Time-Shifted Prefetching and Edge-Caching of Video Content: Insights, Algorithms, and Solutions

A Thesis
Presented to
The Academic Faculty

By

Shruti Lall

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

August 2022

TIME-SHIFTED PREFETCHING AND EDGE-CACHING OF VIDEO CONTENT: INSIGHTS, ALGORITHMS, AND SOLUTIONS

To the loving memory of my father - my biggest cheerleader

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Siva. He is an exemplary mentor; being wise, insightful, kind and flexible of mind. His supervision has helped me shape my research while granting me the autonomy to explore and innovate. He nudged me to look beyond the obvious and motivated to engage in critical thinking.

I would like to thank the Fulbright Program for their financial support and enabling me to attend Georgia Tech; it is a privilege to be a part of the Fulbright family. Thank you also to the Wayne J. Holman Endowed Chair and the National Science Foundation for their funding support.

I would also like to thank Prof. Fekri, Prof. Blough, Prof. Ramachandran and Prof. Sundaresan for serving on my proposal and dissertation committee. I am most thankful for their valuable feedback and insights in steering my research.

I would like to thank my lab-mates Bhuvana, Ching-Lun, Ekansh, Mohit, Shyam, Uma, and Yubing, who have been a major source of support during my Ph.D. My days at our lab were littered with insightful discussions, captivating brainstorming sessions, interesting lunch breaks, and entertaining coffee runs.

My gratitude extends to Prof. Blough, Prof. Fekri, Prof. Ramachandran and Prof. Sundaresan, for serving on my committee and providing invaluable suggestions and feedback that has improved the quality of this work.

I have also had the great fortune of meeting wonderful people and making amazing friends who have provided stimulating discussions and happy distractions outside of research. I would especially like to thank Anish, Anusha, and Shashi, who have become my family away from home; I will forever cherish the fun times we have had together.

I am extremely fortunate to have been born to a very supportive and caring family. I deeply thank my parents for their gentle encouragement, endless patience and limitless love. My mother is my greatest source of inspiration and has taught me the values of hardwork and determination through example. My father's belief in me gave me the strength to tackle anything that came my way; his support and love have been immeasurable. I am so grateful for my sister, Khushi, who true to her name, has been a source of immense joy. I want to thank my father-in-law, mother-in-law and brother-in-law, Amol, who have been unwavering in their support and so kind in their encouragements. I would also like to thank my extended family for always cheering me on- especially to my aunt, Anupa, and uncle, Amitabh, who are like parents to me.

Finally my husband, Adwait, who is my best friend and a great companion, has loved, supported, encouraged, entertained, and helped me get through this grueling period in the most positive way- Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Video content is by far the largest contributor to Internet traffic, with video currently accounting for nearly 80% of overall global Internet traffic [1]. This is expected to grow further due to a variety of reasons which include the projected increased number of Internet users and faster broadband speeds, expected improvement in video quality and growth in the number of video-capable devices, as well as due to the emergence of video dominated technologies like augmented and virtual reality [2]. As a result of this rapid growth and increasing popularity of video content, the network is heavily burdened. If the network traffic load is near or exceeds the available capacity, it can lead to network congestion, increased latency, and even outages, ultimately resulting in poor user quality of experience. Today's service providers are under constant pressure to increase the capacity of their networks and upgrade their infrastructures, which is extremely expensive, in order to keep up with user demand [3].

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 [4, 5]. In this dissertation, we consider the strategy of *time-shifted prefetching* and *edge-caching* of video content to address these peak load conditions. Prefetching is not a new strategy and has been extensively considered in prior related work [6, 7, 8]. 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. In this context, with network traffic load being significantly higher during peak periods (up to 5 times as much), we explore the problem of prefetching video content during off-peak periods of the network even when such periods are substantially separated from the actual usage-time. The objective of this work is to develop a set of data-driven prediction and prefetching sys-

tems, using machine-learning and deep-learning techniques, which accurately anticipates the video content the user will consume, and caches it to the edge which is closest to the user; this can be the user's end devices or small data storage servers and wireless routers.

Currently, both fixed and mobile internet traffic is dominated by two video streaming services- Netflix and YouTube. In fact, these two applications have the largest share of global Internet traffic. YouTube is the leading application and is alone responsible for nearly 16% of global traffic, and is closely followed by Netflix, which is responsible for 12% of global traffic [9]. While both applications provide video content, they vastly differ in terms of the purpose they serve and the type of videos provided. In this dissertation, we develop prefetching systems specifically for YouTube and Netflix content. To this end, we first collect 3 real-world datasets for YouTube usage, Netflix usage, and joint YouTube-Netflix usage; using these datasets, we draw key insights and provide substantiated directions for solutions to problems in the general domains of networking and telecommunications. We then develop two separate data-driven prediction algorithms and systems for prefetching the YouTube and Netflix content, respectively. Specifically, the three datasets we collect are:

1. YouTube Usage: the first dataset we collect comprises of approximately 1.8 million YouTube videos spanning a 1.5 years watched by 243 users. Using this dataset, we perform a user-level watching behavior study and propose a prefetching system, *Mantis*, which anticipates what YouTube videos a user will consume during peak-hours, based on their past viewing behavior and how the user interacts with related videos. We show that we are able to shift 34% of YouTube peak-time viewing traffic to off-peak hours.

2. Netflix Usage: the second dataset consists of Netflix usage from 1060 users and contains 1-year worth of viewing activity for each user, which amounts to over 1.7 million episodes and movies collectively watched. With this dataset, we present key insights related to how users consume Netflix content. We then develop the

associated prefetching system, *CacheFlix*, which learns how the user watches Netflix series videos and then caches the optimal subsequent episodes under storage and bandwidth constraints. We are able to shift nearly 70% of Netflix series videos to off-peak hours.

3. Joint YouTube and Netflix Usage: the final dataset we collect contains both YouTube and Netflix usage from 377 users; the dataset contains at least 1 year's worth of both YouTube and Netflix activity for each user, which amounts to 4.3 million YouTube videos, Netflix episodes and movies collectively watched. From this dataset, we show results regarding the joint consumption of videos across these platforms and present practical implications based on our findings. In particular, we investigate if and how a user's YouTube and Netflix viewing behavior is related to one another. We also propose an interesting way of enhancing each solution by incorporating the watching behavior on the platform we are not prefetching for i.e., we enhance the YouTube prefetching solution with a user's Netflix watching behavior and vice versa.

We also show how to integrate the two systems for prefetching YouTube and Netflix content. Furthermore, based on our findings from our developed algorithms, we develop a framework for prefetching video content regardless of the type of video and platform upon which it is hosted. We show how the core concepts of our algorithms can be applied to prefetch and edge cache beyond just YouTube and Netflix videos. We also show how the framework can be modified to cater for other type of structured data, as well as for unstructured data.

CHAPTER 1
INTRODUCTION

Internet traffic load is not uniformly distributed throughout the day and is significantly higher during peak periods. A simple way to exemplify this imbalance is by conducting a bandwidth (BW) test probe at regular intervals throughout the day. The BW probe measures the available BW of the network by downloading a small file from a web server and uses the download time to estimate the current throughput. The aggregate available BW over both a fixed and cellular network over a 2 week period is shown in 1.1. We see that the available BW varies through the day, indicating that the traffic load also fluctuates through the course of the day- the corresponding shape of the traffic load can be approximated when we assume that the maximum available BW corresponds to a minimum traffic load. We see that the load is significantly higher during peak hours as compared to off-peak hours. Similar evidence of the load imbalance has been shown in other Internet traffic distribution studies; see Figures 1.2 to 1.4.

If this peak traffic load is near or exceeds the available capacity, it can lead to network congestion, increased latency, and even outages, ultimately resulting in poor user quality of experience. Today's service providers are under constant pressure to increase the capacity of their networks and upgrade their infrastructures, which is extremely expensive, to keep up with user demand [3]. Several strategies can be used to defer consequent upgrades by reducing peak-load usage; these include reducing the load using compression and deduplication algorithms, improving the efficiency of the communication through protocol optimization, and even discouraging users from imposing such peak loads by enforcing penalties [4, 10, 5]. In this dissertation, we consider the strategy of *time-shifted prefetching* and *edge-caching* to address these peak load conditions. Time-shifted prefetching refers to predicting content that a user is likely to access in the future and then fetching that ahead of time. In particular, we focus on utilizing the available BW during off-peak hours to

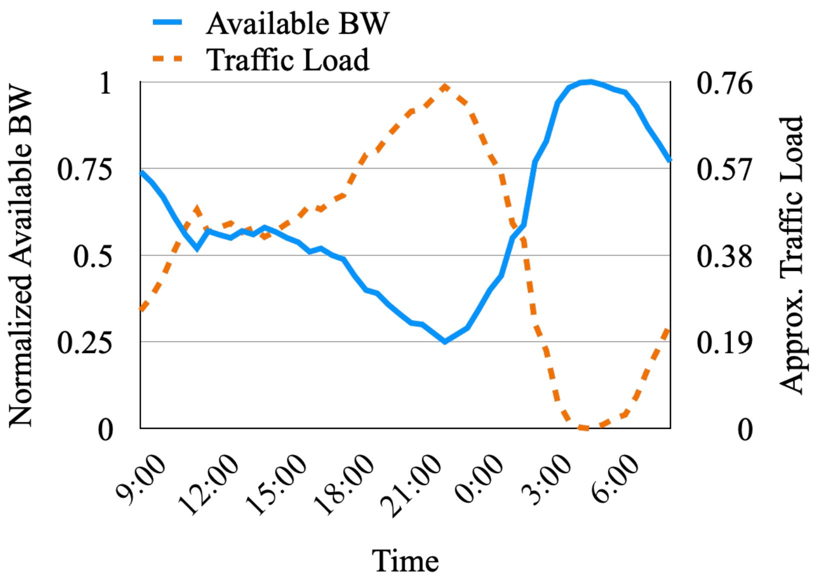


Figure 1.1: Average normalized available bandwidth with the approximate shape of the traffic load

prefetch content and cache it to the edge which is closest to the user; this can be the user's end devices or small data storage servers, and wireless routers.

Video content is by far the largest contributor to Internet traffic, with video currently accounting for nearly 80% of overall global Internet traffic [1]. This is expected to grow further due to a variety of reasons which include the projected increased number of Internet users and faster broadband speeds, expected improvement in video quality and growth in the number of video-capable devices, as well as due to the emergence of video dominated technologies like augmented and virtual reality [2]. We thus narrow the scope of our work to the prefetching and edge-caching of *video* content. Currently, both fixed and mobile internet traffic is dominated by two video providers- Netflix and YouTube. In fact, these two applications have the largest share of global Internet traffic. YouTube is the leading application and alone accounts for nearly 16% of global traffic, and is closely followed by Netflix, which accounts for 12% of global traffic [9]. While both applications provide video content, they vastly differ in terms of the purpose they serve and the type of videos provided.

In this work, we develop prefetching systems specifically for YouTube and Netflix con-

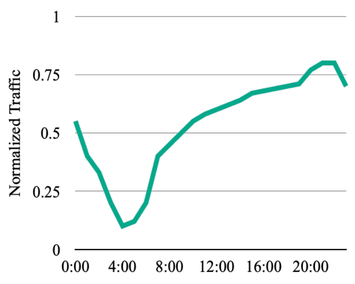


Figure 1.2: Traffic distribution studied by CloudFlare [11]

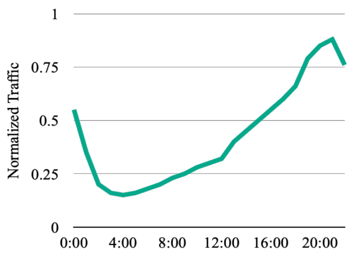


Figure 1.3: Traffic distribution reported by Sandvine [12]

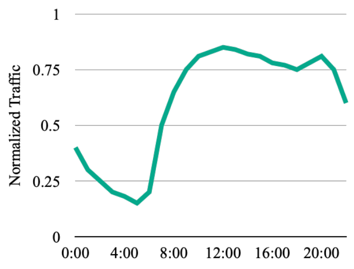


Figure 1.4: Traffic distribution showed by Xu *et al.* [13]

tent. For each system, we first collect 3 real-world datasets for YouTube usage, Netflix usage and joint YouTube-Netflix usage. We use the datasets to analyze and present key insights about user-level usage behavior, and show that researchers can use our analysis can to tackle a myriad of problems in the general domains of networking and communication. Thereafter, equipped with the datasets and our derived insights, we develop a set of data-driven prediction and prefetching solutions using machine-learning and deep-learning techniques (specifically supervised classifiers and LSTM networks). The systems learn the user's watching behavior, anticipate the video content the user will consume in the next peak-hour period, and edge-cache it during the off-peak hours. We also show how to integrate the two systems and develop a framework that extends the prefetching of videos beyond just YouTube and Netflix videos.

1.1 Research Focus

Figure 1.5 shows the research landscape and focus of our work. Specifically, the vertices on the triangle represent the key areas of our work, and the edges represent the relevant intersections of the areas. For example, the edge between the *video traffic* and *prefetching* represents work that encompasses the prefetching of video content.

At the top vertex, there is an abundance of prior work related to video traffic. It can be broadly categorized into 3 areas: video measurement studies which study the characteristics and behavior of video traffic [14]-[16], the prediction of video popularity [17, 18] and also

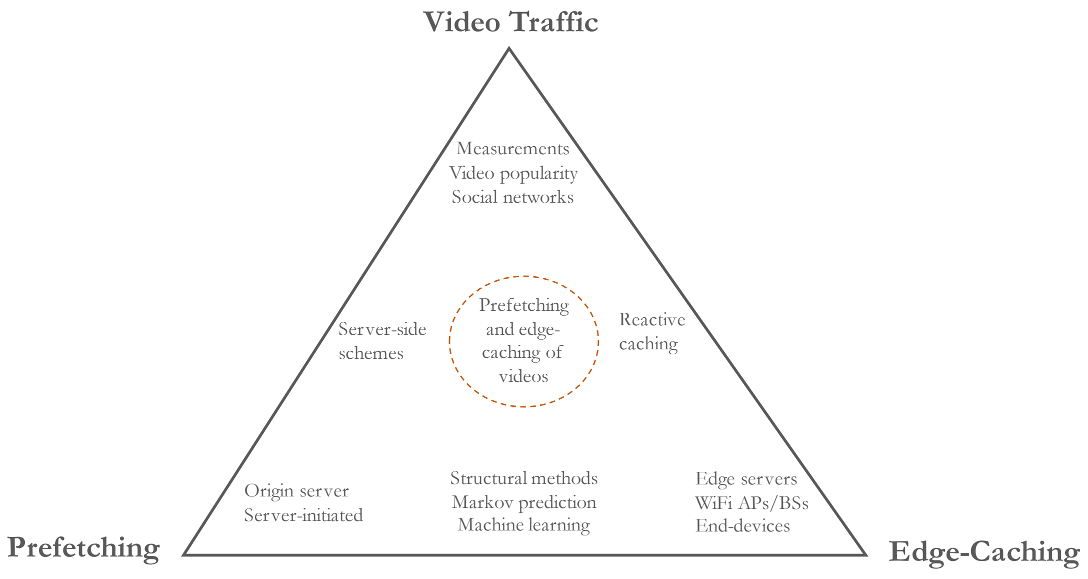


Figure 1.5: Research Focus Landscape

work in understanding the social networking aspects related to video viewing [19, 20]. Concerning the prefetching dominant prior works shown at the bottom left vertex, there has been some early work done for the development of strategies at the origin server and of server-initiated schemes where the computation for the protocols is solely performed at the server [21]-[23]. In terms of edge caching works shown on the bottom right vertex, there is significant work done for the development of algorithms and systems at various points in the network, particularly in content delivery networks [24], edge servers [25], wireless routers or mobile base stations [26] and end-devices [27].

There is considerably less work that lies between the areas of prefetching and video traffic that is *not* cached at the edge, as the benefits of reduced latency and network congestion is greater towards the edge of the network (shown on the edge between the *video traffic* and *prefetching* vertices). The work here primarily involves server-side optimization schemes for prefetching video content [28, 29]. Research for edge-caching video content deals with reactive caching schemes where videos are cached based on what users have previously accessed; there is no prediction component here [30]-[33]. Conversely, the work that lies between prefetching and edge caching is dedicated to predicting content that will be accessed in the future, and then edge-caching that. There are structural methods that are used for the prediction; an example of this is link-prefetching [34]. In more recent times, Markov prediction models and machine learning tools are used for deciding what content to prefetch to edge devices [35]-[37].

The research focus of this dissertation is at the intersection of the areas of video traffic, prefetching, and edge-caching.

1.2 Research Contributions

The research contributions of our work can be summarized as follows:

1. With YouTube being the world's largest video sharing site and reportedly accounts for 38% of a mobile user's cellular data usage [38], we collect and perform an in-depth analysis on a real-world dataset of YouTube usage. The dataset comprises of $1,826,075$ videos spanning a 1.5-year period of watch history. The videos, as an aggregate, represent approximately 65TB of videos watched over a 1.5-year period. With this dataset, we provide a number of insights and associated implications by answering questions regarding a user's interaction with YouTube; these are: i) How often do users watch the same video again? ii) Is a user's watch behavior predictable? iii) What are users' typical YouTube data consumption patterns? These questions pertain to specific representative problems, and our associated analysis provides key insights related to problems in the general area of Internet protocols, algorithms, and systems.

2. Equipped with the YouTube usage dataset and our derived insights, we design and develop a data-driven machine-learning based algorithm, *Mantis*, 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 4 metrics: prefetch accuracy, prefetch efficiency, prefetch selectivity, and overall accuracy. We also implement the proposed prefetching algorithm using a simple strategy of a con-

trol application for the YouTube app on mobile devices. We recruit 10 volunteers and evaluate the results of our solution using the prototype over a 2 weeks and show a subsequent reduction of 42% in peak-time YouTube traffic across the users. Furthermore, we show that the implementation is lightweight in terms of CPU, memory, and network consumption.

3. The second most popular video streaming platform is Netflix which currently accounts for nearly 12% of the global Internet traffic [9]. As a result of the difference in the nature of the videos of YouTube content (relatively short-form videos) and Netflix content (long-form series and movies), a separate set of prediction and prefetching algorithms would need to be developed to prefetch and edge cache Netflix content. To investigate time-shifted prefetching for Netflix content, we collect and analyze Netflix usage from 1060 users. The collected dataset contains 1-year worth of viewing activity for each user, which amounts to over 1.7 million episodes and movies collectively watched. We study the user-level Netflix watch behavior by answering a series of research questions pertaining to the user's watch patterns, watch-session length, user preferences, predictability, and series continuation tendencies. We also implement and evaluate classification models to predict the user's engagement in a series and the likelihood of them continuing to watch a series.

4. We propose a time-shifted edge-caching solution, *CacheFlix*, for prefetching Netflix series content and storing it to the edge that is the closest to users. We first show results from a naive caching solution as well as 2 baseline heuristics that is dependent on the user's past watch behavior. We then design and implement a deep learning caching algorithm that uses global and local learners based on Long Short-Term Memory networks, to cache episodes of Netflix series to the user during off-peak hours. The algorithm is evaluated in terms of how accurately it is able to predict content that the user watches in the future (prediction accuracy) and how efficiently it

consumes bandwidth and stores the content (caching efficiency). We present results for 3 different cache eviction policies and also for edge-caches with storage sizes as small as 2 GB. We also compare our results to related work and show that *CacheFlix* is able to perform 1.8 times better in terms of accuracy and 3.5 times better in terms of efficiency.

5. We collect a dataset for studying the *joint* Netflix and YouTube consumption from 377 users. The collected dataset contains at least 1-year worth of both YouTube and Netflix viewing activity for each user, which amounts to over 4.3 million YouTube videos, Netflix episodes and movies collectively watched. Using this dataset, we investigate if and how a user's YouTube and Netflix viewing behavior is related to one another. We study and derive insights regarding their joint watch patterns, the volume of content consumption, viewing interests, and predictability of their future viewing.

6. Finally, we develop an integrated solution of *Mantis* and *CacheFlix*, and show how *Mantis* can be enhanced with a features representation of the user's Netflix viewing behavior, and vice versa. We also present a framework for prefetching video content during off-peak periods, which is agnostic to the type of videos, the provider, and the platform upon which it is hosted. This framework consists of 5 modules, namely: data processing pipeline, features generator, prediction module, cache manager, and delivery module. We also show how the framework can be modified to cater for other type of structured data, as well as for unstructured data.

1.3 Thesis Statement

Internet traffic load is significantly higher during peak periods as compared to off-peak periods; if this peak traffic load is near or exceeds the available capacity, it can lead to network congestion, increased latency, and outages. *With video dominating global Internet traffic, the available bandwidth during off-peak hours can be efficiently utilized for prefetching* *video content ahead of time so as to reduce peak-time traffic.*

1.4 Thesis Organization

The rest of this dissertation is organized as follows - In Chapter 2, we review related literature in the domains of video traffic characterization, prefetching, and edge-caching. In Chapters 3 and 4, we discuss our contributions to the analysis of YouTube usage, and the developed prefetching system, *Mantis*. In Chapters 5 and 6, we present our work toward developing *CacheFlix*, a time-shifted edge-caching solution for Netflix content. In Chapter 7, we present results related to the joint watch behavior of YouTube and Netflix content. In Chapter 8, we present an integrated solution of *Mantis* and *CacheFlix* and propose a general framework for prefetching video content. In Chapter 9, we delve into some existing challenges of our proposed work and outline additional research directions. Finally, we conclude our arguments in Chapter 10.

CHAPTER 2
LITERATURE SURVEY

In this chapter, we provide a summary of the relevant research pertaining to video traffic characterization, prefetching (also known as “proactive caching”), and edge-caching and compare them with the proposed contributions of this thesis.

2.1 Video Traffic Characterization

There have been several measurement studies performed for characterizing and understanding video traffic. With YouTube contributing to 16% of the world’s internet traffic [39], there have been a plethora of measurement studies in which YouTube video popularity and YouTube video request patterns were investigated [14]-[16]. One of the earlier works investigating the platform was performed by Cheng *et al.* [40] by crawling YouTube’s site and obtaining video meta-data. The authors found that YouTube streaming videos have noticeably different statistics from traditional streaming videos ranging from length, caching strategies to their access pattern and active life space. Similar findings were presented by authors in [14]-[16]. These studies were also alike in their approach to collecting data by either scraping data from the network edge, or by crawling YouTube’s site for publicly available content. The focus of studies from the perspective of users has been limited. Halvey *et al.* [19] examined users’ social behavior with YouTube by analyzing their publicly available online interactions such as commenting and sharing videos. Our work fundamentally differs from previous works in that we are able to present an in-depth, long-term study of how user’s interact with both YouTube and Netflix, and what the implications are for the research community.

Given the short-form nature of YouTube videos, the findings cannot be effectively extrapolated to long-form videos, like Netflix episodes and movies. There has been work towards understanding the characteristics of Netflix traffic as performed by Rao *et al.* [15], in which the strategies that Netflix employs to stream their video traffic is performed, and by Adhikari *et al.* [41], in which the authors perform a measurement study to understand Netflix architecture. These studies, however, are agnostic to users' individual behavior and provide a macro-view of Netflix traffic. Huang *et al.* [42] presented a user behavior analysis on a large-scale video-on-demand system from China, which specifically investigated the individual's temporal characteristics and their interests in popular and recently uploaded content. The dataset which was studied contained a mixture of user-generated content and commercial videos, which were primarily watched on the user's mobile device over mobile networks. While the authors presented an in-depth analysis of their behavior, it was over a considerably shorter period of time (30 days) and did not separate the analysis for short-form vs. long-form content, as we have presented. Yuan *et al.* [43] performed a similar study and focused primarily on the temporal user's access patterns. However, they did not study how the level of activity affects these patterns and neither explored how this differs for different categories and types of videos. The previous works study the user's behavior for a specific video service, Yan *et al.*, on the other hand, attempt to model user video preferences in six popular video websites with user viewing records obtained from a large ISP in China. The authors are unable to present a longitudinal study as their dataset contains viewing records from 2 months, and two-thirds of the users have no more than 10 viewing records and thus suffer from data sparsity.

2.2 Prefetching Content

Prefetching content has extensively been used to reduce user-perceived latency when loading web pages across the internet [44, 45]. 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 the user's prior interactions with web pages [6, 7, 8]. In our work, we focus on the time-shifted prefetching of YouTube and Netflix videos. The motivation for prefetching videos 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: In order to alleviate backhaul congestion, the authors in [46] propose a mechanism whereby files are proactively cached to users' devices during off-peak periods based on file popularity and disseminated to other users via device-to-device communications. The authors assume generic distributions for file popularity to decide what content to prefetch. On the other hand, we focus on a single application that allows for specifically tailored efficient solutions to be developed. Nanopoulos *et al.* [47] proposed a data mining algorithm for generalized web prefetching, in particular, the authors propose a framework that determines the order of dependencies between web-page accesses and factors the noise which affects the method for calculating the appearance frequencies of user access sequences. A closely related work with the goal of reducing peak traffic by prefetching video content specifically was presented in [48], 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.

Prefetching for QoE: There have been prior works that have prefetched video content specifically to end-user's devices with the specific goal of improving the user's QoE. [49] performed prefetching for social media multimedia content to mobile devices, while [50] cached short-form videos to user's devices based on their social network interactions. As these works considered only videos that are shared on social media platforms, the prefetching is limited in scope. [51] 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. [52] developed CP-

Sys, 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. A QoE-driven video segment caching system is proposed by the authors in [53], in which they introduce a two-dimensional user QoE-driven algorithm for making caching and replacement decisions based on both content context (e.g., segment popularity) and network context (e.g., RAN downlink throughput).

2.3 Edge Caching

Edge caching has recently emerged as a promising solution to reduce the burden on data transmission, reduce the overall latency of content distribution, and improve user QoE by caching content near or at the edge of the network. Several works have cached video content specifically to end-user's devices with the specific goal of improving the user's QoE. [49] performed prefetching for social media multimedia content to mobile devices, while [50] cached videos to users' devices based on their social network interactions. Authors in [54] proposed caching segments of videos on the end-users' devices based on the access patterns. [55] utilizes edge servers to prefetch content for image recognition applications, and [56] proposes a selective data object prefetching strategy for mobile apps.

Netflix employs edge caches as a part of their own content delivery network (CDN), named *Open Connect* [57]. Open Connect consists of thousands of geographically distributed data servers that exclusively deliver their content. Even though these servers are placed near the edge, each of these servers caters to millions of users [58]. On the other hand, with the emergence of smart WiFi access points/router, and increased storage space in user end-devices, utilizing these devices as edge caches can allow for fine-grained content demand prediction and significantly improve user's quality of experience (QoE). With the rise in smart WiFi smart-routers or APs, which are equipped with an OS card and has storage space, authors have proposed a content delivery architecture that is based on geographically dispersed groups of “last-mile” CDN servers, e.g., set-top boxes, WiFi APs located within users’ homes [59], [60]. The authors in a closely related work [61], proposed an algorithm for determining *when* to cache episodes from on-demand television series to WiFi access points based on trace-driven simulations. The objective of the proposed algorithm is to balance the trade-off between the cost of fetching the content from the server and the user’s QoE. The authors do not consider the prefetching from the user’s perspective but rather determine when would be the best time to prefetch content from the content servers.

Next-generation networks will be extremely dynamic and complex owing to the 3 principle features it promises:(i) enhanced mobile broadband; (ii) ultra-reliable low-latency communications, and (iii) massive machine-type communications [62]. Edge caching is one of the key elements that can enable next-generation networks to meet stringent latency requirements [63, 64]. Typically, machine learning techniques are used to predict content popularity based on user preferences [65]-[37], cluster users based on similar content interests [67],[68], and optimize caching policies under a given set of constraints and predictions about the state of the network [69]-[71].

There have also been works that specifically utilized LSTM networks for content caching; such a work was presented by Zhang *et al.* [72]. The authors presented an LSTM based edge caching solution for video content. While in our work, the cache node is the end-device or wireless AP, the cache node in the related work is the data-center of the video streaming service provider or at a base station or at the CDN. Narayanan *et al.* [73] claim to be the first to use an LSTM encoder-decoder for content caching; the authors employ the LSTM network to predict the future popularity of heterogeneous content objects for various future time intervals e.g., the next 1-3 hours, 12-14 hours and 24-26 hours. The authors, however, did not evaluate their proposed algorithm on real-world datasets and instead generated synthetic datasets with different popularity Zipf distributions to predict object popularity.

CHAPTER 3

A REAL-WORLD DATASET OF YOUTUBE VIDEOS AND USER WATCH-BEHAVIOR

YouTube is the world's largest video sharing site with more than 2 billion active users [74]. YouTube videos reportedly account for 38% of a mobile user's cellular data usage [38]. This represents the largest share of the cellular bandwidth usage among all applications on the mobile device. Given the prominence of YouTube in terms of the share of wireless resources consumed, it is of much interest to understand the characteristics of YouTube usage that could be of use to researchers. However, beyond the macro-level statistics that YouTube publishes [74], there has been very little work done toward collecting any non-trivial data performing any meaningful analysis on such usage.

This forms the context for this chapter. We use Amazon's Mechanical Turk (mTurk) platform to collect a dataset for YouTube usage from 243 users. The dataset comprises of $1,826,075$ videos spanning a 1.5-year period of watch history. The videos, as an aggregate, represent approximately 65TB of video watches over a 1.5-year period. We believe that the dataset will be of significant use for researchers working on a wide range of problems in the general area of Internet protocols, algorithms, and systems. We perform a baseline analysis of the dataset to identify some interesting standalone nuggets of information such as the average number of videos watched by a user per day, how long a typical video lasts, the typical number of categories videos are watched by a user, the average number of playlists created by a user, and the typical number of channels a user subscribes to.

While we believe that the real value of the dataset lies in other researchers using it for their respective problems, the core contribution of this work includes considering a few representative problems in the domains of networking and communications, and analyzing the dataset to answer key questions pertaining to those problems. Note that the goal of this work is not to solve the problems, but instead provide substantiated directions for solutions *based on insights from the dataset*. Specifically, we consider the following sets of questions:

1. How often do users watch the same video again? If they do see certain videos again, how far apart are the redundant views? Are there any patterns in which videos are likely to be watched again?

2. How much of a user's watch behavior can be predicted? How much of a user's past watch behavior has to be considered to maximize the predictive accuracy while considering the associated costs?

3. How much of a user's watch behavior is influenced by recommendations? Are there certain categories of videos for which the users are more likely to be influenced by recommendations? Are there other attributes of a video (e.g., length, number of likes, etc.) that also influence recommended watch outcomes?

4. How static are a user's video preferences over time? Do they remain static over 1.5 years, or do they change drastically?

5. What are the typical data consumption patterns for YouTube usage for a user? Does this change based on time of day or day of the week? How consistent or bursty is the usage?

For each of these sets of questions, we delve into the collected dataset, extract insights, and provide a summary analysis.

3.1 YouTube and its significance

YouTube is a content community that was founded in 2005, which allows users to post, view, comment, and share videos on the site. It is the most visited website in the world, with just over 2 billion monthly visitors and more than 300 hours of content uploaded every minute [75]. It consumes nearly 12% of global network traffic share (following Netflix and HTTP media streaming), and benefits from being the most commonly embedded video on other sites, including Facebook [76]. YouTube content currently dominates mobile data traffic, and is reported to account for 38% of all mobile traffic [38]. Furthermore, YouTube's data traffic usage is the highest among all other mobile apps. As reported by Cisco, the average mobile YouTube data traffic consumed per smartphone per month is 2.3 GB, and the average usage for PC/tablets is 3.3 GB per month [77].

3.2 Dataset Collection

To collect the dataset, we rely on mTurk to gather anonymized watch-history from the users [78]. 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 [79]. The task we posted required mTurkers to navigate to Google's *Takeout* 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 (we were advised by the IRB that IRB approval was not required as no private or personally identifiable information was collected).

The archived file that was 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. The JSON file for each user-created playlist contains a list of the video IDs for all videos added to that playlist. Similarly, the *subscriptions.json* file contains a list of all channels the user is subscribed to.

3.3 Baseline Characteristics

A high level overview of the statistics of the per-user watch-history data is presented in Table 4.1. In the collected dataset, there are $1,826,075$ videos watched by 243 users. Each video is categorized by the uploader according to 18 predefined categories and added to a particular channel; users can subscribe to the channel (known as subscriptions) and add the video to user-created playlists. 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 shown in the table.

The total number of *unique* videos watched by the users is $1,172,111$ videos. Using YouTube's data API, we obtained meta-data associated with each video in our dataset regarding its video duration, the number of views, the number of likes, the number of dislikes and the number of comments each video has at the time of data collection. Table 4.2 summarizes the metrics associated with the videos watched by the users.

Table 3.1: 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 3.2: Videos statistics

Attribute	Mean	Std Dev.	Min	Max
Duration (min.)	13.2	30.1	0.02	820
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

3.4 Analysis and Key Insights

In the following subsections, we attempt to answer 5 questions to gain insights regarding our users' YouTube watching behavior. Based on the insights, we also present implications and the feasibility of the applications and development of associated technologies. We present results on a per-user basis and also study the effect of various attributes. In particular, we are concerned with the following attributes, namely: 1) video related attributes- the category that a video belongs to, the duration of the video, the number of views the video has; and 2) user related attributes- the hour of day that the video was watched by a particular user, and the day of week that the video was watched by the user. For each of the aforementioned video related attributes, the percentage of videos, from our videos dataset, belonging to each category, video duration window and view count range is computed and is shown as the *overall distribution* in relevant results that follow. Similarly, for user related attributes, we show the overall distribution of videos watched in each hour of the day and day of the week, across all users.

3.4.1 How often do users watch the same video again?

Local caching attempts to speed the access to data by storing data that has recently been accessed by the client. Caching plays a vital role for web traffic and can effectively decrease network traffic volume, lessen server workload, and reduce the latency perceived by end users [80]. A fundamental prerequisite for successful caching is the presence of redundancy in a user's behavior i.e. do users watch the same video again? We seek to capture this redundancy by analyzing how often, as well as when and what types of videos, are watched again by a particular user. We also present the feasibility of local caching based on our findings.

Methodology and Metrics: To explore this aspect, for each user in the dataset, we compute the percentage of videos from their watch-history that are watched more than once by the user. Furthermore, we see whether a video that was watched again belonged to a channel that the user subscribed to or appeared in any of their playlists. We also compute the time difference between subsequent watches. It is also beneficial to understand the characteristics of the videos that are watched again; to this end, for each category, duration window, and views range, we calculate the percentage of videos that are watched again for that parameter value. It is important to mention that for our analysis, a video is considered to be watched again only if the video content is retrieved from YouTube servers and not stored on their device.

Analysis and Discussion: Fig. 3.1 shows the percentage of videos that are watched again by each user, arranged in ascending order. The average percentage of videos that are watched again is 10.9%, and ranges from 0.8% to 33.7%, with a standard deviation of 6.4% and median of 9.2%. With respect to their subscriptions and playlists, we found that 8.4% of user's repeated views are from channels that the user has subscribed to, and 4.3% are from their playlists. We also computed the average time difference between each repeated watch of a video on a per user basis. We found that the average difference between such watches across all users, is approximately 2.8 months, and ranges from 2.7 days to approximately a year, with a standard deviation of 2 months and a median of 2.3 months.

To understand how the video related attributes impact the repeatability of video watches, we further look into this redundancy expressed as a function of video category, duration window and views count. In Fig. 3.2, we see that videos belonging to the "movies" category are the most likely to be watched again (with nearly 10% of all "movies" being watched again), as compared to other categories; we should bear in mind that the fraction of videos in our dataset belonging to the movies category is less than 1%. Following the "movies" category, the videos that are categorized as "shows", "comedy", and "music" are more likely to be watched again. We also analyze whether the video duration affects if a video is watched again or not; this is shown in Fig. 3.3. We see that video duration does not have a large impact on the repeatability, however, the repeated video watches percentage generally increases as the duration increases. Fig. 3.4 shows the percentage of repeated watches of videos with various view counts. We see that in general, videos that have a higher view count are more likely to be watched again.

