were in the related videos set. To see if the related videos set can serve as an effective proxy of the recommended videos set, we perform a similar analysis for the videos that are shown as recommended videos to a currently watched video. We see that the percentage of videos watched from the recommended videos, for 10 randomly selected users, is on average only 4% higher than the percentage of videos watched from the related videos set, as shown in Fig. 7. Thus the related videos can act as a close proxy to the recommended videos set, while avoiding the idle training period that would be required when utilizing the recommended videos as the candidate set.

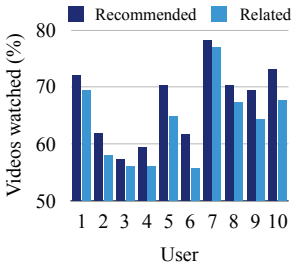


**Figure 7: Recommended vs. related set across 10 users**

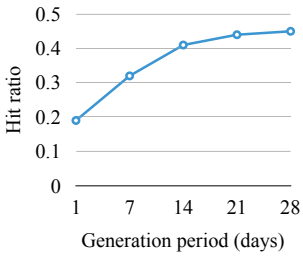


**Figure 8: Hit ratio for varying generation periods**

*4.2.2* **Candidate set generation period.** *MANTIS* generates the candidate set by adding 50 videos that are related to videos previously watched by the user over a certain fixed period in the past; we term this period as the *candidate set generation period*. To determine the optimal *generation period* for the candidate set, for each day of the user's watch history, we compute the fraction of videos that are watched that day which appear in the candidate set (hit ratio) generated over varying generation periods. Figure 8 shows the average hit ratio for 206 users, over their entire watch-history, while varying the generation period from 1 day to 4 weeks. The hit ratio increases from 0.19 to 0.41, when the generation period is increased from 1 day to 2 weeks. Increasing the generation period to more than 2 weeks only exhibits a slight increase on the hit ratio, but considerably impacts the size of the candidate set. Thus, the generation period is set to 2 weeks for *MANTIS*.

## 4.3 Feature Design

For the average user who watches 15 videos every day, *MANTIS* needs to be able to accurately identify 15 videos the user is likely to watch the next day from 10,500 potentially distinct videos (from the related videos the user has watched over the last 2 weeks). To obtain this precision, being able to effectively encapsulate the user's viewing behavior is pertinent. Thus the features that are used for the classification algorithm are of great significance.

*4.3.1* **Feature selection:** The features that we select, which are associated with every *video* in the candidate set, are shown in Table 3. The features are selected to capture the user's preferences (retrieval date, channel ID, category ID, subscribed, repeats, playlist, and tags) and also features that are associated with the inherent nature of the video (time-difference, views, likes, dislikes, comments, subscribers, uploads, and duration).

**Table 3: Features description**

| Feature | Description |
|---|---|
| Retrieval date | Date of when the *video* from which the related videos are obtained, was watched |
| Time difference | Time difference between *video* upload date and prefetching day |
| Views | No. of views of *video* |
| Likes | No. of likes of *video* |
| Dislikes | No. of dislikes of *video* |
| Comments | No. of comments of *video* |
| Channel ID | ID of the channel which uploaded *video* |
| Category ID | ID of the category of *video* |
| Subscribers | No. of subscribers *video's* channel |
| Uploads | No. of videos uploaded by *video's* channel |
| Subscribed | A boolean flag that is set if the user has subscribed to the *video's* channel |
| Repeats | No. of times the *video* has been viewed |
| Duration | Duration of the *video* |
| Playlist | A boolean flag that is set if the *video* appears in a user's created playlists |
| Tags | Tags associated with the *video* |

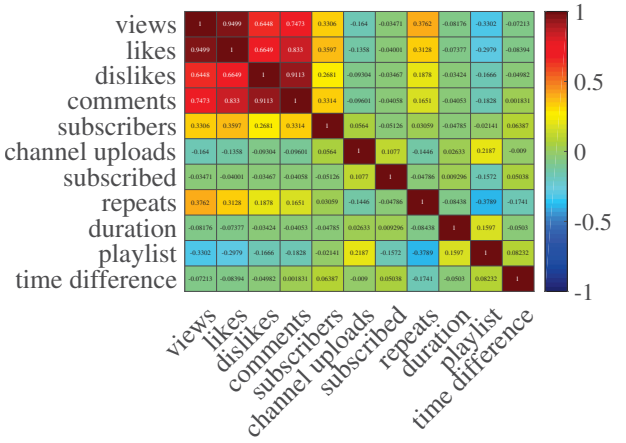


**Figure 9: Correlation Matrix**

*4.3.2* **Dimensionality reduction:** For the numeric features, the Pearson's correlation coefficient, is computed between each feature pair. Figure 9 shows the Pearson correlation matrix containing correlations between every pair of features from Table 3. The coefficients shown are averaged across the watch history of 206 users, made up of 1,116,271 unique videos. We can observe from the correlation matrix that there is a strong relationship between four features for a video - the number of views, likes, dislikes and number of comments. This demonstrates that there is considerable redundancy present in the features.

