CHARACTERIZING AND OPTIMIZING INTERNET VIDEO STREAMING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yun Seong Nam

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Sanjay G. Rao, Chair

School of Electrical and Computer Engineering, Purdue University

Dr. Ramesh Govindan

Viterbi School of Engineering, University of Southern California

Dr. Bruno Ribeiro

Department of Computer Science, Purdue University

Dr. Felix Xiaozhu Lin

School of Electrical and Computer Engineering, Purdue University

Approved by:

Dr. Dimitrios Peroulis

Head of the School Graduate Program, Purdue University

For my family.

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Sanjay Rao, who has been a great mentor and a sincere friend. I am indebted to Prof. Rao for the helpful guidance he gave me and the opportunities he created to help me grow my research skills. I have learned a lot from Prof. Rao, not only the way to conduct first-class research, but also the way to pursue success and to handle failure.

I am grateful to my Committee members, Prof. Ramesh Govindan, Prof. Felix Xiaozhu Lin, and Prof. Bruno Ribeiro, for the valuable advice and feedback they gave on my research and dissertation.

I am thankful to my research collaborators, Prof. Ramesh Govindan, Prof. Bruno Ribeiro, Dr. Zahaib Akhtar, Ehab, Dr. Ethan Katz-Bassett, Jibin Zhan, Dr. Hui Zhang and Dr. Jessica Chen for their hard work and contributions to my research.

I am fortunate to have worked with my colleagues at Internet Systems Lab: Ashiwan, Chuan, Ehab, Harsh, Russ, Shankar, Sruthi, Yiyang and Zaiwei for those enlightening discussions and kind support.

Last but not least, I would like to thank School of Electrical and Computer Engineering, Purdue University for funding me as a teaching assistant; and I would like to thank NSF for funding my research.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Nam, Yun Seong Ph.D., Purdue University, May 2020. Characterizing and Optimizing Internet Video Streaming. Major Professor: Sanjay G. Rao.

Internet video comprises a major portion of Internet traffic today. While core and access network capacities continue to grow, optimizing Internet video delivery will remain a challenge, as new forms of video and technology keep emerging, and content publishers continue to seek higher Quality of Experience(QoE) of users due to its correlations with user engagement and revenue. The goals of this thesis are to create a deeper understanding of the Internet video ecosystem and to propose novel methodologies to improve QoE of Internet video delivery.

In this thesis, we make the following contributions. First, we create a deeper understanding of video management plane by characterizing it, at scale, along its key dimensions based on more than 100 content publishers data spanning 27 months, and we propose new metrics to measure complexity of video management plane. Next, in order to enhance video control plane, we propose Oboe, a system that improves the dynamic range of Adaptive Bitrate(ABR) algorithms by automatically tuning ABR behaviors to the current network state of a client connection to improve QoE of a wide range of users. Through testbed experiments, we show Oboe significantly improves performance of several different ABR algorithms. Finally, given that performance of ABRs critically depends on throughput prediction accuracy, we propose a new throughput prediction approach, called Xatu, to address challenges in existing prediction methods used by ABRs. Xatu, a learning based throughput prediction framework, uses richer information (*e.g.,* ISP or chunk size) without apriori partitioning data, and we show that Xatu reduces the prediction error by more than 23% relative to state-of-the-art throughput prediction.

1. INTRODUCTION

Video dominates today's Internet and will play a larger role going forward. The average adult in the United States watches over an hour of Internet video a day [1], and this consumption accounts for over half the traffic on residential networks being video [2]. The volume of video traffic is projected to double by 2020, when a million minutes of video traffic will cross the Internet every second [3]. Also delivering high *quality of experience* (QoE) is critical since it correlates with user engagement and revenue [4–6].

Figure 1.1 shows an overview of the Internet video delivery ecosystem. It has evolved to have several *entities*. *Content publishers* serve their videos from *content delivery networks*(CDNs), which deploy servers around the world and peer with many ISP core and access networks, enabling users to access nearby servers and to consume videos using various *playback devices*. Since videos must be delivered over various networks that are highly variable, a conventional approach to delivering videos today is splitting a video into chunks, encoding chunks with multiple bitrates, and allowing

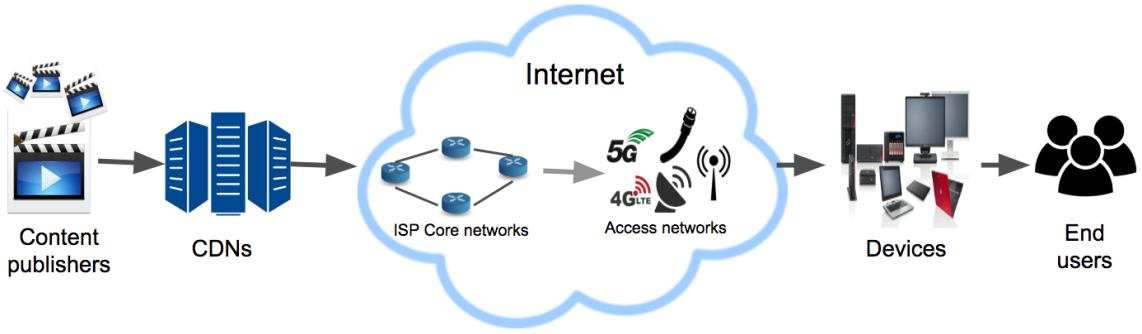


Fig. 1.1.: The Internet video ecosystem.

clients to pick bitrates for each chunk based on client's network conditions through *adaptive bitrate* (ABR) algorithms.

While core and access network capacities continue to grow, optimizing Internet video delivery will remain a challenge for two reasons. First, new forms of video, technology, and devices continue to emerge—such as an interactive video content [7], 4K video [8], and virtual reality devices [9]—which will involve more stringent requirements. For instance, a user action (e.g., a change of orientation) in an interactive video can require more tight deadline to ensure responsiveness to user interactivity. Second, both users and content publishers continue to seek higher QoE. Even though users consume videos with various devices, they want a TV-like, uninterrupted experience and leave if QoE does not meet their expectations. Content publishers also want to provide higher QoE since this is directly related to their revenue [5].

To address challenges in a systematic way, we decompose the Internet video delivery ecosystem into a series of three planes based on its operations: *video management plane* for preparing videos(transcoding/encoding and packaging videos with various qualities and streaming protocols) and distributing them to content delivery networks(CDNs), *video control plane* for selecting a CDN and video quality to a user, and *video data plane* for transporting video chunks through network.

This thesis focuses on video management and control planes for two reasons. First, understanding video management plane is necessary to help learning its implications for complexity, efficiency, requirements and QoE impact on video delivery. While there have been some industrial efforts for understanding video management plane [10–14], we are just scratching the surface in this area, and this plane is relatively unexplored despite of its impacts. Second, video control plane is one of the key components to improving QoE of users and not mature to ensure good QoE of all users in a wide range of network conditions.

The goals of this thesis are to create a better understanding of video management plane and to enhance video control plane in order to improve QoE of users. To this end, we make the following contributions:

- Characterizing video management plane based on more than 100 content publishers data spanning 27 months in order to understand how it evolved over time and across publishers, proposing new metrics to quantify the impact of diversity on video management plane operations, and unveiling video management inefficiency in today's practice.

- Proposing Oboe, a system that enhances video control plane by adapting ABR behavior to user network conditions in order to improve QoE of users.

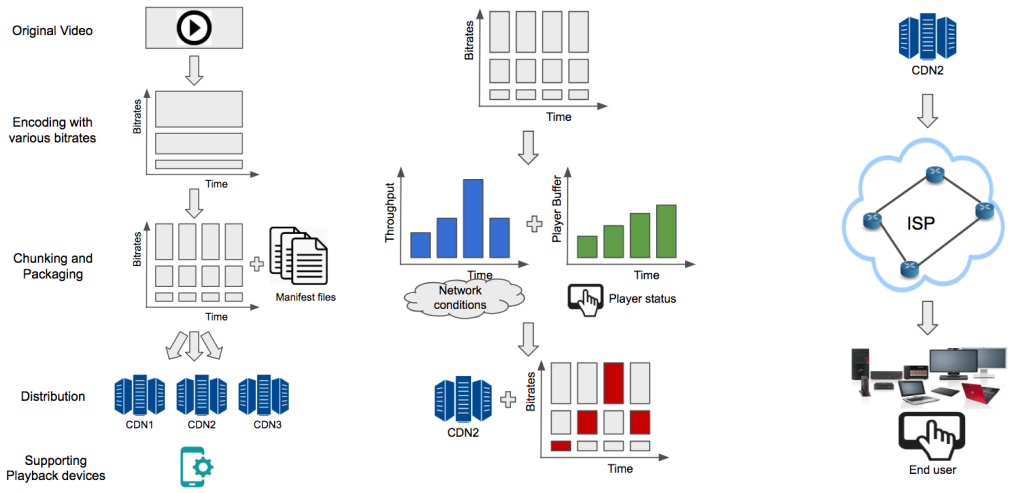
- Developing Xatu, a leaning-based throughput prediction framework, in order to improve video control plane since performance of ABRs critically depends on accuracy of throughput prediction.

1.1 Overview of video delivery planes

The Internet video delivery ecosystem consists of three planes based on its operations: video management, video control, and video data plane.

Video management plane. Figure 1.2(a) shows video management plane(§2.2) which performs two primary functions. The first function prepares video content for delivery to users. Preparation involves (i) splitting the video into chunks, (ii) encoding each chunk at one or more bitrates and encapsulating chunks using *HTTP chunk-based streaming protocols*(encoding and packaging) and (iii) distributing video to CDNs. The second function is developing and maintaining playback software for the wide range of user devices.

Video control/data plane. Figure 1.2(b) illustrates video control plane(§3.2.1). The key operation of video control plane is selecting at what bitrate level to fetch a chunk based on conditions such as the amount of video the client has buffered and the recent throughput achieved by the client through ABR algorithms in order to improve QoE of users. It also involves choosing which CDN to direct a user to.



(a) Video management plane (b) Video control plane (c) Video data plane

Fig. 1.2.: Video management, control and data planes

Finally video data plane relates to transporting each chunk to the end user as shown in figure 1.2(c).

Those series of operations and decisions impact QoE metrics such as a video start-up latency, average bitrate in a video session, average bitrate changes(smoothness) during the video session, or rebuffering ratio (the fraction of time video is paused because the playback buffer has drained).

1.2 Motivation

This thesis focuses on video management and control plane since video data plane is relatively well studied. An exploration of important considerations and associated challenges follows.

First, understanding video management is key to learn about operational complexity, efficiency, and requirements of Internet video delivery. The Video management plane comprises a few key dimensions such as streaming protocols, playback devices and CDNs, and the state of practice for the video management plane is in the curse

of choice. There are numerous types of playback devices, streaming protocols, and CDNs, and the choice of each dimensions (*e.g.,* what devices/stream protocols to support and how many CDNs to use) significantly impacts user reachability to users, operational complexity, computation and storage requirements, application performance, and QoE of users. Even though understanding video management plane is important to learning about its impacts, it has been relatively unexplored compared to the other planes. Even though there are a couple of prior industry reports that explored related aspects of video management plane [10–14], these studies have several shortcomings such as small scale of data, lack of evolutional trend, and little understanding about content publisher characteristics.

Second, although video control plane has received much attention (e.g., [15–22]), ABR algorithm design, which is one of key components to improving QoE in video control plane, remains an active research area because content providers continue to be interested in *improving the performance of video delivery.* Current ABR algorithms perform well on average, but some users can experience poor delivery performance, as measured by QoE metrics. These users suffer because ABR algorithms *have limited dynamic range*: they do not perform uniformly well across the range of network conditions seen in practice because their parameters are sensitive to throughput variability (§3.2).

Last, even though the performance of ABR algorithms for video streaming critically depends on accurately predicting application-perceived throughput [23, 24], existing prediction approaches for ABRs have limitations that impact their prediction accuracy. Based on data analysis from real-world video streaming sessions, we show two key limitations: (i) their reliance on apriori clustering of video sessions (e.g., based on ISP and CDN) prevents them from learning from related clusters, and (ii) their inability to effectively incorporate important, yet rarely considered, factors such as the Time to First Byte (TTFB) and the chunk size in the prediction.

1.3 Contributions

The main goals of this thesis are creating a deeper understating of video management plane and advancing video control plane in order to improve QoE of Internet video streaming across various conditions and users. To achieve these goals, we have been working with one of industrial leading companies [25], and this thesis makes the following three contributions.

Creating a deeper understanding of video management plane. We shed light on video management plane by characterizing it, at scale, along three key dimensions (§2.4): streaming protocols, playback devices and platforms and CDNs based on *more than 100 content publishers data spanning 27 months*. We provide a deeper understanding of how each dimension/how many instances of each dimension have evolved over time and across video publishers. We also take an initial step towards proposing new metrics to measure impacts of diversity of three dimensions on *complexity* of video management plane operations such as software maintenance, failure triaging, and packaging overheads (§2.5). Additionally, we demonstrate that today's management plane practices may not be well suited for content *syndication* (§2.6), in which *syndicators* license and redistribute content from a *content owner*. We found significant diversity with respect to video packaging, playback device support and CDN use, and current trends suggest increasing diversity in some of these dimensions. Also we found that this diversity adds complexity to management, and we showed that the complexity of many management tasks is sub-linearly correlated with the number of hours a publisher's content is viewed. Moreover, today each publisher runs an independent management plane, and this practice can lead to sub-optimal outcomes for syndicated content, such as redundancies in CDN storage and loss of control for content owners over delivery quality.

Enhancing video control plane by increasing dynamic range of ABRs. We developed Oboe, a system that improves the dynamic range of ABR algorithms by automatically tuning ABR behavior to the current *network state* of a client con-

nection, specifically to throughput and throughput variability to improve QoE of a wide range of users. Oboe *pre-computes*, for a given ABR algorithm, the best possible parameters for different network conditions, then *dynamically adapts* the parameters at run-time for the current network conditions. We demonstrate several aspects of Oboe performance through real testbed experiments and trace driven simulations, and show the practical viability of this architecture with results from a pilot deployment. We integrated Oboe with several existing ABR algorithms [16, 18] such as MPC, BOLA and HYB and showed significant improvement in several QoE metrics by 7.2% to 38%. Oboe also betters a recently proposed reinforcement learning-based ABR [17], Pensieve in part because it is able to better specialize ABR behavior across different network states.

Improving video control plane by a better throughput framework. We propose a new throughput prediction approach, Xatu, to address the challenges in existing prediction methods. Xatu jointly learns a neural network sequence model with an interpretable automatic session clustering method. Xatu learns clustering rules across all sessions it deems relevant, and it models sequences with multiple chunk-dependent features (e.g., TTFB, size) rather than just throughput. We evaluate Xatu over datasets of real video sessions along with trace-driven experiments. Our results show that Xatu significantly improves throughput prediction accuracy by 23.8% relative to CS2P [23] (the state-of-the-art predictor). We also show, through an example of hierarchical CDNs, that Xatu is extensible, and can achieve better accuracies if additional information was exposed to video streaming algorithms. To demonstrate the potential impact of Xatu when integrated ABRs, we combine Xatu with MPC [16] (a well-known ABR algorithm) and show QoE improvement based on a heuristic approach, while a more thorough solution is left for a future work.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 presents a deeper understanding of video management plane that is based on more than 100 video publishers data over 27 months. Chapter 3 presents Oboe, which can improve dynamic range of ABR algorithms to improve QoE of a wide range of users in different network conditions. Chapter 4 proposes Xatu, a learning based throughput prediction approach, which improves network throughput prediction accuracy of ABRs. Finally, Chapter 5 summarizes the contributions of this thesis and presents future directions.

2. UNDERSTANDING VIDEO MANAGEMENT PLANES

2.1 Introduction

Video forms the overwhelming majority of Internet traffic [4,26–28]. The deluge in video traffic is due both to the popularity of large services like YouTube, Netflix, and Facebook [29–31] and to the significant increase in Internet video services provided by publishers who traditionally produced content for broadcast television [32].

An Internet video publisher must (i) split the video into chunks, encode each chunk at one or more bitrates, and encapsulate chunks using a *streaming protocol*; (ii) develop and maintain playback software for the wide range of user devices; and (iii) distribute video to Content Delivery Networks (CDNs). We refer to these tasks as *video management plane* operations (§2.2), as distinct from *control plane* operations that involve selecting which CDN to direct a user to and what bitrate to choose for each chunk, and *data plane* operations that involves transporting each chunk to the end user. Whereas the data and control planes have received much attention (e.g., [15–22]), video management plane decisions have been relatively unexplored, even though they impact how many users and devices a publisher can reach, the computation and storage requirements of content publishers, the complexity of troubleshooting, application performance, and the effort needed to incorporate control plane innovations such as new bitrate selection algorithms [15–18, 33].

This thesis characterizes aspects of video management planes for more than 100 content publishers (§2.3), including 7 of the top 10 subscription video publishers [34], as well as prominent sports and news broadcasters and on-demand video publishers. Our dataset comes from a CDN broker [35] and contains metadata for over 100 billion video views, including metadata about the client (device and application used), video (publisher, URL, playback duration), and delivery (CDN, performance metrics). The

aggregate daily view-hours across all our publishers are comparable to reported values for Facebook and Netflix.

Two aspects make our data unique relative to published industry reports [10,12,34] (§2.7). Our data spans 27 months, enabling analysis of management plane practices over time. It also lets us assess *view-hours* (the total number of hours content is viewed) and *views* (the total number of video sessions) for any slice of the data (*e.g.,* how many view-hours or views can be attributed to mobile apps).

Contributions. First, we characterize video management planes along three key dimensions (§2.4): streaming protocols, playback devices and platforms, and CDNs. For each dimension, we characterize (i) how each instance (*e.g.,* a specific streaming protocol, or a specific platform category such as the set-top box) has evolved across publishers, and over time; and (ii) the number of instances of each dimension used by a given publisher and its evolution, and how this correlates with the publisher's view-hours.