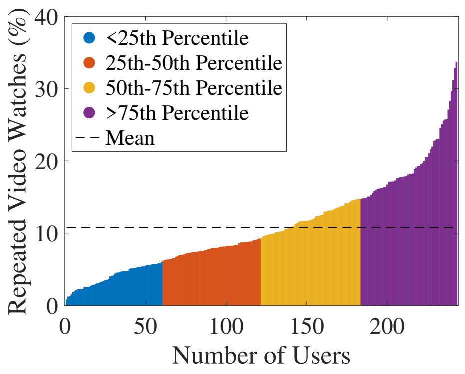


Figure 3.1: Repeated video watches per user

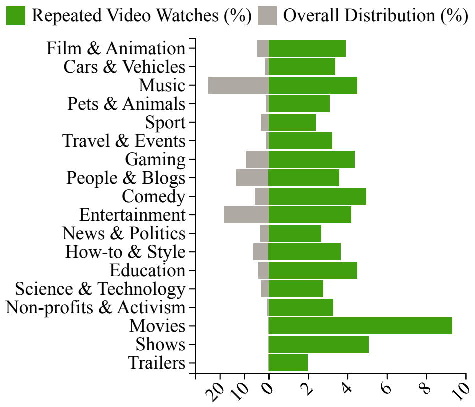


Figure 3.2: Repeated watches per category

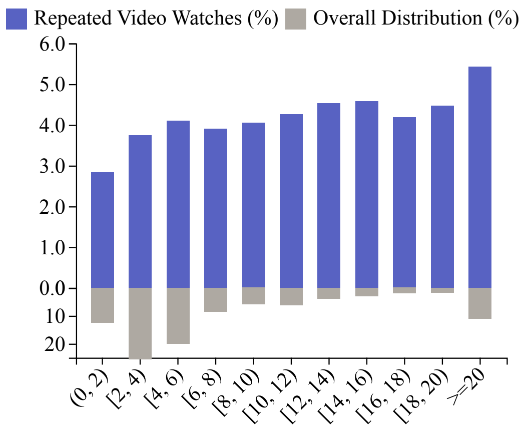


Figure 3.3: Repeated watches for varying duration

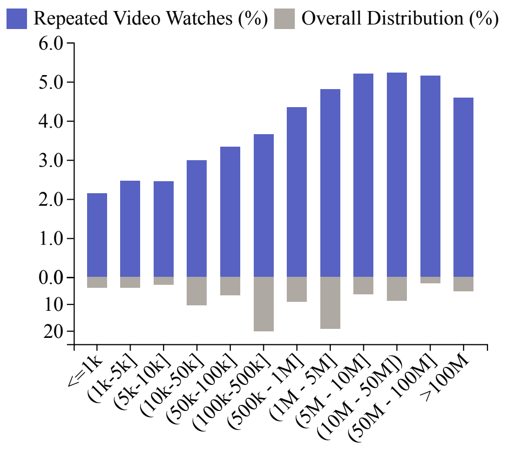


Figure 3.4: Repeated watches for changing view counts

Key Insights: With the use of local caching, even a minute reduction of YouTube traffic volume can lead to savings of tens of millions of dollars for carriers that operate under severe resource constraints [81]. There are also several benefits from the user's perspective; two notably being an improved quality of experience, and reduction in costs associated with network data transfers. Typically, YouTube content is not locally cached beyond caching only video chunks as stipulated by YouTube's employment of the MPEG-DASH protocol [82]. From the results presented in this section, we see that there is a scope for local caching of video content with nearly 11% of videos watched again by the user; algorithms to determine what videos and for how long they should be kept in the cache, can be developed.

With approximately 11% of videos being watched again by a user, local caching can yield an acceptable hit rate; however, the cached content will need to be stored for 2.8 months on average.

3.4.2 Can a user's YouTube watch behavior be predicted?

Being able to predict what a user will watch in the future is particularly useful for prefetching strategies. Prefetching content has extensively been used to reduce user-perceived latency when loading web pages across the internet [44, 45]. 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. To explore the feasibility of prefetching, we consider how a user's YouTube watch behavior is influenced by videos they have seen in the past. Specifically, we see whether videos that are related to videos that have been seen by a user in the past, is consumed by the user in the future.

Methodology and Metrics: YouTube algorithmically determines videos that are related to one another using the video's meta-data, and also by employing collaborative filtering methods. We 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 term this set as the "related set". We perform this analysis for all the users in our collected dataset for their entire watch-history, and present the per-user results, as well as the results pertaining to several video related attributes.

Analysis and Discussion: Fig. 3.5 shows the percentage of videos that are found in the related set of videos they have seen in the past. We find that the average percentage is 59.1%, and ranges from 36.5% to 91.6%, with a standard deviation of 16.2% and median of 58.6%. In addition, 9.6% of these videos are from channels the user has subscribed to, while 2.9% appear in their playlists. In addition, we also study the time difference between when a video was consumed, and when a video related to it, was watched in the future. The average time difference between such watches is 25.9 days, and ranges from 6.4 days to 45.7 days, with a standard deviation of 7.3 days and median of 25.9 days.

The video related attributes we investigate are the category, duration and view count. In Fig. 3.6, we compute the percentage of videos watched in each category that was in the related set of a video that a user watched in the past. We observe that related videos belonging to the "entertainment" category are more likely to be watched as compared to any other category. In Fig. 3.7, we similarly perform the analysis for videos of varying duration; here we see that related videos that are between 4-6 minutes long are most likely to be watched; the least likely are videos from 0 to 2 minutes. Fig. 3.8 shows that, in general, related videos with a higher view count are watched more.

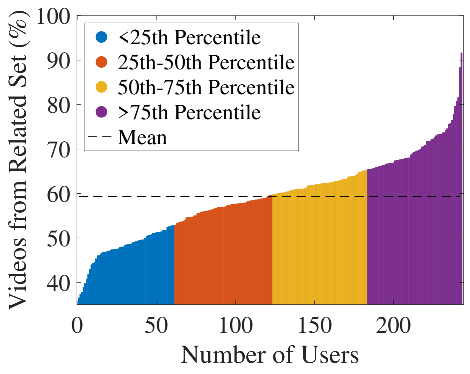


Figure 3.5: % Videos from related set per user

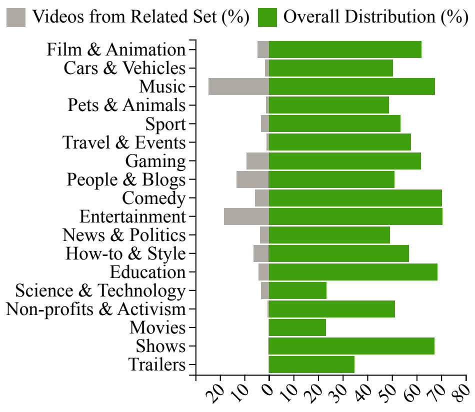


Figure 3.6: Related video watches per category

Key Insights: The motivation for prefetching videos 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. The results presented in this section show that YouTube content is indeed predictable, across categories and especially for more popular videos. Hence, there is a potential for developing successful prefetching systems which can be used to fetch content during low-cost periods (such as over WiFi or off-peak periods).

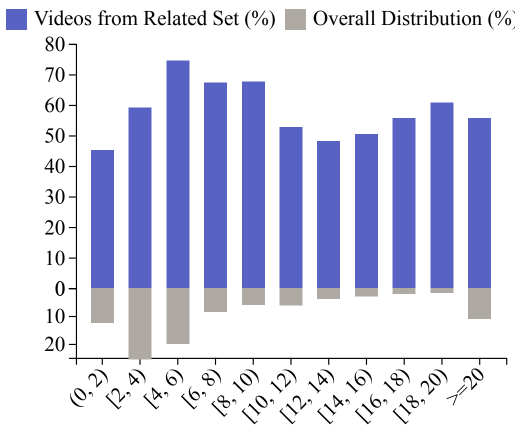


Figure 3.7: Related watches for varying duration

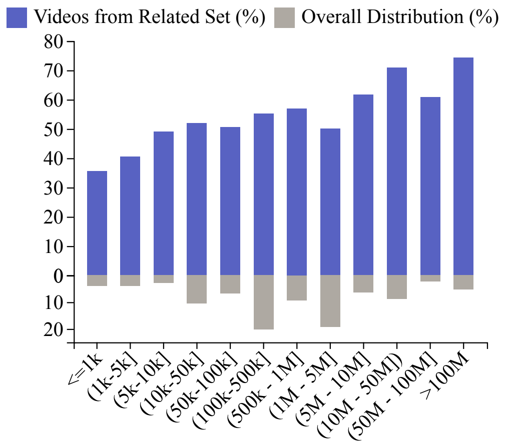


Figure 3.8: Related watches for changing view counts

With 59% of videos watched by a user being present in the related set of videos that the user has previously watched, YouTube watch behavior is predictable and can be used in the development of effective prefetching systems.

3.4.3 Do users consume videos suggested through YouTube's recommendation engine?

YouTube's recommendation engine (RE) uses sophisticated algorithms to understand user preferences and suggest videos that the user is likely to watch. The understanding of how video views are driven through the RE is beneficial for not only the research community (in serving as a case study of how video content is discovered), but also advertisers and content providers [83]. Using our collected dataset, we attempt to independently quantify the effectiveness of YouTube's RE. Furthermore, we provide insights about the types of videos that are better referrers.

Methodology and Metrics: The recommended videos are based on the user's past watch-history and videos identified as "related videos" (as discussed in the previous section) through collaborative filtering and other association algorithms. Specifically, YouTube's RE consists of two neural networks: the candidate generation network and the ranking network [84]. For each video watched by a user, there are recommended videos shown alongside; we term this as the "recommended set". Due to the RE being dependent on the user's live actions and the prioritization of fresh content, there is no simple approach to obtain the recommended set for a user.

In order to approximate the RE'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 for 1 year (for the full video length). Even though the recommended videos will not be an exact match of the videos shown to the user at their time of viewing, we see that across 10 randomly selected users, a relatively large fraction (67%) of their future video watches have previously appeared as a recommendation. We also compute the percentage of videos watched from their related video sets, and find that on average, 63% of their future video views appear in the related video sets; this is only 4% lower than their recommended video sets. Due to the complexity and computational inefficiency associated with emulating the RE, we use the related videos as a close proxy for the recommendation videos set.

We obtain the RE effectiveness by computing the percentage of videos which appeared in the related videos set of the video previously watched; this provides an indication of whether the user clicked on a video that appeared in the recommendation list of a video they were currently watching. We also study how the recommendation system performs across video categories, duration, number of views and also, the hour of day the video was consumed.

Analysis and Discussion: The RE effectiveness per user is shown in Fig. 3.9; the average effectiveness among all users is 21.4%, and ranges from 3.2% to 47.7%, with a standard deviation of 7.5% and median of 21.2%. 8.1% of the recommended videos were watched from the user's subscribed channels, and 2.2% from their user-created playlist. Fig. 3.10 shows the RE effectiveness for different video categories; we see that the RE for the "shows" category is found to be the most effective where the recommended videos from just over 35% of videos watched from this category, is watched by users. Fig. 3.11 shows the effectiveness as a function of view count; in general, recommended content from more popular referrer videos are likely to be be seen. Fig. 3.12 shows how the time of day a particular video was watched affects if a recommended video was watched; we find that the recommendation system tends to be more effective from 5am to 2pm. This corresponds to a decrease in the overall distribution of video traffic. Furthermore, we compute the effectiveness for different video duration, and find that this has no significant impact on the performance of the RE.

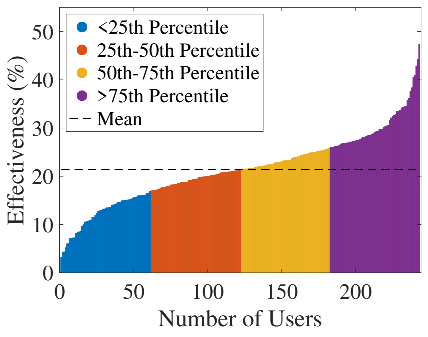


Figure 3.9: RE effectiveness per user

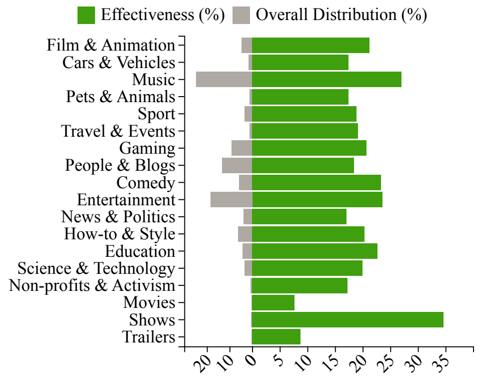


Figure 3.10: RE effectiveness per category

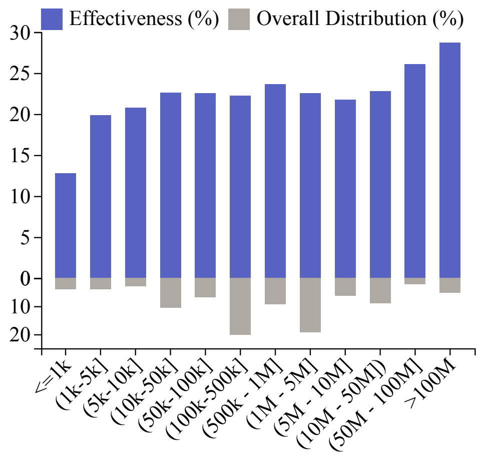


Figure 3.11: RE effectiveness for changing view counts

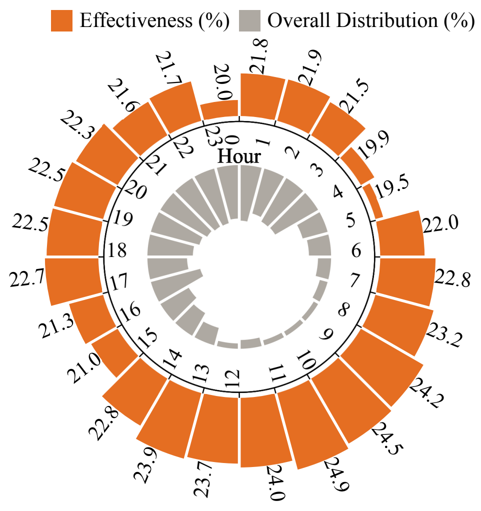


Figure 3.12: RE effectiveness per hour of day

Key Insights: The RE plays a vital role in attracting and retaining users, and also increasing video popularity. Advertisers and content providers will be able to plan their strategies to increase visibility and predict their effectiveness by understanding how and when video's recommended content is consumed. We find that we can anticipate that a user will watch a recommended video over one fifth of the time, even though it is not watched within the same watch session. We also find that there are certain videos that serve as better referrers than others (e.g. more popular videos in "shows" category), and that the RE is more effective during off-peak periods.

For a video currently being watched by a user, approximately 21.4% of videos watched next, appear as a recommendation. Furthermore, the more popular a referrer video is, the more likely a recommended video will be consumed thereafter.

3.4.4 Do users' YouTube video preferences change over time?

The immense prevalence and widespread consumption of YouTube have influenced advertisers to design their strategies incorporating this platform. Advertising revenue on YouTube is estimated to be up to $4.5 billion [85]. User preferences and how this evolves would thus be of interest to advertisers for targeting and personalizing adverts. To gauge how dynamic users' preferences are, we explore how video duration, channel and category preferences change with time.

Methodology and Metrics: We study how the duration of a video influences a user's preferences and whether this changes depending on when they watch the videos. Furthermore, we analyze the user's category and channel preferences, and how this changes over time. The preference strength is proportional to the volume of video content consumed, i.e. the more video content that is consumed from a particular channel or category, the more preferred that channel or category is.

Analysis and Discussion: Fig. 3.13 shows the average duration of videos watched across all users; the average per video duration across all users is 12.8 minutes, and ranges from 9.9 minutes to 13.8 minutes, with a standard deviation of 0.7 minutes and median of 13 minutes. Performing this analysis on a per month basis, we see that the average duration differs by only 0.8 minutes from month-to-month; this is equivalent to a 6.3% change across 1.5 years. Fig. 3.14 and Fig. 3.15 show how the average duration of videos watched by users differs for the time of the day they are watching it, and the day of week the video is watched. We find that during off-peak periods, the average video length is approximately

2.5 minutes longer per video than during peak periods. Over weekends (Friday to Sunday), the average video duration is only slightly higher than during the rest of the week.

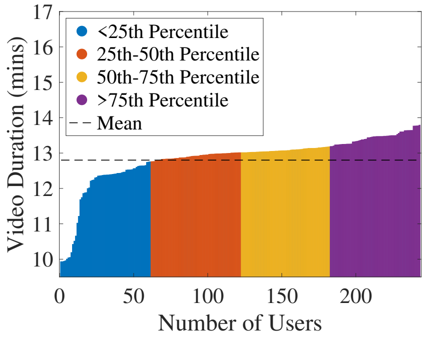


Figure 3.13: Video duration preference per user

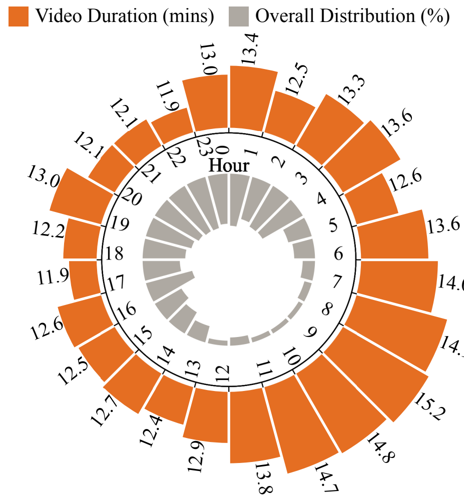


Figure 3.14: Video duration preference per hour

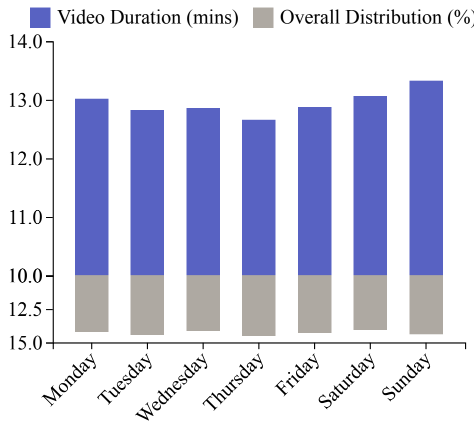


Figure 3.15: Video duration preference per day

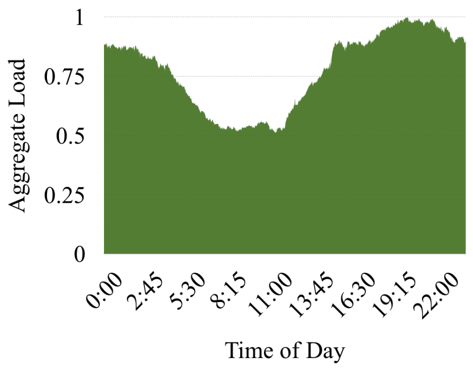


Figure 3.16: Aggregate load across all users

We also evaluate the dynamic nature of users' preferences by studying how their channel and category preferences change over time. We find that users are actually fairly static with 95% of all their watched videos, belonging to their 3 most preferred categories. Similarly, we find that 38% of all video watches are from user's 10 most preferred channels, while 63% are from their 30 most preferred channels. When we perform this analysis on a month-to-month basis for each user, and compute the percentage change of videos watched from their 3 most preferred categories and 30 most preferred channels of the previous month, we find that their consumed video content changes by 4.6% and 32.4% per month for category and channel preferences, respectively.

Key Insights: Learning about user preferences makes it possible to model user information needs and adapt services to meet these needs. Our results suggest that users tend to watch videos between 12 to 13 minutes of length. We also see that user preferences related to the types of videos they watch (characterized by their category and channels) does not vary significantly across time, and so there is potential for time-invariant personalized advertising.

User preferences in terms of the video duration, their 3 most preferred categories, and 30 most preferred channels change by 6.3%, 4.6% and 32.4%, respectively over their 1.5 years of watch history.

3.4.5 What are the typical data consumption patterns for YouTube usage for a user?

Internet access provisioning or network load provisioning is the process of preparing and equipping a network to allow it to handle the anticipated load and provide new services to its users. Predicting the peak workload of an Internet application and capacity provisioning based on these estimates is notoriously difficult [86]. This is because typically, the peaks of individual users are uncorrelated, and so, the network peak load grows much more slowly than the sum of the peak loads of the individual users. To investigate how the user peak load affects overall traffic, we provide results to show the distribution of YouTube traffic across time and how bursty the usage is.

Methodology and Metrics: To understand how YouTube specific network load is distributed through the day, for each user in our dataset, for each minute of day a video was seen, we check to see whether a video is being watched during that minute (here we assume that the video was watched in its entirety unless the start time of the next video watched by the user is before the current video has finished playing). We also calculate the length and gap between watch sessions for each user; we deem a watch session as the period of time that videos are watched within 5 minutes of each other. In addition, we show how the hour of day, and the day of week that videos are watched, affect the watch session length.

Analysis and Discussion: Fig. 3.16 shows the normalized aggregate load across all users for each minute of the day. Here we see that from approximately 5am and 12pm, the load drops significantly. During the rest of the day, the load is nearly twice as much. With regard to the watch session length, Fig. 3.17 shows this on a per user basis. The mean is 26.9 minutes, and ranges from 5.6 minutes to 106.7 minutes, with a standard deviation of 14.4 minutes and a median of 24.8 minutes. In addition, the average time difference between such watch sessions for each user is shown in Fig. 3.18. The mean is 61.3 hours, and ranges from 1.1 hour to 592.8 hours with a standard deviation of 114.9 hours and a median of 20.6 hours.

Fig. 3.19 shows how the watch session length varies for hour of the day, and Fig. 3.20 shows how it varies for day of the week. We see that there is an increase in the watch session length during off-peak periods (5am to 12pm), and also a slight increase on Fridays and Saturdays.

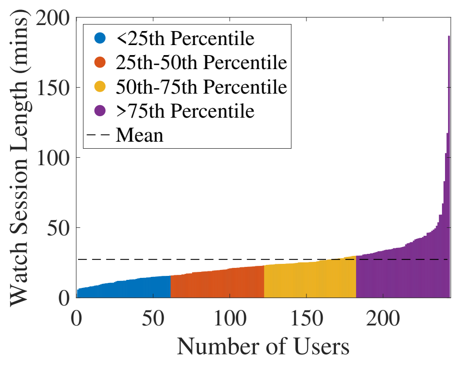


Figure 3.17: Watch session length per user

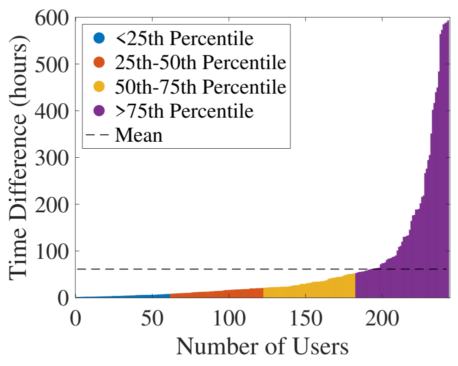


Figure 3.18: Watch session difference per user

Key Insights: With YouTube having a severe influence on network traffic, it is beneficial to understand the distribution of YouTube traffic and users' access patterns. From our analysis, we see that user's access patterns are similar in that their aggregate usage results in a clear distinction between off-peak and peak periods (shown in Fig. 3.16). We find

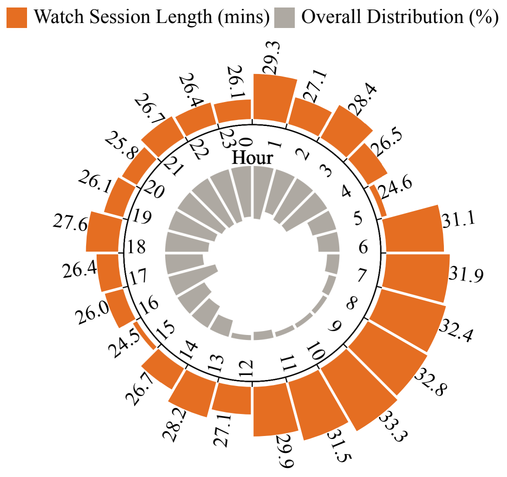


Figure 3.19: Watch session length per hour

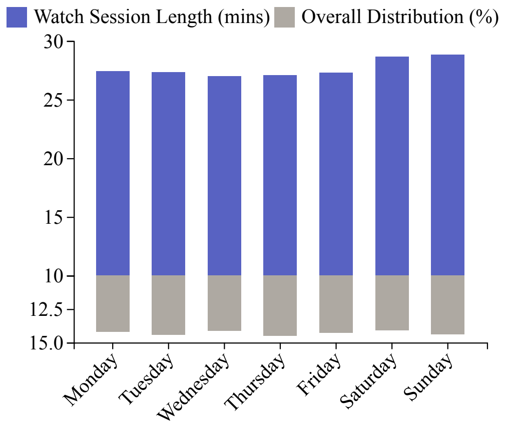


Figure 3.20: Watch session length per day

that YouTube specific network traffic almost doubles during peak hours, while the average watch session length increases by 17% during off-peak hours. Hence, we see that the time of day plays an important role in the burstiness and overall traffic load. Network service providers would thus need to take this into account when developing their provisioning strategies.

YouTube traffic is nearly 2x as much during peak periods as compared to off-peak periods, and the average watch session length increases by 17% during off-peak hours.

CHAPTER 4

TIME-SHIFTED PREFETCHING OF YOUTUBE VIDEOS TO REDUCE PEAK-TIME CELLULAR DATA USAGE

Wireless spectrum is expensive. The federal communications commission's AWS-3 auction (in 1700 MHz and 2100 MHz blocks) netted approximately \$45B for 65 MHz 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 [3].

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 [4, 10, 5]. In this chapter 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 [6, 7, 8]. 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 chapter 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 fetched when watched during peak periods. YouTube videos reportedly account for 38%

[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 chapter.

of a mobile user's cellular data usage [38]. 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 chapter.

Thus, the key question we answer in this chapter 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 chapter 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.* [87].

- 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.

4.1 Background & Motivation

4.1.1 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. 4.1 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 [76, 13]. Using Fig. 4.1 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.

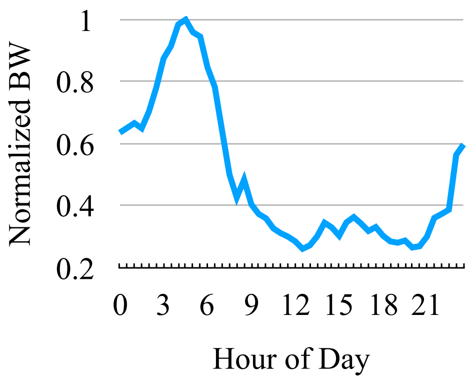


Figure 4.1: Normalized available BW across the day

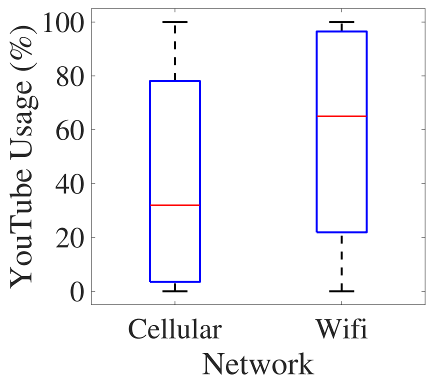


Figure 4.2: Usage over Cellular and WiFi networks for 90 users

4.1.2 On YouTube Data Usage

YouTube content currently dominates mobile data traffic and is reported to account for 38% of all mobile traffic [38]. 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 [77]. 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. 4.2. 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.

4.1.3 Problem Definition

The problem we address in this chapter 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 4.4, 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.

4.2 Quantitative Analysis of YouTube Usage

4.2.1 Methodology

Dataset collection:

To collect the dataset, we rely on Amazon Mechanical Turk (mTurk) to gather anonymized watch history from the users [78]. 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 [79]. The task we posted required mTurkers to navigate to Google's *Takeout* 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

Table 4.1: 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

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.

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.* [87]. The authors in this chapter 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.

4.2.2 Data Highlights

Users dataset:

A high-level overview of the statistics of the per-user watch-history data is presented in Table 4.1. 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]We were advised by the IRB that IRB approval was not required as no private or personally identifiable information was collected.

Table 4.2: 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

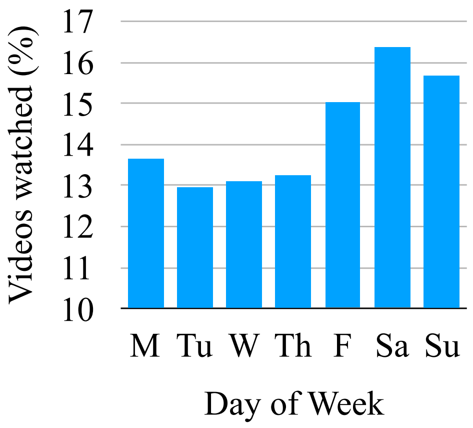


Figure 4.3: Percentage of videos watched by day across all users

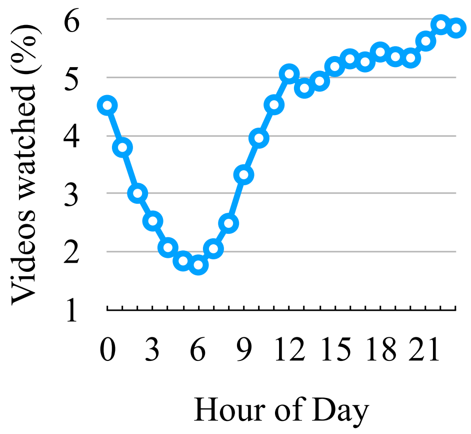


Figure 4.4: Percentage of videos watched by hour across all users

Videos dataset:

The total number of *unique* videos watched by the users is $1,116,271$ videos. Table 4.2 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. 4.3 and across hour of day in Fig. 4.4. 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).

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.

4.2.3 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.*

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, 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

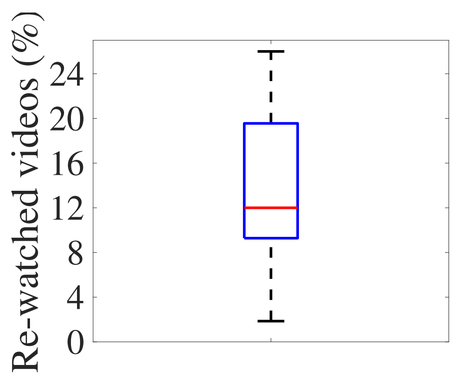


Figure 4.5: Percentage of re-watched videos across all users

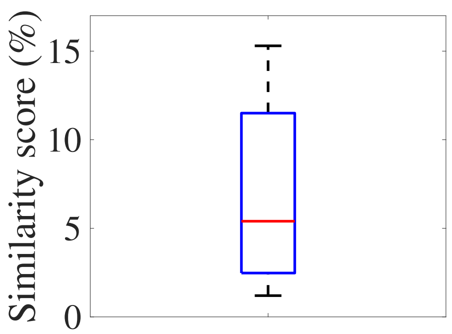


Figure 4.6: Watch-profile similarity scores across all users

repeated viewings is shown as a box-plot in Fig. 4.5. 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.

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. 4.6.

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.

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 [88].

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.

4.3 The MANTIS Prefetching Solution

4.3.1 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*.

4.3.2 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.

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. 4.7. 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.

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 4.8 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*.

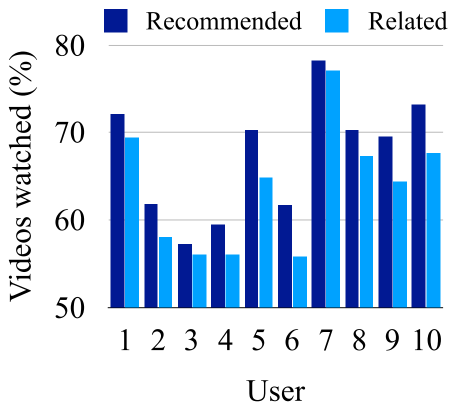


Figure 4.7: Recommended vs. related set across 10 users

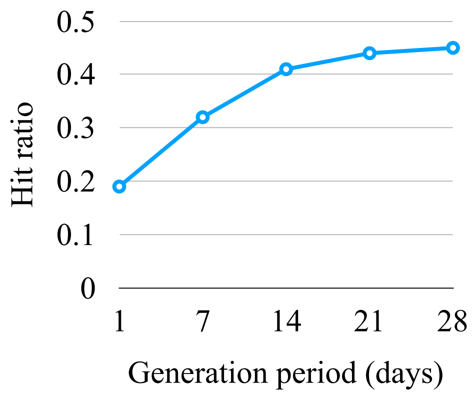


Figure 4.8: Hit ratio for varying generation periods

4.3.3 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.

Feature selection:

The features that we select, which are associated with every *video* in the candidate set, are shown in Table 6.2. 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 4.3: 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*

Dimensionality reduction:

For the numeric features, the Pearson's correlation coefficient, is computed between each feature pair. Figure 4.9 shows the Pearson correlation matrix containing correlations between every pair of features from Table 6.2. 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 -

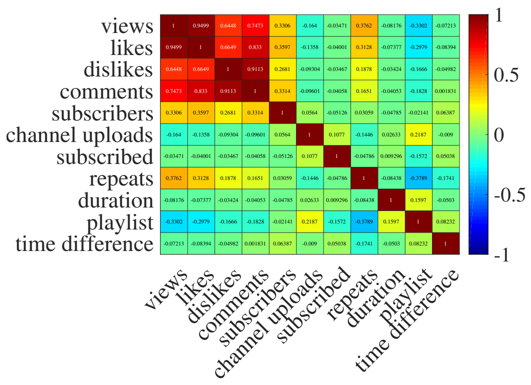


Figure 4.9: Correlation Matrix

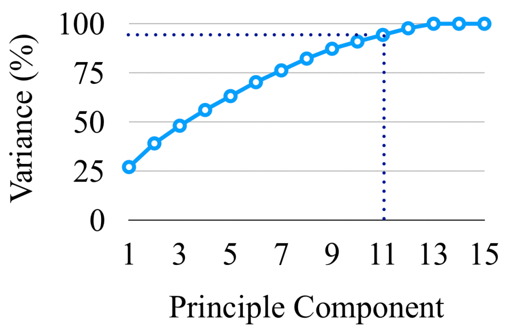


Figure 4.10: Cumulative Variance PCA

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) [89]. 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 [89]. Fig. 4.10 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.

4.3.4 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 4.2.2) 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.

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.