Given that there is a sizeable amount of data to be considered for accurate prediction of videos a user may watch in the future, utilizing all the features is computationally expensive. *MANTIS* eliminates this redundancy with the application of principal component analysis (PCA) [26]. PCA is a statistical technique that uses an orthogonal transformation to convert a set of features into a set of linearly uncorrelated variables called principal components. The principal components are computed in such a way, that the greatest variance by some projection of the original features, lies on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so forth. The details of the algorithm can be found in [26]. Fig. 10 shows the cumulative variance contributed by each of the components for the 1,116,271 videos. We can observe that approximately 95% of the variance is captured within the first 11 principal components. We can thus reduce the number of dimensions from 15 to 11, thereby reducing the computational complexity associated with large datasets, while still capturing significant variance within the dataset. The 11 principal components containing 95% of the variance are used by *MANTIS* to classify which videos to prefetch.

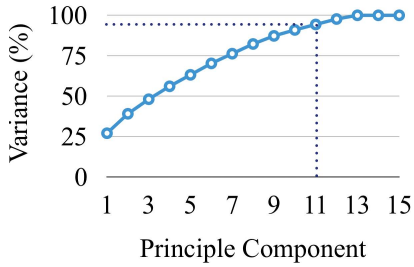


**Figure 10: Cumulative Variance PCA**

## 4.4 Classifier Design

There are two facets to *MANTIS*'s classification algorithm: training, and prediction. These are described as follows:

**Training:** *MANTIS*'s training algorithm involves populating the training set which will be used during the prediction phase, and is shown as Algorithm 1. For each user in the *mTurk* dataset, and for each day in the training period, the following steps are performed:

*1) Populate candidate set:* A list of related videos of all videos watched in the candidate set generation period, is obtained through the appropriate YouTube API call (line 8). The metadata for each related video is fetched, either from the videos database (which stores the video's metadata, line 11) or through the appropriate YouTube API call (line 13).

*2) Prune candidate set:* These videos are further filtered to reduce the candidate set by only selecting videos within the user's preferred channels and category (line 15). This is motivated by the insight that (see section 3.2) users watch 95% of their videos from their 3 most preferred categories, and 63% of the videos from their 30 most preferred channels.

*3) Set classification attribute:* Once the candidate set has been populated for the generation period, a *class* attribute for the video, that indicates whether this video was indeed watched by the user on the day, is set to *chosen* if it is found in the candidate set, otherwise it remains as *discarded* (lines 23-25).

This set is then added to the training data for the classifier (line 27). Once the training data has been obtained, PCA is applied (line 29), and this data is loaded into the classifier (line 30), after which the prediction will commence.

**Algorithm 1:** Prefetching training algorithm

```
1  INPUT: videos DB; preferred channels and categories
     per user; training_period; generation_period
2  OUTPUT: trained classifier
3  PROCEDURE
4  for each day in training period
5    generation_period_start← day - generation_period
6    past_videos← videos watched
     (generation_period_start to day)
7    for each video in past_videos
8      related_video_list← relatedToVideo(video)
9      for each rv in related_video_list
10       if rv is in videos DB
11         metadata← corresponding videos DB entry
12       else
13         metadata← videos_list_by_id(rv)
14         Add metadata to videos DB
15       end if
16       if rv's channel or category is preferred
17         Add rv metadata to candidate set
18       end if
19     end for
20   end for
21   current_videos← videos watched on day
22   Set class to discarded for all videos in candidate set
23   for each video in cadidate set
24     if video in current_videos
25       class ← chosen
26     end if
27   end for
28 Add candidate set to training data for classifier
29 end for
30 Apply PCA to training data
31 Load training data into classifier
```

**Prediction:** For each prefetch day, *MANTIS* aims to predict what the user is likely to watch that day. It performs this prediction by populating the candidate set over the generation period, as was done during the training phase (lines 8 to 13), and then prunes this candidate set (line 15). The classifier intelligently selects, from this candidate set, the videos that will be watched by the user.

*4.4.1* **Classifier selection:** Once the candidate set has been populated, *MANTIS*'s classifier will predict which videos from the candidate set will likely be watched by the user on a particular day. From the data highlights, we found that users tend to watch videos that are similar in nature (for example, 95% of all videos watched by a user was from 3 of their most preferred categories); a classifier that we hypothesize will be well suited for data is the k-nearest neighbor classifier (KNN). KNN is a supervised neighbors-based learning method that predicts the label for a sample based on the labels of a predefined number of training samples ($K$) closest in Euclidean distance to the sample to be classified [27]. It then assigns a class (*chosen* or *discarded*) to the video based on the majority of classes present in the closest $K$ points. As the KNN classifier directly classifies data samples based on feature similarity, it applies well to the prefetching problem.