Several common themes run across our analysis of these three dimensions. We find that, despite significant changes over the 27 month period, *no single dominant alternative* has emerged along any dimension. Among streaming protocols, HLS and DASH have significant usage, while view-hours are almost evenly distributed across 3 CDNs and across 3 platforms (browser, mobile, and set-top). Moreover, more than 90% of view-hours can be attributed to publishers who support more than 1 protocol. The same is true of publishers who use more than 1 CDN, and publishers who support more than 1 platform. Publishers with more view-hours tend to support more choices of protocols, platforms, and CDNs.

Beyond these, our analysis uncovers some new findings: set-top boxes dominate by view-hours; almost 80% of view-hours are from publishers that support 4-5 CDNs; and a significant fraction of publishers who use multiple CDNs segregate live and on-demand traffic by CDN. Our analysis also adds color to known findings. For example, DASH usage has increased, but this growth is being driven entirely by large

publishers. Moreover, while mobile app views have indeed increased, view-hours have not proportionally increased because view durations on mobile tend to be short.

Second, we take an initial step towards quantifying the impact of three dimensions of diversity on the *complexity* of management plane operations such as software maintenance, failure triaging, and packaging overheads (§2.5). We find that metrics that approximate the complexity of these operations for a publisher are sub-linearly correlated with the publisher's view-hours. For example, a publisher with 10× as many view-hours as another will tend to maintain 1.8× as many versions of its video playback software.

Third, we demonstrate that today's management plane practices may not be well suited for content *syndication* (§2.6), in which *syndicators* license and redistribute content from a *content owner*. Syndication is prevalent in Internet video, yet syndicators run video management planes that are independent from those of content owners. As a result, we find cases where, for the *same* content, owners' clients observe significantly different delivery performance than syndicators' clients. We also find that more integrated management planes between owners and syndicators can reduce CDN origin storage requirements for a popular video series by 2×.

Our results further our understanding of video management planes and open the door for research into new syndication models, complexity metrics, and approaches to cope with diversity and reduce management complexity.

2.2 The Video Management Plane

A video *publisher* makes available online live and/or stored video content. Video content is encoded in different *formats*, delivered by one or more *CDNs*, and delivered to *playback software* on user devices. The video control and data planes together achieve *chunked adaptive streaming*: the data plane streams video *chunks* over HTTP, and the control plane adaptively determines, based on network conditions, which *bitrate* a chunk is downloaded at, and from which CDN.

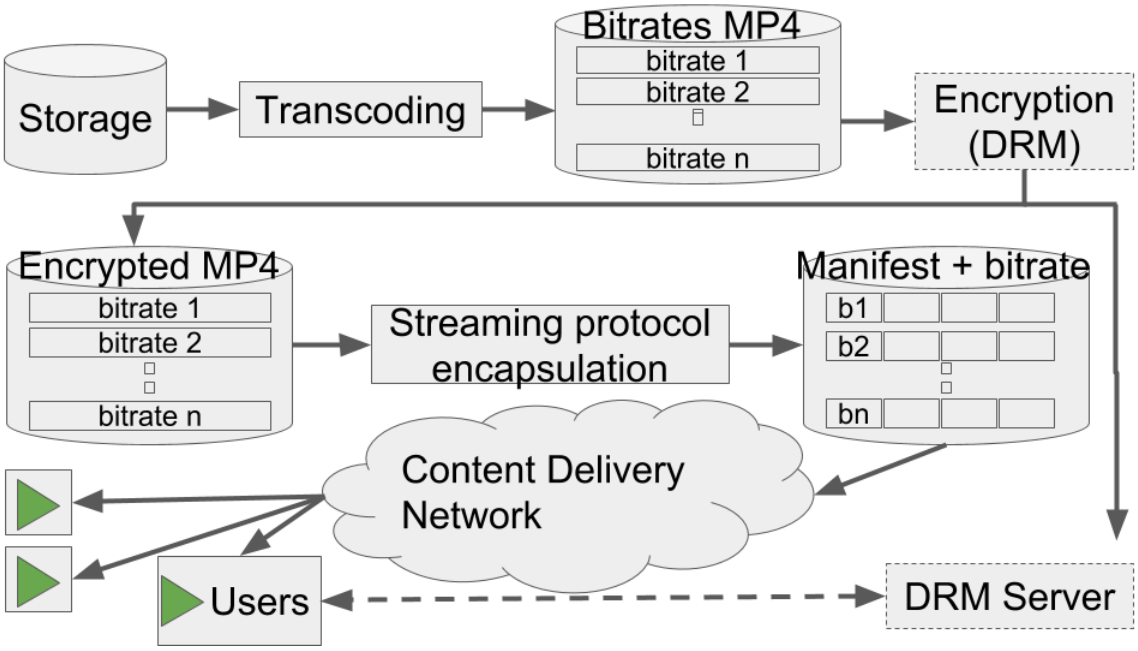


Fig. 2.1.: A video delivery pipeline.

Each publisher operates a *video management plane*, a term we use for a pipeline of automated systems (some with humans in the loop), Fig. 2.1 shows one such pipeline, that perform two primary functions. The first function *prepares* video content for delivery to users. Preparation involves *packaging* the video content and *distributing content to CDNs* for delivery to users. The second function is to *develop and maintain* playback software for the wide range of devices on which video is consumed by users.[1]

Packaging. Packaging achieves two goals: 1) preparing content for adaptive streaming and 2) generating the necessary information for an end user device to perform playback.

Encoding. The first packaging step transcodes the master video file into multiple bitrates of encodings such as H.264 [36], H.265 [37] or VP9 [38]. A video bitrate encodes the video at a certain *resolution* and a certain *quality*. A given resolution can be encoded at different qualities, which differ in the degree of lossy compression

[1]Other management plane functions, including accounting, billing, and fault isolation, are beyond the scope of this thesis.

applied to trade-off perceptual quality for reduced bandwidth. Publishers optionally use *DRM* (Digital Rights Management) software to encrypt the video so that only authenticated users can access it.[2]

Each encoded bitrate of the video is then broken into chunks (a chunk is a fixed playback-duration portion of the video) for adaptive streaming and encapsulated using a *Streaming Protocol* (discussed below). Some publishers support byte-range addressing, where clients can request an arbitrary byte range for a given bitrate instead of chunks.

Streaming protocols. Streaming protocols define the encapsulation format for video chunks to enable delivery over the network. A number of streaming protocols are in use today including Apple's HLS [39], Microsoft's SmoothStreaming (MSS [40]), Adobe's HDS [41] as well as an open standard MPEG-DASH protocol (DASH) [42]. Of these, Apple's devices only support HLS, though recent Apple devices allow limited support for DASH [43]. Some protocols like DASH [42] can support any video encoding format, while others like HLS only support a fixed set of codecs [44].

Streaming protocols also specify metadata about the video necessary for adaptation by the control plane. This metadata is stored in a *manifest* file. The manifest contains information about a number of attributes including the values of available bitrates for adaptation, the audio bitrates, the time duration of an individual chunk and the URLs to fetch video chunks *etc.*

Device Playback. The next function of the management plane is to support the range of devices on which a user can view the publisher's content. To enable playback on them, publishers either provide *Video Players* embedded in web pages to permit browser-based viewing or *Apps* on devices that permit app-based content delivery.

Browser-based video players today are either implemented using JavaScript inserted into webpages using native HTML5 support, or using external plugins such as Flash or Microsoft's Silverlight. However, several types of devices such as set-top boxes (e.g. Roku, AppleTV), game consoles (e.g Xbox), smart TVs (e.g. Samsung

[2]This is orthogonal to TLS encryption of the video during transmission over HTTPS.

TV), and mobile devices use app-based playback. To build these apps, publishers use device-specific SDKs (Software Development Kits, sometimes called Application Frameworks) which provide support for frame rendering, user controls *etc.* as well as bitrate adaptation logic [15–18, 24, 33]. Because publishers may have to support different devices, and, for a given device, different SDK versions (since users may take time to upgrade their device SDKs), at any given time publishers may have to maintain several versions of their app (one for each device-SDK version combination).

Content Distribution. Publishers employ Content Distribution Networks (CDNs). Some content publishers such as YouTube and Netflix deploy their own CDNs. The publishers in our dataset serve their videos via *third-party CDNs* (though some also use private CDNs). To improve performance and availability, some publishers serve content through multiple CDNs [19, 20, 45]. Some publishers use a CDN broker to select the best CDN for a given client view [46]. Even some publishers who only use a single CDN use a CDN broker for management services such as monitoring and fault isolation.

Most publishers proactively *push* content to CDNs. A publisher may either push packaged chunks to each of its CDNs, or may use a packaging service provided by a CDN. In the latter case, the publisher pushes the master video file (or live video stream), and the CDN performs the packaging on behalf of the publisher. The client playback software retrieves chunks using URLs in the manifest file.

2.3 Goals, Methodology & Dataset

Goals. We want to characterize, at scale, publisher video management plane practices (with respect to packaging, CDN use, and device support) and how they have evolved over time. We also present preliminary analyses to understand the implications of these findings on the complexity of video management, and the performance of video delivery.

Prior industry reports have explored related aspects of video management planes [10–14] (see §2.7 for details). These studies have four shortcomings that we address in this thesis. First, they lack a *publisher-centric* focus, even though publishers make video management decisions. As a simple example, these studies do not reveal how many streaming protocols or how many CDNs a publisher uses, but these factors affect management complexity (§2.5). Second, these studies do not *contextualize* their results. For example, consider a finding that few publishers use DASH. If these publishers are large, they are more likely to drive adoption of DASH than if they are all small publishers. Third, most studies were one-off, but the video landscape is continuously evolving. Longitudinal trends can help understand how the video delivery ecosystem is likely to evolve in the short to medium term. Fourth, in part because they lack a publisher-centric focus, these studies do not shed light on a common practice in video delivery, *content syndication* (§2.6).

Extracting management plane practices from a CDN broker dataset. We use data from one of the largest brokers in the world. The broker is used by publishers to monitor playback quality and to select CDNs. The broker provides a monitoring library which publishers integrate with their video players. The monitoring library reports per-view information to the broker's backend. The broker collects data for devices including desktops, mobiles, and smart TVs.

Our dataset spans 27 months (January 2016 to March 2018) and contains over 100 billion views. For each view, it identifies the video publisher; the manifest's URL; device model (*e.g.,* iPhone, Roku); the operating system (*e.g.,* iOS, Android); HTTP user-agent (for browser views) or SDK and SDK version number (for app views); the CDN(s) that were used to deliver the content;[3] viewing time; and delivery performance (average bitrate and rebuffering time).

From this data, we can extract, for any given time window (*e.g.,* a month), and for each publisher: which CDNs the publisher uses, and which devices the publisher's content was viewed on. We also infer which streaming protocols a publisher uses by

[3]During a single view, chunks may be downloaded from multiple CDNs.

Table 2.1.: Streaming protocol file extensions and sample URLs

Protocol	Extension	Sample URL
HLS	.m3u8, .m3u	http://[...].akamaihd.net/master.m3u8
DASH	.mpd	http://[...].llwnd.net//Z53TiGRzq.mpd
SmoothStreaming	.ism, .isml	http://[...].level3.net/56.ism/manifest
HDS	.f4m	http://[...].aws.com/cache/hds.f4m

parsing the URL for the manifest file. Different streaming protocols use pre-defined file extension types for their manifest files (Tab. 2.1): for example, HLS manifest files typically use the `.m3u8` file extension[4].

For each of these dimensions, we can associate three measures that can help contextualize the dimension. Our primary measure, and one used most often in the video industry [29–31], is the number of *view-hours* (*i.e.,* the total viewing time in terms of hours). Using our data, we can examine, for example, the number of view-hours of a publisher's content delivered from a given CDN, over HLS, to iPhones. In some of our analysis, we use *the number of views* associated with a particular dimension (*e.g.,* the number of video plays delivered to Roku players). This is helpful to understand if view-hours were accumulated from a few long video views or many short video views. In some analyses, we also measure importance by the number of distinct videos seen. We do not have this data for all our publishers (some do not report video *titles*), so when we use this measure, it is an under-estimate of the total number of distinct videos.

Dataset limitations. We only have data when publishers have integrated the broker's monitoring library. So, we do not have data from publishers that do not use the library, including the 3 largest video publishers, YouTube, Netflix and Facebook.

[4]There are two exceptions to this. RTMP can be detected from the protocol specification in the URL (RTMP instead of HTTP). Progressive downloading uses file extensions corresponding to video encodings, like `.mp4` or `.flv`.

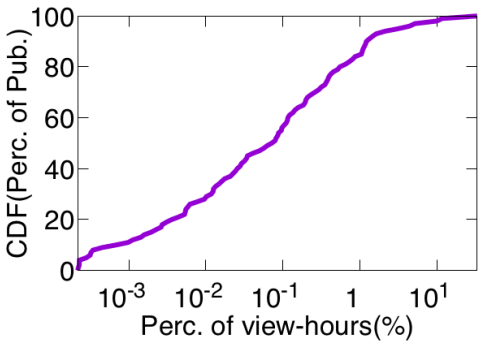
For publishers that do use the library, we cannot definitively differentiate whether the appearance (or increase in view-hours) of a publisher in our data is due to actual growth or due to progressive on-boarding of the publisher. Publishers that use the broker may be predisposed towards using multiple CDNs, and the broker's role in CDN choice means that trends in CDN usage in our data may or may not be indicative of trends in the larger Internet video ecosystem.

The dataset does not include data necessary to investigate three aspects of video management: Digital Rights Management (DRM) usage, monetization, and encoding format. To study the video encoding codec, we could have downloaded all unique manifest URLs in our dataset. This is not only logistically difficult (13 million unique URLs in March 2018 alone) but also often requires user authentication, and we do not have subscriptions for all the publishers in our study.

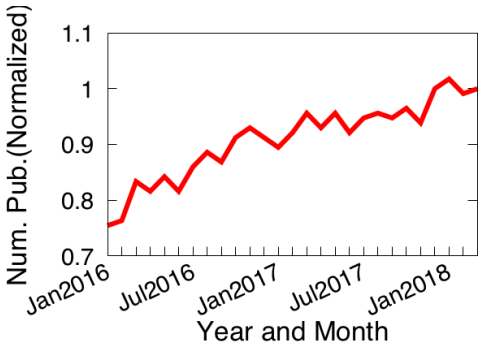
A macroscopic view of the dataset. Video consumption today is dominated by YouTube, Facebook, and Netflix, which contributed (according to 2016/2017 studies [28, 30]) 1 billion, 0.1 billion and 0.14 billion view-hours per day respectively. Beyond these three, there are a large number of video publishers that deliver online content. In March 2018, our dataset had more than hundred of publishers each with at least 3000 view-hours in a month. Across all our publishers, the aggregate *daily* view-hours is 0.06 billion per day, comparable to Facebook and Netflix in 2016 and 2017. Finally, the publishers in our study together serve 180 countries.

Prominence. Our dataset includes many prominent publishers, including 7 of the top 10 US subscription-based video publishers [34]. Further, the publishers in our dataset are diverse, and include international sports broadcasters, major news outlets, on-demand movies/subscription TV providers and entertainment content providers. Within these broad categories, the publishers in our dataset are prominent, including 3 of the top 5 Sports broadcasters and 4 of the top 5 Breaking News publishers according to Alexa category rankings [47].

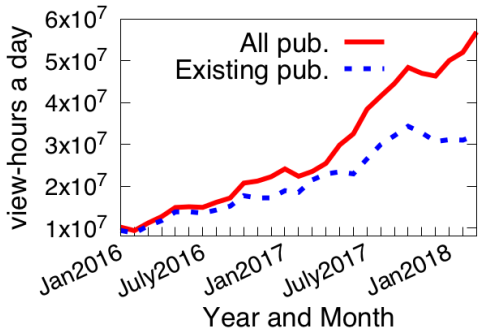
Distribution of view-hours. In March 2018, the average daily view-hours of publishers in our dataset ranged from hundreds to tens of millions (actual numbers



(a) CDF of percentage of view-hours a day across publishers in March 2018



(b) Normalized number of publishers per month for past 27 months



(c) Aggregated average view-hours a day for past 27 months

Fig. 2.2.: Dataset characteristics with respect to view-hours and number of publishers.

omitted due to confidentiality). Fig. 2.2(a) shows a CDF of percentage of view-hours a day across publishers. 84% of publishers contribute less than 1% of total view-hours. However, the top 5% of publishers contribute 65% of the total daily view-hours, while the biggest publisher contributes 33% of the daily view-hours. This trend is similar to studies of the web that indicate that the most popular web pages account for a large fraction of accesses [48, 49].

Longitudinal view. Fig. 2.2(b) shows the number of publishers in our dataset over time (for these, and subsequent analyses, we only count publishers that exceed 3000 view-hours per month). Over the course of 27 months, the number of publishers

increased by 32%. Fig. 2.2(c) shows the average daily view-hours aggregated across publishers over the entire period. The total number of view-hours (solid line) grew by 5× over 27 months. Some of this growth was driven by new publishers that appeared over time, some by publishers on-boarding their viewers to our broker over time, and the rest by growth in viewership of existing publishers. The dotted line shows the view-hours over time for the approximately 45% of publishers that are present for the entire time period. These publishers experienced a 3× increase in view-hours.

2.4 Characterizing Video Management Planes

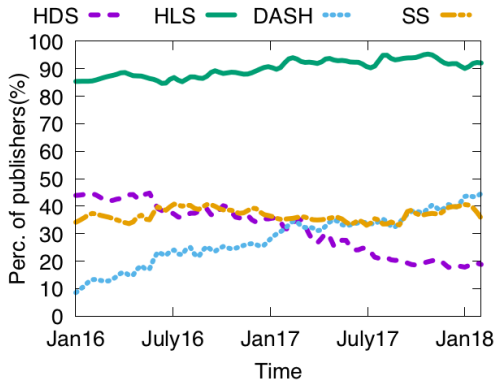
We characterize video management planes along three dimensions: *packaging*, measured by the streaming protocols used; *content distribution*, measured by the CDNs used; and *device playback*, measured by types of devices and number of application frameworks. For each dimension d, we ask:

- How has d evolved across publishers?
- How has d evolved in terms of view-hours? For example, does a dominant practice result from a few big publishers or many small publishers?
- What is the distribution across publishers of number of instances of d? Is it correlated with publisher view-hours?