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
```

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 [90]. 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 5.2). 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) [91].

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*.

Table 4.4: 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

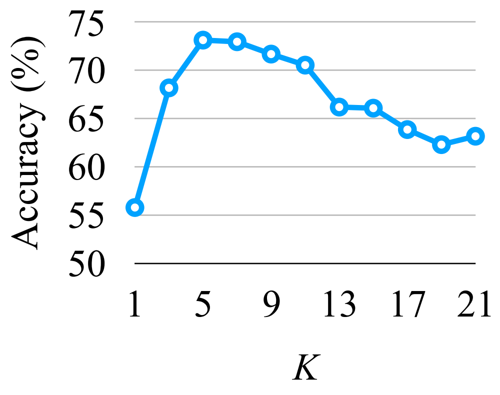


Figure 4.11: Accuracy for varying *K* across 206 users

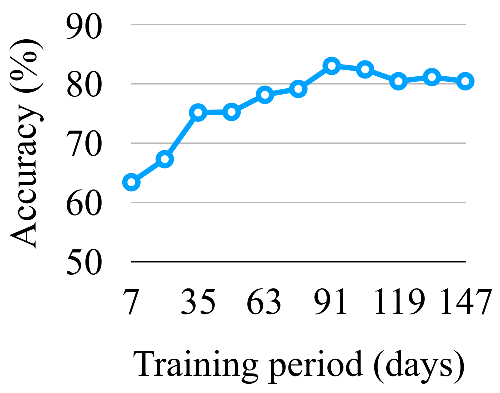


Figure 4.12: Accuracy for varying training period across 206 users

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. 4.11; 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 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. 4.12). 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.

4.3.5 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.

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. 4.13 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 [92]. 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.

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

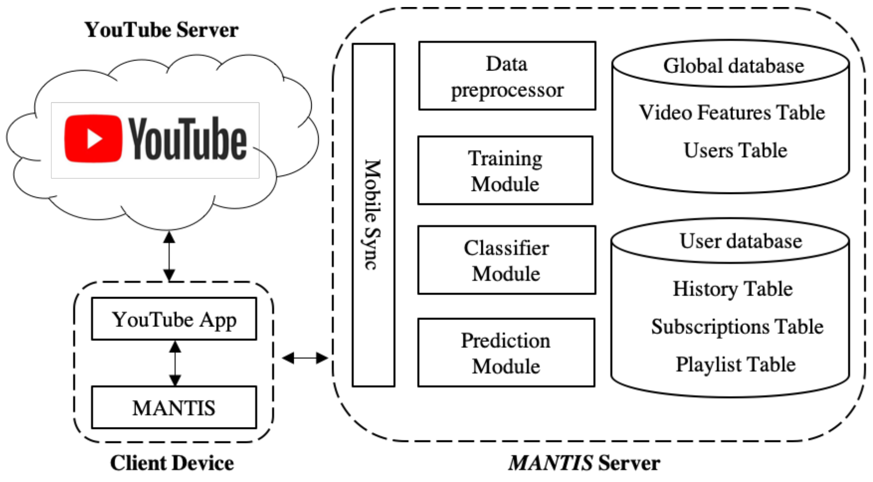


Figure 4.13: Primary system architecture

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).

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 6.2. 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 4.3.4. 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.

4.4 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 4.2.1. 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 31. *Mantis* was evaluated for 30 continuous viewing days for each user in the dataset.

4.4.1 Macroscopic performance of *Mantis*

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. 4.14; 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. 4.15 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. 4.1, 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 [93]), which is well within the chosen 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

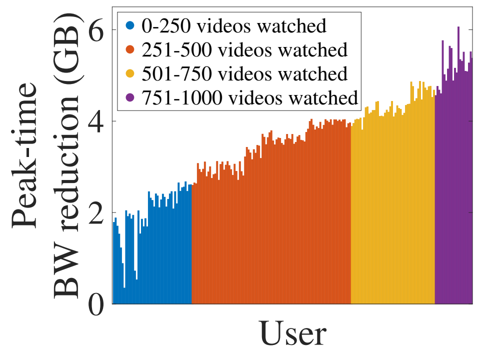


Figure 4.14: Peak-time BW reduction across 206 users for 1 month

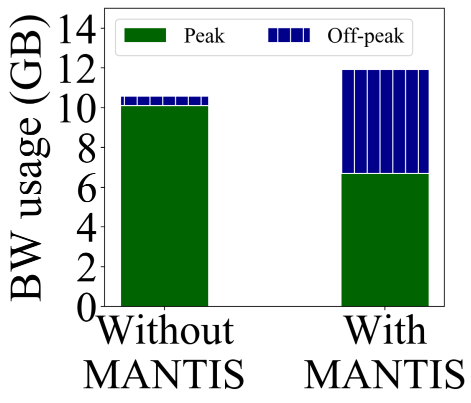


Figure 4.15: Average BW usage for 206 users for 1 month

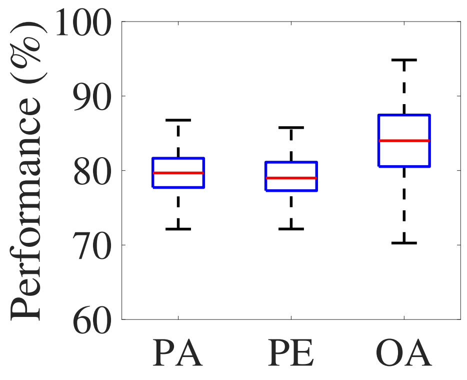


Figure 4.16: Performance of *Mantis* across 206 users

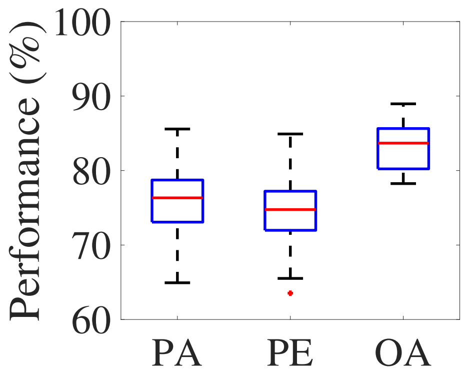


Figure 4.17: Performance of *Mantis* on dataset in [87]

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.

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.4.16. 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. 4.16.

Mantis is also evaluated on data that was collected by Park *et al.* [87]. 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. 4.17 shows the evaluation metrics for the dataset collected in [87]. *Mantis* performs well for this dataset with PA= 76.2%, PE= 74.2%, OA= 83.1% and PS of 26.7%.

4.4.2 Prototype Results

We developed a proof-of-concept prototype of *Mantis* for Android devices. The prototype uses Macrodroid [94] 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.

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.

[4]iOS devices can use workflow [95] for this purpose

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 4.3.4. 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 4.5: Prototype Results

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

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 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 4.5.

4.4.3 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 4.3.5. 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. 4.18 shows the results for each of the 10 volunteers over 2 weeks.

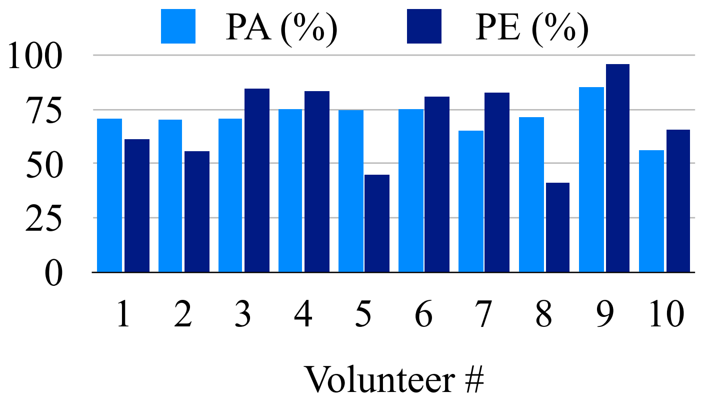


Figure 4.18: PA and PE across 10 volunteers over 2 weeks

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*).

CHAPTER 5

A REAL-WORLD DATASET OF NETFLIX VIDEOS AND USER WATCH-BEHAVIOR

Video streaming accounts for over 60% of downstream Internet traffic, and is expected to grow to 82% by 2022 [1]. Netflix, the world's most popular video streaming service, is alone responsible for nearly a quarter of the world's video traffic [9]. Given the dominance of Netflix on Internet resources, it is valuable to derive insights on Netflix usage which can be useful to not only the research community, but to network operators, content providers, marketing agencies, content creators as well as users themselves. This serves as the primary motivation for our work in which we conduct a meaningful analysis and provide key insights using a real-world dataset of users' Netflix behavior.

To this end, we use Amazon's Mechanical Turk (mTurk) platform to collect a dataset for Netflix usage from 1060 users. The collected dataset contains 1-year worth of viewing activity for each user, which amounts to over 1.7 million episodes and movies collectively watched. Beyond high-level statistics published by Netflix [96], there has been little work done towards collecting and deriving insights using real world usage data spanning a significant period of time.

Equipped with this dataset, we provide an in-depth analysis on user's watching behavior for movies and series content. Movies and series vary vastly in their form with movies being 3 times longer than episodes from series, and also are non-episodic. It follows that the way a user watches movies will be different to how they consume series content. We thus separate the analysis of user's watching behavior for movies and series. We derive key insights for individual user behavior related to their watch patterns, watch-session length, preferences, predictability of their future viewing, and their series continuation tendencies. Furthermore, we implement and evaluate classification models to predict the user's engagement in a series, and the likelihood of them continuing to watch a series. We present our results by grouping our users into 3 categories based on the amount of content they consume: *low* active user, *moderately* active user, and *high* active users. We believe that the real value of the dataset lies in researchers using it for their respective problems. A core contribution of this work includes presenting results in the context of problems in the domain of networking and communications. Specifically, we consider the following sets of research questions (RQs):

1. Do users have a preferred day of viewing for movies/series? What is the number of days between subsequent watches? How does this differ for varying user activity levels?

2. How many episodes do users watch each day? Do active users tend to consume the same amount of content each day?

3. Do users watch the same genre(s) content regularly? Are users inclined to binge watch certain genres over others? Do active users prefer more popular and higher rated content? How much content is related to content the user has previously seen?

4. How much of a user's future watches are predictable? Is it easier to predict for less active users? Are certain genres easier to predict that others?

5. How much of a series does a user watch to its entirety? At what point does a user stop watching a series if they don't complete it? Can we predict when this point will arise? Which classification model is the most well suited for this prediction?

5.1 Data Collection

5.1.1 Methodology

To collect our dataset, we rely on Amazon Mechanical Turk (mTurk) to gather anonymized Netflix viewing history from 1060 users for a 1-year period [78]. The mTurk platform allows a task to be posted for a fee, which in turn can be completed by users known as mTurkers. Studies have shown that mTurk samples can be accurate when studying technology use in the broader population [79]. The mTurkers were required to navigate to their *viewing activity* page associated with their profile, and download their Netflix viewing activity as a csv file; the file was then anonymously uploaded via a dropbox link[1]. The viewing activity file uploaded by the mTurker contains 2 fields: *title* and *date*. The *title* field consists of the name of the feature film or TV series/documentary, as well as the season and episode name where applicable, separated by colons. The *date* field consists of the most recent date that the title was viewed (there is no time of day given).

We then use *The Movie Database* API [97] (TMDB) to obtain the following metadata for each title watched by a user: the release date of the title, the IMDB rating, the number of IMDB votes for the title, the run-time in minutes, the genre(s), director(s), writers, actors, the language of the title, country of production, and related titles. For series, we obtain the number of seasons, and number of episodes each season has through appropriate API calls. A Postgres SQL database is used to store the user's viewing history and metadata.

5.1.2 Baseline Characteristics

A high-level overview of the collected dataset is presented in table 6.1. We show the total number of movies and episodes watched by all the users in the dataset, as well as the total number of seasons and series watched by all our users. We also show the average number of hours a user spends watching series during each watch session (WS). We define a (WS) as a day on which at least one episode of some series is watched by the user.

5.2 Analysis and Key Insights- Movies

In this section, we perform an analysis on the user's movie watching behavior. We group the users into 3 categories based on the number of movies they have watched in their submitted 1-year history. Users in *low* active category have watched less than 20 movies (11% of

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

Table 5.1: Dataset Overview

Description	Value
Users	1060
Movies	63,296
Episodes	1,632,980
Seasons	121,101
Series	30,224
Hours per WS	1.8

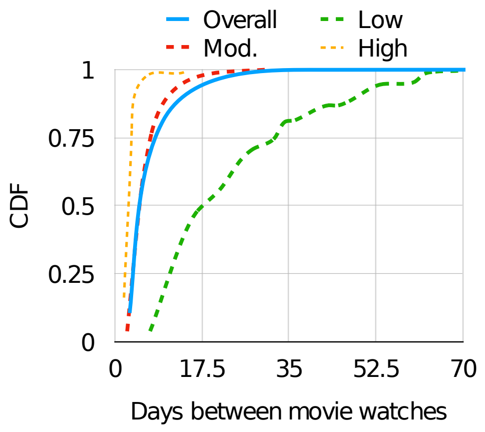


Figure 5.1: Days between subsequent movie watches

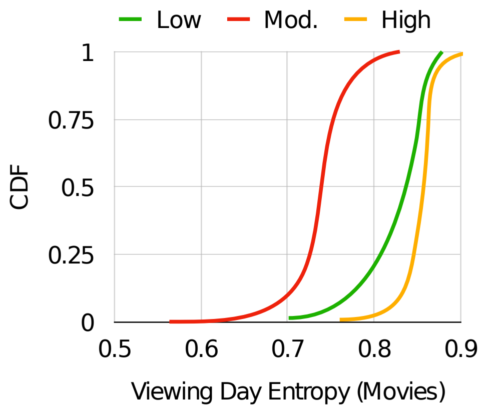


Figure 5.2: Viewing day entropy for movies

users), users in *moderately* (mod) active category have watched between 21 to 100 movies (81% of users), and *high* active users have watched more than 100 movies (9% of users).

RQ1: *How often do users watch movies?*

An important question to consider for load estimation and content delivery systems is how much and how often the user consumes content. The typical user in our dataset watched 56 movies in 1 year, this equates to approximately 1.1 movies per week. For the users 1-year viewing history, we show the number of days between subsequent movie watches in Fig. 5.1 (outliers were removed). We find that 75% of the low, mod, and high active users watch movies every 33.2 days, 6.6 days and 3.5 days respectively.

RQ2: *Do users watch movies on the same day(s)/week?*

To quantify whether users tend to watch movies on the same day(s) each week, we define the *Viewing Day Entropy* (VDE) as given in Eq. 7.1.

$$VDE = \frac{-\sum_{d \in D} p_d \times log(p_d)}{log(N)} \tag{5.1}$$

where
$$p_d = \frac{\text{Number of movies watched on day } d}{\text{Total number of episodes watched by user}} \tag{5.2}$$

and N is the total number of days in a week (N= 7). The VDE is a value between 0 and 1, where a VDE closer to 0 indicates that the user has a more regular request pattern, and a value close to 1 indicates that the user uniformly watches content across the week. Interestingly, we find that less active and highly active users have a higher VDE than moderately active users. This implies that moderately active users tend to watch movies around the same day of the week as compared to other users.

RQ3: *Are movies more often re-watched by active users?*

Local caching attempts to speed the access to data by storing data that has recently been accessed by the client. A prerequisite for successful caching is the presence of redundancy in a user's behavior. Here we analyze if and how often a user re-watches, either parts or the entire, movie. For every user, we compute the fraction of movies that appear in the user's viewing activity more than once (i.e. it was watched on more than 1 day). We find that for low active, moderately active, and high active users, approximately only 3.2%, 7.4% and 8.7% of movies, respectively, are watched more than once. Thus, we conclude that active users tend to re-watch more movies than less active users.

5.3 Analysis and Key Insights- Series

With 96% of the user's Netflix titles being episodes of series, we perform a larger and more in-depth analysis of users' series watching behavior. In the following subsections, we answer questions categorized into 5 groups to gain insights regarding the users' Netflix series viewing behavior. The groups are related to the user's watch patterns, watch-session length, user's preferences, the predictability of Netflix series videos and the continuation of watching series. In order to analyze the user's behavior, and how their levels of activity impact our derived insights, we group our users into 3 categories based on the number of episodes they have consumed in their 1-year viewing history, namely, *low* active users that have watched less than 100 episodes, *moderate* (mod) active users who have watched between 101 and 800 episodes, and *high* active users that have watched more than 800 episodes. Approximately 13% of the users are in the *low* active category, 17% in the *high* active category, and the remaining 70% of the users are in the *moderate* active category.

5.3.1 User Watch Patterns

RQ1: *Do users have a preferred day of viewing?*

We explore whether users have a regular schedule in terms of when they view content; knowing what day a user is likely to access content is particularly helpful for load estimation and caching systems, and consequently can improve the user's Quality of Experience (QoE). We first show the distribution of content watched across the day of the week; this can be seen in Fig. 5.3. In general, the highest % of episodes watched occurred on a Sunday (16.3%), and the lowest on Friday (13.4%). In contrast, low active users watch their least content on Thursdays (12.8%). To quantify whether users tend to watch series content on the same day(s) each week, we compute the VDE as given in Eq. 7.1, where

$$p_d = \frac{\text{Number of episodes watched on day } d}{\text{Total number of episodes watched by user}} \tag{5.3}$$

The CDF of the VDE across users is shown in Fig. 6.7. We find that low active users are slightly more regular in terms of their day of viewing than high active users; however, 50% of all users, regardless of their activity level, have a VDE between 0.6 and 0.83, implying that in general users, do not have a regular schedule in terms of their watch pattern.

RQ2: *What is the number of days between subsequent watches?*

A further important insight related to a user's watch patterns, is what the number of days between subsequent WSs, termed as *time between watch sessions* (TBWS), is. Fig. 5.5 shows the CDF of the TBWS days across the 1 year viewing history for all the users. We find that the TBWS days for 75% of the users is less than 6 days i.e. a typical user watches Netflix at least every 6 days. The TBWS days is nearly 3 days for 75% of the highly active

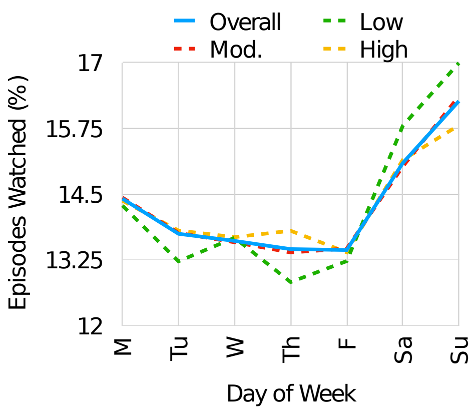


Figure 5.3: Distribution of episodes watched per day

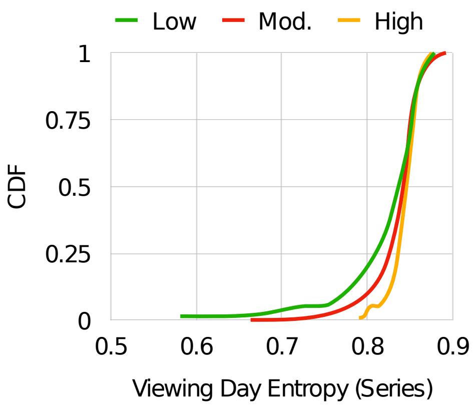


Figure 5.4: Viewing day entropy across 1 year history

users, and 13 days for low active users.

Similar to computing the VDE to see if a user's watch pattern follows a regular schedule, we also compute the entropy for the TBWS days. That is, we compute if the user tends to leave the same number of days between watching Netflix series content, regularly. The *TBWS entropy* (TBWSE) is computed as

$$TBWSE = \frac{-\sum_{i \in I} p_t \times log(p_t)}{log(N)} \tag{5.4}$$

where

$$p_t = \frac{\text{No. of instances when TBWS was } t}{\text{Total number of WSs} - 1} \tag{5.5}$$

and N is the total number of possible TBWS days (N= 28, as the maximum number of days between any 2 WSs was 28 days across in our dataset). Essentially, p_t is the probability that the days between 2 WSs for a specific user is t days. Fig. 5.6 shows the CDF for the TBWSE. We see that highly active users are more regular in terms of the days between subsequent WSs (a smaller TBWSE means a more regular behavior), whereas for the least active users, the TBWSE is closer to 1, implying that the user's watch pattern is sporadic. This is line with the findings from Fig. 6.7 where the highly active users have a larger VDE, indicating a smaller and more regular TBWS.

5.3.2 User Watch-Session Length

RQ3: *How many episodes do users watch per day?*

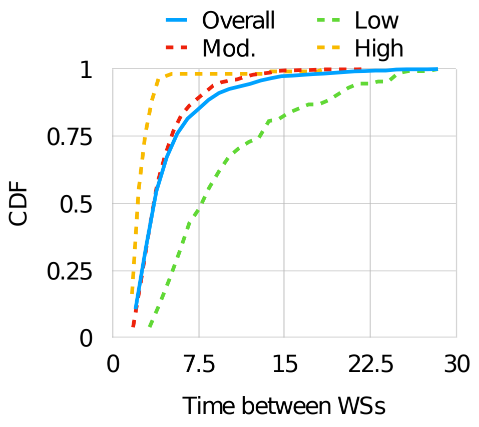


Figure 5.5: CDF of time between WSs days

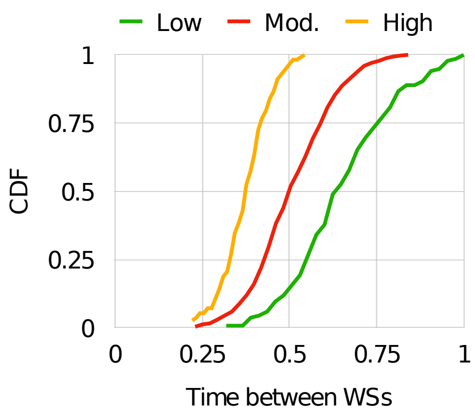


Figure 5.6: TBWS entropy across 1 year history

In effort of understanding user's viewing behavior as well as for the design of content delivery, caching and load estimation systems, it is crucial to know about how much content is consumed by a user. We show the CDF of the number of episodes watched in each WS across all our users' viewing history in Fig. 5.7. For 75% of the users in our dataset, at most 4.5 episodes are watched per day. For highly active users, 75% of the users watches at most 6 episodes per day and for the least active users, it is 3.5 episodes per day. The typical user in our dataset watches 2.7 episodes each day- using the runtime associated with watched episodes, this is equivalent to spending approximately 1.5 hours during each WS. This corresponds to 4.5 GB of a user's data when streaming in HD [98]. Furthermore, for a highly active user, the average user watch 5.3 episodes per day, spends 2.9 hours on Netflix series, and uses 9 GB (streaming at HD) of data each day.

RQ4: *Do active users watch the same no. of episodes daily?*

An important consideration for prefetching and caching systems, is being able to effectively predict how much content a user will see, usually based on their past behavior. It follows that users with uniform behavior will be easier to predict for than users with inconsistent behavior. We observed that some users drastically increase or decrease the number of episodes they watch in 1 WS as compared to previous WSs. To quantify if the user tends to watch the same number of episodes during each WS, we compute the *episode consumption entropy* (ECE) as given in Eq. 5.6.

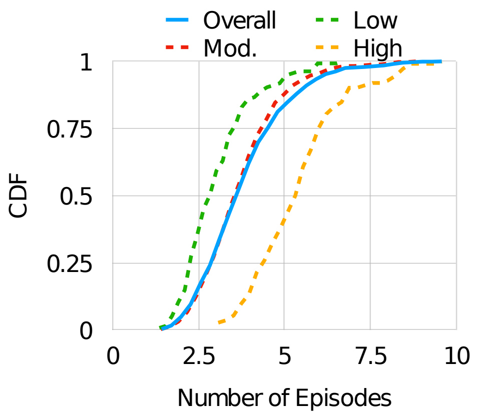


Figure 5.7: CDF of No. of episodes watched per WS

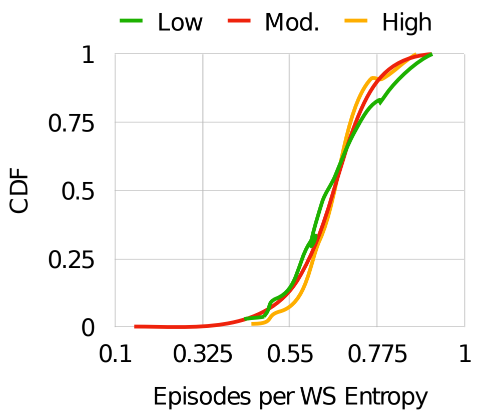


Figure 5.8: Episodes consumption entropy across 1 year history

$$ECE = \frac{-\sum_{w \in W} p_e \times log(p_e)}{log(N)} \tag{5.6}$$

where
$$p_e = \frac{\text{No. of WSs when episodes watched was } e}{\text{Total number of WSs}} \tag{5.7}$$

and N is the total number of possible episodes that the user can watch in a WS (N= 9 as the maximum number of episodes watched by a user during a single WS). The CDF of the entropy is shown Fig. 5.8; here we find the entropy is very similar across users with different activity levels. We find that 75% of all the users in our dataset have a ECE of more than 0.5 which indicates that the users do not have a regular pattern in terms of the number of episodes consumed during each WS; we observe that users have a large variance in the number of episodes they watch in consecutive WSs.

This insight leads us to investigate the “burstiness” of the amount of content consumed during WSs; this parameter is computed as in Goh and Barabasi [99]. The Burstiness parameter is defined in equation 7.8 as,

$$B = \frac{\sigma_t - m_t}{\sigma_t + m_t} \tag{5.8}$$

where σ_t is the standard deviation and m_t is the mean of the user's episodes per WS, over a period of t days. The parameter is a value between -1 and 1, where a value closer to 1 means that the standard deviation is larger than the mean, implying that the user's behavior is bursty with regard to the number of episodes they consume in consecutive WSs. A

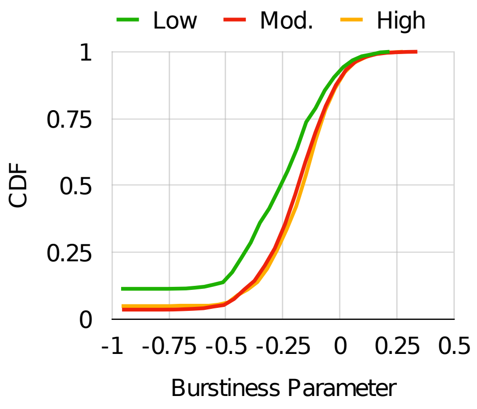


Figure 5.9: CDF of burstiness on a per month basis

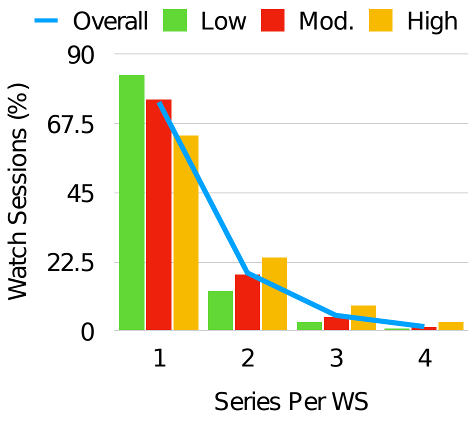


Figure 5.10: Distribution of No. of series watcher per day

value closer to -1 indicates the user watches almost the same number of episodes each WS. As an example, if user A watches the following number of episodes from Monday to Friday: [M=2, Tu=3, W=2, Th=2, F=3], then the burstiness parameter is -0.6; whereas if user B watches episodes as follows: [M=0, Tu=5, W=0, Th=10, F=0], then the burstiness parameter is 0.2. Fig. 7.39 shows the average monthly burstiness parameter (t= 30) for the entire viewing history for the users. We find that the more active the user is, the more bursty their behavior is i.e. there is more variance in the number of episodes consumed per WS for active users.

RQ5: *How many series does a user watch in a singe day?*

It can be argued that the number of episodes a user watches from a particular series will vary depending on what else the user is watching at that time. We explore this by determining the number of different series the user watches episodes from in a single sitting. Fig. 5.10 shows the distribution of the number of series watched across all WSs of the users. We find that, most of the time (nearly 68% of WSs), a user watches episodes from a single series in one sitting. In general, we see that less active users have a more concentrated viewing experience in that they only watch episodes from a single series in nearly 88% of their WSs, whereas, for highly active users, they only watch content from a single series for 65% of their WSs, and 23% of the time, they watch episodes from 2 series during the same sitting.

5.3.3 User Preferences

RQ6: *Do active users watch the same genre(s) regularly?*

This is an important question for recommendation engines and proactive caching systems, where a prediction of what to cache is made based on the user's preferences. Understanding users' preferences would also be useful for targeted advertising. There are 27 genres of Netflix series that are watched by the users in our dataset, and a series can be assigned multiple genres. A distribution of the episodes watched by all the users in our dataset, and the genres of the associated series, is shown in Fig. 5.11. We have shown the % of episodes watched belonging to the top 12 genres that make up 98% of all series' genres consumed; the remaining 15 genres are included in "other". As seen in the figure, the largest % of episodes watched (nearly 27%) are of the "drama" genre; this is the largest for all levels of user activity. Furthermore, we see that regardless of user activity level, the distribution of episode genres is very similar.

This, however, does not tell us if users in different activity levels have a concentrated preference in terms of the genre of content (i.e. they tend to watch content only from 1 or 2 genres) or a more diverse genre preference (i.e. they watch content from multiple genres). To quantify this, we compute the user's *viewing genre entropy* (VGE) as given in Eq. 7.9.

$$VGE = \frac{-\sum_{g \in G} p_g \times log(p_g)}{log(N)} \tag{5.9}$$

where

$$p_g = \frac{\text{Number of episodes in genre } g}{\text{Total number of episodes watched by user}} \tag{5.10}$$

and N is the total number of genres (N= 27). The VGE is a number between 0 and 1; a value closer to 0 means that the user has more stability in terms of their preference (they prefer content from a few genres only), whereas a larger VGE means that the user watches content from various genres. We computed the VGE for each month of the user's viewing history, and obtained the average across all the months; the results are shown in Fig. 5.12. We find that the more active the user is, the higher the VGE and thus, the more diverse the

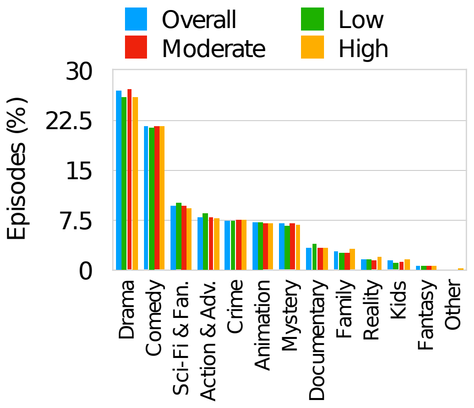


Figure 5.11: Episodes distribution per genre

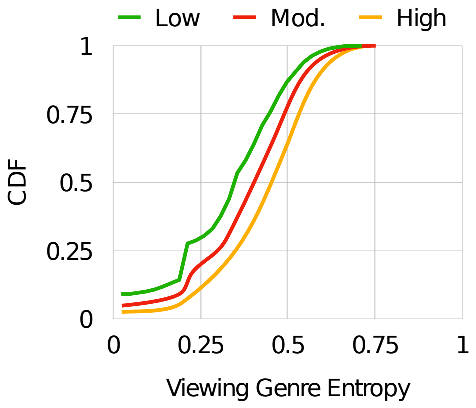


Figure 5.12: Monthly genre entropy for all users

preferred genres (75% of the users in low, moderate and high activity levels have a VGE of less than 0.45, 0.49, and 0.54 respectively).

RQ7: *Do active users prefer more popular and higher rated content?*

Gaining insight into how the popularity and ratings of content affect the consumption for different activity levels, is helpful for caching and content delivery. For each series watched by users in our dataset, we obtained the number of IMDB votes the series had at the time of retrieval. IMDB is an extensive online database of information related to movies, TV series and streaming content- including rating and reviews that are given by registered IMDB users. A rating that a series has received by IMDB registered users is counted as a vote; thus the number of votes a series received can serve as a indication of how popular that series is. Fig. 5.13 shows the distribution of the votes that users' watched series have; we find that 33% of series that highly active users watch has between 20,000 and 30,000 votes, whereas 29% of low active users' series fall in this range. We see that for votes higher than 30,000; users in low active categories watch the largest % of series (39%) as compared to moderately and highly active users (32% for both). The average number of votes for series watched by users in low, moderate and high categories are 30786, 29985, 28734 respectively. This implies that less active users tend to watch slightly more popular content than more active users.

Fig. 5.14 shows the distribution of the ratings (a score out of 10 given by registered

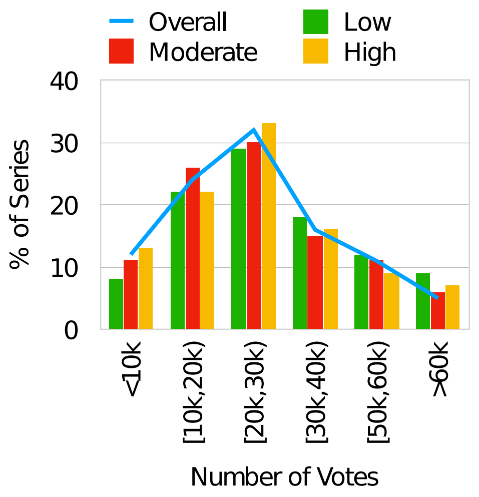


Figure 5.13: Distribution of number of votes received

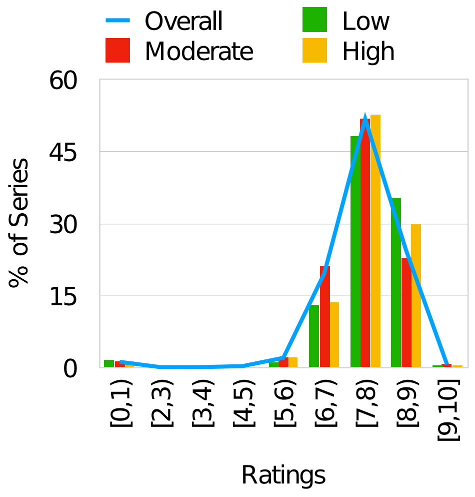


Figure 5.14: Distribution of ratings received

IMDB users) of series watched by users. Here we see that less active users prefer content with higher ratings than more active users; 35% of series watched by low active users have a rating of above 8, whereas 30% of highly active user have a rating of above 8. In conclusion, we find that less active users, even though watch less content, prefer more popular and higher rated content than more active users.

RQ8: *How much of user's watched series are related to series they have seen in the past?*

Recommendation engines predominantly recommend content that is related to what the user has watched in the past. Although we are unable to retrieve the content that is recommended to the user when they are watching content on Netflix, we obtain an approximation of the effectiveness of the engine by computing the fraction of series watches that are related to series that the user has previously seen. This analysis can further aid in the prediction of what content the user will watch. During the meta-data retrieval process, we obtained the 12 related series as listed by IMDB; using this information, for every series that a user has watched, we see if this series is related to any series watched previous to this. We find that approximately 42% of a series watched by a user, was related to a previously watched series. Furthermore, we find that this percentage is similar for users across activity levels; with low, moderate and high active users, watching 41.4%, 42.3% and 40.1% of series that was related to a series they had seen before.

5.3.4 Predictability

RQ9: *How much of user's future watches are predictable?*
Predicting what, and how much, a user will watch next, is crucial for prefetching and caching strategies. These strategies anticipate the content a user is likely to consume, downloads the content ahead of time, and makes the content available at the time of consumption. To see whether we can predict what the user will watch next based on what they have consumed in the past WSs, we do the following: for every WS that appeared in a user's viewing history, and for each episode watched in that session, we check if its preceding episode was watched within a certain number of previous WSs. For example, if episode 20 of series A was watched today, we check if and how many WSs prior, episode 19 was watched. We compute this for all users in our dataset across their entire history, the average is shown in Fig. 7.62 for various WS intervals.

For the average user in our dataset, we see that nearly 58% of episodes proceed an episode that was watched in the previous WS (1 WS), a further 13% of episodes proceeded an episode that was watched between the previous 2 and 10 WSs, 1% between 11 and 20 WSs ago, 1% between 21 and 30 WSs prior, and 3% was watched more than 30 WSs prior. In general, we find that approximately 77% of the user's episode watches follows an episode that the user has seen in the past. We also find that as the activity level of the user increases, the larger the predictable % of episodes. Thus, we conclude that nearly 77% of a user's future episode watches can be predicted as it proceeds a previously watched episode from the series.

RQ10: *Are certain genres easier to predict for than others?*
Fig. 5.16 shows the % of episodes that is predictable for different genres. We consider all WSs in the user's viewing activity for this analysis. We find that nearly 85% of episodes from "Fantasy" series follows a previously watched episode; this is the highest for any

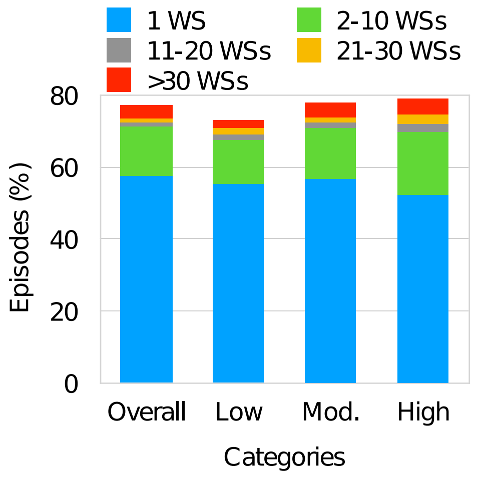


Figure 5.15: Predictability from past WSs

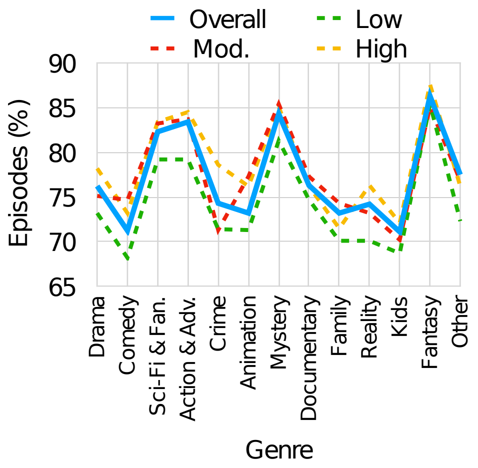


Figure 5.16: Predictability across different genres

genre. We find that the "comedy" genre and "kids" genre has the least % of episodes that are predictable (71.4% and 71.1% respectively), this is the same for low and high active users as well. We speculate that these differences arise due to the episodic (such as for "Fantasy" series) vs non-episodic (such as "kid" shows) nature of series. This insight can further aid prediction and prefetching systems to determine if and how many episodes from a particular series, the user will watch in the near-future.

5.3.5 Continuity of User Watch-Behavior

RQ11: *How many seasons does a user watch to its entirety?*

An effective way of gauging a user's interest and engagement in a particular series, which will be helpful for content creators, marketing agencies and content providers, is to see if they watch a series season to its entirety. Fig. 5.17 shows that % of seasons users watch to its completion across various series genres. Overall, nearly 55% of series seasons are watched entirely, with series seasons in the "Animation" genre watched to its entirety the most as compared to other genres (60%). We find this to be similar across low and high active users.

RQ12: *At what point does a user stop watching a series season if they don't complete it?*

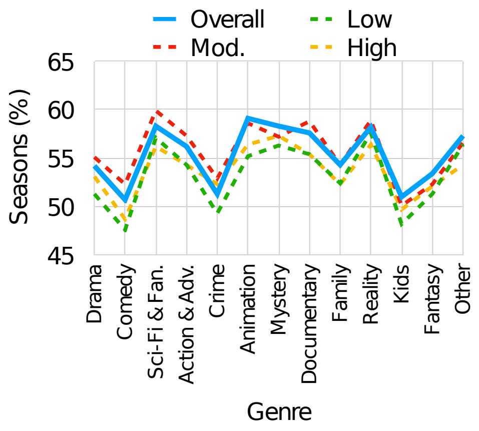


Figure 5.17: Seasons watched to its entirety for different genres

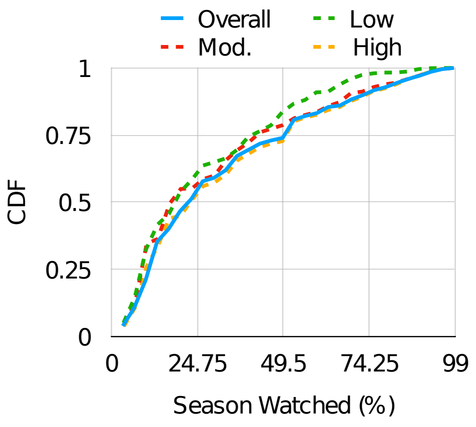


Figure 5.18: Point of departure of seasons not watched to its entirety

Interestingly, we found that a large percentage of seasons, nearly 45%, are abandoned at some point, and not watched to completion. For the series seasons that are not watched to its entirety, we explore the point at which a user stops watching a season (we only consider seasons of episodes that are watched contiguously). Fig. 5.18 shows the CDF of how much a season a user has watched before abandoning it- we term this as the "point of departure". We see that 50% of seasons are abandoned when less than 25% of the season is watched; this is consistent across users of all activity levels. The remaining 50% of the seasons has a point of departure from 25% to 99%, and this is nearly uniformly distributed.

RQ13: *Can we predict when a user will abandon a series?*

Given that nearly 45% of series seasons are not watched to completion, this leads us to investigate if we can predict the time at which the user will stop watching a series- this could be due to a variety of reasons, but particularly a waning interest in continuing the season. To this end, we employ 4 popular machine learning classification models to answer the following question: *For the latest episode of a series watched in a particular WS, will the user watch proceeding episodes in subsequent WSs?* The models we employ are as follows: Binary Logistic Regression (LR), Support Vector Machine (SVM), Naive Bayes model (NB) and Random Forest (RF). The models use the following features for prediction: % season watched, number of votes the season's series has, the IMDB rating the season's

Table 5.2: Classifier Comparison