To validate our hypothesis regarding the choice of the classifier, we implement the aforementioned training and prediction (after feature scaling), and using KNN (for the default value of K=3) and compare it to 3 other popular machine learning algorithms: Gaussian Naive Bayes (GNB), linear support vector machine (SVM) and also random forests (RF). 10-fold cross-validation is applied where the models are trained on 90% of the data, and is tested on the remaining 10%; we compute the classifier accuracy (how accurately it is able to correctly predict which videos from the candidate set are watched) and the area-under-curve (AUC) score (see Table 4). The AUC score provides an aggregate measure of how well the classifier can distinguish between the 2 different classes, and is especially useful for classifiers that are trained on an imbalanced dataset, such as is characteristic with this problem (there are significantly more videos that are not watched from the candidate set than are watched) [28].

**Table 4: Classifier comparison results**

| | KNN | GNB | SVM | RF |
|---|---|---|---|---|
| Accuracy (%) | 77.6 | 38.1 | 57.6 | 69.1 |
| AUC (%) | 84.1 | 42.6 | 64.5 | 78.2 |

It can be seen that the KNN classifier does indeed perform better than the other methods (with the AUC score being 41.5%, 19.6% and, 5.9% higher than GNB, SVM and also RF respectively); for proceeding results, we thus utilize KNN as the classifier for *MANTIS*.

*4.4.2* **Classifier parameter tuning:** There are two important parameters for the KNN classier, namely the value of K, the number of neighbors in the KNN algorithm, and also the training period. The effect of varying the number of neighbors used by the classifier (averaged across 10-fold cross-validation), is shown in Fig. 11; the optimal value for K is found to be 5. Training the classifier on the user's entire watch-history data can result in over-fitting and may be computationally inefficient. Furthermore, we will not be able to take into account the temporal variance of the data

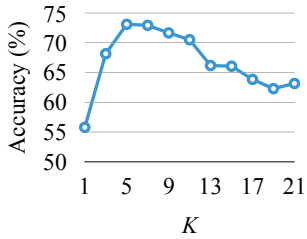


**Figure 11: Accuracy for varying $K$ across 206 users**

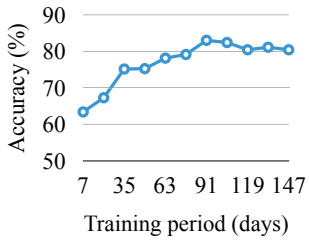


**Figure 12: Accuracy for varying training period across 206 users**

i.e. the way the user's viewing behavior changes over time. We thus evaluate the classifier when the training period is selected as the period immediately preceding the test period. To determine the optimal training period, we utilize the 30 most recent viewing days in their watch-history and vary the value of the training period (see Fig. 12). It can be seen that the classifier performance peaks at a training period of approximately 90 days, after which there is a slight decrease. We thus set the *training period* to be 90 days.

## 4.5 System design

In this subsection, we present a system architecture, as well as the design for a prototype for *MANTIS*. The architecture presented here requires the user to download and install a prefetching app. In Section 6, we discuss an alternative architecture that does not require user involvement.

*4.5.1* **Architecture:** The architecture consists of a *MANTIS* server and a *MANTIS* client, with the server residing on a cloud infrastructure (either on a centralized cloud or a mobile edge cloud), and the native client app on the user's mobile device. *MANTIS* is provided as service to which a user subscribes to, from the *MANTIS* client app during the *start-up* process. After the user has subscribed to the *MANTIS* service, the *MANTIS* client performs in *steady-state* mode. Fig. 13 depicts a high-level overview of the system architecture for a single server-client scenario. In this architecture, YouTube's offline download feature is utilized as means of prefetching the videos. This offline feature is available only with YouTube Premium subscriptions in the U.S.A, however, this feature is freely available in 125 other countries [29]. This feature enables seamless injection of the prefetched video content into the YouTube app's native cache. The *MANTIS* client app interacts with the YouTube mobile app to fetch the videos predicted by the *MANTIS* server.

The *MANTIS* server consists of a mobile sync module that is used to interface with the users. It is responsible for registering the user with *MANTIS* and instantiating the four modules responsible for predicting the content to prefetch for the user. These modules are (i) *Data preprocessor* module, which is used for processing the user's history; (ii) *Training module*, which contains the KNN classifier used to train the network with the user's history; (iii) *Classifier module* that is used for classifying related videos; (iv) the *Prediction module*, which determines the videos to be prefetched by the client.