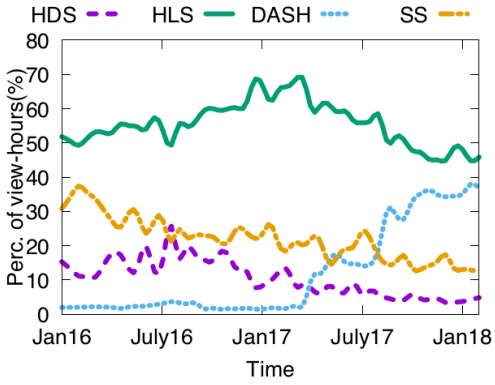
Our two-year dataset is too large to process every view, so we use a sequence of two-day snapshots taken bi-weekly. We use the last snapshot, taken in March 2018, for the third question.

2.4.1 Packaging

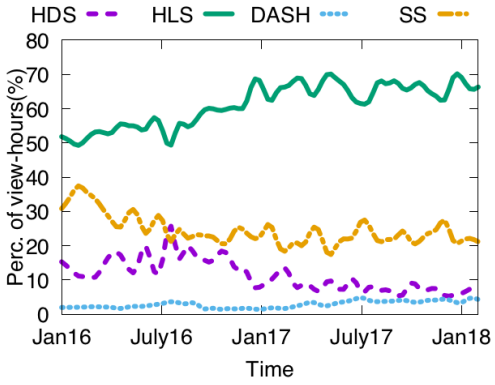
Understanding the prevalence of different streaming protocols is important for several reasons. First, the amount of work/resource needed to package content is proportional to the number of streaming protocols supported by a publisher. Also



(a) Percentage of publishers that supported each streaming protocol over time



(b) Percentage of view-hours by each streaming protocol over time



(c) Percentage of view-hours (excluding largest publisher) by each streaming protocol over time

Fig. 2.3.: Streaming protocols used in terms of percentages of publishers and view-hours for past 27 months

the time taken to package content can add delay to live content distribution. Second, in some cases, support for a streaming protocol can directly impact the set of devices that can be supported: *e.g.,* until recently publishers needed to support HLS to work with Apple devices (§2.2).

Prevalence by streaming protocol. Streaming protocols include HTTP-based protocols as well as RTMP, a protocol for low latency video streaming services [50–52].

In our dataset, RTMP only accounted for 1.6% of the view-hours in January 2016 and 0.1% in March 2018. RTMP has compatibility issues with network middleboxes, scalability limitations [50, 51], and limited device support. For these reasons, our publishers prefer HTTP-based streaming protocols even though these protocols may add a few seconds of encoding and packaging delay to live streams. The rest of our analysis focuses on HTTP-based protocols.

Across publishers. Fig. 2.3(a) shows the percentage of publishers that supported a given streaming protocol over time (the sum of percentages at any given point in time exceeds 100% because publishers can support multiple protocols). The *rightmost point* of each curve *indicates the latest snapshot*. In this latest snapshot, 91% of publishers support HLS, likely because many devices and players support it [53–55]. DASH and SmoothStreaming are currently supported by around 40% of publishers, but HDS is only supported by 19% of the publishers. Over time, support for DASH has increased from 10% of publishers to 43%, corroborating a recent survey of video developers [10]. HDS has steadily lost support. The growth of DASH has not been at the expense of HLS or SmoothStreaming. Over time, HLS and SmoothStreaming support across publishers has remained steady.

By view-hours. We can quantify usage of streaming protocols in terms of *view-hours*, unlike existing industry surveys [10, 12]. Fig. 2.3(b) shows the percentage of view-hours served by different protocols over time. In our latest snapshot, HLS and DASH are dominant, each accounting for about 38-45% of the view-hours, with the other two being relatively small. Longitudinally, the most noticeable trend is the growth in use for DASH from 3% to 38% view-hours. We found that this is due to a single large publisher that starts to become visible in the dataset from March 2017 onwards. This increase in DASH support noticeably reduces the fraction of view-hours attributable to the other protocols.

When we remove this publisher (Fig. 2.3(c)), we observe that DASH support from other publishers only accounts for less than 5% of view-hours overall. To explain this further, Fig. 2.4 shows the distribution across publishers of the percentage of

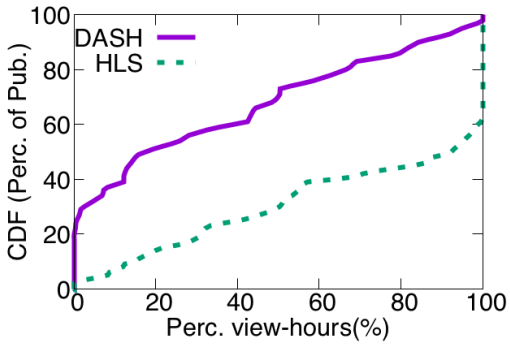


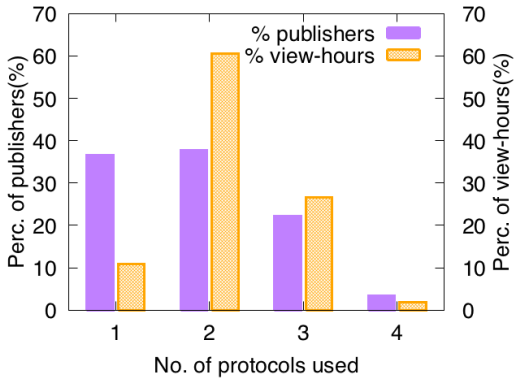
Fig. 2.4.: CDF across publishers of percentage of view-hours served via DASH and HLS in the latest snapshot.

view-hours that used a given protocol, only considering publishers that support that protocol. Even though 40% of the publishers support DASH (Fig. 2.3(a)), half of them employ it for at most 20% of their view-hours (Fig. 2.4). In contrast, among the 90% of publishers that support HLS, half use it for at least 85% of their view-hours.

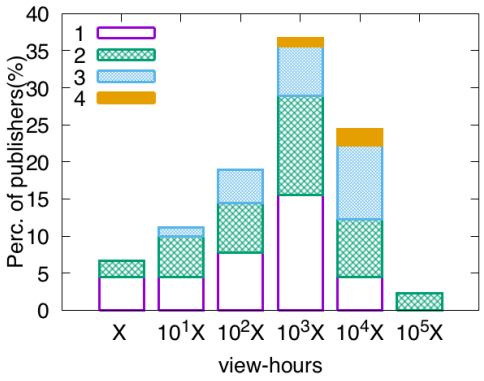
Finally, in Fig. 2.3(b), the growth in DASH due to a single publisher also illustrates the complexity of a management plane operation: *onboarding*. In this case, this publisher slowly moved their clients to our broker over a period of *one year*, by gradually integrating the broker's monitoring libraries into each of its device players.

Number of protocols per publisher. Fig. 2.5(a) explores *how many* streaming protocols each publisher supports in the latest snapshot. Each group of bars corresponds to a given number of protocols n and shows the percentage of publishers that used n protocols (left) and the percentage of view-hours from publishers that used n protocols (right). While 38% of publishers support 1 protocol, these publishers account for less than 10% of view-hours. The use of 2 protocols is dominant (38% of publishers, accounting for nearly 60% of view-hours), and the use of 3 protocols is also significant.

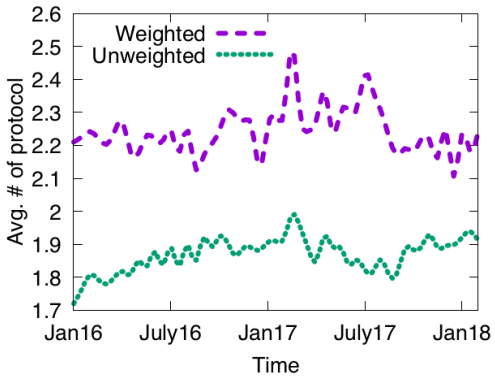
Fig. 2.5(b) presents the number of protocols used by publishers when bucketed by their view-hours. The left most bar corresponds to publishers with X view-hours or less (we do not specify X for confidentiality reasons). The second bar corresponds to



(a) Number of protocols supported by publishers in latest snapshot, as percentage of publishers and when weighted by their view-hours



(b) Number of protocols supported by publishers in latest snapshot, bucketed by publisher view-hours



(c) Average number of protocols supported per publisher over time

Fig. 2.5.: Number of streaming protocols used by publishers (by % of publishers and by their view-hours).

publishers with X to $10X$ daily view-hours, the next bar to publishers with $10X$ to $100X$ view-hours, and so on. Each bar corresponds to the percentage of publishers in a given bucket, broken down by the number of protocols used by the publishers. The tallest bar indicates that (i) over 35% of publishers have 100X to 1000X daily view-hours; and (ii) publishers in this bucket use from 1 to 4 protocols. Outside of

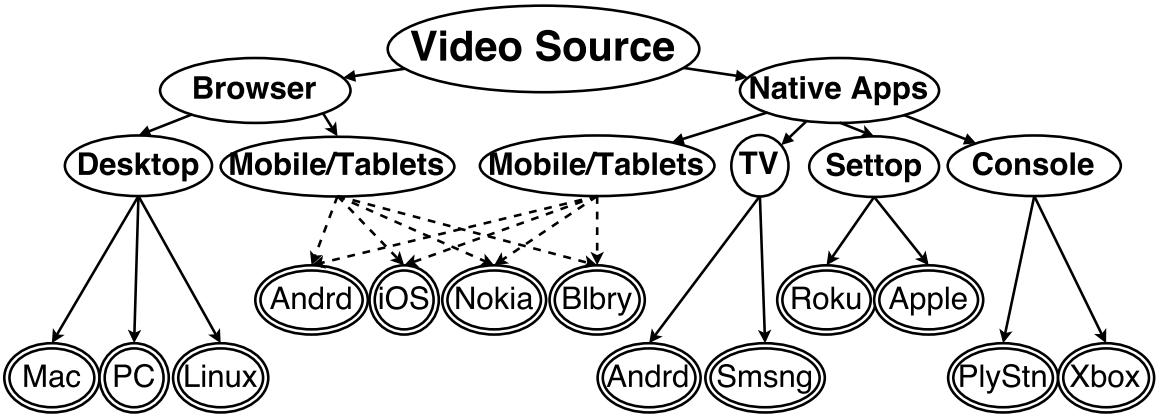


Fig. 2.6.: Target platforms for video publishers

publishers in the bucket with the least view-hours, more than 50% of publishers in all buckets use at least 2 protocols, with all publishers in the $10^4X - 10^5X$ bucket (right-most bar) using 2 protocols. A significant number of publishers in the intermediate buckets use 3 or 4 protocols.

Fig. 2.5(c) shows changes in the number of protocols used over time. The lower curve shows the number of protocols averaged across publishers. The upper curve is the average weighted by the publisher's view-hours. The weighted average is always higher, indicating larger publishers tend to use more protocols. Despite fluctuations, the average number of protocols has remained a bit below two, and the weighted average higher than two. This consistency is likely because the growth in DASH has coincided with the decline of HDS.

2.4.2 Device Playback

Understanding the set of user devices that a publisher supports is important because (i) implementing and maintaining video players for a range of platforms requires

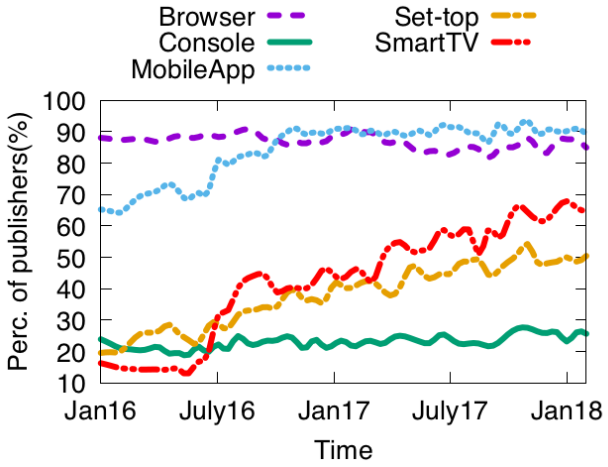


Fig. 2.7.: Percentage of publishers supporting each platforms

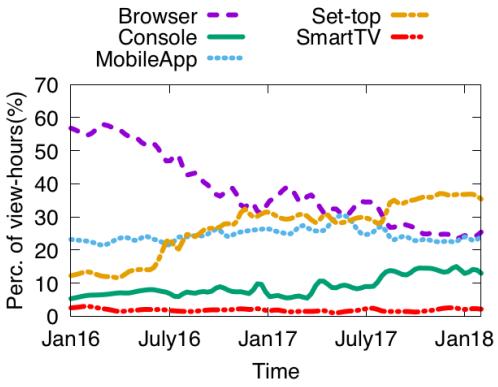
significant effort; and (ii) the popularity of a platform can impact the publisher's decision of whether to support it.

Prevalence by platform. Video is consumed (Fig. 2.6) on a variety of devices which can broadly be classified into two *platform types*: browsers and apps. Video is consumed on desktops, laptops, tablets and mobile devices using browsers on these devices. Video is also consumed using apps on mobile devices, smart TVs, set-top boxes and gaming consoles. We now explore the prevalence of video consumption across these 4 app-based platform categories, and across browsers (which includes browser usage on mobile devices[5]).

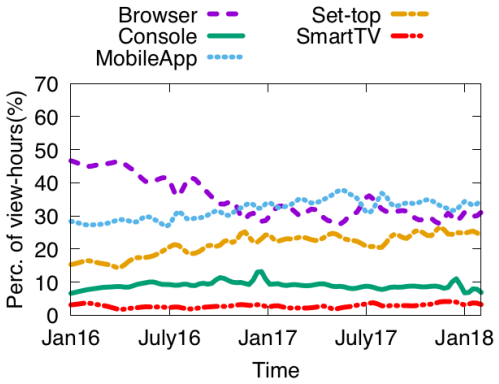
Across Publishers. Fig. 2.7 shows that, over the 27 month period, support has grown most significantly for set-top boxes and smart TVs (from under 20% of publishers to above 50% and 60% of publishers respectively today). There has also been growth in mobile applications, and, as expected, almost all publishers support browsers and mobile apps today.

By view-hours. Fig. 2.8(a) shows that, by percentage of view-hours served by different platforms, browser viewership has declined from nearly 60% to less than 25%

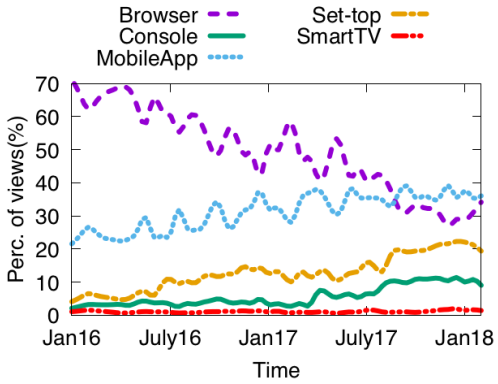
[5]While smart TVs and set-top boxes support browsers, we see very little video consumption on these browsers.



(a) Percentage of view-hours on each platform



(b) Percentage of view-hours on each platform, excluding 3 largest publishers



(c) Percentage of views on each platform

Fig. 2.8.: Over time, percentage of view-hours, view-hours excluding 3 largest publishers, and views on each type of platform

today. Despite publishers aggressively increasing support for smart TVs during the last two years, their share of view-hours has stayed at less than 5%. Interestingly, the set-top category has grown the most, with the largest share of view-hours (nearly 40%) in the latest snapshot, while mobile app viewership has stayed steady at about 20-25%.

To understand whether large publishers bias our observations, Fig. 2.8(b) shows view-hour trends when we remove the three largest publishers, which account for half

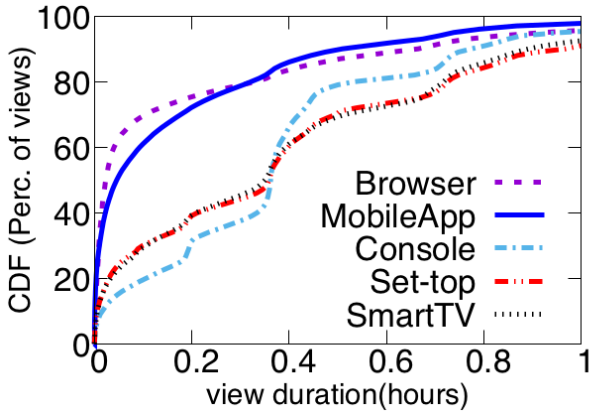
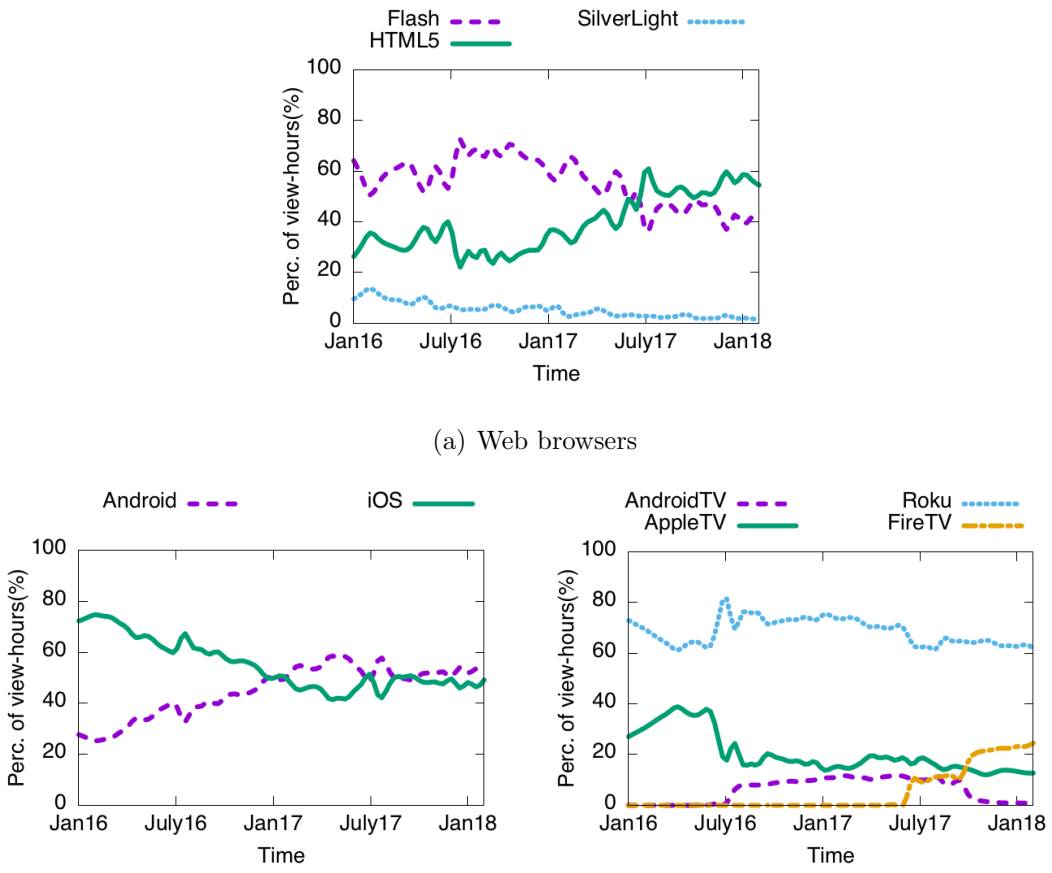


Fig. 2.9.: CDF of individual view duration for each platform

the view-hours in our dataset. There are some differences in trends, with mobile app viewing surpassing all other platforms over time, and set-top viewing growing at a slower rate. However, the results overall are qualitatively similar, indicating that platform usage trends in our dataset are *not* being driven by the largest publishers alone, unlike the trend with DASH adoption.