Method	Accuracy	Precision	Recall	AUC
LR	66.2	0.57	0.69	0.61
SVM	59.2	0.53	0.59	0.62
NB	65.1	0.56	0.67	0.63
RR	68.1	0.61	0.73	0.69

series has, series genre, episode runtime, year of release and number of seasons. In essence, for every series watched in a particular user's WS, we obtain the latest episode watched from that series, extract the appropriate features of the episode's series, feed this into the trained classification model and obtain one of two possible outputs: 1) "continue"- the model predicts that the user will continue watching the seasons, 2) "abandon"- the model predicts that the user will stop watching the series.

Table 5.2 shows the results of the classification model using the following performance metrics: the accuracy, the precision, the recall, the and the AUC value. The descriptions of the classification metrics can be found in [100]. We train the models on the first 9 months of the user's data, and perform the testing on the remaining 3 months. To ensure a balanced dataset i.e. approximately the same number of "abandon" instances as there are "continue" instances, we perform under-sampling of the "continue" class during training. We find that we are able to achieve the highest prediction accuracy with the RF model- we are able to correctly predict 68% of the instances of when the user either abandons or continues watching a seasons. In general, we find that the classifiers perform similarly in terms of their classification.

CHAPTER 6

TOWARD EFFECTIVE PREDICTION OF WATCH BEHAVIOR FOR TIME-SHIFTED EDGE-CACHING OF NETFLIX SERIES VIDEOS

Video streaming services dominate global Internet traffic because of the tremendous rise in the number of cord-cutters, which has grown by 48% in the last 8 years and is predicted to rise to 55 million by 2022 in the U.S. alone. As a result of the increasing growth and popularity of video streaming services, the network is heavily burdened. To cope with this, Internet Service Providers either have to spend several millions of dollars for infrastructure upgrades, or employ congestion-reducing mechanisms like bandwidth throttling and data caps, which negatively impacts the overall user experience [101]. Typically, upgrades are triggered when there is a reasonably sustained peak usage that exceeds 80% of capacity [3].

Edge-caching is often used to overcome this problem by storing content nearer to the clients, thereby eliminating redundant traffic flows and, reducing overall traffic consumption and latency. Typically, recently accessed content (traditional web caching [102]), or popular content accessed by users over a geographical region (content delivery networking [103]) is cached. In this chapter, we explore the strategy of time-shifted prefetching, or caching during off-peak periods of the network even when such periods are substantially separated from the actual usage-time. Prefetching, or proactive caching, is not a new strategy, and has been extensively considered in prior related work [52]. The uniqueness 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.

Furthermore, with the increasing storage space available in end-user's viewing devices along with the emergence of smart WiFi access points (APs) [59], we consider caching content at the edge closest to the client; at the end-user's viewing device (set-top box, computer) or a storage server attached to an AP (such as a WiFi router). This architecture has significant benefits for both content providers as well as users. It allows for requests to be served locally which in addition to reducing latency and improving user's Quality of Experience (QoE), alleviates content server traffic and cross-traffic among ISPs. However, these devices have stringent storage and bandwidth constraints, and as a result, the selection of what and how much content to cache is extremely important.

With Netflix being the most popular video streaming service (it has approximately 195 million global subscribers and accounts for the largest share of global application traffic [39]) and also consuming the largest portion of global network traffic share, we restrict our focus of edge caching to *Netflix* content. Given the dominance that Netflix traffic has on global Internet traffic and the correspondence of peak-time traffic with prime time for television viewing, accurate and efficient time-shifted caching can have a meaningful impact in tackling the problem of network traffic imbalance. With 96% of the average user's Netflix titles being episodes from series, we restrict the scope of our study to Netflix series and documentaries which together account for 65% of a typical user's Netflix load in terms of bytes fetched. Thus, the key question we answer in this chapter is the following: *For a given Netflix user, can Netflix series videos be prefetched and edge-cached during off-peak periods so that the actual cost of fetching videos during the peak periods is reduced?* The key contributions made in the chapter are summarized as:

- We collect a dataset of Netflix viewing history from 1060 users. The dataset is comprised of $2,465,276$ Netflix titles over a 1-year period- $2,132,980$ of which are TV shows and documentary episodes, while the remaining are movies.

- Using the dataset, we divide the users in 4 categories depending on how active the user is. We then perform an extensive analysis on users' content preferences, their request patterns and series continuation tendencies. We also explore the extent to which a user's Netflix usage can be predicted, and we show that a significant percentage of their series watching behavior (77%) is predictable by relying on the past viewing history of that user.

- We present results from a naive caching solution which caches proceeding episodes, regardless of the user's past viewing behavior, and we show that we achieve a cache efficiency of 6% with this method- this means that 94% of content that is cached, is not consumed by the user in the future. Given the limited edge-caching resources, this is an unacceptable cache efficiency rate. In effort to improve this efficiency, we present 2 baseline heuristics that is dependent on the user's continuation tendencies.

- Finally, we design and implement a deep learning caching algorithm, *CacheFlix*, that uses global and local learners based on Long Short-Term Memory (LSTM) networks, to cache episodes of Netflix series to the user during off-peak hours. Based on the user's past viewing patterns and preferences, *CacheFlix* predicts how many episodes from previously watched Netflix series to prefetch to the storage constrained edge-cache for future viewing. The algorithm is evaluated in terms of how accurately it is able to predict content that the user watches in the future (prediction accuracy) and how efficiently it consumes bandwidth and stores the content (caching efficiency). We present results for 3 different cache eviction policies, and also for edge-caches with storage sizes as small as 2 GB. We also compare our results to related work, and show that *CacheFlix* is able to perform 1.8 times better in terms of accuracy, and 3.5 times better in terms of efficiency.

6.1 Background & Motivation

6.1.1 Peak vs. Off-peak Load

Traffic load on networks is significantly higher during peak periods. To illustrate this, we performed a bandwidth (BW) probe on an Apple MacBook Pro, and measured the available BW over a university campus WiFi network as well as over a home WiFi at different times during the day. The probe downloads a small file from a nearby Comcast web server to the mobile device, and uses the download time to estimate the throughput; the same server is used for both the home and campus networks. This test was conducted every 30 minutes for

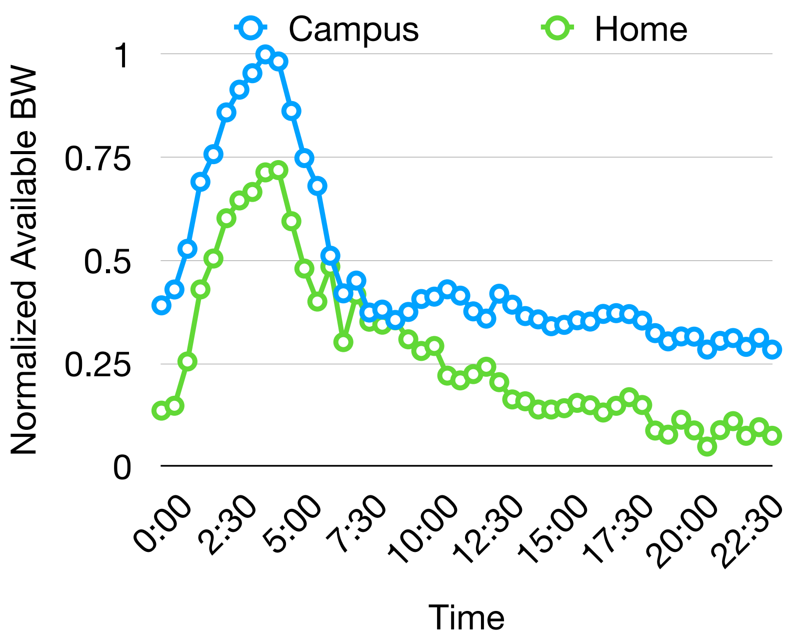


Figure 6.1: Normalized available BW across the day

10 consecutive days, while the device was connected to the WiFi network; Fig. 6.1 shows the average of the normalized (with respect to the maximum value) measurements across 10 days. We observe a sharp increase in the available BW between 12:30 AM to 4 AM, and a subsequent decrease till 8 AM. 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 Internet traffic distribution studies [104]-[106]. Using Fig. 6.1 as a reference, the *off-peak period* is defined as 2 AM to 6 AM, and the *peak period* is defined as 6 AM to 12 AM, and 12 AM to 2 AM. There is thus potential to utilize the available BW during off-peak periods for prefetching content.

6.1.2 On Netflix usage

Netflix is the most popular streaming entertainment service with over 195 million paid memberships in over 190 countries. The number of subscribers is currently growing at an unprecedented rate, with Netflix adding 15.8 million new subscribers worldwide during the first three months of 2020, more than doubling its growth forecast for the quarter [107]. It consumes 12.6% of global network traffic share- the largest for any single application [39].

Furthermore, it is estimated that Netflix users collectively stream 165 million hours using nearly 500 million GB of data per day [108]. It was recently reported by Netflix, that an average users watches 2 hours of Netflix videos per day [109]. This equates to approximately 60 GB of data usage per month if content is watched on standard definition, otherwise 180 GB is watched in high definition [108]. This corresponds to the average viewing time computed for the users in our collected dataset; the average user in our dataset watches 1.8 hours of content per day. It was further reported, by Netflix, that users mainly watch Netflix on TV and multiple devices rather than on mobile devices only [109]. Given the importance of Netflix content on global Internet traffic and the correspondence of peak-time traffic with prime time for television viewing (8pm-11pm), accurate and efficient prefetching has potential in tackling the problem of network traffic imbalance.

6.1.3 Problem Definition

The problem we address in this chapter is shifting peak-time traffic to off-peak periods by caching *Netflix* series content at the edge closest to the user. The edge device can be the user's viewing devices themselves or smart WiFi routers that have limited storage space. The key question we answer is: *For a given Netflix user, can Netflix series videos be effectively prefetched and cached to storage-constrained edge devices during off-peak periods, so that the actual cost of fetching videos during the peak periods is reduced?* We use the following metrics, which are later defined in section 6.3.3 to evaluate our proposed solution: (i) *Prediction Accuracy* (PA)- the fraction of watched Netflix series episodes that have been cached, among the total number of watched episodes; (ii) *Caching Efficiency* (CE)- the fraction of watched Netflix episodes among the cached Netflix episodes. The goal of our proposed solution is to decrease the peak-period data traffic consumed by the end-user by ensuring the caching solution has both a high PA, while having an acceptable CE given the storage and bandwidth constraints of the edge-caches.

6.2 A Real-World Dataset

6.2.1 Dataset Collection

In order to study user behavior and the feasibility of caching Netflix content, we rely on a dataset collected from 1060 users spanning a total of 1 year. We utilize Amazon Mechanical Turk (mTurk) to gather anonymized Netflix viewing history from users over the required time period [78]. 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 [79]. The task we posted required mTurkers to navigate to their *viewing activity* page associated with their profile, and download their viewing activity; the file was then anonymously uploaded[1]. Netflix allows a user to download their past viewing activity as a CSV file which contains 2 fields: *title* and *date*. The *title* field consists of the name of the feature film or TV series/documentary, as well as the season and episode name where applicable, separated by colons. The *date* field consists of the most recent date that the title was viewed; there is no associated time of viewing.

6.2.2 Metadata Retrieval

A Postgres SQL database is used to store the data for the mTurkers. The SQL database consists of 3 tables, namely, *tblUsers*, *tblSeasons*, and *tblTitles*. The *tblUsers* table is used to store the title, watch-date as well as the season number and episode number if an episode from a series is watched. These values are populated from the user's submitted viewing activity file, and through appropriate API calls from The Movie Database (TMDB) API [97]. The *tblSeasons* contains the season number and total number of episodes in each season for every series watched by the users. The *tblTitles* table contains a number of attributes related to the series and movies watched by all the users; the TMDB API is used to obtain this metadata. The attributes obtained are: the release date of the title, the IMDB

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

Table 6.1: Dataset Overview

Description	Value
No. of Users	1060
No. of Movies	332,296
No. of Episodes	2,132,980
No. of Hours (movies)	631,351
No. of Hours (episodes)	1,172,510
No. of Seasons	121,101
No. of Series	30,224

rating, the number of IMDB votes for the title, the run-time in minutes, the genre(s) (there are 29 genres that Netflix uses to classify its content, and a title can have multiple genres), director(s), writers, actors, the language of the title, country of production and related titles (as determined through TMDB). There is also a field which is used to indicate if the title is a movie or a series.

6.2.3 Data Insights

Overview

A high-level overview of the collected dataset is presented in table 6.1. We show the total number of movies and episodes watched by all the users in the dataset, the number of hours of viewing time for TV series episodes, the number of hours of viewing time for movies, as well as the total number of seasons and series watched by all our users. In the proceeding sections, we will group users in 4 categories based on their level of viewing activity, and present results related to what type of content they view and when they consume it.

Activity Levels

We divide the users in our dataset into 4 levels from activity level 1, which are the users that watch the least amount of Netflix content, to activity level 4, which is the subset of users that watch the most content. The users are categorized based on the total number of episodes that the user consumes over their 1 year history submitted. The distribution of the number of episodes watched by users during the year and the percentage of users belonging

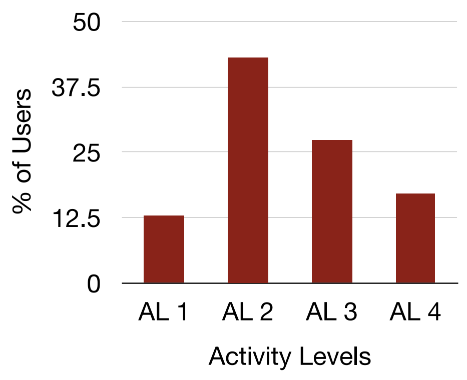


Figure 6.2: User activity levels distribution

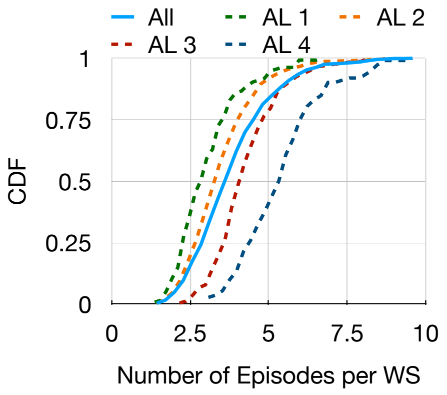


Figure 6.3: CDF for the number of episodes watched per WS

to each category is given in Fig. 6.2. The users in activity level 1 (AL 1) have watched less than 100 episodes, users in activity level 2 (AL 2) have watched between 101 and 400 episodes, users in activity level 3 (AL 3) have watched between 401 and 800 episodes and finally, the most active users in activity level 4 (AL 4), have watched more than 800 episodes. In the following sections, we present results for the entire user dataset as well as for the different activity levels as defined here.

User request pattern

Here we present insights, through questions, regarding the request patterns for users in our dataset.

How much content does a user consume each day? We define a watch-session (WS) as a day on which at least one episode of some series is watched by the user. Fig 6.3 shows the CDF for the number of episodes watched per WS for all the users in the dataset together, as well as for different activity levels. On average, a user watches 2.7 episodes per WS (approximately 1.5 hours), with the number of episodes watched per WS increasing with increasing activity level.

How often do users watch content? We explore the number of days between consecutive WSs, termed as the *time between watch sessions* (TBWS); the CDF is shown in Fig. 6.4. For 80% of the users, the TBWS is approximately 6 days- i.e. a user watched Netflix content every 6 days. We see that for 80% of the users in activity levels 3 and 4, the TBWS

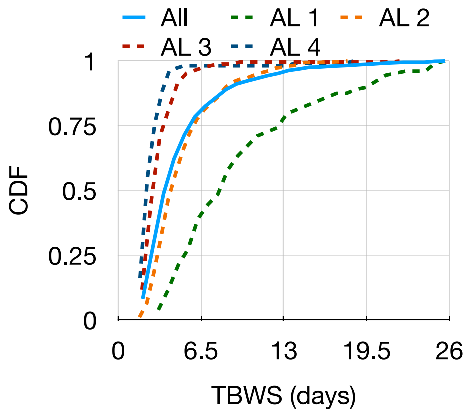


Figure 6.4: CDF of TBWS between WSs

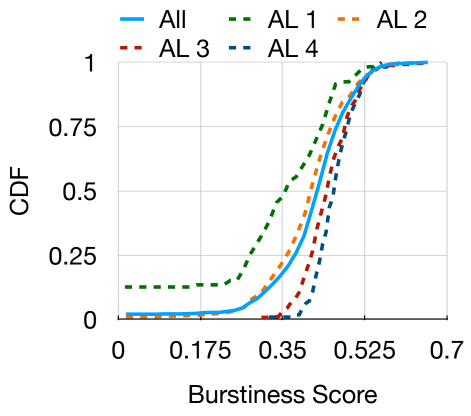


Figure 6.5: CDF of per-month burstiness score

is approximately 2 days, and for users in activity level 1, it is 14 days.

How consistent is the amount of content the user consumes each day? The observation that a user drastically increases or decreases the number of episodes they watch in 1 WS as compared to previous WSs, leads us to investigate the burstiness, in terms of the number of episodes watched in a single WS, of a user. In order to quantitatively capture this burstiness, we compute a burstiness parameter as in Goh and Barabasi [99]. The Burstiness parameter is defined as in equation 7.8,

$$B = \frac{\sigma_t - m_t}{\sigma_t + m_t} \tag{6.1}$$

where σ_t is the standard deviation and m_t is the mean of the user's number of episodes they consumed in each WS. The parameter ranges from -1 and 1, and is then scaled to a value between 0 and 1, where a value closer to 1 indicates that the standard deviation is larger than the mean, implying that the user's behavior is bursty with regard to how many episodes they consume in each WS. Fig. 7.39 shows the CDF of the average per-month burstiness parameter computed across 1 year of the user's viewing history; we find that the more active the user is, the more bursty their behavior is i.e. there is more variance in the number of episodes consumed across WSs for more active users.

How consistent is the time that content is consumed by the user? For our entire dataset, in Fig. 6.6, we plot the % of episodes viewed on each day of the week; the most amount

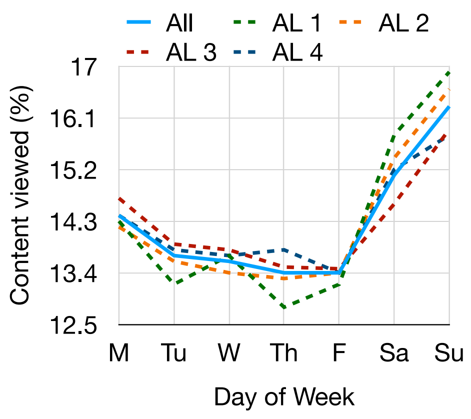


Figure 6.6: Distribution of content watched per day

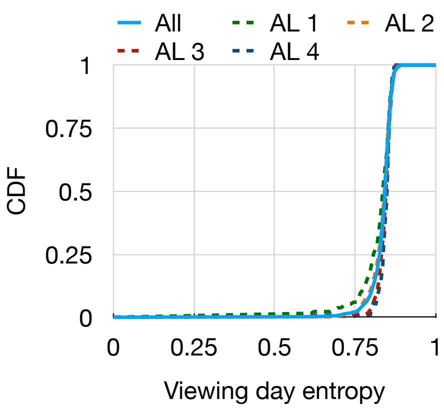


Figure 6.7: VDE across users for their 1-year history

of content is viewed on Sundays, and the least being on Friday. There is no significant differences across the activity levels. To quantify if a user tends to watch Netflix content on the same day each week, we define a Viewing Day Entropy (VDE) metric as follows [110]:

$$VDE = \frac{-\sum_{d \in D} p_d \times log(p_d)}{log(N)} \tag{6.2}$$

where

$$p_d = \frac{\text{Number of episodes watched on day } d}{\text{Total number of episodes watched by user}} \tag{6.3}$$

and N is the total number of days in a week (N= 7). The VDE is a value between 0 and 1, where a smaller VDE indicates that the user mostly watches content on the same day of the week, and that this user may have a regular request pattern (note that VDE=0 when p_d tends to 0). A larger value, on the other hand, indicates request days more uniformly distributed across the week. The CDF of the VDE is shown in Fig. 6.7; there is a slight difference between the curves across activity levels. We find that active users tend to watch content on the same day(s) each week as compared to less active users.

Preferences

We present insights related to the users viewing preferences for genres and popularity of series.

Do users watch the same genre of content regularly? To quantify whether the users prefer a certain genre of series than others (recall, there are 29 different genres and a series can have multiple genres), we define Viewing Genre Entropy (VGE) as follows:

$$VGE = \frac{-\sum_{g \in G} p_g \times log(p_g)}{log(N)} \tag{6.4}$$

where

$$p_g = \frac{\text{Number of episodes in genre } g}{\text{Total number of episodes watched by user}} \tag{6.5}$$

and N is the total number of genres (N= 29). The larger the value of the VGE, the more genres the user regularly consumes (when $p_g = 0$, then VGE is 0). As shown in Fig. 6.8, the CDF of the VGE across the user's entire viewing history shows that 50% of the users have a VGE of greater than 0.7, meaning that users tend to consume content of various genres. We also find that there is not much difference between active and less active users- all users, regardless of the level of activeness, tend to consume content of a variety of genres.

Do users tend to watch more popular content? During the metadata retrieval process, we also obtain the number of ratings, termed as votes, that a series receives on IMDB. IMDB registered users can rate series on a score from 1 to 10, thus the number of votes is the number of users that have rated a particular series. The number of votes can serve as an indication of how popular a series is, where the more number of votes a series has, the more popular it is. In our dataset, the series with the highest number of votes has 836,117 votes at the time of metadata retrieval, and the series with the lowest number of votes has 65 votes. For each user's viewing history, we compute the average number of votes that each watched episode's series obtained; we found that the mean number of votes for activity levels 1 to 4 are 30786, 30398, 30121 and 29212 respectively. This indicates that less active users tend to prefer content that is slightly more popular than active users.

Do users tend to watch content with higher ratings? The aggregate of the ratings that IMDB registered users give a particular series is also obtained during the metadata retrieval process. In our dataset containing all series that users have watched, the highest rating is

10, and the lowest rating is 0.02. As in the case of number of votes, for every user's viewing history, we compute the average rating that each watched episode's series obtained; for the entire dataset, the average rating is 5.81. For activity levels 1 to 4, the average ratings is 6.16, 5.85, 5.72, 5.64 respectively. This shows that less active users, although watch less content, prefer higher rated content than active users.

Season Completion

We investigate the season completion percentage of series that users watch. We attempt to answer the following questions: (i) *How often do users watch a particular series to completion?* (ii) *At what point do users stop watching a season of a series if they do not complete it?* To do this we compute the number of contiguous episodes a user watches from the beginning of a season (subsequent watches of episodes from a season can be interleaved with movie watches or episode watches from other series), and compute the percentage of the season that the user watched during the course of their 1 year viewing activity. We find that across our entire dataset, approximately 55% of series seasons are watched in its entirety. This value is similar across different activity levels, with users in activity levels 1 to 4, completing 53.2%, 55.1%, 54.3% and 53.7% of TV series seasons.

For the remaining seasons that are not completed to its entirety (approximately 45%), we compute the "point of departure"- that is, the % of the season watched until which the user abandons the season, and does not watch any more episodes. The CDF across all our users for their entire viewing history is shown in Fig. 6.9. We find that 75% of seasons are abandoned when less than 50% of the season was watched by the user, and this is fairly consistent across different activity levels.

6.3 Baseline Solutions

6.3.1 Predictability of Netflix Content

Being able to predict what a user will watch in the future is particularly useful for prefetching strategies. These strategies anticipate the content a user is likely to consume, download

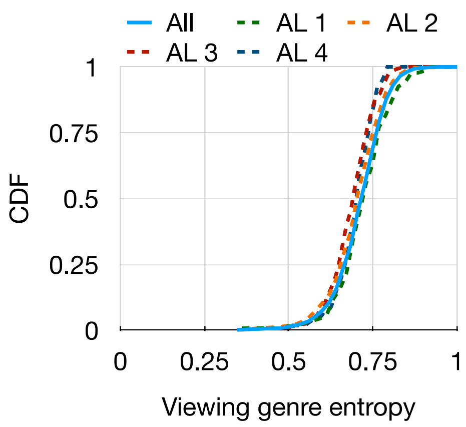


Figure 6.8: VGE across user activity levels for 1-year history

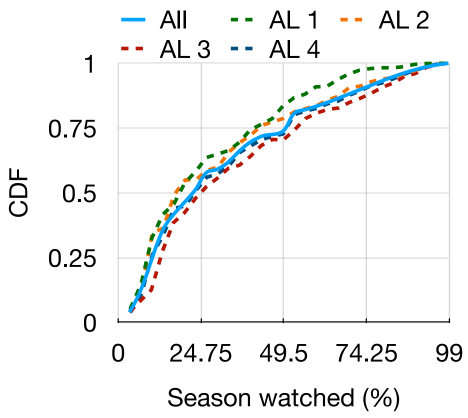


Figure 6.9: CDF of point of departure for seasons not watched in its entirety

the content ahead of time, and make the content available at the time of consumption. To explore the feasibility of prefetching, we consider how a user's Netflix watch behavior is influenced by content they have seen in the past. We restrict the scope of our study to Netflix series and documentaries that together account for 65% of a typical user's Netflix load in terms of bytes fetched (computed from the collected dataset across all users where the runtime was used as a proxy for the data consumed) and 96% of the Netflix titles watched by a user (a Netflix title is either an episode or a movie). For every WS that appeared in a user's viewing activity, and for each TV show/documentary watched in that session, we compute the fraction of episodes (across following watch-sessions) which proceed the last viewed episode in the series. We see that nearly 77% of all episodes that a user watches in the future, follows an episode that the user has seen in previous WSs. Thus, we conclude that nearly 77% of a user's future episode watches can be predicted as it proceeds a previously watched episode from the series. These results serve as an upper bound for the accuracy the caching algorithm can achieve, if it were to cache all proceeding episodes of TV series and documentaries that the user has seen.

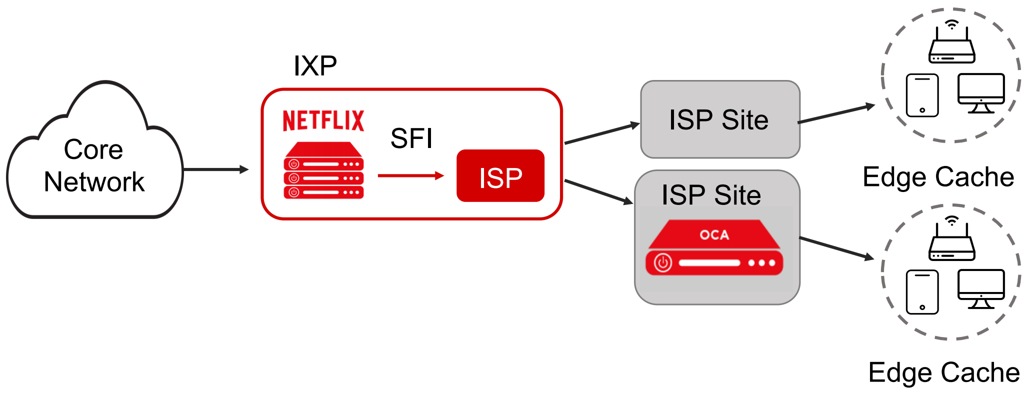


Figure 6.10: Logical Architecture

6.3.2 Logical Architecture

Netflix deploys its own content delivery infrastructure, named *Open Connect* [57], which are used to exclusively deliver their video content. Their global network of thousands of Open Connect servers, called Open Connect Appliances (OCA) are deployed in 2 ways: (i) the OCAs are installed within internet exchange points (IXP) which are interconnected with mutually-present ISPs via settlement-free public or private peering (SFI); (ii) the OCAs are deployed directly inside ISP networks [111].

The logical architecture that we consider for the edge-caching of Netflix videos is shown in Fig. 6.10. We cache contents to the node closest to the user, beyond the OCAs. This can be on the devices themselves or a server attached to an access point (e.g. a WiFi AP), a network gateway, or even a micro-datacenter available for use by nearby devices. The prefetching algorithm will reside in these edge cache nodes as well. This allows the caching to be tailored for specific users, and negates latency during time of consumption. With storage being a pertinent constraint in our edge caches, as compared to traditional edge caches (i.e. CDN servers), the proposed prediction algorithm needs to be highly efficient so that it does not unnecessarily prefetch content that the user does not end up watching.

6.3.3 Naive Caching Strategy

Here we present a naive caching solution which blindly caches proceeding episodes during off-peak periods based on episodes that the user has previously seen. Specifically, we consider the episodes that the user has watched in the watch-session prior to the prefetch time (we arbitrarily select 4 AM as the prefetch time during off-peak hours); we then cache proceeding episodes accordingly. We begin by first defining the following metrics which will be used to evaluate the prefetching solutions:

- *Prediction Accuracy (PA)*= the ratio of watched episodes present in the cache to the number of *predictable*[2] episodes watched by the user on a particular date.
- *Cache Efficiency (CE)*= the ratio of watched episodes present in the cache to the number of episodes in the cache.

The naive caching algorithm can be summarized as follows: for each day that a user watches episodes from some series, we cache *N* proceeding episodes and store it in the cache for *M* WSs (if the number of episodes left in the series is less than *N*, we only cache the remaining proceeding episodes). The PA and CE is computed for the WS proceeding the prefetch time. The results, across all users, is shown in fig. 6.11 and fig. 6.12 for *N* from 1 to 10, and *M* from 1 to 4.

We find that as we increase the number of WSs we store the content for (*M*), the accuracy also increases with the difference between 1 stored WS and 2 stored WSs being the most significant. A maximum of 58.1% in the prediction accuracy is achieved when we store 10 proceeding episodes (*N*= 10) for 4 future watch sessions (*M*= 4). We see that there is an increase in the accuracy as *N* increases, but it tends to saturate once *N* surpasses 6. A PA of 58.1% corresponds to 44.6% of Netflix traffic associated with series (recall, that 77% of the traffic was found to be predictable). The CE however takes a hit when we cache more episodes and store them over a longer time period, with the CE being only 6% for the

[2]Predictable episodes are the subset of the user's watched episodes which follow episodes watched in previous watch-sessions

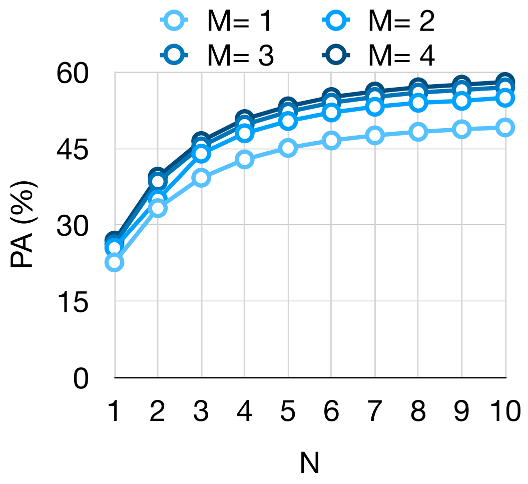


Figure 6.11: PA for varying values of *M* and *N*

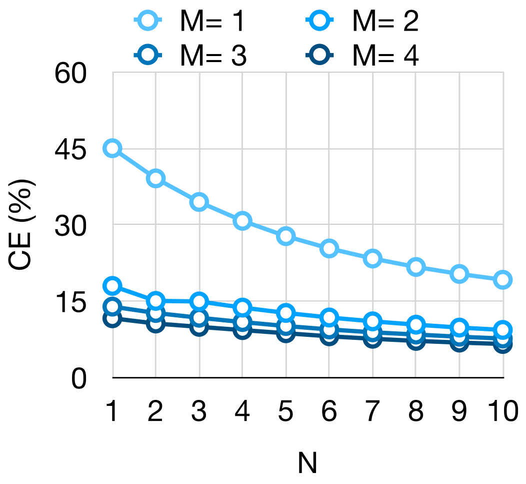


Figure 6.12: CE for varying values of *M* and *N*

highest PA.

With the naive caching strategy, 45% of a user's series traffic can be shifted to off-peak periods, however the associated CE is 6%.

6.3.4 Heuristic 1: User Continuation

Overview

From the naive caching solution, we find that while we can achieve an acceptable accuracy, the efficiency is extremely low at 6%, which means that 94% of content that is being fetched and stored, is not being consumed. This is unacceptable given the stringent storage and bandwidth constraints of the end-devices on which content is cached. In effort to improve the efficiency, we present a heuristic solution which caches episodes depending on the user's perceived interest in the series. A way of gauging the user's interest can be based on how much of a particular series the user has seen prior to the prefetching time. For this heuristic, we use the % of a season the user has watched prior to the prefetching time as a means of predicting how many more episodes the user will see in the future. We present 2 methods of prediction: (i) hash map scheme, and (ii) employing linear regression.

Hash map scheme

In this method, for each user, 75% of the the user's viewing activity is used to create $\langle \%seasonwatched, numberofepisodeswatchedthenextWS \rangle$ tuples ($\langle key, value \rangle$) for the highest episode of every season watched during a WS. These tuples will act as the *training set* to populate a hash table; for the remaining 25% of the user's viewing activity, we use the tuples to perform a matching where we use the % season watched as an input (or key), and find all tuples that have the % season watched within 5% of the input, and take the average of the corresponding values. We use this number to cache the next episodes and store it for 4 WSs. If there is no such tuple present in the hash table, we search for the nearest % season watched tuple.

The results of the PA and CE are shown in Fig. 6.13. We find we can achieve an CE of 12.8% (an increase of 6.8% from the naive caching scheme), but the PA decreases to 56.2%. This CE means that we are caching approximately 87% more content than the user actually consumes. In particular, we find that CE decreases the more active the user is. A reason for this is that the users in AL 1 on average watch between 1 to 2 episodes per WS, with a std dev of 0.13, which indicates that their amount of consumption is fairly regular, and so this heuristic tends to perform better than for more active users.

Linear regression prediction

Another way of implementing this heuristic is with the use of a machine learning model to predict the number of episodes to cache based on the % season watched by the user. For this, we implement a linear regression model (ordinary least squares regression), using Python's scikit learn library to model and predict the relationship between the season watched % and the number of episodes to cache. We fit the model on our training data tuples and predict the values for the number of episodes to cache for our testing data. We found that the efficiency increases to 18.7%, however the accuracy drops by around 6% to 50.1%. This indicates that it is simply not sufficient to estimate a user's interest as a way of determining how much content they will consume in the proceeding sessions. This leads us to the development of our second heuristic.

With a CE of 19%, it is not sufficient to prefetch episodes simply based on how many episodes the user previously watched.

6.3.5 Heuristic 2: Season Continuation

Overview

As was discussed in section 6.2.3, only 55% of seasons are watched to completion, and around 45% are abandoned at some point (on average, when 20% of the season is watched). It follows that if we can predict at what point a user will stop watching a season, then we need not prefetch episodes thereafter, thereby improving the efficiency.