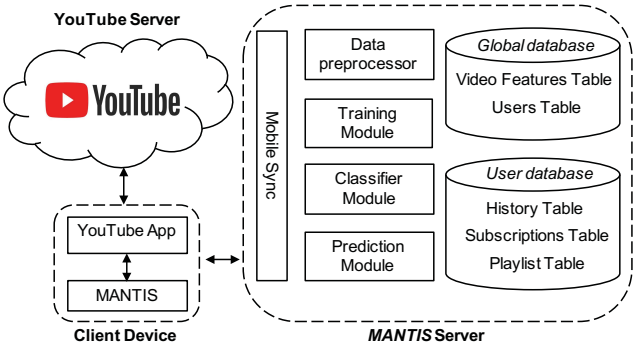


**Figure 13: Primary system architecture**

The prefetching server also consists of a *global database* and *user database*. Each user subscribed to the *MANTIS* service will have a specific instance of the *user database*. Unlike the *user database*, there is only one *global database* from which metadata related to videos are obtained for every user. The *global database* contains a *videos features table* with metadata about videos. The database also contains a *users table* which is a list of the users, with each user identified by a user ID. The *user database* contains details about the user's viewing behavior. The *History table* is used to store the watch history for each user, along with whether the watched video appears in one of their playlists and whether the user is subscribed to the channel which uploaded the video. There are also 2 other tables in this database - the *subscriptions* table containing the list of channel IDs of the channels the user is subscribed to and the *playlist* table containing the list of video IDs from user-created playlists.

Note that this architecture places the burden on the user for its deployment, as the user is required to install the app on their phone. Cellular network providers can reduce this burden by 1) implementing an incentive-model in which the user is offered benefits for installing the application, or 2) the network provider can bundle *MANTIS* with bloatware (software installed on phones by network carriers).

*4.5.2* **Operating modes:** *MANTIS* operates in two modes - the *start-up mode* and the *steady-state mode*. *Start-up mode* occurs when the user first installs the client app and uses it to start the *MANTIS* service. During this mode, there are four main actions which occur:

1) *Create user database and modules:* This process involves instantiating the prefetching modules and creating an instance of the user database on the *MANTIS* server.

2) *Fetch user's data:* The *MANTIS* client on the user's mobile device retrieves the user's watch-history, as well as their current channel subscriptions and videos that appear in their playlists. This information is securely sent to the server over

an encrypted channel.
3) *Populate databases:* The current channels to which the user is subscribed to, and video IDs appearing in their playlists, are used to populate the *subscriptions table* and *playlist table*, respectively. The received watch-history file, is also used to populate the *history table* which has fields corresponding to the features described in Table 3. Furthermore, the global video database is populated with the video IDs received in the watch-history, and metadata is obtained.
4) *Train KNN classifier:* The KNN classifier is trained on the user's watch-history using Algorithm 1.

Once the start-up mode is complete, the *steady-state mode* begins during the off-peak period, and will continue daily until the user decides to stop the service. These actions take place during the steady-state mode:
1) *Fetch user's data:* As in start-up mode, the history from the previous day, subscriptions and playlists are retrieved and securely sent to the server.
2) *Populate databases:* This is the same as with start-up mode, with the respective databases on the *MANTIS* server being updated with the received information.
3) *Train KNN classifier:* The KNN classifier is trained with the user's history, using Algorithm 1.
4) *Predict videos:* Using the trained KNN classifier, the video titles of the predicted videos are retrieved using the algorithm described in Section 4.4. These titles are communicated to the client securely.
5) *Download predicted videos:* Utilizing YouTube's offline download feature, the client app interacts with the YouTube mobile app and downloads the videos sent by the server.

# 5 PERFORMANCE ANALYSIS

We begin this section by defining the following metrics for evaluating *MANTIS*:

- $TP$= Number videos that are prefetched and watched by the user (*true positives*)
- $FP$= Number of videos that are prefetched but not watched by the user (*false positives*)
- $TN$= Number of videos that are not prefetched and also not watched by the user (*true negatives*)
- $FN$= Number of videos that are not prefetched but are watched by the user (*false negatives*)
- *Prefetch Accuracy PA= TP/(TP+FN)*: the ratio of videos correctly classified and watched by the user, to the total number of videos watched by the user from the candidate set; indicates how accurately the prefetching algorithm predicts what a user will watch in the near future.
- *Prefetch Efficiency PE= TP/(TP+FP)*: the ratio of videos correctly classified and watched by the user, to the total number of videos prefetched from the candidate set; indicates how efficient the prefetching algorithm is in prefetching content that is actually consumed by the user.
- *Prefetch Selectivity PS= (TP+FP)/(TP+FP+TN+FN)*: the ratio of videos prefetched to the total number of videos in the candidate set (after the related videos set been pruned); indicates how effectively the algorithm selects videos from the candidate set.
- *Overall Accuracy OA= (TP+TN)/(TP+FP+TN+FN)*: the ratio of videos correctly classified, to the total number of videos in the candidate set; indicates how well the prediction algorithm can classify a video as going to be watched by the user, or not.