By views. The growth in view-hours with set-top boxes could be caused by longer view durations or by more views. To investigate this further, Fig. 2.8(c) depicts the fraction of *views* served across different platforms (including the three large publishers). While views with set-top boxes have grown to 20% of views in the latest snapshot, they lag behind the set-top view-hour growth (to nearly 40% in Fig. 2.8(a)). Taken together, Fig. 2.8(a) and Fig. 2.8(c) suggest that mobile app views are of shorter duration, while set-top views are of longer duration. Fig. 2.9 confirms this intuition, depicting the CDF of individual view duration (in hours, with the X-axis truncated at 1 hour) for each platform in our latest snapshot. Only 24% of mobile and browser views last longer than 0.2 hours, while more than 60% of set-top views last longer than 0.2 hours.

Trends within platforms. An examination of trends of device usage within each of the top 3 platforms also shows interesting trends, some well-known, others



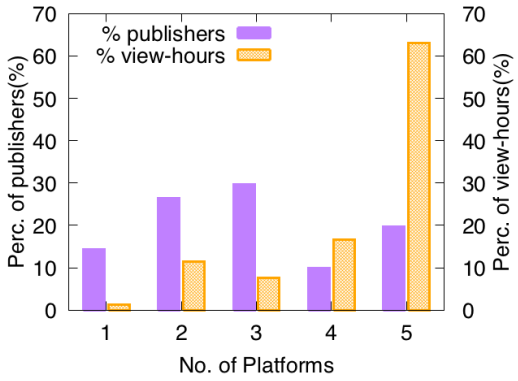
(a) Web browsers

(b) Mobiles/Tablets

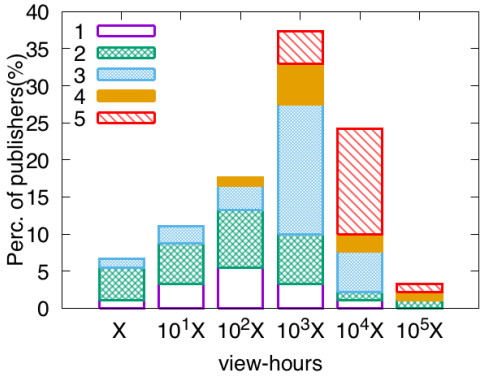
(c) set-top boxes

Fig. 2.10.: Percentage of view-hours served by specific devices belonging to the same platform.

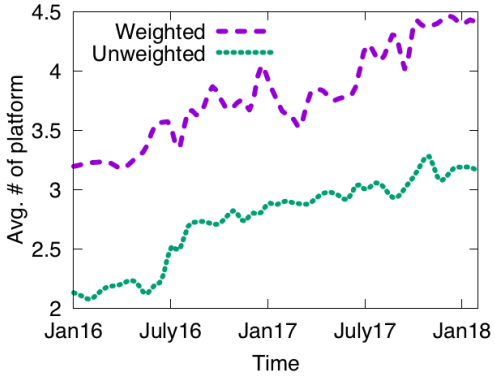
less so. Among browsers, the view-hours for HTML5 increased from about 25% to nearly 60% within the two year period (Fig. 2.10(a)). This increase came at the expense of a reduction of view-hours in other browser-based players, especially Flash. Among mobile devices, view-hours for Android devices have increased significantly (Fig. 2.10(b)), and both Android and IOS have comparable viewership in the latest snapshot. Finally, among set-top boxes (Fig. 2.10(c)) Roku devices are dominant in terms of view-hours, but AppleTV and FireTV account for a non-negligible percentage of view-hours. Overall, the results indicate that a publisher must not only cope



(a) Number of platforms supported by publishers in latest snapshot, as % of publishers and when weighted by their view-hours



(b) Number of platforms supported by publishers in latest snapshot, bucketed by publisher view-hours



(c) Average number of platform supported per publishers over time

Fig. 2.11.: Number of platforms supported per publisher (by % of publishers and by their view-hours)

with multiple platforms but also multiple devices within each platform, which can contribute to significant management complexity (§2.5).

Number of platforms per publisher. Fig. 2.11(a) characterizes the number of platforms supported by publishers. Over 85% of publishers support more than one

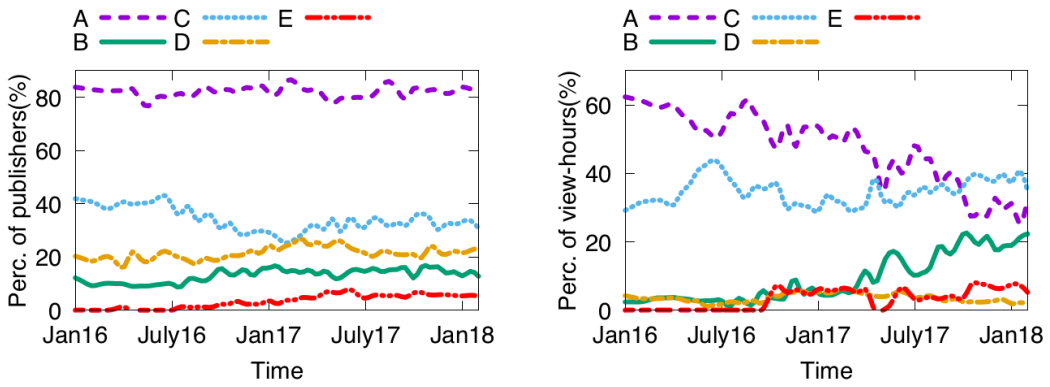
platform, and over 95% of view-hours are attributable these publishers. 30% of publishers support all 5 platforms and these publishers account for over 60% of the view-hours. As with other dimensions, the number of platforms supported increases with view-hours (Fig. 2.11(b)). For instance in the bucket corresponding to 10^3X to 10^4X view-hours, the vast majority of publishers support at least 3 platforms, and nearly half support all 5 platforms. Finally, Fig. 2.11(c) shows that the average and the view-hour weighted average of the number of platforms supported by publishers have increased by 48% and 37% respectively over the two year period. Publishers support more than 3 platforms on average in the latest snapshot, with the weighted average being nearly 4.5.

2.4.3 Content Distribution

Once the content is packaged, it is distributed to end users using content distribution networks (CDNs). CDNs work by situating the content closer to the end user. Understanding this dimension is important because CDN usage can have significant performance impact [19–21,45]. Further, publishers can employ multiple CDNs which can lead to complexity in video management (§2.5).

Prevalence by CDN. Across all publishers we observed 36 different CDNs in our dataset. This list included both regional and international CDNs. Further, some publishers had their own internal CDNs (sometimes used in conjunction with external CDNs). Of these, over 93% of the view-hours were served by 5 CDNs, indicating that video viewership is concentrated among a handful of CDNs. We analyze the opportunities arising from this consolidation in §2.6.

Across publishers. Fig. 2.12(a) shows the percentage of publishers across time that use each of the top 5 CDNs (anonymized). One CDN, A dominates, with nearly



(a) Percentage of publishers by a CDN they used over time

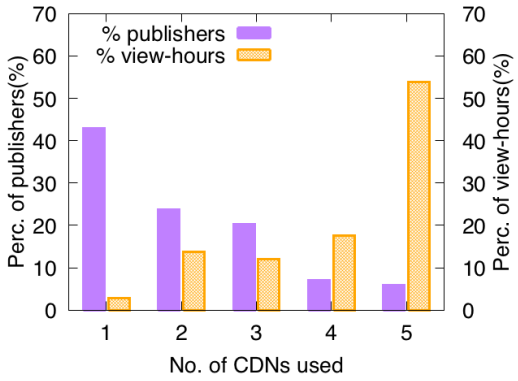
(b) Percentage of view-hours by each CDN over time

Fig. 2.12.: Analysis of CDNs based on percentage of publishers and view-hours for past 27 months

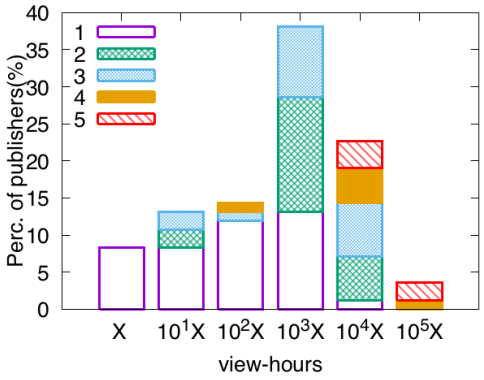
80% of the publishers using it, while only 30% use the second most dominant CDN C. Longitudinally, these numbers have remained more or less steady.

By view-hours. Fig. 2.12(b) shows the percentage of the view-hours served by each of these CDNs. In the current snapshot, 3 CDNs (A, B and C) each account for 20-35% of the view-hours, while the other 2 account for about 5% or less each. Some CDNs use anycast to direct a client to a particular server [56], but anycast is susceptible to BGP route changes that sever ongoing TCP connections, raising concerns that anycast may not be suitable for large transfers [57]. We discovered that 1 of the top 3 CDNs in our dataset uses anycast, suggesting that anycast route instability has not been a blocking factor in the reliable delivery of video chunks.

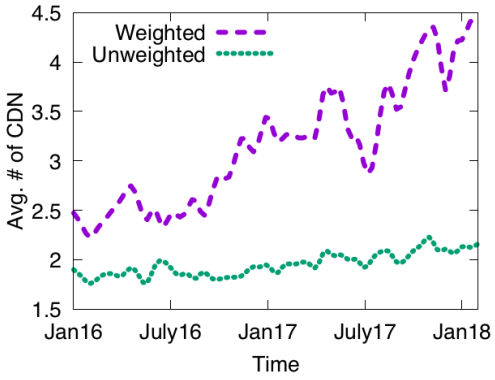
Longitudinally, CDN A's share of view-hours has halved, while CDN B's share has increased from a few percent to about 20%. We examined the data a little further, and found that this change is driven by one large publisher who recently became our broker's customer. This publisher uses CDNs B and C, driving down A's share of



(a) Number of CDNs used by publishers, as percentage of publishers and when weighted by their view-hours



(b) Number of CDNs used by publishers in latest snapshot, bucketed by publisher view-hours



(c) Average number of CDNs used across publishers over time

Fig. 2.13.: Number of CDNs used by publishers (by % of publishers and by their view-hours).

view-hours. Again, these results highlight the importance of considering view-hours in the analysis, an aspect not considered by prior work (§2.3).

Number of CDNs per publisher. We next characterize publishers by the number of CDNs they use.

Fig. 2.13(a) shows the percentage of publishers that use a given number of CDNs, and the percentage of view-hours that may be attributed to these publishers in our latest snapshot. While more than 40% of publishers use only a single CDN, they account for less than 5% of the view-hours. In contrast, less than 10% of publishers use 5 CDNs, but these publishers account for more than 50% of view-hours.

Fig. 2.13(b) shows the number of CDNs used by publishers classified by their view-hours. The results indicate that the percentage of publishers that use multiple CDNs increases with the number of view-hours attributable to the publisher. For example, at the extremes, all publishers with more than 10^5X view-hours use at least 4 CDNs, while all publishers with less than X daily view-hours use a single CDN. In the $10^3X - 10^4X$ bucket, the number of CDNs used ranges from 1 to 3, while in the $10^4X - 10^5X$ bucket, the number of CDNs ranges from 1 to 5.

Finally, Fig. 2.13(c) shows the longitudinal trend for the average number of CDNs used by publishers, and the weighted average (weighted by the publisher's view-hours). While there is some growth in the average CDNs per publisher (with the average exceeding 2 in the latest snapshot), the weighted average grows much faster and is nearly 4.5 in the latest snapshot.

Live video has some different demands than video-on-demand (VoD), especially low end-to-end latency from video capture to viewing, and so we were interested in whether publishers favored particular CDNs for one type of content versus the other, perhaps due to different CDN features or latency. Of publishers which use multiple CDNs and serve both live and VoD traffic, 30% use at least one CDN only for VoD traffic, and 19% use at least one CDN only for live traffic. In one extreme case, a publisher used one CDN for all its VoD traffic and a different CDN for all its live video. However, most CDNs that were used exclusively for live content by one publisher were used exclusively for VoD content by another publisher. Thus, no CDN dominated others for live video, and our results seem to reflect opaque management plane decisions of publishers. We have left it to future work to explain the rationale for CDN choices amongst publishers.

2.4.4 Summary

Several common themes run through our analysis of these three dimensions of management complexity.

- In no dimension does a single alternative dominate in terms of view-hours. View-hours are roughly equal between HLS and DASH; across browser, mobile, and set-up; and across three large CDNs.
- More than 90% of view-hours can be attributed to publishers who support more than 1 protocol. The same is true of publishers who use more than 1 CDN, and publishers who support more than 1 platform.
- Publishers with more view-hours support more choices in each dimension. The average number of choices, weighted by view-hours, is 2.2 for protocols, 4.5 for CDNs and 4.5 for platforms.
- At least two of our trends (increase in DASH usage, and the emergence of 3 CDNs with comparable view-hours) are driven by a large publisher. However, large providers alone do not drive trends in platform usage.

By assessing the distribution of view-hours, we observe new trends and provide additional insights on known trends:

- By view-hours, set-top box usage is significant, even exceeding browsers and mobile apps. This sharp rise of set-top box usage is not well documented and can drive the adoption of higher resolution video such as 4K video.
- Prior work has not quantified the distribution of multi-CDN usage. We find that almost 80% of view-hours are from publishers using 4 and 5 CDNs. While 2 or 3 CDNs are sufficient for resilience or load balancing, additional CDNs appear to be necessary for improved coverage.
- Given industry excitement with DASH [10], we expected to find significant DASH support among our publishers. While over 40% of our publishers support DASH,

one large publisher accounts for almost all DASH view-hours. In working with publishers, we have experienced quality issues with DASH implementations, so it might take some more time for the DASH ecosystem to mature to the point where small publishers also use more DASH.

- Our dataset shows negligible use of RTMP even though several of our publishers serve live content. RTMP provides low latency live streaming, but it has some scalability issues and lacks widespread device support.
- Other studies report high mobile view shares [13], and we find that these have indeed risen over time, but mobile app view-hours have not increased by a corresponding amount, because view durations on mobile devices tend to be small.
- Prior work has quantified the demise of Flash, reporting a 96% drop in Flash views for one browser [58]. We find a much more modest drop, with about 40% of browser view-hours attributable to Flash, down from 60% at the beginning of our study.

2.5 Understanding Management Complexity

Our results in §2.4 have shown that publishers must deal with significant *diversity* across all components of the management plane. This diversity can impact the *complexity* of management tasks. In this section, we propose measures to capture this complexity, and explore how these measures correlate with publisher view-hours, an approximate indication of publisher size. A correlation indicates that management complexity is higher for large publishers and low for small publishers. If, however, even small publishers incur high management complexity, this may indicate a high barrier to entry since a publisher who targets modest viewership early in its business growth must still pay for high management costs.

Some examples of video management tasks include:

Software development and maintenance. Video publishers must build and maintain players for different devices and browsers. Typically, content publishers use device specific SDKs (released by device vendors, or third parties) [59–61]. A publisher

may not only need to maintain multiple code bases corresponding to the different supported devices (§2.4), but may also need to support multiple versions of the SDKs to support legacy devices. Besides the one-time development cost, there is an ongoing maintenance cost associated with rolling out new features, and fixing software bugs.

Packaging. For each video title, a publisher needs to package the content for different streaming protocols. Packaging may be performed by the CDN or other third parties [12, 62–64], but the associated overheads remain irrespective of who does the packaging.

Failure Triaging. Troubleshooting video performance problems is challenging, and poor performance may be due to a CDN, the network or the user's device [65, 66], or a combination of these factors [66]. In addition, performance problems may be associated with a particular streaming protocol (*e.g.,* manifest files may have errors for specific protocols).

Measures of management complexity. Each of these tasks has an associated *complexity* that depends on the three components we study (protocols, CDNs, and devices). We list below *complexity measures* for video management, motivated in part by prior work on quantifying complexity in other domains such as web page complexity [67], and router configuration complexity [68]. Other measures are possible, and we have left an extended exploration to future work.

Management plane combinations. One measure of complexity is the number of unique combinations of CDN, streaming protocol, and the end user's device that a publisher supports. This relates to the complexity of triaging failures. A failure can be caused by one of the components (*e.g.,* CDN or protocol), an interaction between two components (*e.g.,* a specific CDN's implementation of HLS), or an interaction across all three components (*e.g.,* we have observed a failure caused by the interaction between a Chromecast implementation using SmoothStreaming on a specific CDN). In the worst case, it might be necessary to examine all combinations to triage a failure: indeed, the broker whose data we use triages failures automatically by aggregating

failure reports across all management plane combinations [69]. More broadly, failure triaging may also depend on other factors such as the choice of ISP which we do not consider in this thesis.