Methodology

For this prediction, we employ a binary logistic regression model which will predict if a user will continue watching a season or abandon the season (this is a standard classifier that is often used for predicting categorical outcomes [112]). The question we attempt to answer is the following: *Given the % of a season watched by the user, what is the likelihood that the user will continue watching the season?* The logistic regression model will be used to predict if a user will continue watching the season based on what % of the season they have already watched; it will also use content specific features for training and testing. The model uses the following features: % season watched, number of IMDB votes, IMDB rating, genre, runtime, year of release and number of seasons.

Prediction Results

With 75% of the user's dataset used for training, and 25% for testing, the logistic regression model is able to accurately predict 66.2% of the instances of when a user abandons a season. We overlay this model with the linear regression model described in heuristic 1 to improve the efficiency by first predicting whether a user will continue watching the season using the logistic regression model, and if so, then using the linear regression model, we predict

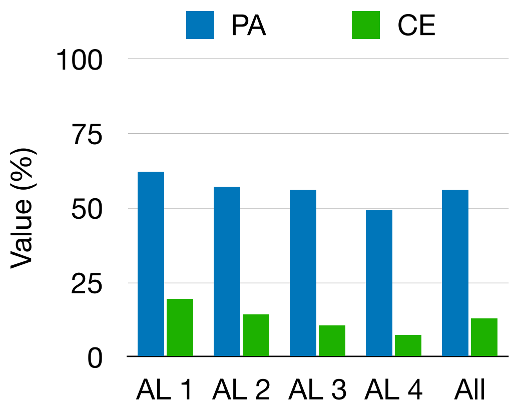


Figure 6.13: Heuristic 1 performance across varying activity levels

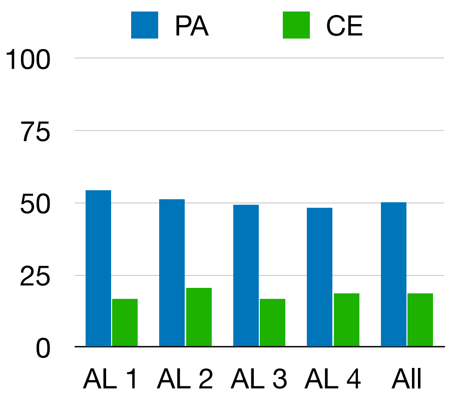


Figure 6.14: Heuristic 2 performance across varying activity levels

and cache the associated number of episodes in that instance. The results for the different activity levels are shown in Fig. 6.14. Here we see that that the overall efficiency increases to 22.6% (from 18.7%), and the accuracy decreases slightly to 48.6% (from 50.1%) due to the incorrect predictions of the logistic regression model (when it incorrectly predicts that the user will not continue watching the season, and thus no episodes are cached thereby decreasing the accuracy).

Even while being able to predict 66% of the instances when a user abandons a season of a series before completion, the efficiency only increases slightly to 22%.

6.4 *CacheFlix*: Edge-caching of Netflix series episodes

We have shown that the naive caching strategy can shift nearly 45% of peak-time Netflix traffic, however, the associated CE is only 6%. We also showed, through 2 heuristics which are based on the user's continuation of series, that while we are able to improve the efficiency slightly, approximately 80% of content that is cached is not watched. With storage being a major limiting constraint, there is a need for an intelligent algorithm that is able to accurately predict how much content a user will watch so that excess bandwidth and storage is not utilized for content that the user will not watch (given the average user's Netflix consumption, approximately 10GB of additional unnecessary content would need to be downloaded during every single off-peak period for the naive caching strategy, and 8

GB for heuristic 2). To this end, we propose the *CacheFlix* caching algorithm, which not only takes into account how much a user will continue to consume (as in heuristic 1 and 2) and content specific features (as in heuristic 2), but also capture the temporal dependencies in the amount of content consumed by the user on a daily basis.

6.4.1 Overview

We model the prediction problem as a sequence problem, where the user's past viewing behavior is encapsulated as a sequence of episodes they watch from series in past WSs. A prediction of how many episodes the user will watch in proceeding WSs is then made based on their history. For every WS, and for every series watched in that WS, we cache proceeding episodes during the following off-peak hours for the next WS. If there are multiple series watched in the same WS, for example episode 1 of series *a*, and episode 1 of series *b*, then caching will be triggered for both series *a* and series *b*. *CacheFlix* predicts the number of proceeding episodes to prefetch, and the optimal time to cache it for, while optimizing for caching efficiency under storage and bandwidth constraints.

For the remainder of this section, we present *CacheFlix* which predicts how many proceeding episodes to cache from a series they have watched in the WS prior to the prefetching time, for future watch-sessions. The algorithm uses an *local* learner which is boosted by predictions made by an *global* learner. The details on the features used for prediction, the learners and the boosting algorithm, is provided in the proceeding subsections. The cache eviction policies we consider are also discussed.

6.4.2 Feature Design

Informed by our insights into the user's content preferences (section 6.2.3), their temporal access patterns (section 6.2.3), and the results from the heuristics, we consider 3 categories of features: content specific features, series popularity features, and temporal features. These features will be used to predict the number of episodes to cache during off-peak periods for a particular user.

Content specific features

The features that are specific to the content that the user watches fall under this category. These features include the genres of the series (there are 29 possible genres for Netflix series, and a series can have multiple genres), the runtime of the series (the episode length time in minutes), the first air date of the series (considered as a ¡month, year¿ tuple), the number of seasons the series has, and also if the series is related to any series that was previously watched by the user (series that was watched in WSs prior to the WS for which we are caching).

Series popularity features

There are 2 features that we use as an indication of how popular a particular series is, these are: IMDB votes (the number of IMDB votes or ratings that series has received), as well as IMDB rating (the aggregate score that IMDB registered users have given these series out of 10). In addition to using these features as is, we also compute a series popularity score as shown in eq. 6.6.

$$\text{Popularity score} = \text{IMDB rating} \times \text{IMDB votes} \tag{6.6}$$

By multiplying the features to compute the popularity score, we are mapping the feature's impact as a whole, and adding another dimensionality (known as "feature crossing" [113]).

Temporal features

The temporal features are features that are related to when the content for which we are caching, was consumed. The features include the day of the week that the WS we are caching for occured on, the % of season watched until that point, the % of series that has been watched until that point, and also the number of series that the user has been watching over the past 3 watch sessions (users tend to watch 1.2 episodes less per series than they typically do if they watch more than one series in the same WS). In addition to

Table 6.2: Features description

Feature	Description	Category
Genres	Genre(s) of the *episode's* series	Content
Runtime	Length of *episode* in minutes	Content
Air date	Month and year that the *episode's* series aired	Content
No. of Seasons	No. of seasons of the *episode's* series	Content
Related	Boolean flag indicating if this *episode's* series is related to previously seen series	Content
IMDB votes	No. of IMDB votes the *episode's* series has	Popularity
IMDB rating	IMDB rating the *episode's* series has	Popularity
Popularity score	Popularity score given in eq. 6.6 for the *episode's* series	Popularity
Burstiness score	Burstiness parameter over the past w WSs	Temporal
Weekday	Day of week *episode* was watched	Temporal
Season watched %	% of *episode's* season watched	Temporal
Series watched %	% of *episode's* series watched	Temporal
No. series	No. of series watched over 3 prev. WSs	Temporal

these features, a burstiness score, computed over the past w WSs, is used as a means of identifying if there will be drastically less or more content consumed in the next WS. The burstiness parameter is computed as in eq. 7.8 and is scaled from a value of 0 to 1 resulting in a burstiness score.

6.4.3 Prediction model

Overview

The structure of the *CacheFlix* caching algorithm is shown in Fig. 6.15. The viewing history of the user is used as input to train as well as predict how many episodes to prefetch. Each entry in the viewing history, shown as a single block, consists of the the name of the series, the season and episode number, as well as the date that it was viewed. The history until the day of caching is used as input. This is sent to a feature layer which extracts the features shown in table 6.2. Once the appropriate features have been extracted, this is fed into the global long short-term memory (LSTM) network which makes a prediction on the number of episodes to prefetch for the next WS for that series, i. For the local LSTM network, the viewing history only pertaining to series i is extracted (these are shown as

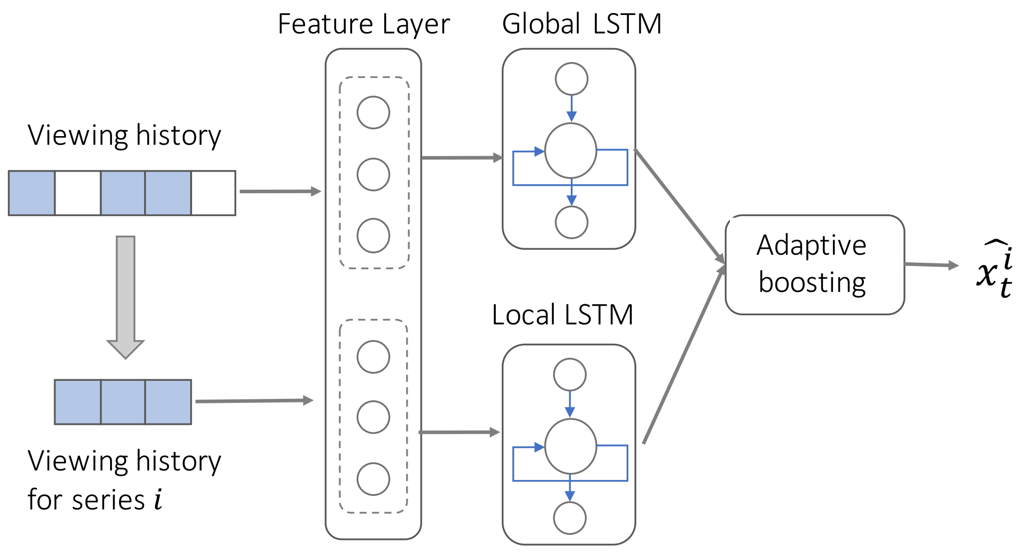


Figure 6.15: *CacheFlix* structure

the blue blocks in the viewing history sequence); this is then sent through to the features layer and subsequently into the local LSTM network which also makes a prediction on the number of episodes to cache. The output from the global LSTM learner is then boosted by the prediction made from the local LSTM learner using an adaptive boosting algorithm. Essentially, the global LSTM network attempts to learn the user's overall viewing pattern, whereas the local LSTM network, captures the viewing pattern for that particular series only. The specifics of the learners are discussed in the proceeding subsections.

LSTM Networks

With the immense success of long short-term memory (LSTM) networks in being able to capture dependencies between items in a sequence, we select LSTMs as the core component upon which the learners are built. LSTMs are a class of neural networks used in the field of deep learning that allow previous outputs to be used as inputs while having hidden states. At its core, the sequence prediction utilizes a type of recurrent neural network (RNN) called long short-term memory (LSTM) network. RNNs have recently proven to be successful for sequence prediction tasks such as for handwriting recognition, speech generation and image classification. RNNs are a class of neural networks that allow previous outputs to be used as inputs while having hidden states; this allows RNNs to have a temporal dimension as well. However, RNNs have been proved to suffer from the vanishing gradient problem which hampers the learning of long sequence of data. To address this LSTMs were created. LSTMs have internal mechanisms which regulate the flow of information, and allow the network to learn what information is important, and what should be forgotten [114]. Each neuron in a LSTM network is called a memory cell and includes a multiplicative forget gate, an input gate and an output gate. The gates, each of which are structured as neural networks with either *tanh* or *sigmoid* activation functions, are used to control access to the cells.

Global LSTM Learner

In order to capture the overall user's viewing patterns, we make a prediction $\hat{x}_t^i$ which is the number of episodes watched during watch session t belonging to series i. For each series i watched in the previous watch session (watch session $t-1$), a prediction $\hat{x}_t^i$ is made based on the user's history from watch session m until watch session $t-1$ where $m < t$:

$$\hat{x}_t^i = f(\mathbf{X}_m, \mathbf{X}_{m+1}, \mathbf{X}_{m+2}, ..., \mathbf{X}_{t-1}) \tag{6.7}$$

where $\mathbf{X}_q = \{x_q^n | n \in R\}$ where R is the set of all series titles watched by the user. $\mathbf{X}_q$ is a sequence that contains the number of episodes from each series watched during watch session q. For example, if 3 episodes were watched from series a and 2 episodes from series b in the first watch session, then $\mathbf{X}_1 = \{x_1^a = 2, x_1^b = 3\}$. This learner attempts to capture viewing trends in the user's watching behavior as the prediction is influenced by their previous viewing patterns from some watch session m.

Local LSTM Learner

As we can see from Eq. 6.7, the sequence contains episodes watched from all series. We therefore train a local learner which specifically is trained on the sequence of episodes for a series the prediction is being made for. We can make a prediction, $\hat{x}_t^k$, for TV series k for

WS t based on the user's viewing history from WS m where $m < t$:

$$\hat{x}_t^k = f(X_m^k, X_{m+1}^k, X_{m+2}^k, ..., X_{t-1}^k) \tag{6.8}$$

where X_q^k is the number of episodes watched in WS q for TV series k. For example, if we are making a prediction for series a for the user's fourth WS, then $\hat{x}_4^a = f(1,0,3)$ if the user has watched 1 episode of series a during their first WS, 0 episodes of series a during their second WS, and 3 episodes from series a in the third WS. This learner captures the behavior of the user specific to the series for which the prediction is being made.

Boosting algorithm

The prediction from the local LSTM learner as defined in Eq. 6.8 can be boosted by the predictions made by the global learner defined in Eq. 6.7. The boosting procedure trains the local learner based on incorrect predictions made by the global learner. Specifically, given our input training sample sequence and output value pair : $(\mathbf{x_1}, y_1)...(\mathbf{x_m}, y_m)$, where each training sample (in the form of eq. 6.7) consists of a sequence of prior history, and the prediction thereof, we initialize the sample weight as:

$$D_1(i) = 1/m \text{ for } i = 1, ..., m \tag{6.9}$$

We train the global learner using the training samples which are sampled according to the distribution of D_1. We then obtain the error from the global learner for each sample as:

$$e_1(i) = \frac{|y_i - \hat{y}_i|}{y_i} \text{ for } i = 1, ..., m \tag{6.10}$$

The weights of each of the training samples are updated as follows:

$$D_2(i) = \frac{D_1(i)exp(-\beta_1 e_1(i))}{\sum_{i=1}^m D_1(i)exp(-\beta_1 e_1(i))} \text{ for } i = 1, ..., m \tag{6.11}$$

where we compute the learning rate as:

$$\beta_1 = \frac{1}{2} ln \left(\frac{1 - \sum_{i=1}^m e_1(i)}{\sum_{i=1}^m e_1(i)} \right) \tag{6.12}$$

The updated weights given in Eq. 6.11 is used to train the local learner with the training samples sampled according to the the distribution of D_2. This procedure essentially trains the local learner to correct the wrong predictions made by the global learner.

6.4.4 Eviction Strategies

The end-user devices on which the cached items will be stored has stringent storage constraints. To fully understand how this impacts the performance of *CacheFlix*, particularly the caching efficiency, we consider 4 different caching eviction policies for which *CacheFlix* will be evaluated. Based on the policy, the episodes are stored in the cache with an associated time-to-live (TTL) parameter based on the cache eviction policy; this parameter indicates after how many watch-sessions a stored and not-watched episode in the cache, should be evicted. The cache eviction policies are:

- Simple: The cache is emptied at the end of each WS before any new content is cached. The TTL parameter is set as 1.
- User specific: For each user, the average number of watch sessions between subsequent episode watches from the same series is computed across their viewing history (for example, if episode 1 and 2 of series *a* is watched in WS 1, and the proceeding episode 3 is watched in WS 4, then the difference between the WSs is 3). This value is then used as the TTL parameter for each item that is cached.
- Unlimited: Across all users, we found that for our dataset, approximately 90% of all episodes that proceed previously watched episodes is consumed within the next 20 watch sessions. We thus set the TTL parameter to 20 for each item in the cache that is prefetched.
- FIFO cache eviction: In this policy, we evict contents in a cache with a fixed storage size on a first-in-first-out (FIFO) basis once it gets full.

6.5 Performance Evaluation

We evaluate *CacheFlix* on the data collected from 1060 users for 4 different cache eviction policies in terms of the PA and CE. For every WS, *CacheFlix* is triggered to cache, during off-peak hours, the proceeding episodes of the series that were watched in that WS. To evaluate *CacheFlix*, the user's data obtained through mTurk, is first parsed and stored in a Postgres PSQL database with the tables described in section 6.2.2. The caching algorithm was implemented on a macOS Mojave system with a 2.5 GHz Intel Core i7. We split the user's viewing history in 2 halves: their first 6 months, and then the second 6 months. For each set, we train *CacheFlix* on the first 75% of the user's WSs, and evaluate on the remaining 25%. This effectively doubles the test data for our evaluation. For the user specific cache eviction policy, the time-to-live parameter associated with each episode is computed from the training data (see section 6.4.4). Each LSTM is trained for 30 time-steps with 3 hidden layers, and proceeded by a dropout layer (dropout rate = 0.4) to avoid over-fitting, and then followed by a fully connected dense layer for the output (the hyper-parameters were determined through 10-fold cross-validation). The burstiness window, w was empirically determined and set to 3 days. The results are evaluated over 100 epochs. The deep-learning models are implemented using Keras with Tensorflow as the back-end.

6.5.1 Bandwidth Implications

When *CacheFlix* 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 flattening of the network traffic demand curve. As there is no time of viewing shown in the user's viewing activity file, we make the assumption that episodes are watched during peak hours i.e. episodes are not watched between 2 AM to 6 AM (this corresponds with the overall Internet Traffic Usage distribution as discussed in section 6.1.1).

We compute the average BW per month that was shifted to off-peak periods for each user's test datasets when using the *user specific* cache eviction policy; the corresponding CDF of the BW reduction during peak-periods is shown in Fig. 6.16. According to Netflix, watching videos uses about 1 GB of data per hour for each stream of standard definition (SD) video [98]. Using the runtime for episodes that are cached, we can compute the BW consumption for SD video consumption at 1 GB/hour. We see that for the average user,

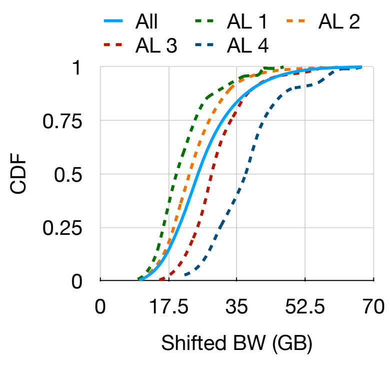


Figure 6.16: Average per month BW shifted to off-peak hours

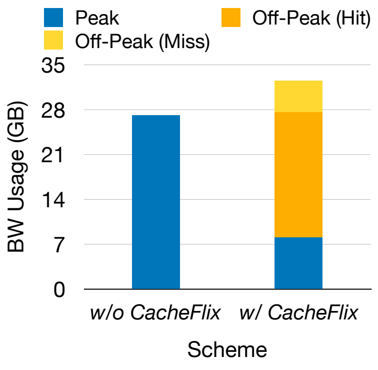


Figure 6.17: Average per month BW usage without and with *CacheFlix*

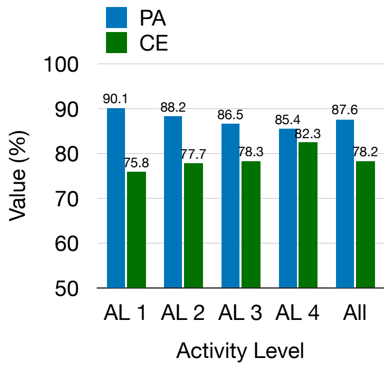


Figure 6.18: Prefetching algorithm results across all users

approximately 19 GB of their Netflix data consumption is shifted to off-peak periods; the average user watches approximately 27 GB per month. This translates to 70% of their Netflix BW usage being shifted to off-peak hours.

Furthermore, in order to understand the impact on overall BW consumption, we need to also take into account the data that is used for caching content that is not watched by the user. This is shown in Fig. 6.17; we can see here that for the average user, without *CacheFlix* the user consumes 27 GB per month, and with *CacheFlix*, approximately 24 GB is consumed during off-peak periods- this includes the 19 GB of prefetched and watched content (off-peak (hit)), in addition to 5 GB of cached content that is not watched by the user (off-peak (miss)); 8 GB of content, that was not cached by *CacheFlix*, is downloaded at time of consumption (shown as peak hours). Thus, with *CacheFlix*, we are able to shift 70% of peak-time traffic to off-peak hours while increasing the overall BW consumption, during off-peak hours, by 21 % (from 27 GB without *CacheFlix* to 32.5 GB with *CacheFlix*).

6.5.2 *CacheFlix* Prediction Performance

The PA and CE across all users in our dataset is shown in Fig. 6.18; the results are shown for the *user specific* cache eviction policy. The PA and CE of *CacheFlix* across all the users is 87.6% and 78.2% respectively- this is a significant improvement from the results obtained in the proposed heuristics. In general, the PA decreases for more active users (from 90.1 %

to 85.4%), while the CE increases (from 75.8 % to 82.3 %). A PA of 87.6% implies that *CacheFlix* is not able to prefetch 12.4% of the Netflix content that the user consumes. We computed that approximately 5.3% of is due to *CacheFlix* under-predicting the number of episodes to cache, 3.2% is due to users watching future episodes that do not immediately follow the episodes they have watched on the prefetching day and the remaining 2.9% is because the cache eviction policy removes the episode before it is consumed.

Algorithm Performance

Here we present results pertaining to the machine learning model that is used for prediction in 6.3. Instead of utilizing the dual LSTM global and local learners for the prediction, we compare the results achieved across all the users in our dataset for the *user specific* eviction strategy when using competing ML models- specifically, support vector machine (SVM), random forest and a artificial neural network with 5 layers (ANN). The models are trained using the features given in table 6.2. We find that the competing models perform poorly largely as they are not able to capture the temporal relationships in viewing activity.

Furthermore, we show the performance when using only the local LSTM, and only the global LSTM. The effectiveness of the adaptive boosting algorithm can be seen here with both LSTM's individually performing around 10-15% poorer than *CacheFlix* which boosts the output from each of the LSTMs. We also present the results of *CacheFlix* with some features removed (shown with a "-" symbol, i.e "*CacheFlix* - Content Features" means that *CacheFlix* is trained with all the features except feature in the "content" category shown in table 6.2). The importance of the features are highlighted here, with the popularity features, temporal features and content features having importance in increasing order based on the performance metrics.

Eviction Strategies

In this section we present the results across all users in the dataset for the 4 different cache eviction policies described in section 6.4.4. Fig. 6.19 shows the average PA and CE for

Table 6.3: Comparison to competing methods

Method	PA [mean, std]	CE [mean, std]
SVM	[63.2, 10.2]	[34.5, 25.4]
Random Forest	[66.7, 7.5]	[42.6, 4.7]
ANN	[68.2, 6.5]	[40.3, 8.7]
Local LSTM	[78.6, 4.5]	[64.3, 2.3]
Global LSTM	[79.6, 8.7]	[69.3, 4.5]
CacheFlix - Content Features	[59.2, 3.4]	[51.3, 4.2]
CacheFlix - Temporal Features	[61.4, 8.9]	[58.7, 10.2]
CacheFlix - Popularity Features	[80.6, 5.6]	[74.5, 3.4]
CacheFlix + All features	[87.6, 4.6]	[78.2, 6.2]

the *simple*, *user specific* and *unlimited* eviction policies. We are able to achieve the highest prediction accuracy of 90.36% with the *unlimited* eviction policy, however this is at the cost of the lowest CE of 56.2%, as due to the long TTL attached to each episode, the number of episodes in the cache at any given time is high as compared to other eviction policies. The simple eviction policy performs the worst in terms of accuracy with a PA of 70.2%; this is because we found that approximately 27% of prefetchable content, is not consumed in the immediately proceeding watch-session. Fig. 6.20 shows the performance of the FIFO policy for various cache storage sizes (1 hour of SD Netflix content consumes 1 GB data [108]). As the cache size increases from 4 GB onwards, the CE drops slightly as we continue to store content that is not consumed in future WSs (recall CE is the ratio of cached episodes to the number of episodes stored in the cache). However the PA increases as the size increases, and is comparable to the unlimited cache policy when the storage size is 10 GB.

The cache eviction policy can be appropriately tuned depending on whether a strong preference for a high PA is desired, or if a conservative CE is desired. Furthermore, we see that the predictions from *CacheFlix*, for any cache eviction strategy, significantly improves the CE (from 6% as was seen in section 6.3.3) and yields high PA with a fairly small cache sizes (in the order of units of GBs). This makes *CacheFlix* suitable for end-user device

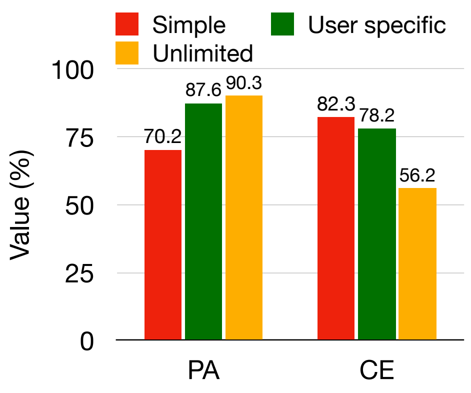


Figure 6.19: PA and CE for different eviction strategies

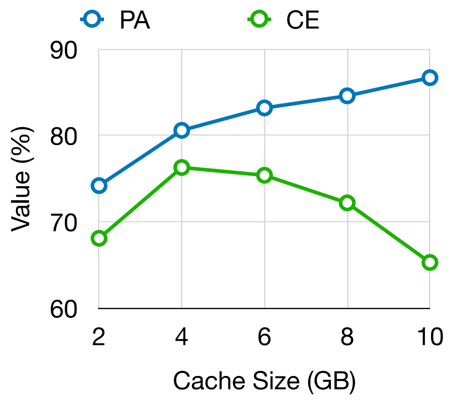


Figure 6.20: PA and CE for FIFO eviction strategy

caching as it utilizes a fraction of built-in storage avaiable in smart WiFi routers [115].

6.6 Comparison to *MANTIS*

In this section, we present a comparison to a closely related work in [116]; the authors present a proactive caching scheme that caches YouTube videos to the user's viewing device based on their viewing history. The goal of the work is similar to ours in that it aims at reducing peak-time cellular network traffic by prefetching content during off-peak hours. The authors propose a prefetching scheme specifically for YouTube which leverages YouTube's related videos suggestions to prefetch videos that is related to videos that the user has previously seen. The authors employ a K-nearest neighbor classifier (KNN) which determines which videos from the related videos set to prefetch for a particular user on a particular day. 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. Here we perform a comparison between the prefetching algorithm presented in [116] and *CacheFlix*.

To predict the number of episodes to prefetch for the next WS, we employ a KNN classifier to find a similar series, to the series that is being prefetched for, in the user's past viewing activity. We present the results for training the KNN classifier on a combination of features described in table 6.2. Once a similar series has been found, we use the watching pattern of that series as a way of predicting how the user will watch episodes for the series we are prefetching for. We evaluate the KNN classifier when trained on the following combination of features: A) content related features, B) content and popularity related features and C) all the features given in table 6.2. If we assume that we are prefetching for some series *Y*, then we use the KNN classifier to find the nearest neighbor, let us call this series *X*. Then, in order to find the corresponding time in series $X's$ watching pattern, we obtain results for the following methods:

1. % series watched: We find the corresponding point when approximately the same % of series *X* has been watched, as was watched for the point in time we are prefetching for series Y. We then find the number of episodes that were watched in the following WS for series *X*, and prefetch this number of episode for series *Y*.

2. % season watched: Here we find the approximate % season watched for the closest season number of series *X* to the season we are prefetching for series *Y*.

3. Deviation: Here we find the corresponding point in series *X* where the deviation from the average number of episodes per series/WS for the user is approximately the same as for the series WS we are prefetching for. For example, if a user, on average, watches 2 episodes per series every WS and if 2 more episodes than the average (i.e. 4 episodes) of series *Y* was watched in some WS, then to determine how many episodes to prefetch, we search for the WS in which 4 episodes, or the closest thereof, of series *X* was watched, we then obtain the number of episodes of series *X* that was watched in a proceeding WS, and use this value to predict the number of episodes to prefetch for series *Y*.

The results for the PA and the CE are shown in table 6.4 for the *user specific* eviction strategy. If a similar series cannot be found in the past viewing activity of the user, then the average number of episodes watched per WS for the user, is the number of episodes that is prefetched. As with the evaluation of *CacheFlix*, we train the KNN classifier with

Table 6.4: Comparison to prefetching algorithm in [116]

Method	Content [PA, CE]	Content & Popularity [PA, CE]	All features [PA, CE]
% series watched	[45.6, 18.3]	[46.1, 17.2]	[45.7, 19.1]
% season watched	[45,1, 20.8]	[46.2, 21.2]	[48.1, 21.5]
Deviation	[44.7, 15.4]	[39.2, 12.7]	[43.2,9.8]

75% of their viewing activity, and test it on the remaining 25%. We find that the highest PA we are able to achieve based on the algorithm presented in [116], is 48.1% with a CE of 21.5%. This is when the KNN is trained with all the features, and the % season watched is used to find the corresponding point in the similar series's watching timeline. In contrast, *CacheFlix* is able to achieve a PA of 87.6% and a CE of 78.2% with the same eviction strategy. This shows that the prefetching methods applied to short-form videos (e.g. YouTube), cannot be effectively applied to long-form videos (e.g. Netflix) given the difference in their nature and way of consumption.

CHAPTER 7
A REAL-WORLD DATASET OF JOINT NETFLIX AND YOUTUBE USER WATCH-BEHAVIOR

Fixed and mobile internet traffic is dominated by two video streaming services- Netflix and YouTube. In fact, these two applications have the largest share of global Internet traffic [117]. YouTube is the leading application and is alone responsible for nearly 16% of global traffic, and is closely followed by Netflix, which is responsible for 11% of global traffic [117]. While both applications provide video content, they vastly differ in terms of the purpose they serve. YouTube is a platform for user-generated content and largely relies on advertising revenue and its creator bases, whereas Netflix is a subscription-based service that invests billions of dollars into scripted and unscripted entertainment. Furthermore, the type of videos available on these platforms differ in their nature where YouTube provides short-form videos, that are on average 12 minutes long, whereas content on Netflix are episodes from series or documentaries (typically range from 30 to 60 minutes), and full feature length films (80 to 120 minutes).

Given the dominance of both YouTube and Netflix on Internet resources, it is valuable to derive insights on YouTube and Netflix usage which can be useful to not only the research community, but to network operators, content providers, marketing agencies, content creators as well as users themselves. This serves as the primary motivation for our work in which we conduct a meaningful analysis and provide key insights using a real-world dataset of users' Netflix and YouTube watching behavior. This work studies the behavior of consuming both YouTube and Netflix content on a per-user basis. Not only is the behavior of a user's Netflix and YouTube viewing studied so that independent conclusions can be drawn, but we also present a unique perspective of how the joint watching behavior of YouTube content and Netflix content relates, including temporal access differences or how these videos are watched together or separately within a session. With our study, we hope to shine a spotlight on the user behavior consumption and provide findings and results upon which researchers can develop problems and associated solutions. Furthermore, related work has shown how online video behaviors have quickly changed in the past and has further highlighted the need for updated work in this area [118].

To this end, we use Amazon's Mechanical Turk (mTurk) platform to collect a dataset for Netflix and YouTube usage from 377 users. The collected dataset contains at least 1-year worth of both YouTube and Netflix viewing activity for each user, which amounts to over 4.3 million YouTube videos, Netflix episodes and movies collectively watched. Beyond high-level statistics published by Netflix [96] and by YouTube [75], there has been little work done towards collecting and deriving insights using real-world usage data spanning a significant period of time. Furthermore, this is the first longitudinal study which investigates the joint YouTube and Netflix watching behavior. Equipped with this dataset, we derive key insights for individual user behavior related to their watch patterns, amount of content consumption, viewing interests, and predictability of their future viewing. We also implement and evaluate classification models to predict the user's engagement in a series, and the likelihood of them continuing to watch a series. Specifically, we consider the following sets of research questions (RQs):