**Evaluation methodology:** To evaluate *MANTIS*, the user's data obtained through mTurk, is first parsed and stored, in a Postgres PSQL database with the tables described in section 3.1. The *MANTIS* algorithm was implemented on a macOS Mojave system with a 2.5 GHz Intel Core i7. The parameter values used for evaluating *MANTIS* are: $K$= 5, number of users= 206, and prefetching time= 4 am, *generation period*= 2 weeks unless stated otherwise. The *training period* was empirically determined for each user as described in section 4.4.2. *MANTIS* was evaluated for 30 continuous viewing days for each user in the dataset.

## 5.1 Macroscopic performance of *MANTIS*

*5.1.1* **Bandwidth implications:** The impact of *MANTIS* can be shown in terms of the bandwidth (BW) consumption for the users. When *MANTIS* is implemented, and videos are prefetched during off-peak hours, there is a decrease in the BW consumed by the users during peak periods. This decrease corresponds to a smoothening of the network traffic demand curve. We compute the BW required to download the videos for each user for their test period. The BW reduction per user during peak periods is shown in Fig. 14; this is shown as a function of the number of videos watched by the user over their test period. On average, a BW saving of 3.3 GB across the 206 users is observed (this is computed based on prefetching and watching videos at 480p quality) Fig. 15 summarizes the per-user BW consumption for peak and off-peak periods, with and without the use of *MANTIS*, across all users. We see that *MANTIS* is able to achieve a peak-time BW reduction of 34% while increasing the overall BW consumption by 12% (from 10.6 Gb to 11.9 Gb).

Furthermore, during the off-peak hours, the average amount of YouTube data downloaded is about *half* the data downloaded during peak periods. As we saw in Fig. 1, there is on average *four-times* more available BW during off-peak periods than during peak-periods, making the off-period prefetching feasible without causing an unmanageable burden on the network providers. Also, we found that with *MANTIS*, 180 MB of video data, on average, is downloaded per person, per day during off-peak periods. Thus, it will only take 90 seconds to prefetch 180 MB data at 16 Mbps (typical LTE data-rate [30]), which is well within the chosen

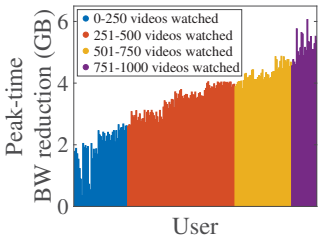


**Figure 14: Peak-time BW reduction across 206 users for 1 month**

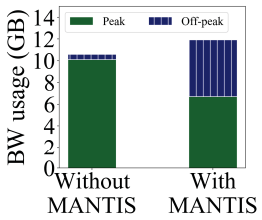


**Figure 15: Average BW usage for 206 users for 1 month**

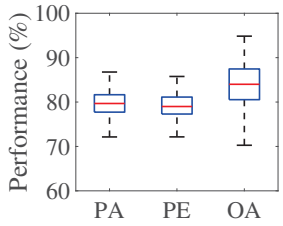


**Figure 16: Performance of *MANTIS* across 206 users**

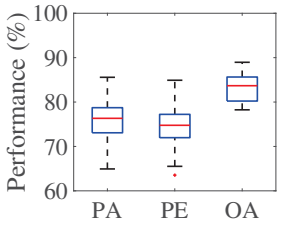


**Figure 17: Performance of *MANTIS* on dataset in [9]**

3-hour off-peak period. As the videos are evicted and replaced after each day from the cache, there is on average 180 MB of video data at 480p quality (at 720p it would be 318 MB, and at 1080p it would be 438 MB) that is stored on the user's mobile device; with mobile devices currently having 16-256 GB available for storage, even at full HD quality, less than 3% of the device's storage is dedicated for storing prefetched videos.

*5.1.2* **Prefetching performance:** In terms of the performance metrics, the results of *MANTIS*, averaged over the 30 day testing period, for all 206 users are shown in Fig.16. We see that *MANTIS* is able to accurately select 79.3% of the predictable videos from the candidate set, and of the videos that are prefetched, 79.1% of the videos are watched by the user. In, addition the average PS is 29.3%, which means that the algorithm is able to fetch 29.3% from the candidate set; the PE from the entire unfiltered dataset is less than 0.001%. Furthermore, *MANTIS* is able to correctly classify 83.2% of the videos in the filtered candidate set. The distribution of the metrics for each user across the 30 days is shown as box-plots in Fig. 16.