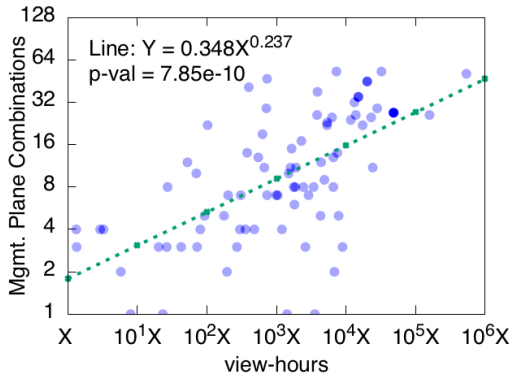
Protocol-titles. The product of protocols used by a publisher and the number of unique video titles for a publisher captures the packaging costs for the publisher's content[6]. Intuitively, each publisher has to package each title separately for each protocol. This measure determines the compute and storage resources needed to package the publisher's contents and can impact the lag experienced by users for live content.

Unique SDKs. Defined as the number of unique versions of SDKs and browsers supported by a publisher across all devices, this measure captures the software development and maintenance complexity. The metric may also relate to the complexity of triaging a failure related to the device, if the failure is specific to an SDK version or browser.

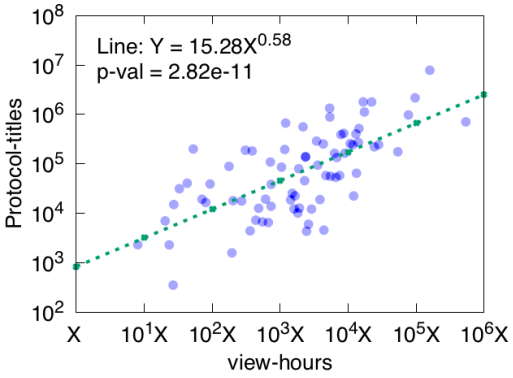
Correlation between management complexity and publisher view-hours. Fig. 2.14(a) presents a scatter plot that shows how the *management plane combinations* metric (on a log scale) correlates with the view-hours (on a log-scale) served by the publisher. We also add a line of best fit using linear regression. The slope of the line shows that when the view-hours increase by a factor of 10, the number of management plane combinations increases by a factor of 1.72×, indicating a sub-linear growth in complexity with publisher size.

Fig. 2.14(b) and Fig. 2.14(c) respectively show similar scatter plots and lines of best fit for both the *Protocol-titles* metric, and the *Unique SDKs* metric. Again, both graphs indicate that the complexity measures increase sub-linearly with publisher view-hours: when view-hours increase by an order of magnitude the *Protocol-titles* grows by 3.8×, while the *Unique SDKs* metric grows by 1.8× with the biggest publishers having to maintain up to 85 different code bases.

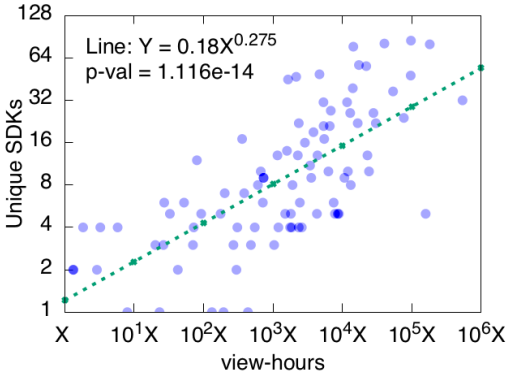
[6]To a first approximation. Packaging cost may also depend on the *length* of each title, which we do not have.



(a) Management plane combinations



(b) Protocol-titles



(c) Unique SDKs

Fig. 2.14.: Correlation between different measures of complexity and publisher view-hours

In each case, the linear fit is statistically significant, with p-values at the 0.05 level of significance smaller than 10^{-9}.

2.6 Management of Syndication

We next explore how today's structure of management planes, where each publisher makes independent decisions on the choice of protocols, CDNs and playback devices, can result in sub-optimal performance when content is *syndicated*. *Syndi-*

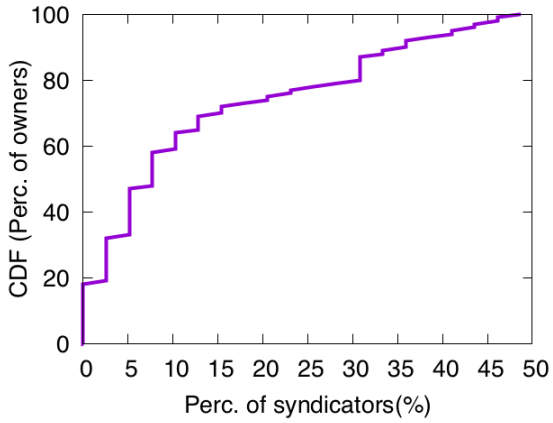


Fig. 2.15.: Content syndication is prevalent in our dataset, with some content owners syndicating to nearly half the full syndicators in our dataset.

cators license and serve content obtained from multiple *content owners*. Conversely, content owners may distribute their content through multiple syndicators.

The prevalence of syndication. By manually inspecting the websites of each of our publishers, we classified about 44% of them as content owners and 43% as *full syndicators* who syndicate the entire content from an owner. The rest are *partial syndicators* which syndicate some, but not all content of an owner (*e.g.,* a single series instead of an entire video library).

By inspecting full syndicators' websites, we can determine, for a given content owner, what fraction of full syndicators have syndicated that owner's content. Fig. 2.15 shows the CDF of the percentage of syndicators used by each content owner. The figure shows that syndication is prevalent – more than 80% of content owners use at least one syndicator, and 20% of content owners syndicate their content to almost 1/3rd of all full syndicators. These numbers are conservative, since a content owner may use syndicators not in our dataset, and since we have not considered partial

syndication. Overall, these numbers suggest that content syndication is significant in online video delivery.

Incorporating syndication in management planes. Today, because each publisher runs an independent management plane instance, the easiest way to syndicate content is that the content owner provides a master or "mezzanine" copy of the content to each of its syndicators which then packages and distributes the content through its video management plane.

In this *independent syndication* model, sub-optimal outcomes might result because syndicators can make independent decisions on video packaging choices. In this thesis, we illustrate two such outcomes to motivate why it is important to study video management planes: (a) different performance for the same syndicated content resulting from different bitrate choices; (b) redundancy in CDN storage usage because multiple copies of the same content can be stored on a CDN using different encodings or protocols.

To quantify these, we focus on a popular series. In our dataset, we observe that this series is served independently by the content owner and by 10 other syndicators. For our bitrate analysis, we focus on a particular episode in the series across all publishers since the bitrates used to encode the video may vary based on the content [70].

Bitrate choices for syndicated content. As described before, bitrate choices decide, for each platform, the resolutions and qualities at which the video is available. With independent syndication, an owner and a syndicator can make different bitrate choices for the same episode of the same series. To obtain these bitrate choices, we downloaded the manifest files for the episode of interest. The manifest files provide information about the audio and video bitrates used in the encoding, as well as other information such as the duration of each chunk. In general, each publisher may have several manifest files for the same video based on factors such as the streaming protocol, type of device, and network connectivity (WiFi, 4G, Wired). For fair comparison,

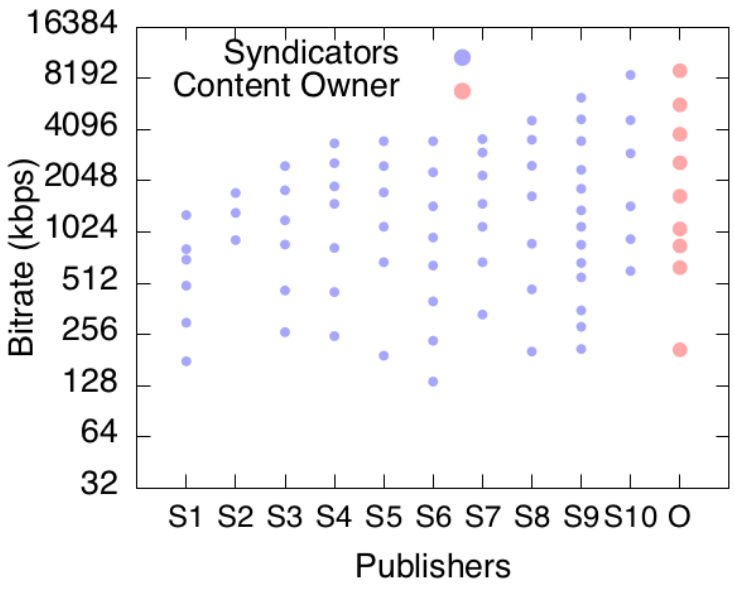
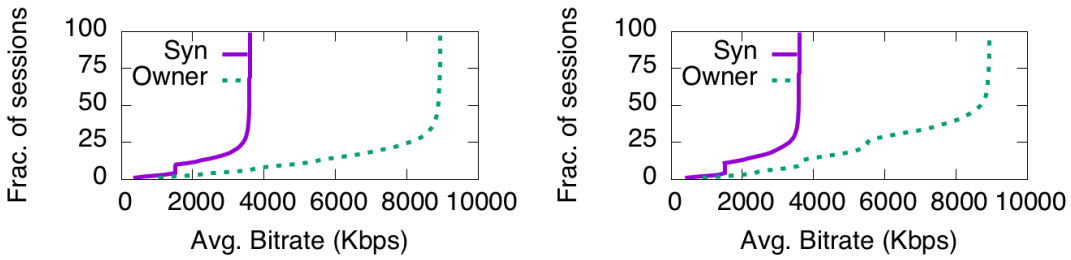


Fig. 2.16.: Bitrate selection decisions for an episode of a popular series by the owner and ten syndicators.

we compare the manifest files served to the same type of device over the same type of Internet connection (WiFi, 4G, Wired).

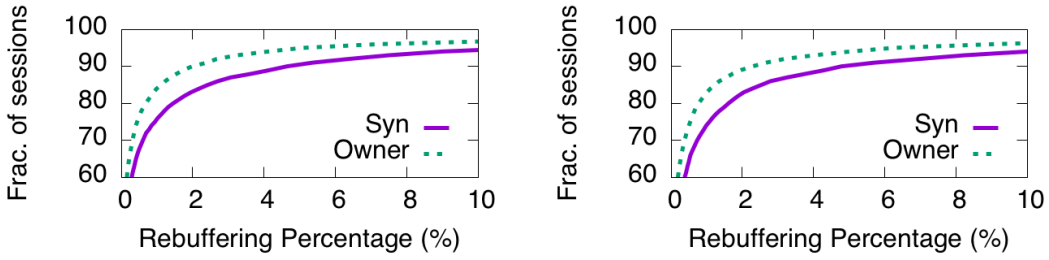
Bitrate choices can vary widely. Fig. 2.16 shows the video bitrates used by the syndicators (S1 to S10), as well as the bitrates offered by the original content owner (O) for iPad devices over a WiFi network. The figure shows a significant difference both in the number of bitrates, the range of bitrates, and individual bitrate choices. At one extreme, S2 encodes video into only 3 different bitrates, while at the other, S9 employs 14 bitrates. The owner uses 9 different bitrates and offers a bitrate that exceeds 8192 Kbps, while the highest bitrate offered by S1 is 7x lower and only a little above 1024 Kbps. We have also performed a similar comparison for other device types, and observed similar heterogeneity in bitrate decisions made by the content owner and various syndicators.

Bitrate choices can impact performance. From our dataset, we can also observe the *performance* achieved by clients of some of these syndicators. Two widely



(a) On ISP X, CDN A (b) On ISP Y, CDN B

Fig. 2.17.: Average bitrate performance of California based iPad clients of owner and of syndicator across different ISPs and CDNs.



(a) On ISP X, CDN A (b) On ISP Y, CDN B

Fig. 2.18.: Rebuffering performance of California based iPad clients of owner and of syndicator across different ISPs and CDNs.

used [5, 71] measures of video delivery performance are the average bitrate of each view, and the rebuffering ratio (the fraction of the view that experiences rebuffering).

Fig. 2.17 shows the distribution of average bitrates observed by iPad clients of a syndicator (S7 in Fig. 2.16) and the owner, for our selected episode, across two different ISP/CDN combinations in March 2018. Further, we restricted the geo-location of clients to California, USA. Consistently, the content owner's clients get much better average bitrates: at the median, the average bitrate of the owner's clients is 2.5× that of the syndicator. Interestingly, clients of the owner also perceive

lower rebuffering ratios (Fig. 2.18), with almost 40% lower rebuffering at the 90-th percentile. We observed similar results for other device, ISP and CDN combinations.

While we cannot comprehensively answer why syndicators select widely varying bitrates, in this case, discussions with the owner and syndicator revealed that the owner selected its bitrates to provide better experience for its users, while the syndicator's choices were dictated in part by the increased storage and encoding costs required for higher bitrates.

Redundancy in CDN storage. Independent syndication can also result in redundant storage in CDNs. In this section, we explore this for a popular series syndicated by two syndicators from the owner. In this case, the owner stores the series content on two CDNs A and B, and uses 9 bitrates. One of its syndicators stores the same series on 3 CDNs A, B and C, but encodes the content using 7 bitrates. Another syndicator stores the content on A, B and another CDN D, but encodes the content using 14 different bitrates. These different management choices for the same content arise largely because of the independent syndication model.

We focus on a setting where publishers proactively push video content to a CDN origin server which serves cache misses from CDN edge servers [72]. This setting is commonly used in practice, especially for popular video content. We quantify the redundancy in storage in CDN origin servers. While there is likely some redundancy in edge servers as well, this is harder to quantify as that depends on content access patterns.

Quantifying storage redundancy with independent syndication. To quantify redundant storage, we first compute the total storage required for the entire video series by (i) downloading the manifest files of each episode of the series for each of the three publishers; (ii) multiplying each bitrate in the manifest file by the duration of the episode, and summing over all bitrates to obtain storage per episode; and (iii) summing across episodes. This results in a total storage requirement of 1916 TB across the 3 publishers (content owner and two syndicators) for each of the common CDNs (A and B).

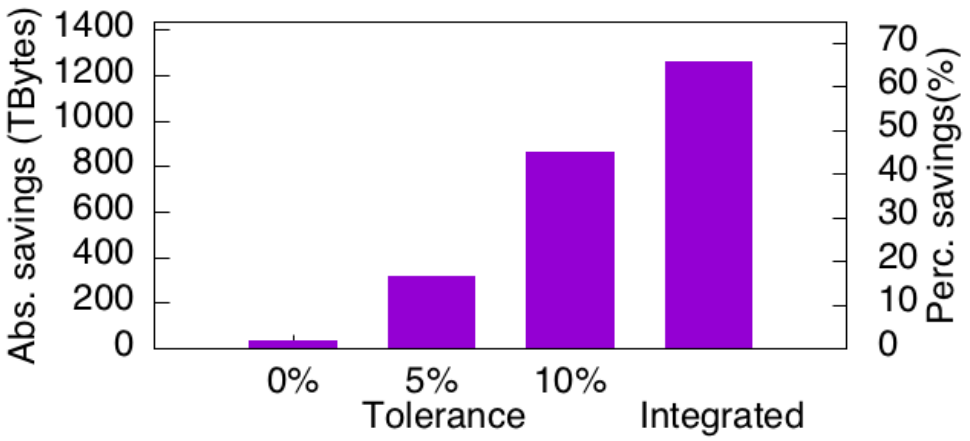


Fig. 2.19.: Storage savings under different syndication models for content served by an owner and two syndicators.

We next explore the storage savings achievable for this series if a CDN removes redundant copies of chunks with the same, or similar bitrates (those within a small *tolerance* factor). This is motivated by the observation that the syndicators and content owners often have similar or identical bitrates (Fig. 2.16). This occurs in practice because, even though publishers make independent bitrate choices, they tend to follow guidelines recommended by streaming protocol specifications. For example, the HLS specifications recommend that publishers make available at least one bitrate under 192 Kbps and that each successive bitrate be within a multiplicative factor of 1.5-2× of the previous [44].

Fig. 2.19 (left three bars) shows the absolute and percentage storage savings with the above model. Even with a 5% tolerance, CDNs A and B can save 316.1 TBs (16.5%) each, and at 10%, they can each save 865 TBs (45.2%).

Integrated syndication. Our broker reports that at least one or two of the publishers in our dataset have attempted *integrated syndication*, in which the owner's content delivery mechanism is integrated into the syndicator's playback software. There are two variants of integrated syndication: (i) *API integration*, where the

syndicator uses the owner's manifest file and CDN; and (ii) *app integration*, where the syndicator embeds the owner's app into its own.

The rightmost bar of Fig. 2.19 shows that with integrated syndication, A and B each save 1257 TB (65.6%). In addition, especially with app integration, syndicators cannot choose different bitrates than content owners, so performance differences similar to Fig. 2.17 are unlikely to arise.

While integrated syndication has potential, many logistical challenges must be addressed to make it a reality. For instance, with app integration, syndicators have to integrate apps for every owner they syndicate from. While API integration is potentially logistically easier, accounting mechanisms must be developed to distinguish CDN usage by clients of the syndicator and the owner. Future work should explore better ways to improve the management of syndication.

2.7 Related Work

Characterizing Video Services. YouTube and Netflix alone have been the subject of numerous studies over the years [73–83]. This body of work has studied several aspects, including 1) architecture, serving strategy and its evolution, 2) characterization of videos in terms of encoded bitrates, total number of videos, popularity, caching, and 3) the user access patterns and quality of experience *etc.* Ghasemi *et al.* [65] conducted an in-depth study of Yahoo's video serving infrastructure to reveal problems in different points in the video delivery pipeline. Other work has also examined different types of video services including a Pay-TV [84], cellular video [85], an on-demand service [86] and user-generated live streaming services [50, 51]. These papers focus on one or a handful of online publishers, but our work focuses on characterizing management plane practices across a large number of online video publishers.