1. User watch patterns: Do users have a preferred time of viewing? What is the time between subsequent video watches? How does this differ for varying user activity levels?

2. Amount of content consumption: How much time is spent consuming content? How does this differ across the two platforms? How does the YouTube traffic load vary across time of day? How bursty is the amount of content consumed for Netflix vs YouTube content?

3. Viewing interests: Do users watch the same genre(s) content regularly? Do genre preferences change across YouTube and Netflix for a single user? Do active users

prefer more popular and higher rated content?

4. Predictability: How much content is related to content the user has previously seen? How much of a user's future watches are predictable? Is it easier to predict for less active users?

7.1 A Real-World Dataset

7.1.1 Dataset Collection

To collect our dataset, we rely on Amazon Mechanical Turk (mTurk) to gather anonymized Netflix and YouTube viewing history from 377 users for at least a 1-year period [78]. The mTurk platform allows a task to be posted for a fee, which in turn can be completed by users known as mTurkers. Studies have shown that mTurk samples can be accurate when studying technology use in the broader population [79]. The task we posted required mTurkers to submit both their YouTube viewing history and Netflix viewing history covering the same period (at least 1 year from January 2020 to January 2021). For the Netflix viewing history, the mTurkers were required to navigate to their *viewing activity* page associated with their profile, and download their Netflix viewing activity as a csv file; the file was then anonymously uploaded via a dropbox link[1]. For the YouTube viewing history, the mTurker was required to navigate to Google's *Takeout* 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 the dropbox link.

The Netflix viewing activity file uploaded contains 2 fields: *title* and *date*. The *title* field consists of the name of the feature film or TV series/documentary, as well as the season and episode name where applicable, separated by colons. The *date* field consists of the most recent date that the title was viewed (there is no time of day given). A Postgres SQL database is used to store the data for the mTurkers. The SQL database consists of 3

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

tables, namely, *tblUsers*, *tblSeasons*, and *tblTitles*. The *tblUsers* table is used to store the title, watch-date as well as the season number and episode number if an episode from a series is watched. These values are populated from the user's submitted viewing activity file, and through appropriate API calls from The Movie Database (TMDB) API [97]. The *tblSeasons* contains the season number and total number of episodes in each season for every series watched by the users. The *tblTitles* table contains a number of attributes related to the series and movies watched by all the users; the TMDB API is used to obtain this metadata. The attributes obtained are: the release date of the title, the IMDB rating, the number of IMDB votes for the title, the run-time in minutes, the genre(s), director(s), writers, actors, the language of the title, country of production and related titles (as determined through IMDB). There is also a field which is used to indicate if the title is a movie or a series. For series, we obtain the number of seasons, and number of episodes each season has through appropriate API calls.

For the YouTube watch-history data, the archived file that was 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. The JSON file for each user-created playlist contains a list of the video IDs for all videos added to that playlist. Similarly, the *subscriptions.json* file contains a list of all channels the user is subscribed to. There are 4 additional tables added to the database related to the user's YouTube viewing, namely, a *users* table, a *videos* table, a *playlist* table and a *subscriptions* table. The users table is used to store the watch history for each user, along with whether the video that was watched appears in one of their playlists and whether the user is subscribed to the channel which uploaded the video. The videos table, stores the associated metadata for each of the videos. The *playlist* table and *subscriptions* table contains all the video IDs of videos that appear in the mTurker's playlists, and channel IDs of all the channels that the user is subscribed to, respectively. For every entry in the watch-history.html file of the user, the video ID is retrieved (through the video URL of the associated hyperlink) and the associated watch date and time. The playlist table is then checked to see if this video ID appears in the table for this particular mTurker, if it does, then the playlist field is set to true, otherwise it is set to false. Similary the *subscriptions* table is checked and the *subscriptions* field is set accordingly. Using YouTube's data API, we obtained meta-data associated with each video in our dataset regarding its video duration, uploaded channel, category, the number of views, the number of likes, the number of dislikes each video has at the time of data collection as well 50 of its related videos.

7.1.2 Baseline Characteristics

A high-level overview of the collected Netflix and YouTube datasets is presented in table 7.1.2 and table 7.1.2 respectively. In table 7.1.2, we show the total number of movies, and series collectively watched by the 377 users. In addition, the number of series seasons and associated episodes watched by all the users in their submitted history files is presented. We refer to any Netflix episode or movie watched as a *title*; there were 778,036 titles collectively watched by the user, of which nearly 12% were movies, and the remaining were of episodes watched from series or documentaries. In table 7.1.2, we present the total number of YouTube videos watched by the 377 users, the number of channels from which those videos were uploaded from. Each video is categorized by the uploader according to 18 predefined categories and added to a particular channel; users can subscribe to the channel (known as subscriptions) and add the video to user-created playlists. We also show the number of channels that the users subscribed to; this is termed as a *subscription*. Furthermore, we present the number of playlists created by the 377 users; a playlist is a collection of videos that the user can access and share with other YouTube users. Approximately 3.6 million YouTube videos uploaded by nearly 990 thousand channels, were collectively watched.

Netflix Parameters	**Value**
No. of Movies	89,793
No. of Series	43,147
No. of Episodes	688,243
No. of Seasons	72,244

Table 7.1: Netflix Dataset

YouTube Parameters	**Value**
No. of Videos	3,590,670
No. of Channels	989,814
No. of Playlists	578
No. of Subscriptions	9,265

Table 7.2: YouTube Dataset

7.2 Analysis and Key Insights

In the following subsections, we answer questions categorized into 5 groups to gain insights regarding the users' YouTube and Netflix viewing behavior. The groups are related to the user's access patterns, amount of content consumption, user's viewing interests, the predictability of Netflix and YouTube watching behavior, and the the user's perceived engagement in a series. In order to analyze the user's behavior, and how their levels of activity impact our derived insights. We group our users into 3 categories based on the number of videos they have watched over their submitted viewing history. Table 7.3 and table 7.4 show the activity levels arranged from least active (AL 1) to most active (AL 3) for videos consumption on Netflix and YouTube, respectively. The users are grouped based on the average number of videos watched per year; for example, users that watched less than 200 Netflix titles are categorized as low active users (AL 1 users), whereas users that watched more than 800 Netflix titles, are categorized as the most active users (AL 3). The users that watched, on average, between 200 to 800 videos per year, are categorized as moderately active (AL 2). Approximately 18% of the users are grouped in AL 1, 68% in AL 2 and 14% in AL 3. Similarly, table 7.4 show the number of videos used to label user's activity levels as well as the % of users belonging in each group.

Activity Level	Videos/year	% of Users
AL 1	< 200	18
AL 2	$200 - 800$	68
AL 3	> 800	14

Table 7.3: Netflix Activity Levels

Activity Level	Videos/year	% of Users
AL 1	< 300	19
AL 2	$300 - 4000$	59
AL 3	> 4000	22

Table 7.4: YouTube Activity Levels

7.2.1 User Watch Patterns

Do users have a preferred time of viewing?

We explore whether users have a regular schedule in terms of when they access and view content; knowing what time a user is likely to access content is particularly helpful for load estimation and caching systems, and consequently can improve the user's Quality of Experience (QoE). We first show the distribution of content watched across the day of the week; this can be seen in Fig. 7.1. With movies and series varying vastly in their form (movies being three times longer and non-episodic), we henceforth show results pertaining to Netflix (NF) series viewing and NF movies viewing separately as *NF: series* and *NF: movies*. In general, we find that there is an increase in the amount of Netflix watched over the weekend, whereas for YouTube, there is a slight increase during the middle of the week and, instead, a drop during the weekend. It can be seen that there is a greater variation in how movies are watched on a day-to-day basis as compared to YouTube videos or episodes from series; the variance across the week for series and YouTube videos is 1.31% and 1.34% respectively, whereas the variance for movies is 4.95%. The sequential nature of episodes can be a potential reason for consistency in terms of the number of episodes watched across the week, wheres the short-form content of YouTube leans itself to a media

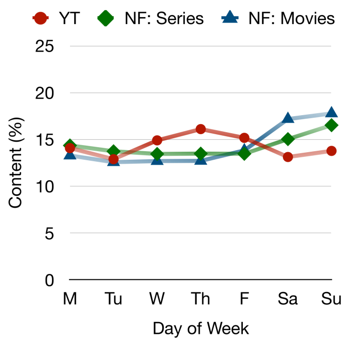


Figure 7.1: Distribution of content watched per day for YouTube and Netflix content

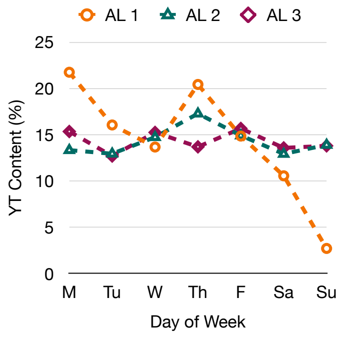


Figure 7.2: Distribution of YT videos watched per day

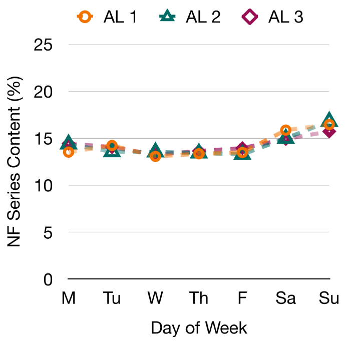


Figure 7.3: Distribution of NF series videos watched per day

which is for ”in-between” moments through the day [119]. To see if this distribution of content watched across the week is the same for all users, despite their level of activeness, we present the results for users in AL1 to AL3 for YT videos (Fig. 7.2), NF episodes (Fig. 7.3) and NF movies (Fig. 7.4). We find that the least active users are more sporadic in terms of when they watch YouTube videos (Fig. 7.2; on the other hand, the watching behavior is fairy consistent irrespective of how active a user is on the Netflix platform (as seen in Fig. 7.3 and Fig. 7.4). In addition to the day of week that a user watches YT content, we also plot the hour of day that the content is watched; this is shown in Fig. 7.5 (note that the time of viewing is not available for the NF content watched; only the date it was watched). As was seen in Fig. 7.2, the least active users (AL 1) are inconsistent across time of day, while there is a peak exhibited from 6pm to 8pm for AL 2 users and a rising increase in the content watched from 6pm till midnight for the most active users (AL 3).

To quantify whether users tend to watch content on the same day(s) each week, we compute the viewing day entropy (VDE) as given in Eq. 7.1.

$$VDE = \frac{-\sum_{d \in D} p_d \times log(p_d)}{log(N)} \tag{7.1}$$

where

$$p_d = \frac{\text{Number of videos watched on day } d}{\text{Total number of videos watched by user}} \tag{7.2}$$

and N is the total number of days in a week (N= 7). The CDF of the VDE across users is shown in Fig. 7.6. The VDE is a value between 0 and 1, where a VDE closer to 0 indicates that the user has a more regular request pattern, and a value close to 1 indicates that the user uniformly watches content across the week. We find that users watch movies are more regular intervals that episodes or YT videos; this was further evidenced in Fig. 7.1 in which we see that users watch movies over weekends more than on other days of the week. We also find that low active users are slightly more regular in terms of their day of viewing than high active users; this is across YouTube and Netflix videos shown in Fig. 7.7, Fig. 7.8 and Fig. 7.9. Furthermore, 50% of all users, regardless of their activity level, have a VDE between 0.7 and 0.85 for YT videos and NF series videos (Fig. 7.7 and Fig. 7.8), implying that in general users, do not have a regular schedule in terms of their watch pattern. In contrast, 50% of users have a VDE between 0.5 and 0.78 when watching movies implying a regular access patterns when watching movies. The viewing hour entropy (VHE) for YT content is defined as in Eq. 7.3

$$VHE = \frac{-\sum_{h\in H} p_h \times log(p_h)}{log(N)} \tag{7.3}$$

where

$$p_h = \frac{\text{Number of videos watched during hour } h}{\text{Total number of videos watched by user}} \tag{7.4}$$

and N is the total number of hours in a day (N= 24). A plot of the VHE across activity levels for YT videos is shown in Fig. 7.10. As was seen in the results for the VDE, the least active users exhibit a slightly more regular behavior than more active users; however, all users have a VHE of greater than 0.5 meaning that users do not typically have a fixed time of day for viewing YouTube content.

To analyze how the access pattern differs for a single user for YouTube vs Netflix, we compute the Netflix VDE for a single user and compare it to their YT VDE; this is shown in Fig. 7.11 where each point in the scatter plot corresponds to a single user. We find that the VDE is fairly concentrated along YouTube VDE dimension (between 0.83 and 0.87

excluding outliers), whereas it is more spread out along the Netflix VDE axis (between 0.75 and 0.9 excluding outliers). While it seems that there are no specific days that only YouTube content is watched, there are certain hours of the day when YouTube videos are consumed; this can be seen in Fig. 7.12 in which the YouTube VHE is scattered more than the Netflix VDE is. We also compute the Pearson correlation coefficient as given in Eq. 7.5:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2(y_i - \bar{y})^2}} \tag{7.5}$$

where the n is total number of users ($n = 377$), x_i is the values of the Netflix parameters values, y_i is the values of the YouTube parameters value, $\bar{x}$ is mean of the Netflix parameters values and $\bar{y}$ is mean of the YouTube parameters values. The Pearson correlation coefficient is a measure of linear correlation between two sets of data. A value of 1 represents a perfect positive relationship, -1 a perfect negative relationship, and 0 indicates the absence of a relationship between variables. The Pearson correlation coefficient for the VDE for Netflix and YouTube videos as shown in Fig. 7.11 is 0.13 indicating a weak positive relationship between the VDE of Netflix and YouTube content. To summarize the key insights:

Users watch movies on a more regular schedule as compared to series or YouTube videos with the average movies viewing day entropy being 0.1 less than the viewing day entropy for YouTube videos and Netflix series content. This is further evidenced by the variance of movies watched on a day-to-day basis being 4.95%, as compared to the variance of watching series and YouTube videos being 1.31% and 1.34%.

What is the time between subsequent watches?

A further important insight related to a user's access patterns, is the time between subsequent watch sessions (WSs), termed as *time between watch sessions* (TBWS). A watch-session is defined as a day on which at least one Netflix or YouTube video was watched. Fig. 7.13 shows the average time difference between WSs on which YouTube videos, Netflix episodes and movies were watched. We find that 75% of users, on average, watch YouTube content at least every 3 days (this corresponds to a TBWS of 2 days), series con-

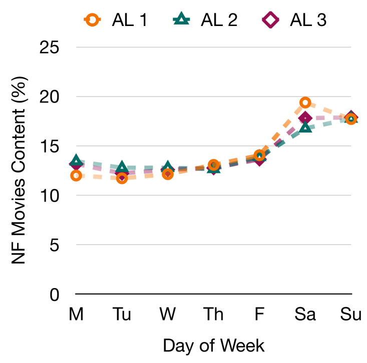


Figure 7.4: Distribution of NF movies watched per day

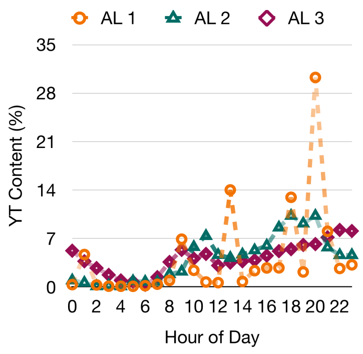


Figure 7.5: Distribution of YT videos across time of day

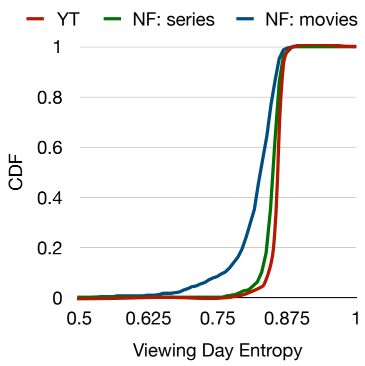


Figure 7.6: Viewing Day Entropy for Video Content

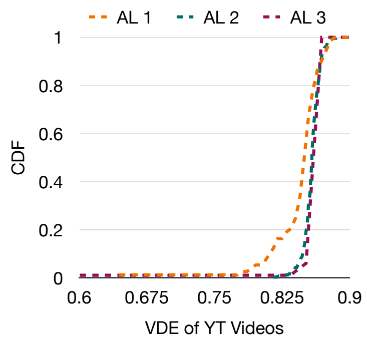


Figure 7.7: Viewing day entropy for YT videos

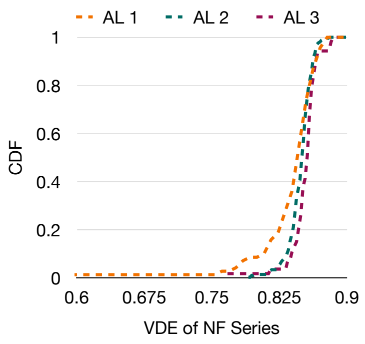


Figure 7.8: Viewing day entropy for NF series

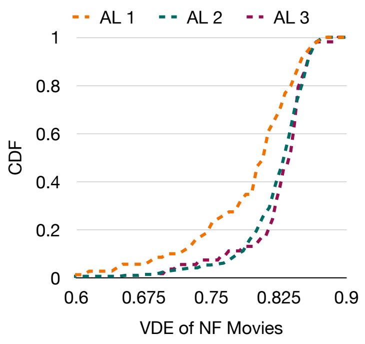


Figure 7.9: Viewing day entropy for NF movies

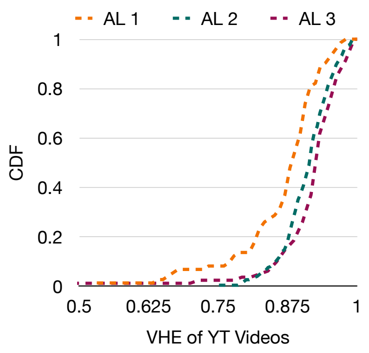


Figure 7.10: Viewing hour entropy for YT videos

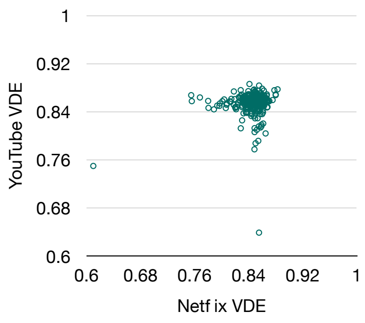


Figure 7.11: Viewing day entropy for Netflix vs YouTube content

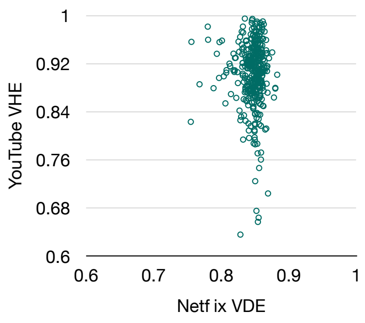


Figure 7.12: Viewing hour entropy vs viewing day entropy

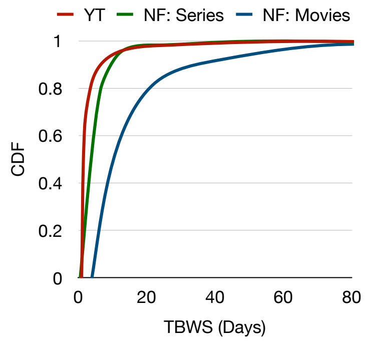


Figure 7.13: Distribution of time between WSs for NF and YT videos

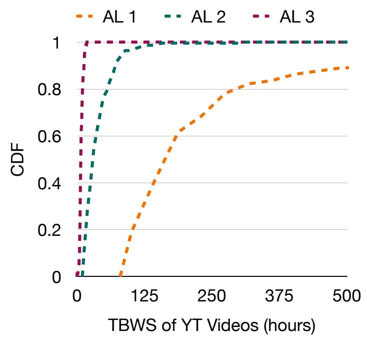


Figure 7.14: Time between WSs for YT videos across all activity levels

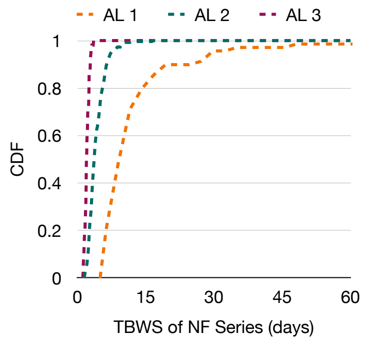


Figure 7.15: Time between WSs for NF series videos across all activity levels

tent is watched at least every 5 days and a movie is watched at least every 20 days. Fig. 7.14 shows the average number of hours between YouTube watch-session, where here, a watch-session is an hour on which at least one YouTube video was watched. We find that 75% of the most active users watch YouTube content at least every 10 hours, moderately active users watch YouTube videos every 1.5 days between watch sessions, and 75% of the least active users watch YouTube videos at least once every 10 days. Fig. 7.15 and Fig. 7.16 show the distribution of the TBWS for watching series and movies content respectively. We find that 75% of the most active Netflix users watch episodes, at least every 2.5 days, moderately active users watch every 5 days, and least active users watch series content every 2 weeks. For movies, 75% of the most active users watch movies at least once a week, moderately active users watch at least one every 2 weeks, and least active users watch movies every 1.5 months.

Similar to computing the VDE to see if a user's watch pattern follows a regular schedule, we also compute the entropy for the TBWS days. That is, we compute if the user tends to leave the same number of days or hours between watching videos. The *TBWS entropy* (TBWSE) is computed as

$$TBWSE = \frac{-\sum_{i \in I} p_t \times log(p_t)}{log(N)} \tag{7.6}$$

where

$$p_t = \frac{\text{No. of instances when TBWS was } t}{\text{Total number of WSs} - 1} \tag{7.7}$$

and N is the total number of possible TBWS. Essentially, p_t is the probability that the days between 2 WSs for a specific user is t days or hours. A TBWSE value closer to 0 implies that the user's watch content at regular intervals whereas a TBWSE closer to 1 implies a non-uniform TBWS. Fig. 7.17 shows the CDF for the TBWSE for YouTube videos, Netflix series content and Netflix movies. It is evident from the figure that the TBWSE is lower for series content and the higher for YouTube videos, meaning that users tend to follow a more regular schedule when watching series content as compared to YouTube videos or Netflix movies. Fig. 7.18, Fig. 7.19 and Fig. 7.20 shows the TBWSE across activity levels; we find that in all three types of content, the least active users have a higher TBWSE than active users, implying that the user's watch pattern is sporadic. This is line with the findings from Fig. 7.7 to Fig. 7.9 where highly active users have a larger VDE, indicating a smaller and more regular TBWS. Fig. 7.21 shows the Netflix TBWSE and YouTube TBWSE where each data point is the average TBWSE for that user over their entire viewing history. Here we can see that the YouTube TBWSE ranges from approximately 0.6 to 1 (average TBWSE is 0.78), whereas for Netflix content, it is from 0 to 0.8 (average TBWSE is 0.36) indicating that users have a more regular schedule for watching Netflix content rather than YouTube videos. The correlation coefficient is 0 indicating no linear relationship between the two values.

A typical user watches YouTube content at least once every 1.2 days, series content at least once every 3.6 days and movies at least once every 10.5 days. The days between subsequent Netflix watch-sessions is more consistent than for YouTube watch-sessions, with the average time between watch sessions for Netflix content being 0.36 and for YouTube content being 0.79.

Practical Implications

Caching of content plays an important role in lowering costs for network operators, increasing network capacity and improving user quality of experience by reducing the distance that data travels within the network. In addition to what content to cache, the temporal dimen-

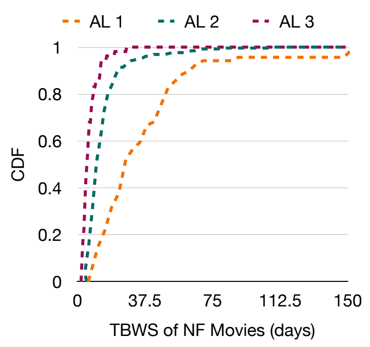


Figure 7.16: Time between WSs for movies across all activity levels

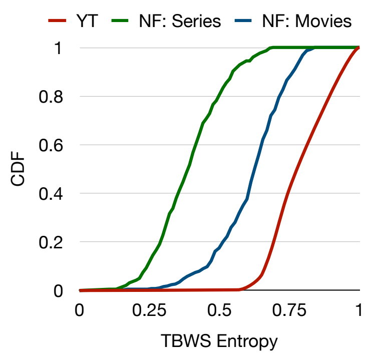


Figure 7.17: Time between WSs entropy for NF and YT content

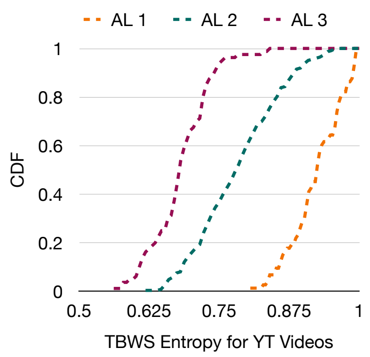


Figure 7.18: Time between WSs entropy for YT videos across all activity levels

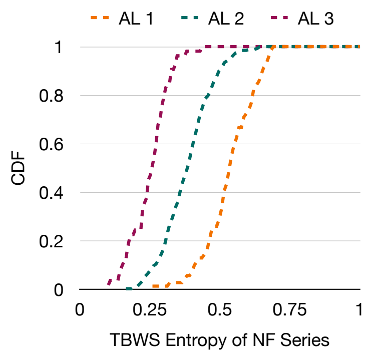


Figure 7.19: Time between WSs entropy for series content

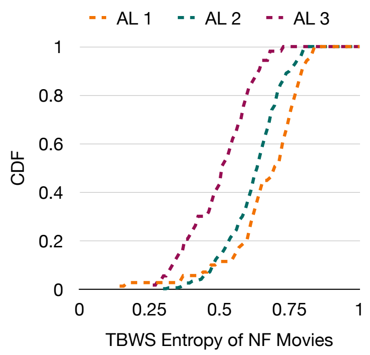


Figure 7.20: Time between WSs entropy for NF movies

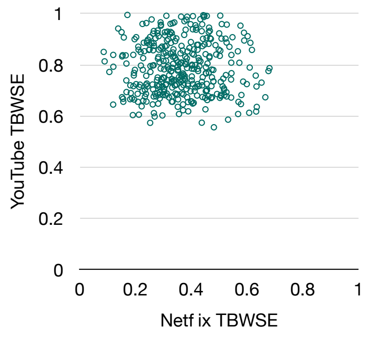


Figure 7.21: Time between WSs entropy for NF vs YT consumption

sion of *when* to cache content is equally important. To design effective caching systems, it is critical to not only understand overall traffic consumption patterns, but also application specific usage [120]; in reality, there exist different types of temporal request trends among different types of content [121]. This phenomena is evident in our collected real-world dataset which shows differing access patterns for long-form vs short-form video content e.g. YouTube content is watched more frequently however Netflix movies are consumed on a more regular schedule. Caching systems should thus understand content specific usage and can then tailor their policies based on the type of content it is catering for [120].

7.2.2 Amount of Content Consumption

How much time is spent watching YouTube and Netflix videos?

In effort of understanding user's viewing behavior as well as for the design of content delivery, caching and load estimation systems, it is crucial to gain insight regarding how much content is consumed by a user. We show the CDF of the number of videos watched each WS day (i.e. a day on which some video was watched) across all our users' viewing history in Fig. 7.22. For 75% of the users in our dataset, at most 15 YouTube videos are watched per day, at most 5 episodes from some Netflix series is watched and less than two-thirds of a Netflix movie is watched during each WS day. Fig. 7.23 to Fig. 7.25 show how this value varies across different activity levels for each WS day. We find that 75% of highly active users can watch up to 20 YouTube videos, 2.5 episodes, and nearly half a movie more than a low active user during a single WS. The greatest variance we see here across difference activity levels is in the average number of YouTube videos watched each day.

Next we investigate the time spent watching content on the different platforms. The CDF average time spent on a WS day is shown in Fig. 7.26. We find that the time spent during each YouTube WS is on average 2 hours less than the time spent during a Netflix WS. Surprisingly, we find that the time spent watching series content as compared to movies content during a single WS, is approximately the same. Fig. 7.27 to Fig. 7.29 show how the time spent during each WS varies across activity levels for YouTube content, Netflix series content and Netflix movies content. We find that there is a larger variance across activity levels for YouTube videos and Netflix series content as compared to Netflix Movies; this is in line with the results obtained regarding the number of videos watched. We find that on average, the watch-session length for least active users is 60% less for YouTube content as compared to AL 2 users, 16% less for Netflix series and 8% less for movies content.

According to Netflix, watching videos uses about 1 GB of data per hour for each stream of standard definition (SD) video, and 3 GB per hour for streaming high definition video [98]. Based on our results, the typical user spends 2 GB (when streaming at SD) or 6 GB (when streaming at HD) of data per WS day either watching Netflix series or Netflix movies content. Our findings corresponds to Netflix's recent reporting stating that the average user spends 2 hours on Netflix each day [109]. For YouTube content, the typical user spends approximately an hour per WS day watching YouTube content; this corresponds to spending around 560 MB of data when streaming at 480p or 1.2 GB when streaming at 720p [122]. While our results show that users consume less data on watching YouTube, the frequency with watching YouTube content is greater than for Netflix content as is evidenced in the discussion in section 7.2.1.

When comparing the average time spent watching Netflix content vs the average time spent watching YouTube content, we find that correlation coefficient is 0.15, indicating a weak positive linear relationship between the average time spent across their viewing history. The scatter plot of the average time spent for each user is shown in Fig. 7.30. As the plot is only comparing the average time spent over more than 1 year, the relationship between how content is watched over the 2 services in a single day is lost. We investigate this aspect in the next section.

The average user spends 27 minutes/day watching YouTube content, while spending a considerably longer 2.5 hours watching Netflix series content during a watch-session, which is also the same for movies content. The watch-session length for least active users is on average 60% less for YouTube content as compared to the moderately active user; however, for Netflix series and movies content, the watch-session length is, respectively, only 16% and 8% shorter for the least active users compared to a moderately active user.

How does time spent differ for watching short-form vs long-form content?

In this section we investigate how for a single user, the time spent watching Netflix and YouTube content differs on a per-day basis. If we plot the time spent of each service on

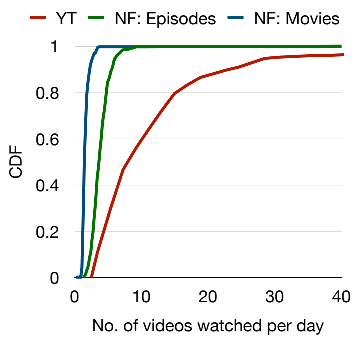


Figure 7.22: CDF of number of NF and YT videos watched each day

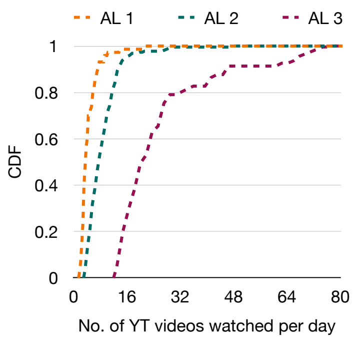


Figure 7.23: CDF of number of YT videos watched each day

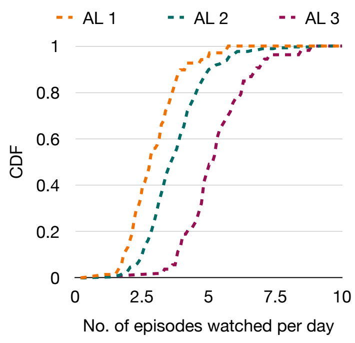


Figure 7.24: CDF of number of NF episodes watched each day

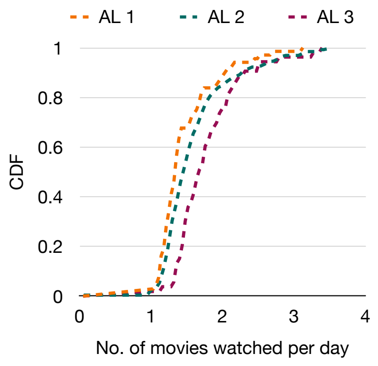


Figure 7.25: CDF of number of NF movies watched each day

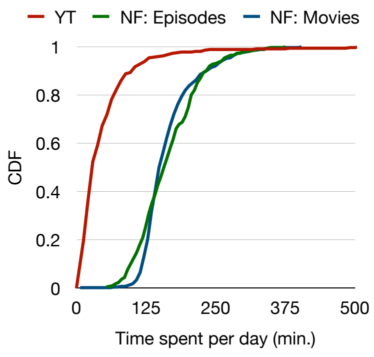


Figure 7.26: Average time spent on YT and NF per day

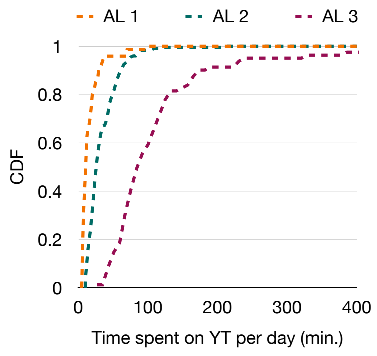


Figure 7.27: Average time spent on YT videos per day

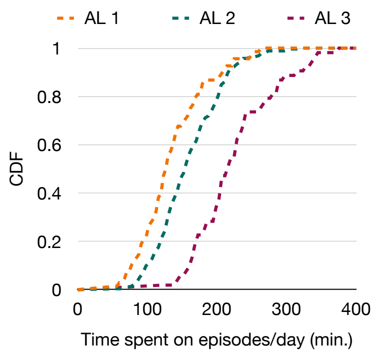


Figure 7.28: Average time spent on NF series content per day

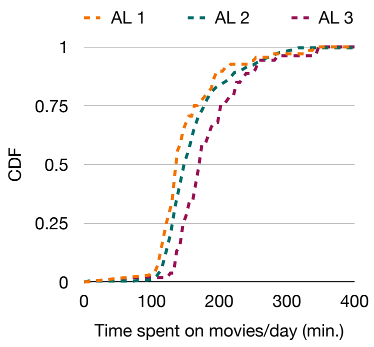


Figure 7.29: Average time spent on NF movies content per day

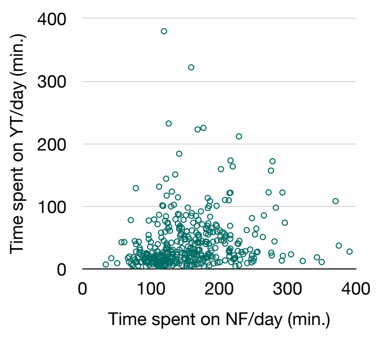
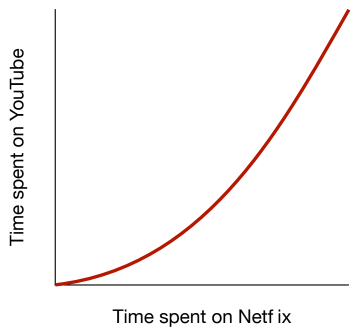
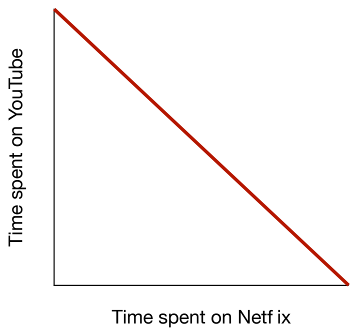


Figure 7.30: Average time spent on YT vs NF per day

a daily basis for a particular user, over some fixed period, there are four possible *time-profiles* that can result. These are shown in Fig. 7.31 to Fig. 7.34. Time profile A captures a user's behavior where on days of increased viewing, there is an increase of viewing on both YouTube and Netflix platforms. A user that fits time-profile B tends to spend a fixed amount of time watching content and thus, the amount of content watched on YouTube is inversely propotional to the amount of content watched on Netflix. A user exhibiting time-profile C's behavior is a user that generally watches the same amount of YouTube content each day, but their time spent watching Netflix content varies across days. Finally, time-profile D captures the behavior of user where the time spent on Netflix is approximately the same each day, however time spent watching YouTube videos varies. An example of 1 month of viewing for 2 users fitting time-profile A and time-profile D is shown in Fig. 7.35 and Fig. 7.36 respectively. Here, the axes are normalized with the respect to the maximum amount of time spent on YouTube and Netflix for that month; and each point on the chart refers to a single WS day in which YouTube and/or Netflix content was watched.

In order to find the best fitting time profile given the daily time spent on Netflix and YouTube over some fixed period, we compute the Pearson correlation coefficient. If the correlation coefficient is a value between 0.5 and 1, then this indicates a positive correlation which matches with the characteristics of time-profile A. If the value is between -0.5 and -1, then there is a negatives correlation and so it is classified as being time-profile B. If there is weak to no correlation (correlation coefficient is between -0.5 and 0.5), the best-fit line is computed based on the least-squares method. If the line has a gradient that is between 0 and 1, then it is classified as time-profile C, otherwise, if the gradient is greater than 5 (implying that the maximum normalized range is 0.2 for the time spent on Netflix), it is time profile D. Finally, if there are data-points that do not conform to any of the time-profiles, we label it as *none*. We compute the time-profiles on a per-month basis for each user across their entire viewing-history; we then categorize a user based on the majority of time-profile labels over their history. Fig. 7.37 shows the distribution of the time-profiles





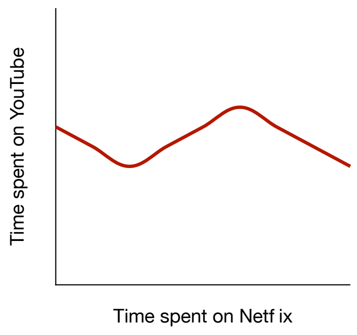


Figure 7.31: Time profile A **Figure 7.32: Time Profile B** **Figure 7.33: Time Profile C**

for all the user in our datasets. Furthermore, we show this distribution for least active and most active users; the least-active user are in the 10th percentile of users based on the number of videos (regardless of whether they were watched on YouTube or Netflix), and the most active users are in the 90th percentile. Interestingly, we find that nearly 38% users tend to spend a consistent time watching Netflix content on a day-to-day basis as compared to YouTube content. This does not translate to the least active and most active users, where in both cases, the users tend to spend time on both platforms sporadically (with nearly 48% of users not being characterized as having a typical behavior which fits time-profiles A to D). To summarize:

On a day-to-day basis, 38% of users tend to spend a consistent time watching content on Netflix while there is a larger variance watching content on YouTube, 26% of the users' time spent watching YouTube videos is inversely proportional to the time spent on Netflix, 13% of users spend approximately the same time on YouTube as compared to Netflix, and 13% spend increased time watching both YouTube and Netflix on days of increased viewing.

How does the amount of YT content consumed vary across time of day?

Internet access provisioning or network load provisioning is the process of preparing and equipping a network to allow it to handle the anticipated load and provide new services to its users. Predicting the peak workload of an Internet application and capacity provisioning based on these estimates is notoriously difficult [86]. This is because typically, the peaks of

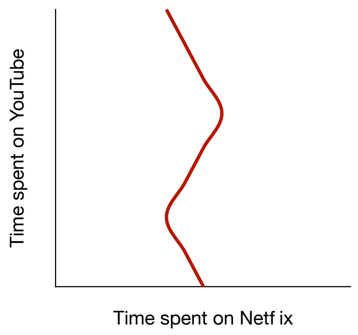


Figure 7.34: Time profile D

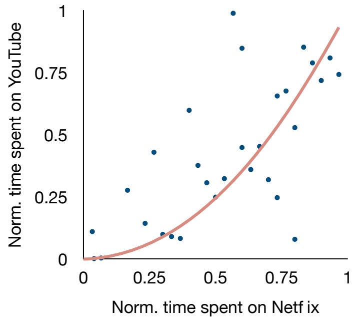


Figure 7.35: Example of time profile A

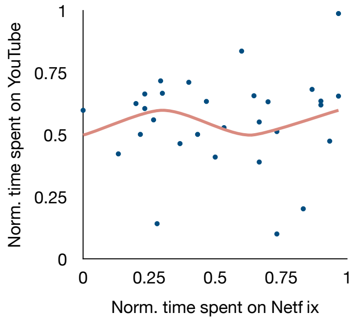


Figure 7.36: Example of time Profile C

individual users are uncorrelated, and so, the network peak load grows much more slowly than the sum of the peak loads of the individual users. To investigate the how user peak load affects overall traffic, we provide results to show the distribution of YouTube traffic across time and how bursty the usage is. Recall that we only have the time of day viewing for YouTube viewing history and not for Netflix viewing history.