*MANTIS* is also evaluated on data that was collected by Park *et al.* [9]. For the 158 users in this dataset, the duration over which the user's history was monitored for, is not consistent and varies from 2 weeks to 14 weeks. *MANTIS* can only be applied to 57 users for whom the required watch-history for the training period was available. For these users, *MANTIS* was used to predict videos for 1 week. Fig. 17 shows the evaluation metrics for the dataset collected in [9]. *MANTIS* performs well for this dataset with PA= 76.2%, PE= 74.2%, OA= 83.1% and PS of 26.7%.

## 5.2 Prototype Results

We developed a proof-of-concept prototype of *MANTIS* for Android devices. The prototype uses Macrodroid [31] to schedule and sequence the various tasks on *MANTIS* client during the steady-state mode [4]. The *MANTIS* server was implemented on a macOS Mojave system with a 2.5 GHz Intel Core i7; the client device and the server exchange information over a wireless FTP.

*5.2.1* **Steady-state operation:** The steady-state operations are given as follows.

*1) Client fetches watch-history*: The client fetches the watch-history at 4 AM each day as a background process. This is performed through a pre-determined sequence of UI interactions with the YouTube App - launching the YouTube app and navigating to the "Manage all activity" page which contains the entire watch-history of the user. This HTML page is downloaded and sent to the *MANTIS* server via FTP. As the page is pulled from the app, there is no request for credentials. The current subscriptions are then fetched by navigating to the subscriptions page, invoking the onClick event associated with the "All" keyword. The playlists are similarly obtained. The HTML history file, subscriptions, and the playlist video titles are sent to the server via FTP.

*2) Server makes prediction:* The various databases are hosted on a PSQL database server in the *MANTIS* server. Upon reception of the data from the client, the server processes the HTML files and appropriately populates the database. The KNN classifier is trained and predictions are made according to the algorithm in section 4.4. Finally, a list of the predicted video titles is sent to the client through FTP.

*3) Client receives predictions:* The client actively listens for messages from the server on the incoming FTP port. If the client receives video title strings from the server, it will loop through the video titles and perform the following steps: (i) launch the YouTube App; (ii) enter title into the search bar; and (iii) prefetch appropriate video by selecting the "download" option from the contextual menu.

**Table 5: Prototype Results**

| Metric | Value |
|---|---|
| Memory Usage | 64 MB |
| Data usage overhead | 65 KB |
| Average CPU usage | 14% |
| Battery usage | 3% |

*5.2.2* **Results:** The proof-of-concept prototype was evaluated for prefetching an average of 6 videos for 10 randomly selected users, for 5 days. Results are shown for the app's memory and CPU usage (when the client is interacting with the YouTube app), data usage overhead (the overhead incurred when uploading data to the server and downloading

[4]iOS devices can use workflow [32] for this purpose

video titles), and also the battery consumption (taken as the difference between battery percentage before the prefetching service starts, until after it has downloaded videos), and is summarized in table 5.

## 5.3 User Study

For the user study, we recruited 12 volunteers from 4 different countries ranging from 16 to 56 years and deployed the *MANTIS* prototype as described in section 4.5. Each volunteer was required to install the prefetching app on their Android device and was told to remain logged into their YouTube account on their mobile device for a period of 2 weeks. The *MANTIS* server was hosted on a macOS Catalina system with a 2.5 GHz Intel Core i7. Prior to the installation of *MANTIS* app on each user's device, we obtained 3 months of their watch-history to train the KNN classifier so that the prediction and subsequent prefetching can occur from 4 AM the next day, and continue so for 13 more days. The average number of videos watched per day by the users is 13.1 (similar to the users in the mTurk dataset). Fig. 18 shows the results for each of the 10 volunteers over 2 weeks.

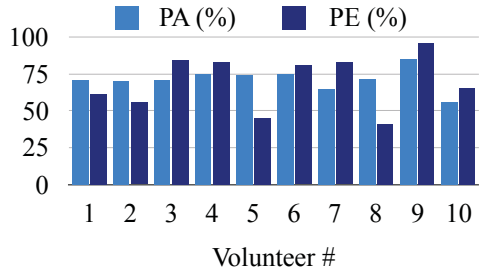


**Figure 18: PA and PE across 10 volunteers over 2 weeks**

The average PA and PE is 71.5% and 69.7% respectively; this is comparable to results achieved across the mTurk dataset (PA= 76.2% and PE= 74.2%). We also found that nearly 48% of peak video traffic was shifted to off-peak periods with the use of *MANTIS*, which is considerably higher than the 34% obtained across the mTurk dataset. This increase can be attributed to the fact that 69% of all future videos watched appeared in the candidate set. A potential reason for this difference could be that the candidate set is populated at the time of prefetching as opposed to ahead of time (which was done in the case of mTurk dataset in simulating *MANTIS*).