Industry surveys. Because of the growing interest in Internet video, several industry surveys [10–13] have examined the video ecosystem. A 2017 industry study by Bitmovin surveyed 380 video developers (individuals or companies associated with

the Internet video business in various ways). This study characterized streaming protocols, encoding formats, devices, DRM *etc.*. An earlier 2016 study [12] characterized aggregate distributions across many of the same dimensions as [10]. Another prior industry survey [14] and an anecdotal report [87] discuss the percentage of publishers that use multiple CDNs (but do not discuss the number of CDNs used, or the fact different CDNs may be used for live and VoD). While valuable, none of these reports weight findings by view-hours, or present trends across publishers categorized by view-hours, or present longitudinal analyses, as we do. Both of these methodological differences result in new findings and add insights to known trends. Finally, while [13] does presents some trends with respect to device usage, other dimensions are not considered. In addition, we go beyond all these reports by considering the implications of these trends for management complexity, and syndicated content.

Quantifying diversity and complexity. Prior work has captured diversity of mobile users [88] and apps [89] and the complexity of web pages [67] and routers [68]. While we draw inspiration from these works, our focus is on video management planes, a different domain.

2.8 Conclusion

The Internet video management plane, which is responsible for packaging video content and for ensuring playback across different devices, has received relatively little research attention. Using data collected by a large broker over a period of two years from over one hundred video publishers, we find that there exists significant diversity across the three aspects of video management we study (packaging, CDN usage, and playback device usage): large publishers support 3-4 protocols, 5 CDNs and 5 different device types. This diversity adds complexity to several management tasks such as failure triaging, software management, and encoding. We find that complexity metrics for these tasks are sub-linearly related to the number of view-hours. Finally, the structure of today's management planes can lead to variable delivery performance

for syndicated content. Integrating management planes for syndicated content can avoid this as well as reduce CDN origin server storage requirements, and future work can explore mechanisms for integrated syndication, as well as analyze new complexity metrics, and approaches to cope with diversity and reduce management complexity.

3. OBOE: AUTO-TUNING VIDEO ABR ALGORITHMS TO NETWORK CONDITIONS

3.1 Introduction

Internet video forms a major fraction of Internet traffic today [26], and delivering high quality of experience (QoE) is critical since it correlates with user engagement and revenue [4–6]. To deliver high quality video across diverse network conditions, most Internet video delivery uses adaptive bitrate (ABR) algorithms [18, 90, 91], combined with HTTP chunk-based streaming protocols (*e.g.,* Apple's HTTP Live Streaming, Adobe's HTTP Dynamic Streaming). ABR algorithms (a) chop a video into *chunks*, each of which is encoded at a range of bitrates (or qualities); and (b) choose which bitrate level to fetch a chunk at based on conditions such as the amount of video the client has buffered and the recent throughput achieved by the client. Within this general framework, ABR algorithms differ in how bitrate level selection decisions are made, and these decisions impact metrics such as the average bitrate or the rebuffering ratio. We call these QoE metrics, because they have been shown to correlate well with QoE [5], but other perceptual video quality metrics [92] may also influence QoE.

ABR algorithm design remains an active research area because content providers continue to be interested in *improving the performance of video delivery.* Current ABR algorithms perform well on average, but some users can experience poor delivery performance as measured by the QoE metrics. These users suffer because ABR algorithms *have limited dynamic range*: they do not perform uniformly well across the range of network conditions seen in practice because their parameters are sensitive to throughput variability (§3.2).

Contributions In this thesis, we present the design of Oboe[1] (§3.3), a system that takes the first step towards overcoming these hurdles. Oboe improves the dynamic range of ABR algorithms by automatically tuning ABR behavior to the current *network state* of a client connection, specifically to throughput and throughput variability.

Oboe's design is based on the observation made by prior work [93–97] that TCP connections are well-modeled as traversing a piecewise-stationary sequence of network states (§3.3.1): the connection consists of multiple non-overlapping segments where each segment is in a distinct stationary network state. For each possible network state, Oboe *pre-computes, offline, the best parameter configuration* for a given ABR algorithm (§3.3.2). It does this by subjecting the algorithm, for each state, to different parameter values, and picking the one that results in the best performance. Then, during video playback, Oboe continuously uses a change-point detection algorithm to detect changes in network state and selects the parameter identified by the offline analysis as best for the current state. Thus, if a video session encounters varying network state during its lifetime, Oboe automatically *specializes* the ABR parameter to each state (§3.3.3).

We have implemented Oboe and demonstrated several aspects of its performance through testbed experiments and trace driven simulations. First, Oboe significantly improves performance of QoE metrics for three qualitatively different ABR algorithms, one that makes bitrate switching decisions on buffer occupancy alone (BOLA) [18], another that uses both throughput and buffer occupancy (HYB, a widely deployed algorithm), and a third that also optimizes decisions across a finite lookahead horizon (RobustMPC) [91]. In each of these cases, Oboe results in significant improvement. For instance, Oboe reduces sessions with rebuffering from 33.3% to 5.3% relative to RobustMPC while also significantly improving a composite QoE metric.

Oboe, when applied to RobustMPC, also performs significantly better than a newly proposed approach called Pensieve that learns, from real traces (using rein-

[1]In orchestras all instruments tune to the Oboe.

forcement learning), how to adapt to a variety of network conditions. For nearly 80% of the sessions in our dataset, Oboe improves the same composite metric, with benefits exceeding 20% for 25% of the traces. Compared to Oboe, which can specialize parameters to individual network states, Pensieve is unable to specialize across the entire range of network throughputs. We have tried training specialized Pensieve models for different ranges of network throughputs and dynamically switching models based on estimated session throughput. This helps, but a significant gap between the two approaches still remains (§3.4.4).

While a variety of viable pathways exist to deploying Oboe, we focus on an architecture where Oboe and the entire ABR logic are deployed on the cloud which enables rapid evolution and fine-grain customizability. We show the viability of this architecture with results from a pilot deployment.

3.2 Background and Motivation

The Internet video delivery ecosystem consists of hundreds of *content publishers* and hundreds of *client side applications* that stream video content to diverse user devices. Publishers, content delivery networks, and users all seek to improve user quality of experience (QoE). There are many factors that affect QoE including start up latency, the average bitrate for a video session, as well as the rebuffering ratio (the percentage of time playback is stalled because of drained buffer) [5]. Video players improve QoE using adaptive bitrate (ABR) algorithms which select bitrates for each chunk while (1) ensuring the bitrate seen by the user is as high as possible and (2) avoiding rebuffering events at the client. Some ABR algorithms may also try to minimize the number of bitrate switches to make the playback smooth.

Content publishers serve different types of content including VoD (Video on Demand) or Live broadcasts. They may also serve streams of different qualities ranging from HD (high definition) to SD (standard definition). These differences impact how they serve videos. For example, publishers who serve VoD content can use player

buffers as large as 4 minutes [90], whereas publishers serving live content may have a time-to-live[2] requirement between 15-45 seconds. Similarly, based on the quality of streams they serve, publishers may use different bitrate levels or chunk sizes. Further, publishers may have different QoE objectives. For example, some may strictly prefer to minimize rebuffering and others may relax their tolerance for rebuffering to prioritize higher bitrates. We use the term *publisher specifications* to denote their choice of bitrate levels, chunk sizes, content type, and rebuffering tolerance.

3.2.1 Background on ABR Algorithms

ABR algorithms fall in two broad categories: (i) those that use both prediction of network throughput and buffer occupancy [33, 91, 98]; and (ii) those that are primarily based on buffer occupancy [18, 90]. Within the above two categories, ABR algorithms can be designed using approaches ranging from heuristics to stochastic optimization. In §3.4, we discuss a recently proposed ABR algorithm based on a qualitatively different approach, reinforcement learning [17].

MPC: Throughput prediction and buffer occupancy with look-ahead. Selects bitrate by solving an optimization problem MPC [91] predicts throughput of future chunk downloads based on throughput samples of recently downloaded chunks, then uses this predicted throughput to select bitrates to optimize a given QoE function (§3.4) over a look-ahead window of 5 future chunks. The aggressive version of the algorithm (FastMPC) directly uses a throughput estimate obtained using a harmonic mean predictor. To compensate for throughput prediction errors, a more conservative version, RobustMPC, reduces predicted throughput by a *discount* factor $1 + d$, where d is the maximum error in throughput predictions experienced in the last five chunk downloads.

[2]For live content, the time between the event and its broadcast to users. This bounds the maximum buffer that a player streaming a live event can build.

BOLA: Buffer occupancy, selects bitrate by solving an optimization problem BOLA is a buffer-based algorithm used in Dash.js [99], so it does not employ throughput prediction in making bitrate decisions [18]. It also models bitrate selection as an optimization problem which it solves for a given value of the buffer. It uses a parameter γ which is a ratio of (i) a minimum buffer threshold, below which it downloads the lowest bitrate and (ii) a target buffer threshold which it tries to maintain. Conceptually γ controls how strongly the ABR should avoid rebuffering [18]. Higher values of γ make the algorithm conservative.

HYB: Throughput prediction without lookahead. Selects bitrate using a simple heuristic An algorithm widely used in production (§3.5), HYB considers both the predicted throughput and current buffer occupancy (HYB is short for hybrid). For each chunk, HYB picks the highest bitrate that can avoid rebuffering. Specifically, if $S_j(i)$ denotes the size of chunk j encoded at bitrate i, B is the predicted throughput based on past samples, and L the length of the buffer. HYB picks the largest bitrate i such that $\frac{S_j(i)}{B} < L \times \beta$. Here, β can have values between 0 and 1 (higher values represent aggressive ABR behavior). β can be tuned to offset prediction errors in throughput and to compensate for the greedy nature of the approach which may make it susceptible to future buffering events owing to unexpected throughput dips.

3.2.2 Ensuring High QoE for All Users

Despite widespread deployment, ABR algorithms continue to be an active area of research [17, 18, 33, 90, 91, 98]. This is because, while deployed ABR algorithms work well on average, they do not work uniformly well across all network conditions. A key reason for this is that ABR algorithms have parameters (which we henceforth refer to as *configurations*) that must be set in a manner sensitive to network conditions. ABR algorithms need to run on many different networks, ranging from cellular and WiFi networks at one end, to high-speed broadband connections at the other. Given

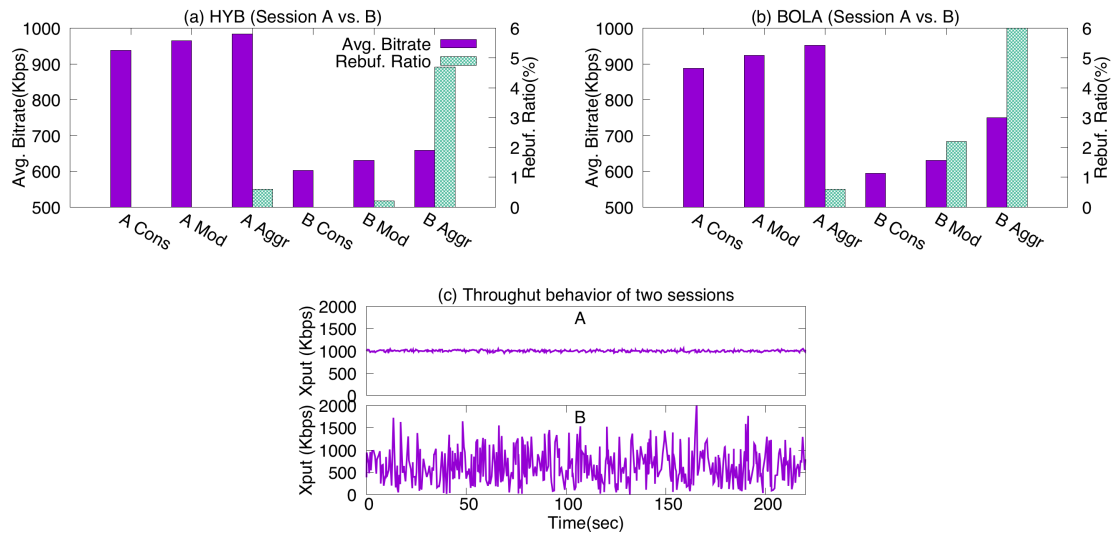


Fig. 3.1.: Performance of ABR algorithms using different configurations for two sessions with different throughput behaviors

this diversity, network conditions can vary significantly. Packet loss conditions can vary by an order of magnitude or more across the globe [100]. Network throughputs can also vary widely: for 90% of traces in a large dataset, the trace's maximum throughput is more than twice its average throughput. Yet, unfortunately, most ABR algorithms today either employ fixed configurations or simple heuristics to adapt these configurations (§3.2.1).

Figures 3.1(a) and 3.1(b) show how the choice of ABR configuration depends on network conditions. Figure 3.1(a) shows the bitrate and rebuffering ratio for two client sessions with the HYB algorithm for three different values of its β parameter, Cons (Conservative), Mod (Moderate), and Aggr (Aggressive). The throughput behavior of the two sessions is shown in Figure 3.1(c). If a publisher prefers to eliminate rebuffering, Mod is suitable for session A, but Cons is better for session B. Figure 3.1(b) shows that BOLA behaves similarly, with Mod being the preferred setting for session A and Cons for session B, to avoid rebuffering.

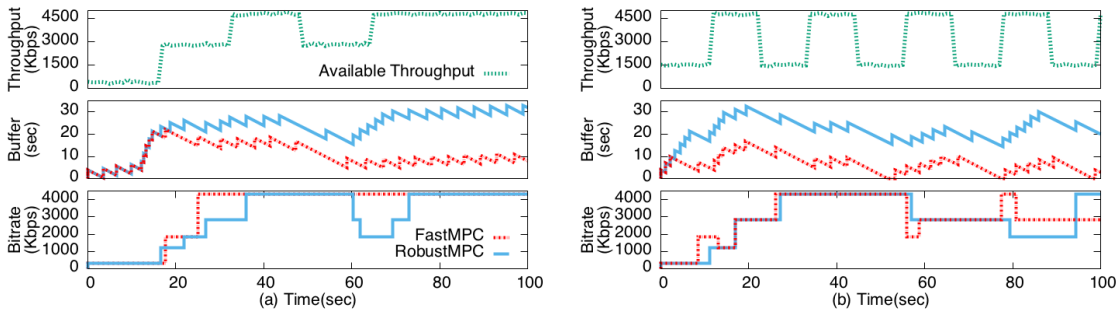


Fig. 3.2.: Illustrating how policy for setting discount factors in MPC impacts performance for different traces

Figures 3.2(a) and 3.2(b) show the difficulty in setting the discount factor with MPC, by comparing the performance of FastMPC (no discount factor), and RobustMPC (discount factor set by local heuristic) for two throughput traces with different characteristics. In each figure, the top subgraphs show the available throughput (green curve) and the throughput estimate of FastMPC (red) and RobustMPC (blue). For the left graph, although the throughput is generally good, the sudden variations force RobustMPC to make overly conservative bitrate decisions, as well as incur more bitrate switches. (bottom subgraph). In contrast, in Figure 3.2(b), the quicker and more frequent throughput changes (top subgraph) result in FastMPC experiencing rebuffering (middle subgraph), while RobustMPC does not. This is just one example illustrating the difficulty in picking parameters – in our evaluations (§3.4), we found that RobustMPC was itself too aggressive when selecting discount factors for some traces.

While this section uses synthetic traces for illustrative purposes, our evaluations with real traces (§3.4) more extensively demonstrate the limitations of current approaches with respect to selecting parameters and the benefits of automatically tuning ABR parameters to network conditions.

3.3 Oboe Design

Oboe aims to ensure good QoE for all users by enabling ABR algorithms to perform better across a wide range of network conditions. The configurations of many ABR algorithms are sensitive to *network state*, specifically to the value and variability of the available throughput between the client and the video server. For example, β in HYB should be smaller when available throughput is highly variable, while γ in BOLA should be higher. This explains why the algorithms perform differently for different values of parameters on a given client trace (§3.2.2). However, a line of prior work [93–97] has observed that network connections are *piecewise stationary*: that is, connections can be in one of several distinct states (§3.3.1), where each state is distinguished by *stationarity* in the statistical sense (informally, a process is stationary if its statistical properties including mean and variance do not change over time - see [101] for a more formal definition).

Oboe leverages the piecewise stationarity of network connections to address the key challenge of sensitivity of configurations to network conditions. It does so using a two stage design: (a) an *offline* stage where it pre-computes the best configuration choice for each (stationary) network state (§3.3.2), and (b) and an *online* stage, where during a session, it detects changes in network state and applies the pre-computed best configuration for the current (stationary) state (§3.3.3). Oboe can also accommodate publisher specifications such as session type (live vs. video-on-demand, time-to-live requirements), bitrate levels or any explicit QoE tradeoffs (*e.g.*, preference between rebuffering and average bitrate) (§3.3.2), by using these to influence the selection of the best configuration for each (stationary) network state in the offline stage.

3.3.1 Representing Network State

Most ABR algorithms today adapt bitrates based on the throughput (more precisely, goodput) achieved by recently downloaded chunks. This perceived throughput

already accounts for network delays and loss-rates, as well as the dynamics of the underlying transport protocol.

The network throughput along a path is *not* necessarily a stationary process [93–97]: flows at the bottleneck along a path may change over time resulting in changes to available throughput, or the bottleneck itself may shift [96]. An analysis of the throughput traces used in our evaluations (§3.4) confirms the lack of stationarity when applied to the entire trace. We analyze throughput traces using the Augmented Dickey-Fuller (ADF [102]) test, a hypothesis test to check for stationarity in a time series. Our evaluations on a dataset of 15,000 video streaming throughput traces show that 59.5% were non-stationary (see §3.4.2 for details of the dataset), implying the presence of distinct mean and/or variance in different segments of the traces.

However, prior work [93–97] shows that TCP connection throughput *can* be modeled as a *piecewise* stationary process; the connection consists of multiple non-overlapping segments where each segment is stationary and often lasts for tens of seconds or minutes (*e.g.*, Figure 3.8). Moreover, Zhang et al. [94] show that the throughput in each segment may be modeled as an i.i.d. process.

Motivated by these observations, Oboe defines network state s by a tuple $< \mu_s, \sigma_s >$, where μ_s is the mean and σ_s the standard deviation of the client-perceived throughput in a (stationary) segment of the underlying TCP connection.