To understand how YouTube specific network load is distributed through the day, for each user in our dataset, for each minute of day a video was seen, we check to see whether a video is being watched during that minute (here we assume that the video was watched in its entirety unless the start time of the next video watched by the user is before the current video has finished playing). Fig. 7.38 shows the normalized aggregate load across all users for each minute of the day. Here we see that from approximately 2AM to 10AM, the load drops significantly for AL 2 and AL 3 users. During the rest of the day, the load is just over twice as much. This is similar across all activity levels.

Overall YouTube traffic is nearly 2x as much during peak periods (10AM to 2AM) as compared to off-peak periods (2AM to 10AM), regardless of user activity level.

How bursty is the users access pattern?

An important consideration for prefetching and caching systems, is being able to effectively predict how much content a user will see, usually based on their past behavior. It follows that users with uniform behavior will be easier to predict for than users with inconsistent

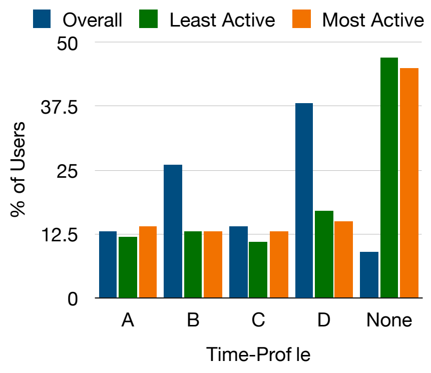


Figure 7.37: Distribution of users according to time-profiles

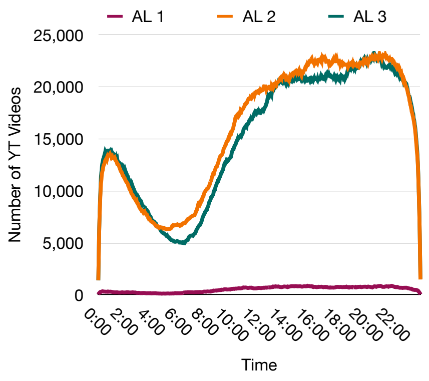


Figure 7.38: YT video load across time of day for all users

behavior. We observed that some users in our dataset drastically increase or decrease the number of videos they watch in 1 WS as compared to previous WSs. This insight leads us to investigate the "burstiness" of the amount of content consumed during WSs; this parameter is computed as in Goh and Barabasi [99]. The Burstiness parameter is defined in equation 7.8 as,

$$B = \frac{\sigma_t - m_t}{\sigma_t + m_t} \tag{7.8}$$

where σ_t is the standard deviation and m_t is the mean of the user's number of videos per WS, over a period of t days. The parameter is a value between -1 and 1, where a value closer to 1 means that the standard deviation is larger than the mean, implying that the user's behavior is bursty with regard to the number of videos they consume in consecutive WSs. A value closer to -1 indicates the user watches almost the same number of videos each WS. As an example, if user A watches the following number of videos from Monday to Friday: [M=2, Tu=3, W=2, Th=2, F=3], then the burstiness parameter is -0.6; whereas if user B watches videos as follows: [M=0, Tu=5, W=0, Th=10, F=0], then the burstiness parameter is 0.2.

Fig. 7.39 shows the average burstiness parameter across the user's entire history and Fig. 7.40 shows the average monthly burstiness parameter (t= 30). We find that in general, the burstiness parameter is lower when users are watching movies as compared to series or YouTube videos. The difference in terms of the number of videos from one WS to the

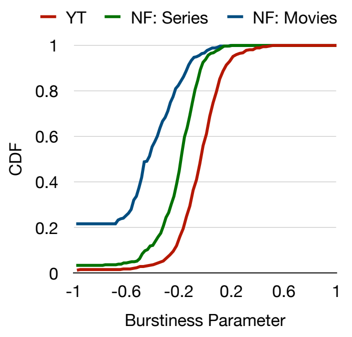


Figure 7.39: CDF of burstiness score for NF and YT videos over entire history

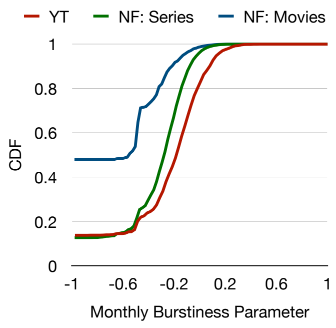


Figure 7.40: CDF of burstiness score on a monthly basis

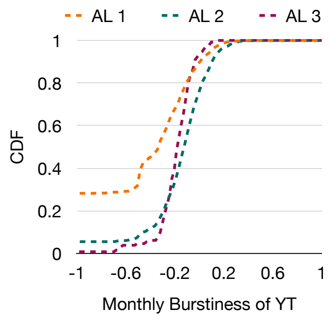


Figure 7.41: CDF of average monthly burstiness score for YT videos

next is larger if the video duration is shorter (i.e. the way in which YouTube videos are watched, which are on average shorter than series episodes and movies, tends to be more bursty). The monthly average burstiness is lower than the average burstiness over at least a year's viewing history; this is expected as there is a larger variance in the number of videos watched over a longer period of time which results in a higher burstiness score. However, to make accurate short-term predictions, it is important to consider the behavior over shorter periods of time. In general, we find that the more active the user is, the more bursty their behavior is i.e. there is more variance in the number of videos consumed per WS for active users; this can be seen in Fig. 7.41 to Fig. 7.46. We also find that users have a higher burstiness score when watching YouTube videos as compared to Netflix videos (the monthly burstiness score on average for YouTube videos is 0.1, for series content it is 0.01 and for movies, this is -0.2). A scatter plot of the average monthly burstiness score per user is shown in Fig. 7.47; the correlation coefficient is -0.03 implying that there in no linear relationship between these two parameters.

The typical user exhibits a more sporadic behavior in terms of the number YouTube videos watched from day to day, as compared to Netflix series and movies content with a average burstiness score of 0.1, 0.01 and -0.2 for YouTube videos, series and movies respectively.

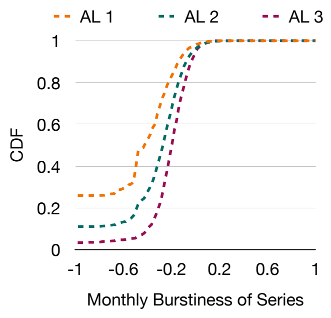


Figure 7.42: CDF of average monthly burstiness score for NF series

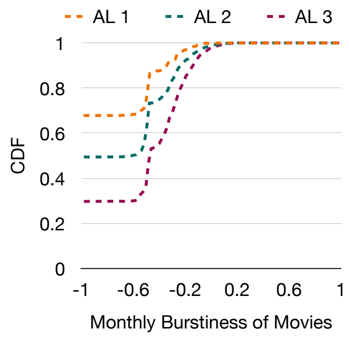


Figure 7.43: CDF of average monthly burstiness score for NF movies

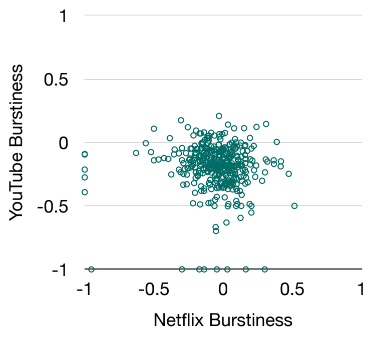


Figure 7.44: Average monthly burstiness score of YouTube vs Netflix content

Practical Implications

Internet access provisioning or network load provisioning is the process of preparing and equipping a network to allow it to handle the anticipated load and provide new services to its users. Predicting the peak workload of an Internet application and capacity provisioning based on these estimates is notoriously difficult [86]. This is because typically, the peaks of individual users are uncorrelated, and so, the network peak load grows much more slowly than the sum of the peak loads of the individual subscribers whose traffic is carried by the network. With YouTube and Netflix having a considerable influence on network traffic, it is beneficial to understand the distribution of such traffic and the users' content consumption patterns. Through our analysis in this section, we compared how on a daily basis, the content consumption for long-form vs short-form video content differs, how individual YouTube traffic varies across time of day, as well as how bursty the different types of content this. For example, we saw that nearly 40% of users tend to watch the same amount of Netflix content each day but there is a larger variance in terms of YouTube watching behavior. Network service providers would need to take these insights into account when developing their provisioning strategies.

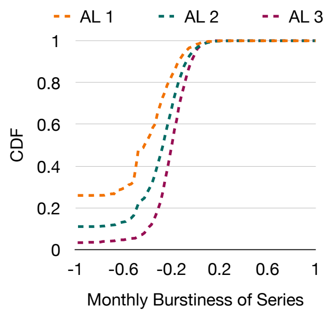


Figure 7.45: CDF of average monthly burstiness score for NF series

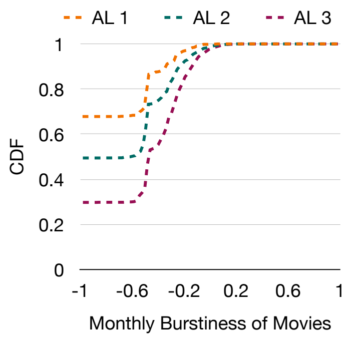


Figure 7.46: CDF of average monthly burstiness score for NF movies

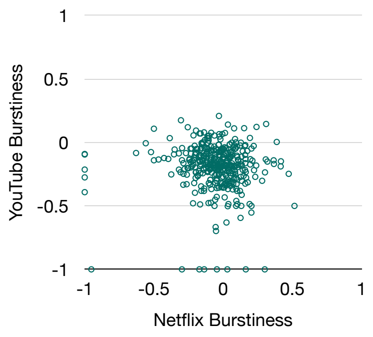


Figure 7.47: Average monthly burstiness score of YouTube vs Netflix content

7.2.3 User Preferences

Do users watch the same genre(s) regularly?

Learning about user preferences makes it possible to model user information needs and adapt services to meet these needs. This is an important question for recommendation engines and proactive caching systems, where a prediction of what to cache is made based on the user's preferences. Understanding users' preferences would also be useful for targeted advertising. There are 21 genres of Netflix series that are watched by the users in our dataset, and a series can be assigned multiple genres; and there are 21 genres that Netflix movies are assigned. Every YouTube video is categorized by the uploader according to 18 predefined categories. A distribution of the videos watched by all the users in our dataset, and the genres of the associated videos, is shown in Fig. 7.48 to Fig. 7.50. We find that the largest % of YouTube videos watched, across all activity levels (nearly 30% overall), is from the "music" genre. For Netflix series we see that the most popular genre is "drama", and "comedy" is the most popular movie genres across all activity levels.

This, however, does not tell us if users in different activity levels have a concentrated preference in terms of the genre of content (i.e. they tend to watch content only from 1 or 2 genres) or a more diverse genre preference (i.e. they watch content from multiple genres). To quantify this, we compute the user's *viewing genre entropy* (VGE) as given in Eq. 7.9.

$$VGE = \frac{-\sum_{g \in G} p_g \times log(p_g)}{log(N)} \tag{7.9}$$

where

$$p_g = \frac{\text{Number of episodes in genre } g}{\text{Total number of episodes watched by user}} \tag{7.10}$$

and N is the total number of genres. The VGE is a number between 0 and 1; a value closer to 0 means that the user has more stability in terms of their preference (they prefer content from a few genres only), whereas a larger VGE means that the user watches content from various genres. We computed the VGE for each month of the user's viewing history, and obtained the average across all the months; the results are shown in Fig. 7.51. We find that the VGE is typically the lowest for series content (0.52 on average) and the highest for movies (0.76 on average). This indicates that users tend to watch series content from the same genre(s), whereas the users are more diverse in their choice of movie genres. Fig. 7.52 to Fig. 7.54 shows the VGE across activity levels; interestingly, we find that the VGE for YouTube videos is fairly similar regardless of activity levels (the average VGE is 0.71), whereas for Netflix content, less active users tend to be comparatively concentrated in their genre/category preferences. The plot in Fig. 7.55 shows the average per-user VGE for Netflix vs YouTube videos. We find that the VGE for YouTube content is more scattered than for Netflix content, indicating that users tend to watch videos for different categories as compared to Netflix videos. The correlation coefficient is 0.1 indicating a weak positive relationship between the VGE of Netflix vs YouTube content meaning that users that tend to have concentrated genre preferences for YouTube, may also have concentrated preferences for Netflix content.

The typical user has more concentrated genre preferences for series content (average viewing genre entropy is 0.52) as compared to YouTube videos (viewing genre entropy is 0.71) and movies (viewing genre entropy is 0.76); interestingly, this holds true across activity levels.

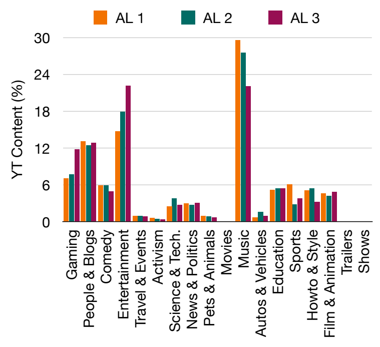


Figure 7.48: Distribution of YT genres watched across all users

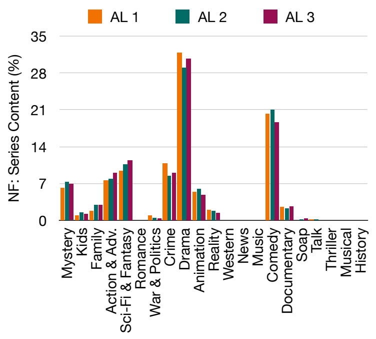


Figure 7.49: Distribution of NF series genres watched across all users

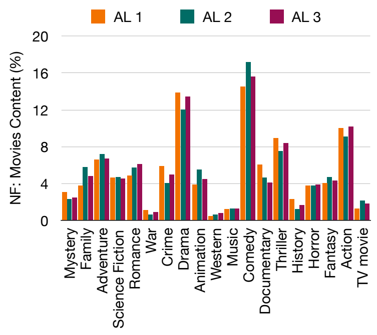


Figure 7.50: Distribution of NF movies genres watched across all users

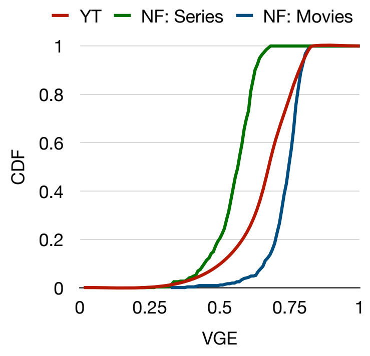


Figure 7.51: Viewing genre entropy for YT and NF content

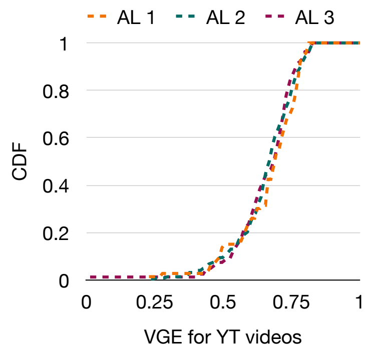


Figure 7.52: Viewing genre entropy of YT content

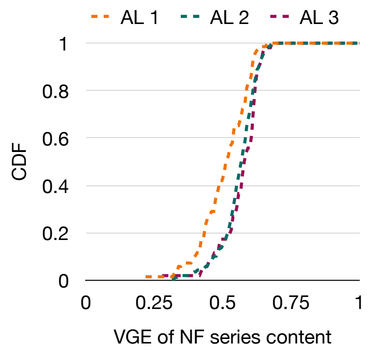


Figure 7.53: Viewing genre entropy of NF series content

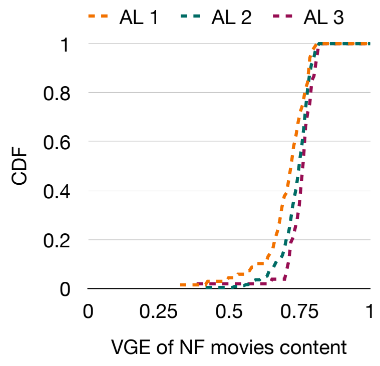


Figure 7.54: Viewing genre entropy of NF movies content

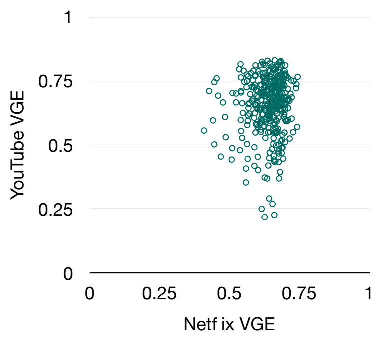


Figure 7.55: Viewing genre entropy for Netflix vs YouTube on a per-user basis

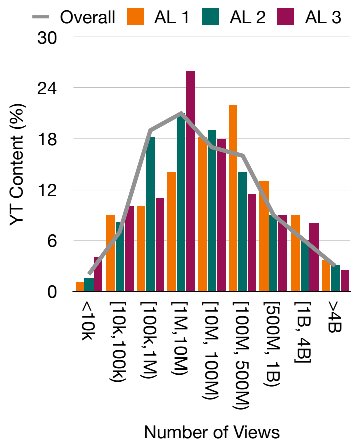


Figure 7.56: Distribution of view count for YT content

Do active users prefer more popular YouTube and Netflix content?

Gaining insight into how the popularity and ratings of content affect the consumption for different activity levels is helpful for caching and content delivery systems. For each series watched by users in our dataset, we obtained the number of IMDB votes the series/movie had at the time of retrieval. IMDB is an extensive online database of information related to movies, TV series and streaming content- including rating and reviews that are given by registered IMDB users. The number of votes a series/movies received can serve as a indication of how popular that title is, and similarly, the number of views that a YouTube videos received can serves as a measure of popularity. Fig. 7.56 shows the distribution of the views that users' watched YouTube videos have; we find that 46% of videos that the least active users watch has more than 100 million views, whereas 29% of highly active users' videos fall in this range. The average number of views for YouTube videos watched by users in AL 1, AL 2 and AL 3 categories are 620 million, 360 million, 202 million views respectively. This implies that less active users tend to watch more popular content than more active users. Similar behavior can be seen for Netflix content too; the distribution of the number of votes for series content and movies content is shown in Fig.7.57 and Fig. 7.58. The average votes of Netflix series content watched for users in AL 1, AL 3 and AL 3, is 6234, 4921 and 4721 votes respectively. We see from Fig. 7.58 that the least active users prefer more popular content; more than 55% of the watched content having more than 6000 votes as compared to 41% for AL 2 users and 36% for AL 3 users.

The least active users prefer to watch more popular content on both Netflix and YouTube platforms; the YouTube videos that least active users watch has 3 times as many views as the videos that highly active users watch, and 1.5 times more votes than Netflix series and movies have.

Practical Implications

Being able to effectively understand users' preferences lends itself to the development of efficient recommendation systems and personalized advertising strategies. With Netflix

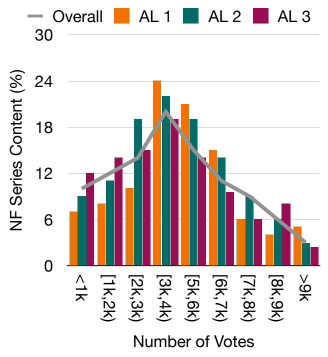


Figure 7.57: Distribution of votes received for NF series

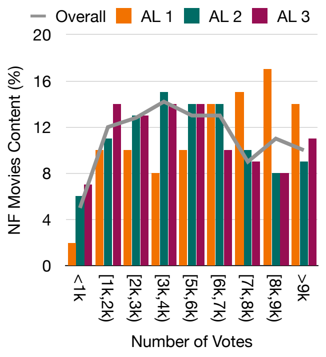


Figure 7.58: Distribution of votes received for NF movies

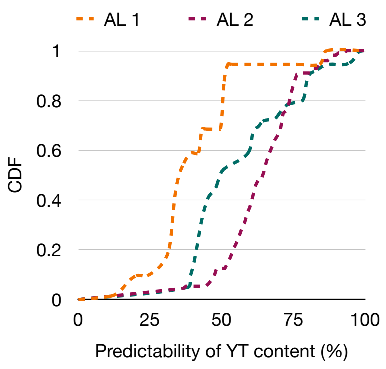


Figure 7.59: CDF of predictability for YT content

being a subscription-based model, it does not rely on personalized ads for its revenue, but rather it heavily relies on its recommendation engine for customer retention; in fact, it is estimated that 80% of stream time is achieved through their recommendation engine [123]. The recommendation algorithm has several components which work in conjunction when determining what to recommend to a particular user. It consists of a personalized video ranker (orders Netflix titles for a specific individual user), a trending now ranker (which takes into account popularity and trending features) as well makes use of collaborative filtering component [124]. The individual user's preferences is key for the personalized video ranker in particular, and being able to study and draw insights from the user preferences as well as how it changes over time is critical. Our results suggest that a user tends to have a more concentrated preference for Netflix series content than for short-form videos or movies, and that typically, the amount of content a user consumes is inversely proportional to the popularity of that content.

With regard to YouTube, the immense prevalence and widespread consumption of YouTube videos has influenced advertisers to incorporate and design their strategies around these platforms [85]. Advertising revenue on YouTube is estimated to be up to \$8.5 billion [75]. User preferences and how this changes with time would thus be of utmost interest to advertisers for targeting and personalizing adverts. Similar to advertisers, content providers are also interested in user preferences with the mutual goal of increasing visibility. Learning about user preferences makes it possible to model user information needs and adapt services to meet these needs. Interestingly, our results show that user preferences related to the types of YouTube videos they watch (characterized by their category) does not vary significantly across time, and so there is potential for time-invariant personalized advertising.

7.2.4 Predictability

How much of user's YouTube future watches are predictable?

Being able to predict what a user will watch in the future is particularly useful for prefetching strategies. Prefetching content has extensively been used to reduce user-perceived latency when loading web pages across the internet [44, 45]. 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. To explore the feasibility of prefetching, we consider how a user's YouTube watch behavior is influenced by videos they have seen in the past. Specifically, we see whether videos that are related to videos that has been seen by a user in the past, is consumed by the user in the future; we term this as the *predictability*.

YouTube algorithmically determines videos that are related to one another using the video's meta-data, and also by employing collaborative filtering methods. We 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 term this set as the "related set". We perform this analysis for all the users in our collected dataset for their entire watch-history, and present the results across the activity levels; this is shown in Fig. 7.59. This figure shows the percentage of videos that are found in the related set of videos they have seen in the past. We find that the average percentage is 60.4%, and ranges from 9.5% to 98.3%, with a standard deviation of 13.7% and median of 59.1%. In addition, we found that 8.3% of these videos are from channels the user has subscribed to, while 3.4% appears in their playlists. We find that moderately active users are more

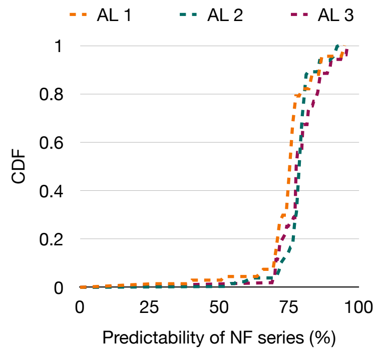


Figure 7.60: CDF of predictability for series content

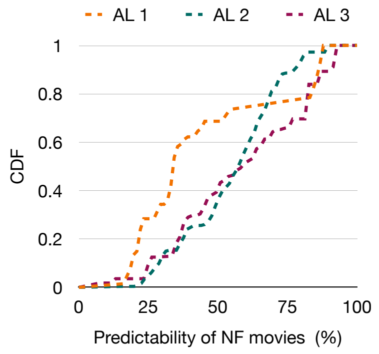


Figure 7.61: CDF of predictability for movies content

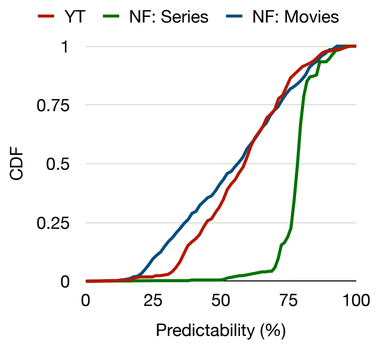


Figure 7.62: CDF of predictability for YT and NF content

predictable, in the sense, than on average, 8.7% more content appears in the related set of their previously watched content as compared to the most active users. When compared to least active users, on average, 23.4% more content is related to previously watched content.

With 60% of YouTube videos watched by a user being present in the related set of videos that the user has previously watched, YouTube watch behavior is predictable and can be used in the development of effective prefetching systems.

How much of user's Netflix future watches are predictable?

To see whether we can predict what the user will watch next based on what they have consumed in the past WSs, specifically for Netflix series content, we do the following: for every WS that appeared in a user's viewing history, and for each episode watched in that session, we check if its preceding episode was watched within a certain number of previous WSs. For example, if episode 20 of series A was watched today, we check if and how many WSs prior, episode 19 was watched. We compute this for all users in our dataset across their entire history, the average is shown in Fig. 7.60 across the various activity levels. In general, we find that approximately 78% of the user's episode watches follows an episode that the user has seen in the past. We also find that as the activity level of the user increases, there is only a slight increase in the predictable % of episodes. The average predictability for users in AL 1, AL 2 and AL 3 is 76.1%, 77.8% and 78.4%. We conclude that nearly 78% of a user's future episode watches can be predicted as it proceeds a previously watched episode from the series. These results serve as an upper bound for the accuracy the caching algorithm can achieve, if it were to cache all proceeding episodes of TV series and documentaries that the user has seen.

To evaluate the predictability of movies we employ a similar method to evaluating the predictability of YouTube content. For every movie watched, we see if it was related to a movie that was previously watched (12 related movies are obtained during the meta-data collection for every movie). Fig. 7.61 show the percentage of movie watches that was related to movies the user had previously watched. On average, approximately 56.2% of future movies were related to previous movies watched; this value increases as the activity level of the user increases. Specifically, the predictability for movie content for moderately active and highly active users is approximately the same (55.8% and 57.9% respectively), however, for the least active users, it is significantly lower with the average being nearly 20% less than for more active user. This in line with our previous results which show than low active users are more diverse in terms of their movie genres that they watch.

Nearly 78% of a user's future series episode watches, and 56% of future movie watches can be predicted based on what the user has seen in the past.

How does the predictability compare for YouTube vs Netflix?

In effort to compare the prefetching potential via the predictability metric, we show the average predictability for all users in our dataset for YouTube content, Netflix series content and Netflix movies content in Fig. 7.62. The predictability is quite similar for YouTube content and Netflix movies content (60.4% and 56.2% respectively), however it is much less than for series content (78%). This is not surprising as Netflix series content is sequential in nature, and thus prefetching and caching systems for Netflix series content can prove to be successful. However, due to limited resource constraints on the caching and prefetching devices, designing an *efficient* prefetching system that does not cache content that the user does not ultimately watch, may prove to be complex. Similarly, for caching systems that

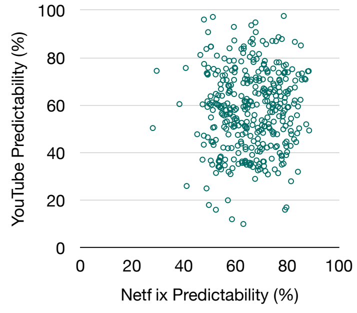


Figure 7.63: Predictability of Netflix vs YouTube content

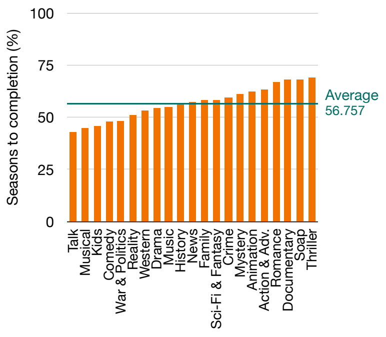


Figure 7.64: Average percentage of seasons watched to completion in each genre

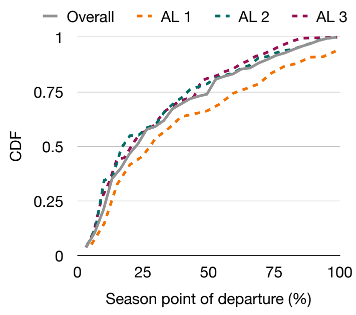


Figure 7.65: Point of departure of seasons not watched to its entirety

rely on the related movies and YouTube videos from which to prefetch, will need to employ intelligence to be able to effectively learn the user's behavior and then cache content in an efficient manner.

Fig. 7.63 shows the average per-user predictability for Netflix vs YouTube content. The Pearson correlation coefficient is 0.04 which indicates that there is no correlation between the predictability from one service to another i.e. if for a specific user, there is a higher prefetching potential for Netflix content, it does not mean it is the same for YouTube content too. We can see that there is a larger range of YouTube predictability (from 10% to 98%) to as compared to Netflix content (from 24% to 90%).

With an average correlation coefficient of 0.04, there is no correlation between the predictability potential for Netflix vs YouTube content for a single user.

Practical Implications

Being able to predict what a user will watch in the future is particularly useful for prefetching strategies. Prefetching content has extensively been used to reduce user-perceived latency when loading web pages across the internet [44, 45]. 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. The motivation for prefetching videos 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. The results presented in this section show that both YouTube and Netflix content is indeed predictable across all activity levels. Hence, there is a potential for developing successful prefetching systems which can be used to fetch content during low-cost periods (such as over WiFi or off-peak periods). However, while there is a considerable higher prefetching potential for Netflix series content (78% for Netflix series and 60% for YouTube videos), the cost of inaccurate prefetching is higher for Netflix series videos as compared to shorter YouTube videos due to bandwidth and storage constraints. Thus, designing a highly efficient prefetching system for Netflix content specifically is imperative.

CHAPTER 8
AN INTEGRATED APPROACH FOR TIME-SHIFTED PREFETCHING OF YOUTUBE AND NETFLIX SERIES VIDEOS

In the previous chapters, we provided an in-depth analysis with associated key insights from our collected datasets of YouTube and Netflix usage (chapter 3 and chapter 5). We then designed two prefetching solutions for YouTube content, namely *Mantis* (chapter 4), and then for Netflix content, namely *CacheFlix* (chapter 6). We then collected another dataset of joint YouTube and Netflix usage, and showed the interaction between these platforms at an individual user level (chapter 7). We also studied the predictability of YouTube and Netflix series videos as well (section 7.2.4); an interesting follow-up would be to determine if and how *Mantis* and *CacheFlix* can be jointly utilized cross-platforms for the prefetching of video content. To this end, we use the joint dataset collected of both YouTube and Netflix usage, and show how we can integrate *CacheFlix* and *Mantis*. We then provide results pertaining to enhanced versions of *Mantis* and *CacheFlix*. We also provide a general framework for prefetching video content, which is governed by the systems that we have developed. This framework can be applied for prefetching video content beyond videos hosted on YouTube and Netflix.

8.1 Integrated Solutions

In this section, we discuss methods to enhance our proposed *Mantis* scheme with *CacheFlix*, and vice versa. For this, we utilize the same dataset that was used for the joint insights study described in chapter 7. To reiterate, we relied on Amazon Mechanical Turk to gather anonymized Netflix and YouTube viewing history from 377 users for at least a 1-year period [78]. The task we posted required mTurkers to submit both their YouTube viewing history and Netflix viewing history covering the same period (at least 1 year from January 2020 to January 2021). A summary of the dataset is in table 7.1.2 and table 7.1.2.

Table 8.1: Accuracy of *CacheFlix+* for users in AL 1 to AL 4

Accuracy (%)	AL 1	AL 2	AL 3	AL 4	Overall
CacheFlix	83.2	82.1	84.3	81.1	82.7
CacheFlix+	86.6	85.3	87.2	83.2	85.6

Table 8.2: Efficiency of *CacheFlix+* for users in AL 1 to AL 4

Efficiency (%)	AL 1	AL 2	AL 3	AL 4	Overall
CacheFlix	70.2	73.3	74.5	76.1	73.5
CacheFlix+	74.6	75.2	77.2	79.5	76.7

8.1.1 *CacheFlix+*: Enhancing *CacheFlix* with YouTube viewing behavior

A simple but effective way to enhance *CacheFlix*, would be to include the user's YouTube watching behavior to determine if that has an influence on how the user consumes Netflix content. We term this system as *CacheFlix+*. Specifically, we include a features based representation of the user's YouTube watching behavior at the time we are prefetching Netflix series videos. As stipulated by Mantis during the candidate set generation period, we include the user's YouTube watching behavior over the 2 weeks prior to the day of prefetching. The features that we include from those videos are the viewing genre entropy (Eq. 7.9), viewing day entropy (Eq. 7.1), viewing hour entropy (Eq. 7.3), time between watch sessions entropy (Eq. 7.6), burstiness parameter (Eq. 7.8), no. of videos watched over the generation period, and also the amount of time spent watching YouTube videos over the generation period. These features are added to the features layer as described in Fig. 6.15. The rest of the operation of *CacheFlix* with the user specific eviction policy is as described in section 6.4 (the way we train and test the algorithm is as described in section 6.5). We show the results of *CacheFlix+* in table 8.1 and table 8.2; we show the results for the users in activity levels 1 to 4 and compare the accuracy and efficiency to *CacheFlix*. We find that overall, *CacheFlix+* is 2.9% more accurate, and 3.2% more efficient than *CacheFlix*. We found this to be true across all activity levels.

Table 8.3: Accuracy of *Mantis*+ for users in AL 1 to AL 4

Accuracy (%)	AL 1	AL 2	AL 3	AL 4	Overall
Mantis	73.2	74.3	77.6	78.4	75.9
Mantis+	74.1	76.3	77.8	78.7	76.7

Table 8.4: Efficiency of *Mantis*+ for users in AL 1 to AL 4

Efficiency (%)	AL 1	AL 2	AL 3	AL 4	Overall
Mantis	72.4	74.2	76.2	75.5	74.6
Mantis+	73.6	75.8	77.9	76.0	75.8

8.1.2 *Mantis*+: Enhancing *Mantis* with Netflix viewing behavior

We also look into how to supplement Mantis with a user's Netflix viewing history. We implement Mantis+ in a similar fashion in that we include a features based representation of the user's Netflix viewing behavior over the past 2 weeks from the day of prefetching. The features that are used to capture the user's Netflix behavior is the viewing genre entropy (Eq. 7.9), viewing day entropy (Eq. 7.1), the time between watch sessions entropy (Eq. 7.6), burstiness parameter (Eq. 7.8), no. of Netflix series videos watched over the generation period, and also the amount of time spent watching series videos over the generation period. These features are added to the features layer as described in Fig. 6.15. The rest of the operation of *CacheFlix* with the user specific eviction policy is as described in section 6.4 (the way we train and test the algorithm is as described in section 6.5). We show the results of *CacheFlix*+ in table 8.1 and table 8.2; we show the results for the users in activity levels 1 to 4 and compare the accuracy and efficiency to *CacheFlix*. We find that overall, *CacheFlix*+ is 2.9% more accurate, and 3.2% more efficient than *CacheFlix*. We found this to be true across all activity levels.

8.2 Framework for Prefetching and Edge-Caching Platform Agnostic Video Content

Here we propose a general framework for prefetching and caching video content in a time-shifted manner for any platform (e.g., Hulu, Disney+, Vimeo). The overall framework,

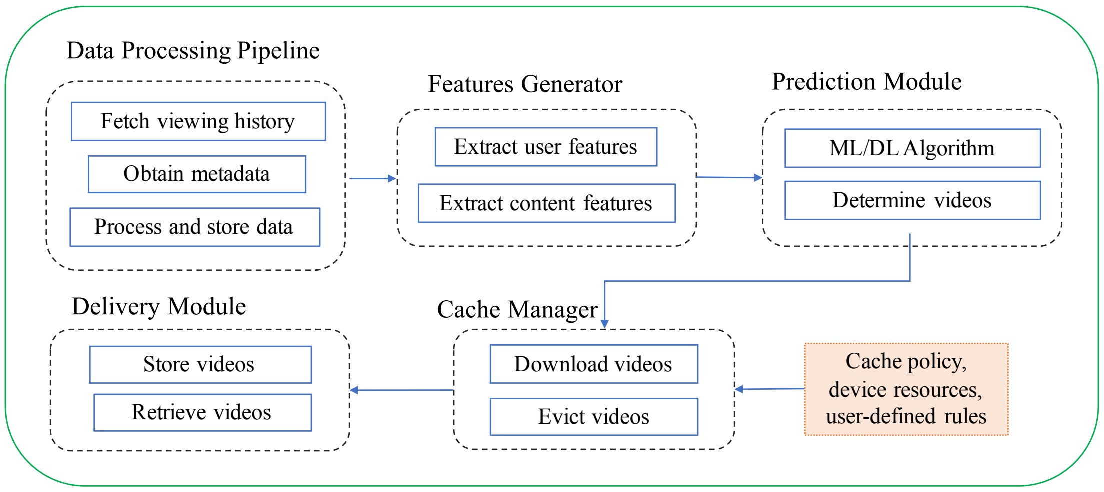


Figure 8.1: General framework for prefetching and caching of video content

governed by our proposed systems for prefetching YouTube and Netflix series videos, is in Fig. 8.1. It consists for 5 components, namely: data processing pipeline, features extraction, prediction model, cache manager and delivery. To summarize, (i) user-level watching data is collected and the associated meta-data is obtained, (ii) after which the appropriate features are extracted, (iii) this is then fed to a prediction model which determines what videos the user is likely to watch, (iv) the videos that are ultimately cached and removed are determined by the cache manager, (v) after which it is stored on the device and retrieved by the user at a later time. In the following subsections, we will discuss each component in further detail.

8.2.1 Data Processing Pipeline

This component relates to the collection and processing of the user's watching behavior which is used during both the training and prediction phases. The main purpose of this component is to learn the user's watching behavior based on their viewing history, so that accurate predictions can be made by the prediction model. This component involves obtaining the relevant metadata for the videos watched by the user so that appropriate features can be extracted. It also includes processing and storing the data which can be stored in a centralized or distributed database depending on the chosen system architecture design.

8.2.2 Features Generator

This component involves extracting both user related features and content specific features. As was discussed in the design for both *Mantis* and *CacheFlix*, there are features that are related to the type of content that is consumed (such as the genre, the release date, the people involved in the content creation etc.), as well as features that are used to capture the way the user consumes videos on that platform, and how they interact with certain videos. User specific features can include the user's temporal patterns (e.g., viewing entropy rates), the amount of content typically consumed by the user, as well as their content preferences (e.g., preferred genre).

8.2.3 Prediction Module

The prediction model consists of the machine learning or deep learning model that is trained on the user's past data, and then makes predictions based on the user's recent viewing history. For both phases, the data processing pipeline as well as the features extraction is the same. The algorithm can be a classification model (as in the case for Mantis), which is used to determine what videos to select from a candidate set, or sequential (such as for *CacheFlix*) which determines how much content to prefetch. It can also utilize a combination of these two types of models as will be discussed in the subsequent sections.

8.2.4 Cache Manager