# 6 ISSUES & DISCUSSION

**System design:** The system architecture presented earlier in the paper requires that the user have a YouTube premium subscription (if used in a country that does not offer offline download as a free feature [29]). To bypass the requirement of a YouTube premium account, a possible alternative architecture for *MANTIS* is to place a transparent HTTPs caching proxy between the *MANTIS* client and the YouTube server, residing on the client device. The *MANTIS* server's operations are the same as section 4.5.1. All the client actions are performed and prefetched by the proxy.

**Expanding candidate set:** In this paper, we use only the related videos set as the candidate set from which to perform the prefetching. This places an upper bound on the overall prefetching hit ratio (of about 40%) that can be achieved. Other sources of videos can be considered to go beyond this bound. Such sources can include the user's social media network and videos within YouTube's recommendation list.

**WiFi offloading:** Although this paper has focused on shifting network traffic from peak to off-peak for cellular networks, WiFi networks can also be used for prefetching content. There are 2 ways in which this can take place: 1) the WiFi network is used to prefetch content during off-peak hours, or 2) *MANTIS* can be configured to trigger the prefetching when a change in connection is detected- content can be prefetched when the user moves to a WiFi network from a cellular network, or if network connectivity is predictable when the user is likely to leave a WiFi network.

**User quality of experience (QoE) implications:** There are several user-related QoE benefits through the employment of *MANTIS* such as improved video quality and reduction in buffer events during peak-time hours when they would otherwise experience network outages or throttling.

**User privacy:** With the current system architecture, the user's watch-history is sent over a secure channel to a server in the cloud; however, this can be prevented if the prediction module and history databases are locally placed on the mobile device itself, which means that the user's data will not be shared with any other service. We leave this consideration for future work.

# 7 CONCLUSIONS

We address the problem of high peak cellular traffic, through a time-shifted prefetching strategy for YouTube content. A dataset containing the watch-history of 206 users was used to study YouTube watch behavior, and aid in the development of the prefetching strategy that relies on prefetching videos related to content that has been previously consumed by the user. *MANTIS* generates the candidate set, selects features to appropriately encapsulate the user's past behavior, and uses a tuned KNN classifier to select videos from the candidate set. *MANTIS* was evaluated across the users, and also compared to data collected by different authors. We also presented and tested a proof-of-concept prototype for the proposed prefetching solution. We found that an overall reduction, of 34%, in traffic during peak periods was achieved through the employment of this algorithm, while increasing the overall BW consumption by 12%.

## ACKNOWLEDGMENTS

This work was supported in part by the Wayne J. Holman Endowed Chair and the National Science Foundation under grant CNS-1813242.

## REFERENCES

[1] (2019) 4 ways service providers can improve capacity forecasts. [Online]. Available: https://www.sevone.com/white-paper/4-ways-service-providers-can-improve-capacity-forecasts

[2] F. Fusco, M. P. Stoecklin, and M. Vlachos, "Net-fli: On-the-fly compression, archiving and indexing of streaming network traffic," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 1382–1393, Sep. 2010. [Online]. Available: http://dx.doi.org/10.14778/1920841.1921011

[3] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional ip routing protocols," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 118–124, Oct 2002.

[4] J. Malone, A. Nevo, and J. Williams, "The tragedy of the last mile: Congestion externalities in broadband networks," NET Institute, Working Papers 16-20, 2016. [Online]. Available: https://EconPapers.repec.org/RePEc:net:wpaper:1620

[5] K. Lau and Y.-K. Ng, "A client-based web prefetching management system based on detection theory," in *Web Content Caching and Distribution*, C.-H. Chi, M. van Steen, and C. Wills, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 129–143.

[6] S. Sanadhya, U. P. Moravapalle, K.-H. Kim, and R. Sivakumar, "Precog: Action-based time-shifted prefetching for web applications on mobile devices," in *Proceedings of the Fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, ser. HotWeb '17. New York, NY, USA: ACM, 2017, pp. 1:1–1:6. [Online]. Available: http://doi.acm.org/10.1145/3132465.3132473

[7] J. Han, X. Li, T. Jung, J. Zhao, and Z. Zhao, "Network agile preference-based prefetching for mobile devices," in *2014 IEEE 33rd International Performance Computing and Communications Conference (IPCCC)*, Dec 2014, pp. 1–8.