3.3.2 Offline Mapping of Network States

To map network states to their optimal ABR configurations, Oboe uses a pipeline (Figure 3.3) consisting of three components – the *ConfigEvaluator*, the *VirtualPlayer* and the *ConfigSelector*. The ConfigEvaluator takes a stationary throughput trace as input, which represents a particular network state, and drives the exploration of different ABR configurations over this trace. It does so by using the VirtualPlayer which models the dynamics of an actual video player. The VirtualPlayer interfaces with the ABR algorithm implementation and outputs the performance of different

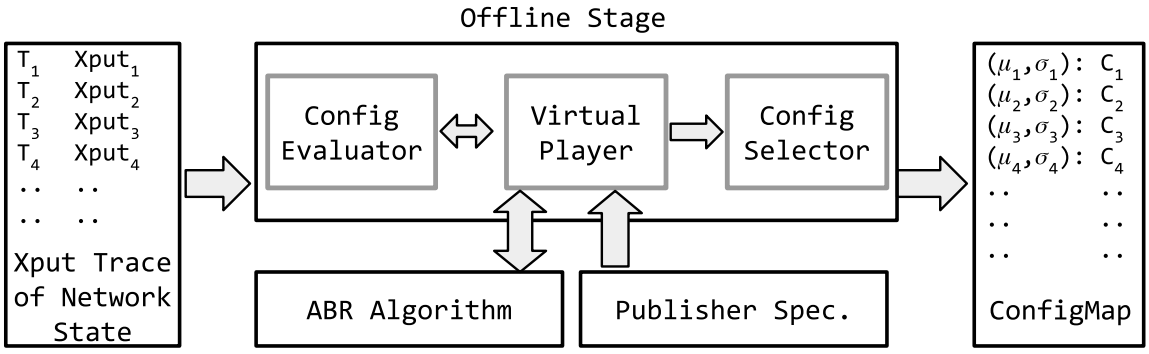


Fig. 3.3.: The logical diagram of the offline pipeline used by Oboe

configurations of the ABR. Finally, the ConfigSelector compares the performance of different configurations and builds a *ConfigMap*, which maps a given network state to the best configuration.

Generating throughput traces for ConfigEvaluator To explore configuration space of an ABR algorithm on each network state s, ConfigEvaluator needs a stationary throughput trace to represent s. To generate such a trace, we explored two different approaches. In one approach, we extracted stationary segments from real traces using offline change point detection ([103], described in §3.3.3). Change points capture points where the distribution changes. However, because we are not guaranteed *coverage* (i.e., not all states might be observable in real traces), we also explored a second approach which involved generating a *synthetic* trace for each s with s's mean and standard deviation, assuming a Gaussian distribution for the throughput samples. This was motivated by Dinda *et al.* [95] who showed that the throughput of TCP flows of the same size in a given stationary segment may be modeled as a Gaussian distribution (also see §3.3.1). More recent work also shows that TCP throughput is well modeled as a Markov process, each of whose states may be modeled as a Gaussian distribution [23]. We found that Oboe with synthetic traces

performed comparably to stationary segments from real traces. So, ConfigEvaluator uses synthetic traces.

Specifically, ConfigEvaluator quantizes both mean and standard deviation of throughput using a quantum (in our experiments, of 50 Kbps), resulting in states (in our experiments, 10,000), spread over a two dimensional space (in our experiments, 0.05-10 Mbps) of throughput and standard deviation. For each state, we generate a synthetic stationary trace. We found that the benefits of finer quantization are marginal.

Estimating ABR performance with VirtualPlayer and publisher specifications Oboe uses VirtualPlayer, a trace-based simulator that mimics the behavior of an actual video player without downloading or rendering actual videos. It takes as input a throughput trace and outputs the QoE performance metrics of a video session for a specified ABR algorithm. We have validated VirtualPlayer in §3.4.7. In designing VirtualPlayer, we have decoupled ABR logic (Figure 3.3), so the same implementation of the ABR logic can be used in Oboe's offline and online stage. Further, this design provides an interface to the ABR designer through which they can easily integrate their ABR algorithm with Oboe without having to know about Oboe's internals.

The VirtualPlayer also takes into account publisher specifications for bitrate levels, player buffer sizes (determined by time-to-live requirements) and chunk size. These specifications are used by VirtualPlayer when it executes ABR algorithms on the input traces, ensuring that the resulting ConfigMap meets the publisher specifications. Finally, Oboe also allows the publisher to optionally express an explicit QoE tradeoff such as maintaining the rebuffering under a desired threshold $x\%$. Oboe derives a ConfigMap that meets the rebuffering threshold in a best effort manner. We evaluate the efficacy of this flexibility in §3.4.7.

Building the ConfigMap using ConfigSelector To build the ConfigMap, the ConfigEvaluator drives the exploration of different configurations for an ABR algorithm. For a given network state s, ConfigEvaluator sweeps through possible configu-

rations of the ABR algorithm using the VirtualPlayer. For example, the β parameter in HYB can take values from 0 to 1, so ConfigEvaluator plays the trace for state s for multiple values of β (quantized for efficiency, see below) in this range.

For each parameter value c_i, VirtualPlayer outputs a *performance vector* $V_i =< v_1, v_2, \ldots v_m >$ where each v_k corresponds to the values achieved by c_i for a QoE metric (*e.g.*, bitrate, rebuffering ratio, and more generally join time and frequency of switching bitrates [5]). This set of performance vectors with the corresponding parameter values are then sent to ConfigSelector for picking the best configuration.

ConfigSelector takes the set of performance vectors and determines the best configuration from them using *vector dominance*. A configuration c_i is said to dominate c_j if V_i element-wise dominates V_j (*i.e.*, each element of V_i is better than or equal to the corresponding element of V_j). This step also takes into account any rebuffering tolerance, and ConfigSelector applies this tolerance to select the maximal performance vector. Deferring the selection of the maximal vector for a given rebuffering tolerance to this stage (instead of filtering vectors in the previous step) is beneficial: it minimizes recomputation by allowing Oboe to quickly compute a new maximal vector if the publisher changes the rebuffering tolerance. At the end of this stage, Oboe obtains the ConfigMap, a complete mapping of each network state to its corresponding optimal ABR configuration.

Two optimizations can be used to quicken the rate of exploration of the ConfigEvaluator. The first is to quantize the parameter sweep, so that configurations are evaluated at a coarser granularity. This trades off some performance for lower computational complexity. The second optimization is based on the observation that there is generally a monotonic relationship between parameter values and the performance. For instance, for HYB (§3.2.1), the rebuffering ratio and average bitrate are monotonically non-decreasing with the parameter β. Based on this observation, we can instead use an $O(\log n)$ binary search of the configuration space instead of doing a full $O(n)$ sweep of all configurations.

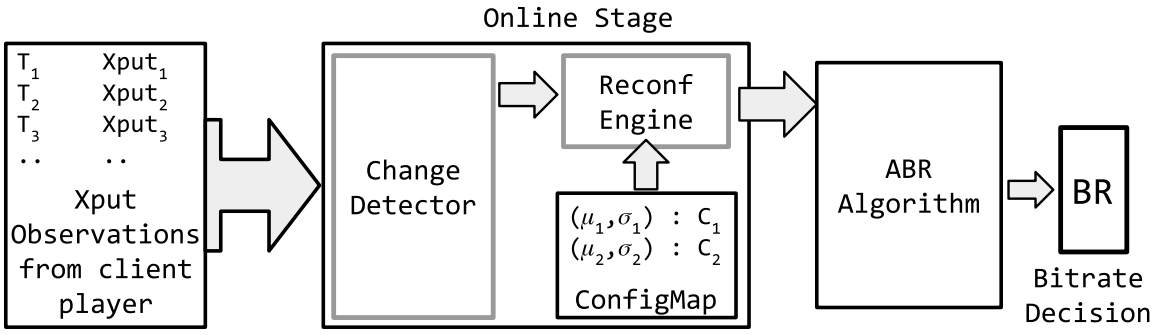


Fig. 3.4.: Logical diagram of Oboe's online pipeline

3.3.3 Online ABR Tuning

Oboe uses the ConfigMap generated offline, and live throughput measurements from the video player to dynamically change ABR configurations during a video playback. It does this by using an *online change point detection algorithm* [104]. This algorithm identifies, in an online fashion, if the distribution of the throughput samples has changed significantly, signaling a state transition. When a change point is detected, the algorithm also provides the new state s's mean and standard deviation. Oboe's ChangeDetector (Figure 3.4) implements the change point detection algorithm, and the ReconfEngine is responsible for updating the ABR configuration based on a new network state and the ConfigMap.

Change point detection algorithms Such algorithms analyze a time series and check if there are regions in the time series where the underlying distribution of the data changes to a different set of parameters. Offline change-point methods require the full time series to be available, whereas online methods work with a continuous stream of samples as they become available. We focus on online methods, since Oboe identifies change points for an in-progress session and dynamically changes configurations.

While several techniques exist for change point detection [105–110], we focus on probabilistic methods [104, 111–113]. Further, we use a Bayesian online probabilistic change-point detector [104] for two reasons. First, in [104], a sequence of observations can be partitioned into non-overlapping states such that the observations are i.i.d. conditioned on a given network state s. This view aligns well with the way we have defined a network state (§3.3.1). Further, the algorithm is fast and requires no prior knowledge about the data stream, matching our scenario. We use the implementation provided in [103] and integrate it with the ChangeDetector.

Detecting changes in network state During a video session, ChangeDetector is continually fed with a series of observations of the network throughput, which it uses to detect state changes. ChangeDetector calculates throughput and standard deviation by only considering those samples which belong to the current state. To generate inputs to ChangeDetector, one approach is to use each downloaded chunk to obtain a single throughput sample. However, this may be too coarse-grained, and prevent detection of changes in network state that occur during the chunk download. Instead, we use fine grained samples recorded at periodic intervals (tens of milliseconds) during the download of each chunk. Players such as Dash.js already periodically log intermediate throughput samples during a chunk download, so obtaining these samples does not incur any additional overhead. We only need to modify players to report these samples to Oboe. The set of samples are provided to ChangeDetector after the chunk download, and any change in state is only detected at the end of the chunk download. This is acceptable since any action that can be taken by the ABR algorithm (such as a bit rate switch) only impacts subsequent chunks. In the rarer case that an ABR algorithm abandons the download of a chunk that takes too long, the report is sent when the chunk download is abandoned. §3.4.8 evaluates the overheads of ChangeDetector.

An alternative approach to changing configurations is to use an exponentially weighted moving average (EWMA) of the mean and standard deviation of throughput

samples and to lookup the corresponding configuration. We experimented with such an approach and found its performance unsatisfactory. The approach can result in continual and unnecessary reconfigurations, since throughput may vary across samples even when the network is (statistically) stationary. Damping these changes can result in slow reaction times when a reconfiguration is actually beneficial. In contrast, Oboe (i) models the underlying TCP connection as a sequence of states; (ii) does not make changes to the configuration within a given network state; and (iii) only reconfigures when a state change is observed.

Reconfiguring ABR Algorithm When a change in the network state is detected, the ChangeDetector signals the change and the new network state s to the ReconfEngine. The ReconfEngine then searches a neighborhood of radius r in the ConfigMap to select the configuration to use for state s. Specifically, if state s is a point in a 2-dimensional space of average throughput and standard deviation of throughput, then it picks the most conservative ABR configuration within a search radius r around s. It does this for two reasons. First, because Oboe quantizes the network states, it might not have precomputed the best configuration for s. Second, the estimated new network state s may have some error, for example, due to inefficiencies in the client download stack [65]. Given these sources of uncertainty, Oboe chooses to be safe in its selection of the best configuration for s. Finally, ReconfEngine configures the ABR algorithm, and the reconfigured ABR algorithm is ready to compute the bitrate decision to be used for the next chunk at this point.

3.4 Evaluation

In this section, we demonstrate Oboe's ability to auto-tune three existing algorithms: RobustMPC, BOLA and HYB. We also compare an Oboe-tuned RobustMPC to Pensieve [17].

3.4.1 Metrics

The performance of a video session depends on multiple factors. *Average bitrate* and *rebuffering ratio* were found to have the most impact on user quality of experience [5], though other factors such as changes in bitrates during a session can play a role [5]. There is no consensus on how to best capture a user's QoE. Consequently, ABR algorithms today are designed to optimize different metrics. For instance, HYB and BOLA primarily maximize average bitrate subject to low rebuffering. In contrast, other algorithms [17, 91] have been designed to optimize a QoE metric which is a linear combination of bitrate, rebuffering and bitrate changes (smoothness).

With Oboe, our primary evaluation goal is to demonstrate the extent to which it can improve the underlying metrics that an ABR algorithm is designed for. Thus, our evaluations with BOLA and HYB focus on average bitrate and rebuffering, while those with MPC+Oboe focus on the linear combination of QoE (which we refer to as `QoE-lin`, [91]), defined as follows. For a video with N chunks, let R_i be the bitrate chosen for chunk i. Then, the magnitude of bitrate changes M may be defined as $M = \sum_i^{N-1} |R_{i+1} - R_i|$. If the session experiences a total of T seconds of rebuffering, then, $\texttt{QoE-lin}(p, c) = \frac{1}{N} * \sum_i (R_i - pT - c * M)$, where p and c represent scaling penalties applied to rebuffering and changes in the session. This function may be viewed as the session QoE averaged over the number of chunks. For our videos that had a maximum bitrate of 4.3 Mbps, we use $p = 4.3$ and $c = 1$ as our default parameters (following previous work that set default rebuffering penalty equal to the maximum bitrate value [17, 91]).

Even when an algorithm optimizes a metric such as `QoE-lin`, *it is important to understand the distributions of underlying factors.* The underlying factors represent concrete application performance that publishers understand how to reason about. Moreover, a unified metric like `QoE-lin` can obscure important differences. For example, two sessions may have the same `QoE-lin` but different performance in underlying metrics, leading to varied user experience. So, we also present graphs of these metrics.

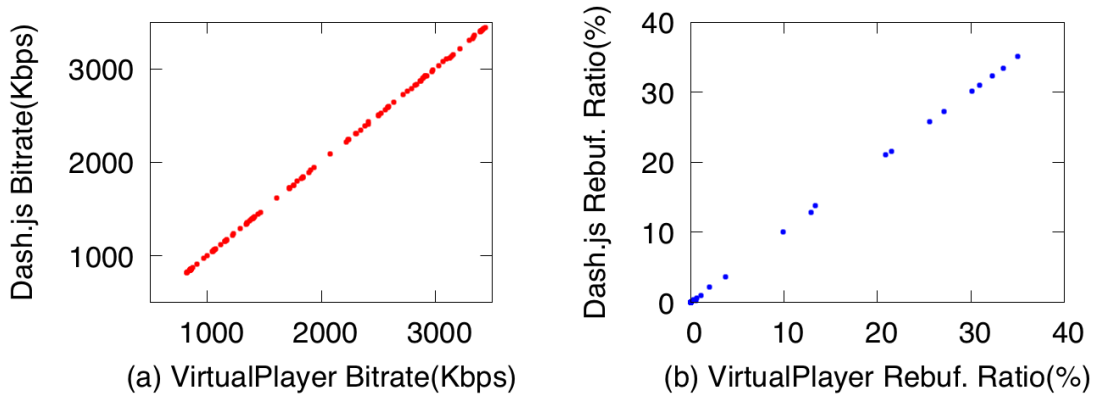


Fig. 3.5.: A scatter plot of average bitrate and rebuffering ratio between the VirtualPlayer and real Dash.js player

3.4.2 Methodology

Implementation For RobustMPC, we used the implementation available at [114]. Our implementation of BOLA [18] is from the Dash.js player. The implementation of HYB is a variant of the algorithm used in a large-scale deployment. These ABR algorithms and Oboe's online stage (change point detection and ABR reconfiguration) run on the server in our experiments. Our client runs the Dash.js video player (version 1.2), a reference player implemented in JavaScript by the MPEG-DASH forum [99]. We modified Dash.js to send client player state information (*e.g.* buffer length, video play state and throughput measurements) to Oboe (§3.5). This player runs on the Google Chrome browser (version 61) in our experiments. In §3.5, we show that Oboe can also be run as a cloud service.

Testbed setup Our evaluations measure ABR performance by delivering a video stream (the "EnvivioDash3" video from the MPEG-DASH reference videos [115]) from a video hosting server to a client, while varying network conditions using throughput traces from real user sessions. We use bitrates $\{300, 750, 1200, 1850, 2850, 4300\} kbps$ with a 4 second chunk duration and total length of 192 seconds. We focus on this

video as it has been used in prior work [17], and we do not consider videos of longer duration because we only have throughput traces available for a video publisher that serves short music videos (as we discuss below). The video is hosted on an Apache server. Both the server and client software run on the same 8-core, 4 Ghz, Intel i7 commodity desktop with 12 GB RAM running Ubuntu 16.04. Between server and client, we emulate different network conditions using the Chrome DevTools API [116]. This allows us to control the upload/download throughput as well as latency using the Chrome-Remote-Interface based on throughput traces [117]. We use 571 throughput traces[3] from our dataset (discussed below) for this emulation. All our testbed experiments use a client buffer of 2 minutes.

Datasets We use throughput traces from real user sessions collected over a three month period. Each trace contains the individual chunk sizes and their download times for on-demand video sessions from a publisher that serves short (4-6 minute) music videos. We derive throughput by dividing the chunk sizes by their download durations. The traces contain sessions that used desktops with wired connections and also sessions on mobile devices using WiFi or cellular connections. Like previous work [17, 91], we primarily focus on traces that have less than 6 Mbps average throughput, since this is the regime where bitrate switching decisions are likely to have QoE impact. We filtered out traces which were too short for playing our entire 192 second video, after which we obtained 5K traces from wired desktops and 4K sessions from WiFi or 3G/4G mobile devices. Our testbed experiments use a subset of 571 traces with roughly the same number of traces sampled from each of desktop and mobile clients.