The cache manager is responsible for fetching the videos that are predicted by the prediction module from the content provider. It also evicts videos that are stored on the device/edge-cache governed by predetermined cache policy (e.g., FIFO, LRU). This policy is used to determine which videos to prefetch based on the available resources (e.g., device storage, available power), and user defined rules (e.g., the user can specify that at most x videos can be downloaded).

8.2.5 Delivery Module

This module stores the videos on the device so that it can be seamlessly fetched and presented to the user, when the user requests it at a later stage. Once the cache manager downloads the videos, the delivery module stores it at an appropriate location and format, so that when the user requests the videos, it is retrieved from the cache and presented to the user. It is important to note that the prefetching system does not alter the user's viewing behavior in any way as the prefetching is performed as a background function, and the videos should be presented to the user in an uninterrupted manner when requested from their choice of video consumption platform.

8.3 Extending Framework for Prefetching and Edge-Caching Structured and Unstructured Data

In the previous section, we presented a general framework for edge-caching video content regardless of the platform on which it is hosted. In this section, we show how the framework can be modified to prefetch other types of data, not only video content. Specifically, we explore two types of data accesses: i) structured data, which refers to data that is organized and searchable (YouTube and Netflix videos fall under structured data); and ii) unstructured data which is raw data that is not searchable such as data from IoT sensors.

8.3.1 Data Processing Pipeline

Structured Data

For structured data, the data processing pipeline would be as shown in Fig. 8.1, where the key principles regarding learning how the user interacts with the structured data, and retrieving appropriate metadata, remains applicable. For example, if the data that is being prefetched is Spotify audio tracks, then the user's history regarding what they listen to and data regarding the audio tracks would be stored.

Unstructured Data

In addition to the sub-modules shows in Fig. 8.1, a sub-module which organizes the unstructured data first would need to be included. For this sub-module, as an example if temperature sensor data were to be collected, then the data would first need to be processed from the readings given by the sensor, and additional information, such as perhaps the time the data was obtained, would need to be attached. Additional metadata of how this relates with the user would need to be included to form an equivalent ”history” of user interactions with the unstructured data.

8.3.2 Features Generator

Structured Data

The sub-modules for the structured data, even though it is not video data, is not modified. The features related to the user’s behavior and how they access the structured data is extracted, as well as features from the meta-data of the structured data is extracted and stored.

Unstructured Data

While the exact method of extracting features will be dependent on the type of data, both user-specific features about how the user interacts with unstructured data, as well as features related to the metadata that is retrieved during the data process pipeline will be extracted, processed and stored accordingly. For example, if the timestamp was attached during sensor data retrieval, then this would be one of the content-specific features extracted.

8.3.3 Prediction Module

Structured Data

The objective of this module is, to use as input, a featurized representation of the user’s history for training the ML or DL module to ultimately make a prediction on what structured data would need to be prefetched. This sub-module also includes generating the ”candidate set" from which data would need to be predicted if applicable (such as in the case of *Mantis*). The module would be only modified for the type of data to predict (for example, audio tracks rather than specific videos).

Unstructured Data

As the features are used as input into the prediction module, the sub-module for training the ML or DL algorithm will be required. The prediction would be in the same structure as formatted during the data-processing pipeline submodule.

8.3.4 Cache Manager

Structured Data

The cache manager would be responsible for downloading the data from the structured data source (e.g. the Spotify server), and evicting content from the cache as stipulated by the cache policies and device resources.

Unstructured Data

While the sub-modules in the cache manager will be required for unstructured data, a separate module which handles the prediction made from the prediction module (this is in a structured form) and converts into the unstructured data would need to be included. Similarly, a module which converts the structured-data based caching eviction policy to cater to the unstructured data would need to be included. This would then evict the unstructured data based on the policy.

8.3.5 Delivery Module

Structured Data

This module would store the structured data on the storage or user device as was used for *CacheFlix* and *Mantis*. The required content would then be retrieved from the device and presented to the user in a seamless manner. The operation of this module would be the same as was discussed in the prior section.

Unstructured Data

As unstructured data is downloaded by the cache manager, an appropriate database which is designed to handle such data, for example non-relational databases such as MongoDB, would be used to store the data. The retrieval sub-module would then retrieve the data from the databases, and present it to the user in its original state.

CHAPTER 9
CHALLENGES AND NEXT STEPS

In this dissertation, we presented 2 prefetching solutions for YouTube videos (*Mantis*) and for Netflix series videos (*CacheFlix*). In this chapter, we discuss some challenges and future directions for the proposed systems.

9.1 *Mantis:* Time-Shifted Prefetching of YouTube Videos to Reduce Peak-time Cellular Data Usage

1. System design: The system architecture presented earlier in the chapter requires that the user have a YouTube premium subscription (if used in a country that does not offer offline download as a free feature [92]). 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 4.3.5. All the client actions are performed and prefetched by the proxy.

2. Expanding candidate set: In this chapter, 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.

3. WiFi offloading: Although this work 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.

4. 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.

5. 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.

9.2 *CacheFlix*: Toward Effective Prediction of Watch Behavior for Time-Shifted Edge-Caching of Netflix Series Videos

1. Incentives and cost-benefits: The problem of reducing peak-time traffic is addressed in this work; network service providers stand to gain the most from this, but there are also several QoS benefits (e.g. reducing buffering events) for clients. In terms of incentives, there are 2 potential ways to address this: (i) ISPs can waive or reduce billing for usage during off-peak periods; (ii) even if the usage will be eventually passed along to the user, if the ISP does 95th percentile billing, then *CacheFlix's* prefetching will bring the 95th percentile usage of the user to a lower tier. Furthermore, *CacheFlix* can help video providers reduce their network burden; recently, several video providers (Netflix, YouTube, Disney+, Apple) complied with a European Commission request to alleviate network strain during the pandemic, by reducing the default quality of the video delivered to users [125]. The incentives and benefits for video and networks providers, as well as for clients should be further explored.

2. Implementation challenges: A possible method of caching content is to store it on

an external storage connected to a smart-TV or WiFi router. However, due to the recent encryption of HTTP transfers using SSL for Netflix content, implementing a transparent caching proxy and seamlessly presenting the prefetched content to the user is challenging and solutions will need to be explored.

3. User privacy: As the user's viewing history is required for the training of *CacheFlix*, methods to ensure user privacy needs to be developed. This can potentially be performed locally on the end-device itself.

4. Sharing Netflix profiles: It is fairly common for households to share a Netflix account with users creating different profiles on the account. However, as *CacheFlix* attempts to learn a user's preferences, if there are multiple users that share the same user *profile*, this would invariably compromise the ability of the algorithm to learn a user's personal viewing behavior.

5. Prefetching Netflix feature films: With feature films constituting 12% of a user's viewing activity in terms of the bytes fetched, methods of prefetching this content should be developed.

In addition to the next steps specific to *Mantis* and *CacheFlix*, a future work for both solutions include extending to a multiple-user scenario. Currently, *Mantis* and *CacheFlix*, learns a single user's behavior and then prefetches content specific for that user. There are interesting challenges that arise if we were to expand these solutions to cater to multiple users with shared storage (e.g. at a CDN storage node) and bandwidth resources. Furthermore, more computationally efficient techniques (e.g. LSTM with attention) for prediction purposes once the solution is scaled up should be adopted.

CHAPTER 10
CONCLUSIONS

In this dissertation, we considered the strategy of *time-shifted prefetching*. We explored the problem of prefetching content during off-peak periods of the network even when such periods are substantially separated from the actual usage-time, and caching the content on edge nodes closest to the user. With video streaming dominating Internet traffic and increasingly placing a burden on the network, we focus on the prefetching of Netflix and YouTube content- the two most popular video streaming platforms. The objective of this work is to develop a set of data-driven prediction and prefetching systems, using machine-learning and deep-learning techniques, which accurately anticipates the video content the user will consume, and caches it on edge nodes during off-peak periods to reduce peak-time usage. To this end, we collected 3 datasets on YouTube and Netflix usage, and presented key insights, and developed associated time-shifted prefetching systems.

In chapter 3, we collected and analyzed a real-life dataset of YouTube watch history from 243 users comprised of over 1.8 million videos spanning over a 1.5-year period. Using this data, we provided a number of insights and associated implications by answering 5 questions regarding a user's interaction with YouTube: i) How often do users watch the same video again? ii) Is a user's watch behavior predictable? iii) What role does YouTube's recommendation engine play in influencing users? iv) How dynamic are user's video preferences? and v) What are user's typical YouTube data consumption patterns? These questions pertain to certain representative problems and our associated analysis provided key insights related to those problems. Furthermore, we considered a few representative problems in the domains of networking and communications, and provided substantiated directions for solutions based on insights from the dataset.

In chapter 4, we address the problem of high peak cellular traffic, through a time-shifted prefetching strategy for YouTube content. Equipped with the dataset and its associated findings from chapter 3, we propose *Mantis*, a machine learning algorithm for prefetching YouTube videos that is trained on a user's YouTube watch history and predicts the user's likely video watch behavior over the next 24 hours. *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%.

In chapter 5, we collected and analyzed a real-world Netflix dataset which consisted of 1-year viewing activity from 1060 users amounting to over 1.7 million watched episodes and movies. Equipped with this data, we derived key insights pertaining to the user's watch patterns, watch-session length, user preferences, predictability and series continuation tendencies. We also implemented and evaluated prediction models that is used to predict if a user will continue watching a series or not. We found that we were able to achieve an overall accuracy of 68% with the Random Forest classifier.

In chapter 6, we explored the time-shifted edge caching of Netflix series videos on edge nodes closest to the user, with the goal of reducing overall peak-time Netflix traffic. We proposed a deep-learning prediction algorithm, *CacheFlix*, using the dataset and insights from chapter 5, which makes a prediction of how much content to cache based on the user's past viewing pattern and their content-specific interactions. We presented results for the prediction accuracy and caching efficiency for our solution across 4 different cache eviction policies, and show that we are able to shift 70% of their Netflix traffic to off-peak periods.

In this chapter 7, we collected and analyzed a real-world YouTube and Netflix dataset, which consisted of at least 1-year viewing activity from 377 users amounting to over 4.3 million watched episodes and movies. Using this dataset, we derived key insights for individual user behavior related to their watch patterns, amount of content consumption, viewing interests, and predictability of their future viewing for both Netflix content (series and movies) and YouTube videos. The results and analysis provided attempt to serve as a basis for tackling several problems in the general area of Internet protocols, algorithms and systems.

In chapter 8, we developed an integrated solution of *Mantis* and *CacheFlix*. We showed how *Mantis* can be enhanced with a features representation of the user's Netflix viewing behavior, and how *CacheFlix* can be enhanced with a features representation of the user's YouTube viewing behavior. We saw that were able to achieve a slight improvement with the enhanced algorithms across all activity levels. We also presented a general framework for prefetching video content during off-peak periods. This framework consisted of 5 modules, namely: data processing pipeline, features generator, prediction module, cache manager and delivery module.

REFERENCES

[1] (2016). “Cisco visual networking index: Forecast and methodology 2016-2021.”

[2] (2017). “Cisco vni complete forecast highlights.”

[3] (2019). “4 ways service providers can improve capacity forecasts.”

[4] 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.

[5] 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.

[6] 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, ISBN: 978-3-540-30471-5.

[7] 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, San Jose, California: ACM, 2017, 1:1–1:6, ISBN: 978-1-4503-5527-8.

[8] 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)*, 2014, pp. 1–8.

[9] (2020). “The global internet phenomena report covid-19 spotlight.”

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

[11] (2020). “On the shoulders of giants: Recent changes in internet traffic.”

[12] (2012). “Sandvine global internet phenomena report 2012.”

[13] 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.

[14] M. Zink, K. Suh, Y. Gu, and J. Kurose, “Characteristics of youtube network traffic at a campus network – measurements, models, and implications,” *Computer Networks*, vol. 53, no. 4, pp. 501 –514, 2009, Content Distribution Infrastructures for Community Networks.

[15] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous, “Network characteristics of video streaming traffic,” in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT ’11, Tokyo, Japan: ACM, 2011, 25:1–25:12, ISBN: 978-1-4503-1041-3.

[16] V. K. Adhikari, S. Jain, Y. Chen, and Z. Zhang, “Vivisecting youtube: An active measurement study,” in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 2521–2525.

[17] Y. Zhou, L. Chen, C. Yang, and D. M. Chiu, “Video popularity dynamics and its implication for replication,” *IEEE Transactions on Multimedia*, vol. 17, no. 8, pp. 1273–1285, 2015.

[18] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, “Analyzing the video popularity characteristics of large-scale user generated content systems,” *IEEE/ACM Transactions on Networking*, vol. 17, no. 5, pp. 1357–1370, 2009.

[19] M. J. Halvey and M. T. Keane, “Exploring social dynamics in online media sharing,” in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW ’07, Banff, Alberta, Canada: ACM, 2007, pp. 1273–1274, ISBN: 978-1-59593-654-7.

[20] X. Cheng, C. Dale, and J. Liu, “Statistics and social network of youtube videos,” in *2008 16th Interntional Workshop on Quality of Service*, 2008, pp. 229–238.

[21] T. Ibrahim and C.-Z. Xu, “Neural nets based predictive prefetching to tolerate www latency,” in *Proceedings 20th IEEE International Conference on Distributed Computing Systems*, 2000, pp. 636–643.

[22] S.-w. Liao, T.-H. Hung, D. Nguyen, C. Chou, C. Tu, and H. Zhou, “Machine learning-based prefetch optimization for data center applications,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–10.

[23] V. A. Siris and D. Kalyvas, “Enhancing mobile data offloading with mobility prediction and prefetching,” in *Proceedings of the Seventh ACM International Workshop on Mobility in the Evolving Internet Architecture*, ser. MobiArch ’12, Istanbul, Turkey: Association for Computing Machinery, 2012, 17–22, ISBN: 9781450315265.

[24] M. Z. Shafiq, A. X. Liu, and A. R. Khakpour, “Revisiting caching in content delivery networks,” in *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, Austin, Texas, USA: Association for Computing Machinery, 2014, 567–568, ISBN: 9781450327893.

[25] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, “Cachier: Edge-caching for recognition applications,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 276–286.

[26] K. Guo, C. Yang, and T. Liu, “Caching in base station with recommendation via q-learning,” in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, 2017, pp. 1–6.

[27] M. Mehrabi, D. You, V. Latzko, H. Salah, M. Reisslein, and F. H. P. Fitzek, “Device-enhanced mec: Multi-access edge computing (mec) aided by end device computation and caching: A survey,” *IEEE Access*, vol. 7, pp. 166 079–166 108, 2019.

[28] A. Gouta, D. Hausheer, A.-M. 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*, 2015, pp. 188–197.

[29] X. Wang, T. Kwon, Y. Choi, H. Wang, and J. Liu, “Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users,” *IEEE Wireless Communications*, vol. 20, no. 3, pp. 72–79, 2013.

[30] S. Bayhan, S. Maghsudi, and A. Zubow, “Edgedash: Exploiting network-assisted adaptive video streaming for edge caching,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1732–1745, 2021.

[31] M. Ma, Z. Wang, K. Su, and L. Sun, “Understanding the power of smartrouter-based peer cdn for video streaming,” in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, 2016, pp. 1–9.

[32] A. Mahzari, A. Taghavi Nasrabadi, A. Samiei, and R. Prakash, “Fov-aware edge caching for adaptive 360° video streaming,” in *Proceedings of the 26th ACM International Conference on Multimedia*, ser. MM ’18, Seoul, Republic of Korea: Association for Computing Machinery, 2018, 173–181, ISBN: 9781450356657.

[33] K. Mokhtarian and H.-A. Jacobsen, “Flexible caching algorithms for video content distribution networks,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1062–1075, 2017.

[34] Z. Jiang and L. Kleinrock, “Web prefetching in a mobile environment,” *IEEE Personal Communications*, vol. 5, no. 5, pp. 25–34, 1998.

[35] Z. Zhao, L. Guardalben, M. Karimzadeh, J. Silva, T. Braun, and S. Sargento, “Mobility prediction-assisted over-the-top edge prefetching for hierarchical vanets,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1786–1801, 2018.

[36] J. Cobb and H. ElAarag, “Web proxy cache replacement scheme based on back-propagation neural network,” *Journal of Systems and Software*, vol. 81, no. 9, pp. 1539–1558, 2008, Gauging the progress of Software Architecture research: three selected papers from Working IEEE/IFIP Conference on Software Architecture (WICSA) 200.

[37] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 28–35, 2018.

[38] (2019). "2019 mobile internet phenomena."

[39] (2019). "2019 global internet phenomena."

[40] X. Cheng, J. Liu, and C. Dale, "Understanding the characteristics of internet short video sharing: A youtube-based measurement study," *IEEE Transactions on Multimedia*, vol. 15, no. 5, pp. 1184–1194, 2013.

[41] V. K. Adhikari, Yang Guo, Fang Hao, M. Varvello, V. Hilt, M. Steiner, and Z. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *2012 Proceedings IEEE INFOCOM*, 2012, pp. 1620–1628.

[42] L. Huang, B. Ding, Y. Xu, and Y. Zhou, "Analysis of user behavior in a large-scale vod system," in *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV'17, Taipei, Taiwan: Association for Computing Machinery, 2017, 49–54, ISBN: 9781450350037.

[43] Y. Yuan, X. Wang, and G. Bin, "Analysis of user behavior in a large-scale internet video-on-demand(vod) system," in *Proceedings of the 5th International Conference on Multimedia and Image Processing*, ser. ICMIP '20, Nanjing, China: Association for Computing Machinery, 2020, 153–158, ISBN: 9781450376648.

[44] 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.

[45] 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, McLean, Virginia, USA: ACM, 2002, pp. 632–634, ISBN: 1-58113-492-4.

[46] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5g wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.

[47] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, "A data mining algorithm for generalized web prefetching," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1155–1169, 2003.

[48] 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)*, 2017, pp. 1–6.

[49] 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, ISBN: 978-3-642-45065-5.

[50] X. Wang, T. Kwon, Y. Choi, H. Wang, and J. Liu, "Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 72–79, 2013.

[51] 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).

[52] 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*, 2015, pp. 188–197.

[53] C. Ge, N. Wang, S. Skillman, G. Foster, and Y. Cao, "Qoe-driven dash video caching and adaptation at 5g mobile edge," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, ser. ACM-ICN '16, Kyoto, Japan: Association for Computing Machinery, 2016, 237–242, ISBN: 9781450344678.

[54] C. Jayasundara, M. Zukerman, T. A. Nirmalathas, E. Wong, and C. Ranaweera, "Improving scalability of vod systems by optimal exploitation of storage and multicast," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 3, pp. 489–503, 2014.

[55] 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, San Jose, California: ACM, 2017, 17:1–17:13, ISBN: 978-1-4503-5087-7.

[56] P. Baumann and S. Santini, "Every byte counts: Selective prefetching for mobile applications," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 2, 6:1–6:29, Jun. 2017.

[57] (2020). "Netflix open connect."

[58] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix cdn," *SIGCOMM Comput. Commun. Rev.*, vol. 48, no. 1, 28–34, Apr. 2018.

[59] W. Jiang, S. Ioannidis, L. Massoulié, and F. Picconi, "Orchestrating massively distributed cdns," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, Nice, France: Association for Computing Machinery, 2012, 133–144, ISBN: 9781450317757.

[60] M. I. A. Zahed, I. Ahmad, D. Habibi, Q. V. Phung, L. Zhang, and A. Mathew, "Security aware content caching for next generation communication networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[61] 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. arXiv: `1703.03530`.

[62] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5G Wireless Network Slicing for eMBB, URLLC, and mMTC: A Communication-Theoretic View," *IEEE Access*, vol. 6, pp. 55 765–55 779, 2018.

[63] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[64] H. Elazhary, "Internet of things (iot), mobile cloud, cloudlet, mobile iot, iot cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions," *Journal of Network and Computer Applications*, vol. 128, pp. 105–140, 2019.

[65] H.-G. Song, S. H. Chae, W.-Y. Shin, and S.-W. Jeon, "Predictive caching via learning temporal distribution of content requests," *IEEE Communications Letters*, vol. 23, no. 12, pp. 2335–2339, 2019.

[66] S. M. S. Tanzil, W. Hoiles, and V. Krishnamurthy, "Adaptive scheme for caching youtube content in a cellular network: Machine learning approach," *IEEE Access*, vol. 5, pp. 5870–5881, 2017.

[67] Y. Liu, Z. Ma, Z. Yan, Z. Wang, X. Liu, and J. Ma, "Privacy-preserving federated k-means for proactive caching in next generation cellular networks," *Information Sciences*, vol. 521, pp. 14–31, 2020.

[68] M. Chen, M. Mozaffari, W. Saad, C. Yin, M. Debbah, and C. S. Hong, "Caching in the sky: Proactive deployment of cache-enabled unmanned aerial vehicles for optimized quality-of-experience," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1046–1061, 2017.

[69] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[70] R. Wang, R. Li, P. Wang, and E. Liu, "Analysis and optimization of caching in fog radio access networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8279–8283, 2019.

[71] Y. Fadlallah, A. M. Tulino, D. Barone, G. Vettigli, J. Llorca, and J.-M. Gorce, "Coding for caching in 5g networks," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 106–113, 2017.

[72] C. Zhang, H. Pang, J. Liu, S. Tang, R. Zhang, D. Wang, and L. Sun, "Toward edge-assisted video content intelligent caching with long short-term memory learning," *IEEE Access*, vol. 7, pp. 152 832–152 846, 2019.

[73] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "Deepcache: A deep learning based framework for content caching," ser. NetAI'18, Budapest, Hungary: Association for Computing Machinery, 2018, 48–53, ISBN: 9781450359115.

[74] (2019). "Youtube for press."

[75] (2019). "The latest youtube stats on audience demographics: Who's tuning in."

[76] (2016). "2016 global internet phenomena."

[77] (2016). "Cisco visual networking index: Global mobile data traffic forecast update, 2017-2022."

[78] (2019). "Amazon mechanical turk."

[79] 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, Denver, Colorado, USA: ACM, 2017, pp. 1092–1099, ISBN: 978-1-4503-4656-6.

[80] J. Erman, A. Gerber, M. T. Hajiaghayi, D. Pei, and O. Spatscheck, "Network-aware forward caching," in *Proceedings of the 18th International Conference on World Wide Web*, ser. WWW '09, Madrid, Spain: ACM, 2009, pp. 291–300, ISBN: 978-1-60558-487-4.

[81] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Web caching on smartphones: Ideal vs. reality," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '12, Low Wood Bay, Lake District, UK: ACM, 2012, pp. 127–140, ISBN: 978-1-4503-1301-8.

[82] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.

[83] R. Zhou, S. Khemmarat, and L. Gao, "The impact of youtube recommendation system on video views," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10, Melbourne, Australia: ACM, 2010, pp. 404–410, ISBN: 978-1-4503-0483-2.

[84] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[85] (2019). "Nielsen social report 2019."

[86] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Dynamic provisioning of multi-tier internet applications," in *Second International Conference on Autonomic Computing (ICAC'05)*, 2005, pp. 217–228.

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

[88] (2018). "Youtube's ai is the puppet master over most of what you watch."

[89] H. Abdi and L. J. Williams, "Principal component analysis," *WIREs Comput. Stat.*, vol. 2, no. 4, pp. 433–459, Jul. 2010.

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

[91] 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.

[92] (2019). "Watch videos offline on mobile in select countries."

[93] (2018). "The state of lte."

[94] (2019). "Macrodroid."

[95] (2019). "Workflow."

[96] (2020). "User behavior analytics."

[97] (2019). "Tmdb api."

[98] (2020). "Netflix help center."

[99] K.-I. Goh and A.-L. Barabási, "Burstiness and memory in complex systems," *EPL (Europhysics Letters)*, vol. 81, no. 4, p. 48 002, 2008.

[100] G. Bonaccorso, *Machine Learning Algorithms: A Reference Guide to Popular Algorithms for Data Science and Machine Learning*. Packt Publishing, 2017, ISBN: 1785889621.

[101] (2017). "America has an internet problem — but a radical change could solve it."

[102] C. Aggarwal, J. L. Wolf, and P. S. Yu, "Caching on the world wide web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, pp. 94–107, 1999.

[103] G. Rossini and D. Rossi, "A dive into the caching performance of content centric networking," in *2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2012, pp. 105–109.

[104] A. Ghosh, R. Jana, V. Ramaswami, J. Rowland, and N. K. Shankaranarayanan, "Modeling and characterization of large-scale wi-fi traffic in public hot-spots," in *2011 Proceedings IEEE INFOCOM*, 2011, pp. 2921–2929.

[105] K. Fukuda, K. Cho, and H. Esaki, "The impact of residential broadband traffic on japanese isp backbones," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 1, pp. 15–22, Jan. 2005.

[106] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè, "Broadband internet performance: A view from the gateway," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, Toronto, Ontario, Canada: ACM, 2011, pp. 134–145, ISBN: 978-1-4503-0797-0.

[107] (2020). "Can netflix's surge last?"

[108] (2019). "Netflix users stream 164 million hours per day."

[109] (2019). "Netflix's cindy holland says subscribers watch an average of two hours a day."

[110] L. Huang, B. Ding, A. Wang, Y. Xu, Y. Zhou, and X. Li, "User behavior analysis and video popularity prediction on a large-scale vod system," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 14, no. 3s, Jun. 2018.

[111] (2020). "Netflix open connect."

[112] A. Agresti, *Categorical Data Analysis*, ser. A Wiley-Interscience publication. New York [u.a.]: Wiley, 2002, XV, 558 S.

[113] Y. Luo, M. Wang, H. Zhou, Q. Yao, W.-W. Tu, Y. Chen, W. Dai, and Q. Yang, "Autocross: Automatic feature crossing for tabular data in real-world applications," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, ser. KDD '19, Anchorage, AK, USA: Association for Computing Machinery, 2019, 1936–1945, ISBN: 9781450362016.

[114] F. Gers, "Learning to forget: Continual prediction with lstm," *IET Conference Proceedings*, 850–855(5),

[115] (2012). "Top five wi-fi routers with built-in network storage."

[116] S. Lall, M. Agarwal, and R. Sivakumar, "A youtube dataset with user-level usage data: Baseline characteristics and key insights," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–7.

[117] (2022). “The global internet phenomena report january 2022.”

[118] F. Bentley and J. Murray, “Understanding video rewatching experiences,” in *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video*, ser. TVX ’16, Chicago, Illinois, USA: Association for Computing Machinery, 2016, 69–75, ISBN: 9781450340670.

[119] O. Budzinski, S. Gänßle, and N. Lindstädt-Dreusicke, “The battle of youtube, tv and netflix: An empirical analysis of competition in audio-visual media markets,” Ilmenau, Ilmenau Economics Discussion Papers 137, 2020.

[120] M. Zeng, T.-H. Lin, M. Chen, H. Yan, J. Huang, J. Wu, and Y. Li, “Temporal-spatial mobile application usage understanding and popularity prediction for edge caching,” *IEEE Wireless Communications*, vol. 25, no. 3, pp. 36–42, 2018.

[121] S. Mehrizi, T. X. Vu, S. Chatzinotas, and B. Ottersten, “Trend-aware proactive caching via tensor train decomposition: A bayesian viewpoint,” *IEEE Open Journal of the Communications Society*, vol. 2, pp. 975–989, 2021.

[122] B. Stegner, “How much data does youtube actually use? explained,” Tech. Rep., 2021.

[123] (2020). “Deep dive into netflix’s recommender system.”

[124] C. A. Gomez-Uribe and N. Hunt, “The netflix recommender system: Algorithms, business value, and innovation,” *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, 2016.

[125] (2020). “Amazon and apple are reducing streaming quality to lessen broadband strain in europe.”