[8] (2019) 2019 mobile internet phenomena. [Online]. Available: https://www.sandvine.com/2019-mobile-internet-phenomena-report

[9] M. Park, M. Naaman, and J. Berger, "A data-driven study of view duration on youtube," in *ICWSM*, 2016.

[10] (2016) 2016 global internet phenomena. [Online]. Available: https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf

[11] F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin, "Understanding mobile traffic patterns of large scale cellular towers in urban environment," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1147–1161, Apr. 2017. [Online]. Available: https://doi.org/10.1109/TNET.2016.2623950

[12] (2019) Cisco visual networking index: Global mobile data traffic forecast update, 2017-2022. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf

[13] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," *SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 3, pp. 22–36, Jul. 1996. [Online]. Available: http://doi.acm.org/10.1145/235160.235164

[14] C.-Y. Chang and M.-S. Chen, "A new cache replacement algorithm for the integration of web caching and prefectching," in *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, ser. CIKM '02. New York, NY, USA: ACM, 2002, pp. 632–634. [Online]. Available: http://doi.acm.org/10.1145/584792.584903

[15] G. Li, Q. Shen, Y. Liu, H. Cao, Z. Han, F. Li, and J. Li, "Data-driven approaches to edge caching," in *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*, ser. NEAT '18. New York, NY, USA: ACM, 2018, pp. 8–14. [Online]. Available: http://doi.acm.org/10.1145/3229574.3229582

[16] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Pefetching for image recognition applications at the edge," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 17:1–17:13. [Online]. Available: http://doi.acm.org/10.1145/3132211.3134456

[17] P. Baumann and S. Santini, "Every byte counts: Selective prefetching for mobile applications," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 2, pp. 6:1–6:29, Jun. 2017. [Online]. Available: http://doi.acm.org/10.1145/3090052

[18] Y. Zhao, N. Do, S.-T. Wang, C.-H. Hsu, and N. Venkatasubramanian, "O2sm: Enabling efficient offline access to online social media and social networks," in *Middleware 2013*, D. Eyers and K. Schwan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 445–465.

[19] C. Koch, B. Lins, A. Rizk, R. Steinmetz, and D. Hausheer, "vfetch: Video prefetching using pseudo subscriptions and user channel affinity in youtube," in *2017 13th International Conference on Network and Service Management (CNSM)*, Nov 2017, pp. 1–6.

[20] W. Hu, Y. Jin, Y. Wen, Z. Wang, and L. Sun, "Towards wi-fi ap-assisted content prefetching for on-demand TV series: A reinforcement learning approach," *CoRR*, vol. abs/1703.03530, 2017. [Online]. Available: http://arxiv.org/abs/1703.03530

[21] S. Khemmarat, R. Zhou, D. Krishnappa, L. Gao, and M. Zink, "Watching user generated videos with prefetching," *Signal Processing: Image Communication*, vol. 27, no. 4, pp. 343 – 359, 2012, modern Media Transport - Dynamic Adaptive Streaming over HTTP (DASH). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0923596511001342

[22] A. Gouta, D. Hausheer, A. Kermarrec, C. Koch, Y. Lelouedec, and J. Rückert, "Cpsys: A system for mobile video prefetching," in *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Oct 2015, pp. 188–197.

[23] (2019) Amazon mechanical turk. [Online]. Available: https://www.mturk.com

[24] F. R. Bentley, N. Daskalova, and B. White, "Comparing the reliability of amazon mechanical turk and survey monkey to traditional market research surveys," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '17. New York, NY, USA: ACM, 2017, pp. 1092–1099. [Online]. Available: http://doi.acm.org/10.1145/3027063.3053335

[25] (2018) Youtube's ai is the puppet master over most of what you watch. [Online]. Available: https://www.cnet.com/news/youtube-ces-2018-neal-mohan/

[26] H. Abdi and L. J. Williams, "Principal component analysis," *WIREs Comput. Stat.*, vol. 2, no. 4, pp. 433–459, Jul. 2010. [Online]. Available: https://doi.org/10.1002/wics.101

[27] L. Bottou and V. Vapnik, "Local learning algorithms," *Neural Computation*, vol. 4, no. 6, pp. 888–900, 1992. [Online]. Available: https://doi.org/10.1162/neco.1992.4.6.888

[28] A. P. Bradley, "The use of the area under the roc curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145 – 1159, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320396001422

[29] (2019) Watch videos offline on mobile in select countries. [Online]. Available: https://support.google.com/youtube/answer/6141269?co=GENIE.Platform%3DAndroid&hl=en

[30] (2018) The state of lte. [Online]. Available: https://www.opensignal.com/reports/2018/02/state-of-lte

[31] (2019) Macrodroid. [Online]. Available: http://www.macrodroid.co.uk
[32] (2019) Workflow. [Online]. Available: https://workflow.is