VirtualPlayer setup Recall that Oboe uses the VirtualPlayer to obtain a ConfigMap for any ABR algorithm. Since the majority of our results use an actual testbed with the Dash.js player, the benefits of Oboe in our evaluation results already arise

[3]Available at https://github.com/USC-NSL/Oboe

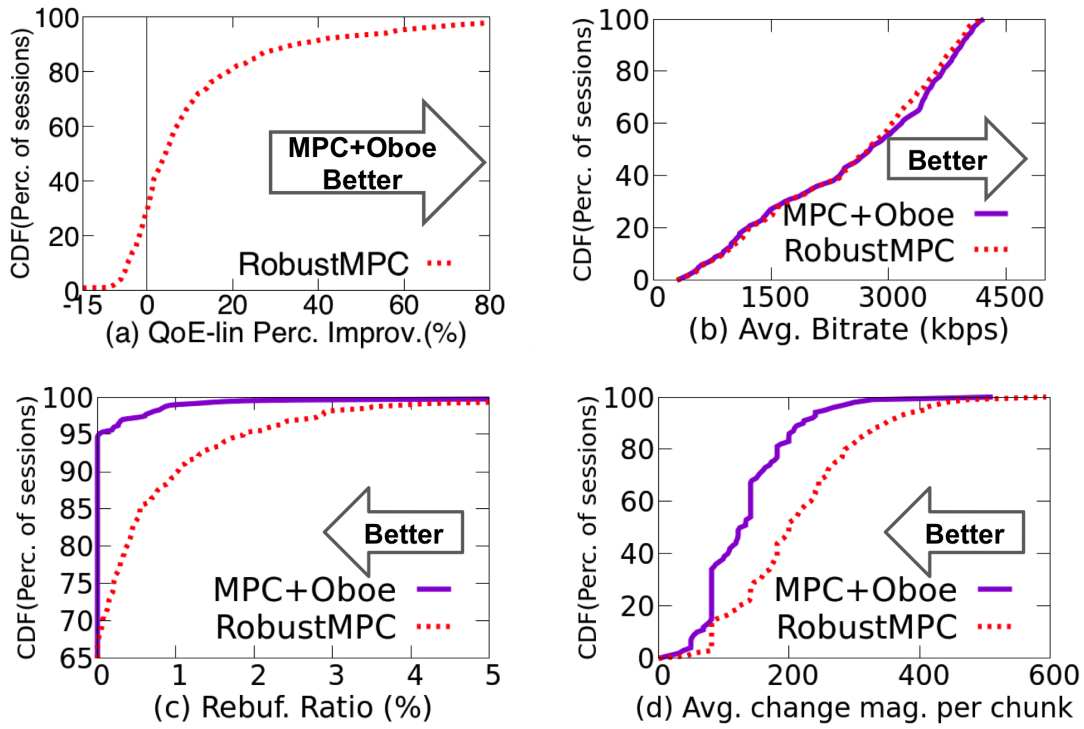


Fig. 3.6.: The percentage improvement in `QoE-lin` of MPC+Oboe over RobustMPC for the Testbed experiment. The distribution of average bitrate, rebuffering ratio and bitrate change magnitude for the schemes is also shown.

despite any inaccuracies in building the ConfigMap on account of using the VirtualPlayer. That said, we have also verified that the VirtualPlayer does a good job of tracking the performance of the actual ABR algorithms. For instance, Figure 3.5(a) and 3.5(b) demonstrates this for the HYB algorithm. The figures shows the correlation for the average bitrate and rebuffering ratio for 100 throughput traces randomly sampled from our dataset using HYB on the VirtualPlayer compared to using an actual Dash.js player. For both metrics, the graph closely tracks the $y = x$ line indicating close correlation. Given these close correlations, we use the VirtualPlayer in §3.4.7 to explore Oboe's performance over a larger range of diverse settings and our entire set of traces.

3.4.3 Oboe with RobustMPC

We now demonstrate that Oboe can be used to auto-tune RobustMPC, the best performing variant of the MPC algorithms. The resulting MPC+Oboe uses the best value of the discount parameter d corresponding to the current network state, replacing RobustMPC's online adaptation based on throughput estimates obtained over the past 5 chunks (§3.2).

Figure 3.6(a) shows the CDF of the percentage improvement in `QoE-lin` of MPC+Oboe over RobustMPC.[4] MPC+Oboe improves `QoE-lin` for 71% of sessions, with an overall average `QoE-lin` improvement of 17.62% across all sessions. In particular, for 19% of the sessions, `QoE-lin` improves by more than 20%. For the sessions MPC+Oboe is unable to improve RobustMPC, its performance degradation is mostly under 8%. Figures 3.6(b), 3.6(c) and 3.6(d) show the constituent QoE metrics. While MPC+Oboe achieves distributionally similar bitrates as RobustMPC as shown in 3.6(b), it *significantly* reduces rebuffering across sessions: the number of sessions with rebuffering reduces from 33.2% to 5.3%. Further, it also achieves better playback smoothness by improving the median per chunk change magnitude by 38% (Figure 3.6(d)). Finally, Figure 3.7 shows the CDF of `QoE-lin` for MPC+Oboe and RobustMPC, and indicates MPC+Oboe distributionally performs better.

Figure 3.8 illustrates, using a single session, why MPC+Oboe performs better than RobustMPC. The top graph shows throughput as a function of time, which includes an initial stable state followed by a drop in throughput. The middle graph shows how the discount factor d of both RobustMPC, and MPC+Oboe vary (the predicted throughput for each system is reduced by a factor of $\frac{1}{1+d}$, where d is shown on the y-axis). During the initial stable state, when prediction errors are low, RobustMPC steadily lowers its discount factor leading to more aggressive bitrate selections (not shown). This results in a *rebuffering event* 44 seconds into the session (lowest graph shows buffer occupancy with 0 indicating a rebuffering event). In con-

[4]The increase in `QoE-lin` over RobustMPC relative to the absolute `QoE-lin` value of RobustMPC expressed as a percentage.

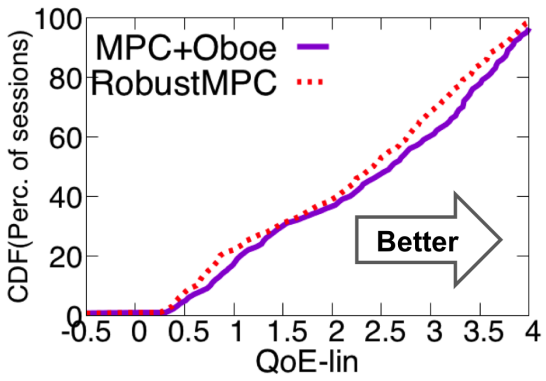


Fig. 3.7.: `QoE-lin` of MPC+Oboe compared to RobustMPC

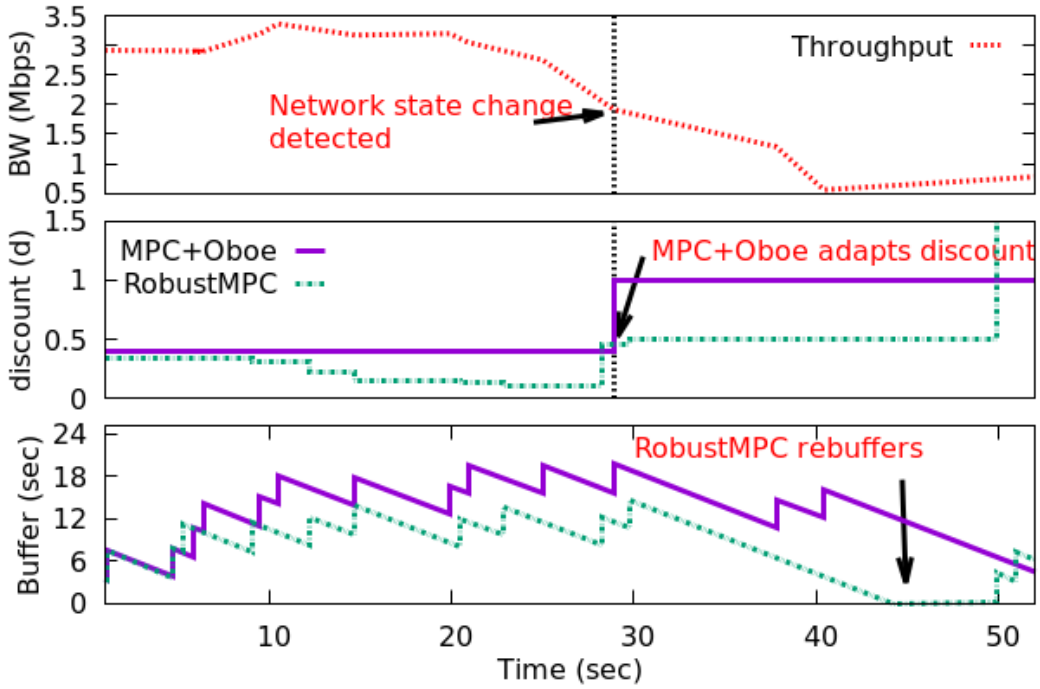


Fig. 3.8.: An example session showing how MPC+Oboe is able to outperform RobustMPC by reconfiguring the discount parameter when a network state change is detected.

trast, MPC+Oboe does not incur a rebuffering event and maintains a fixed d during the initial stable state. At 29 sec, it detects a change in the network state and adapts its discount factor, leading to more conservative bitrate selections.

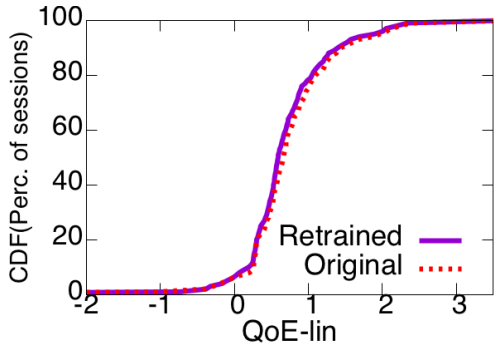


Fig. 3.9.: Validation of our training methodology for Pensieve.

3.4.4 Oboe vs. Pensieve

Pensieve [17] uses deep reinforcement learning [118, 119], a combination of deep learning with reinforcement learning [120], and has been shown to outperform existing ABRs, including RobustMPC [17] in some settings. Since MPC+Oboe outperforms RobustMPC as well, we explore how MPC+Oboe performs relative to Pensieve. Our experiments use the Pensieve implementation provided by the authors [114].

Pensieve Re-Training and Validation Before evaluating Pensieve on our dataset, we retrain Pensieve using the source code on the trace dataset provided by the Pensieve authors [114]. This helps us validate our retraining given that deep reinforcement learning results are not easy to reproduce [121].

We experimented with five different initial entropy weights in the author suggested range of 1 to 5, and linearly reduced their values in a gradual fashion using plateaus, with five different decrease rates until the entropy weight eventually reached 0.1. This rate scheduler follows best-practice [122]. From the trained set of models, we then selected the best performing model (an initial entropy weight of 1 reduced every 800 iterations until it reaches 0.1 over 100K iterations) and compared its performance to the pre-trained Pensieve model provided by the authors. Figure 3.9 shows CDFs of `QoE-lin` for the pretrained (Original) model and the model trained by us (Retrained). The performance distribution of the two models are almost identical over

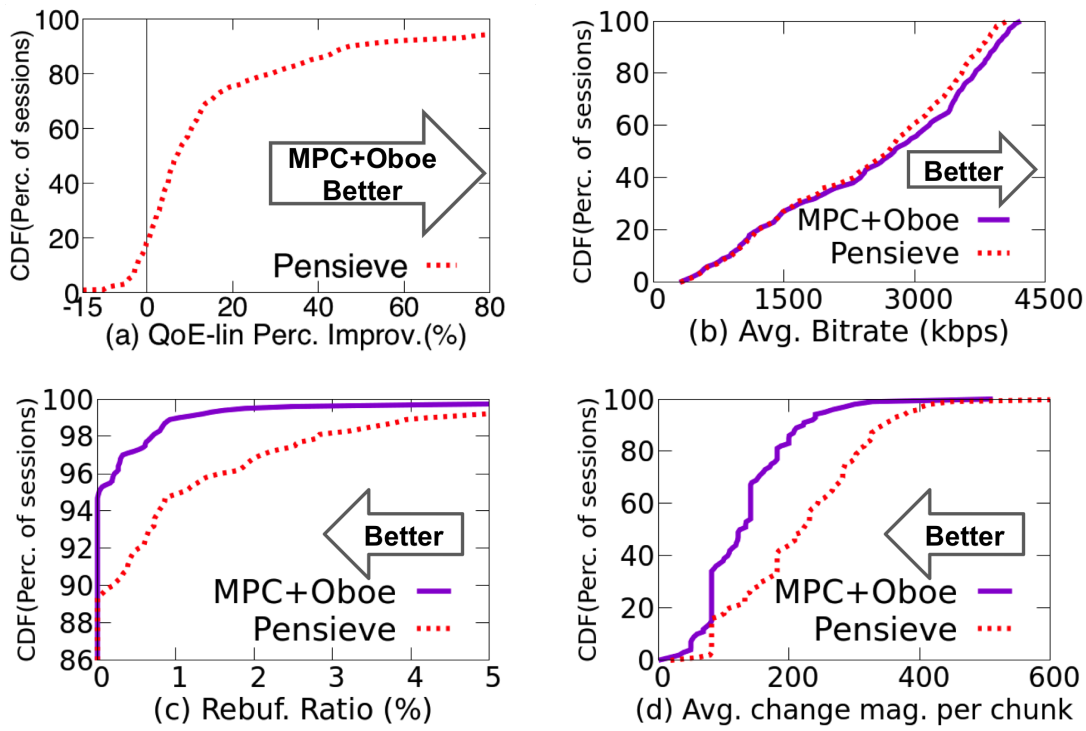


Fig. 3.10.: The percentage improvement in `QoE-lin` of MPC+Oboe over Pensieve for the 0-6 Mbps throughput region. The distribution of average bitrate, rebuffering ratio and bitrate change magnitude for the schemes is also shown.

the test traces provided by the Pensieve authors, thereby validating our retraining methodology.

Having validated our retraining methodology, we trained Pensieve on *our* dataset with the same complete strategy described above. For this, we pick 1600 traces randomly from our dataset with average throughput in the 0-6 Mbps range. The number of training traces, the number of iterations per trace, and the range of throughput are similar to [17]. We then compare Pensieve and MPC+Oboe over a separate test set of traces also in the range of 0-6 Mbps (§3.4.2).

Comparison with Pensieve Figure 3.10(a) shows the CDF of the percentage improvement in `QoE-lin` for MPC+Oboe over Pensieve. MPC+Oboe outperforms

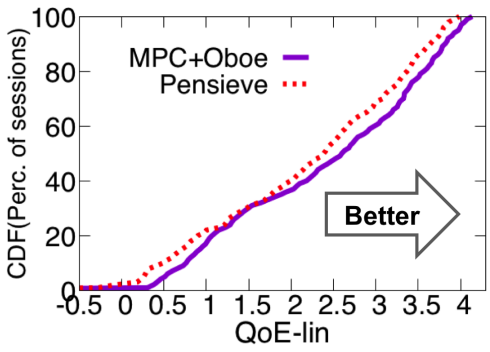


Fig. 3.11.: CDFs of `QoE-lin` for MPC+Oboe and Pensieve

Pensieve for 81% of the sessions, with a `QoE-lin` improvement of 23.9% in average across all sessions. 25% of the sessions achieve more than 20% `QoE-lin` improvement. For the sessions MPC+Oboe is unable to improve over Pensieve, the performance difference is mostly less than 5%. Figures 3.10(b), 3.10(c) and 3.10(d) show that MPC+Oboe distributionally outperforms Pensieve with respect to all underlying metrics. It reduces the number of sessions with rebuffering from 10.7% to 5.3%, reduces the median per chunk change magnitude by 43.9%, and improves median and 95th percentile average bitrate by 2.6% and 4.7% respectively. Finally, Figure 3.11 shows the CDF of `QoE-lin` for MPC+Oboe and Pensieve, and indicates MPC+Oboe performs distributionally better.

Analyzing Pensieve performance To understand where these performance improvements were coming from, we examined the relative performance of these two schemes in the 0-3 Mbps range (*i.e.*, traces having an average throughput between 0-3 Mbps). In this more constrained range of network conditions, we found that MPC+Oboe achieves bigger gains over Pensieve (average `QoE-lin` improvement in 0-3 Mbps is 46.23%). We hypothesize that this performance difference stems from the fact that Pensieve builds a single model which does not *specialize* to different throughput ranges.

To test this, we trained a separate Pensieve model only with traces that have an average throughput between 0-3 Mbps range and compared it with MPC+Oboe. Figure 3.12 shows the per session `QoE-lin` improvement of MPC+Oboe compared to Pensieve models trained for 0-3 Mbps (which we refer to as Pens-Specialized) and for 0-6 Mbps. The median `QoE-lin` improvement with MPC+Oboe over Pens-Specialized is 10.49%, while the median improvement over Pensieve is 19.9%. This indicates specializing the model does improve Pensieve's performance.

Thus, Pensieve's model is as yet unable to create specialized versions of itself based on the session characteristics. By contrast, Oboe specializes parameters for every network state and therefore performs better. We have also validated Pensieve's inability to specialize in several other ways: building a model for the 3-6 Mbps and showing that it performs better with test data in that range compared to a 0-6 Mbps model; checking that a 0-6 Mbps model performs better for data in that range compared to a 0-100 Mbps model; and ensuring that these results hold even when the training set is doubled. It is hard to pinpoint exactly why Pensieve is unable to learn to be more conservative in the 0-3 Mbps range; deep neural network models remain a black box despite efforts by the machine learning community to make these models more transparent [123], and obtaining such understanding may need further advances in interpretable deep learning models.

A model selector with Pensieve One way to improve Pensieve's specialization might be to train different models for different throughput ranges and use the model more suited to the network conditions. To test the efficacy of this approach, we used two models (specialized for 0-3 Mbps and 3-6 Mbps), and tried two different *model selectors*. Pens-SelMultiple switches models throughout the session, using the average throughput of the past 5 chunks. Pens-SelOnce starts with the 0-6 Mbps model, selects either the 0-3 Mbps or 3-6 Mbps model based on the average throughput of the first 5 initial chunks, and does not switch thereafter.

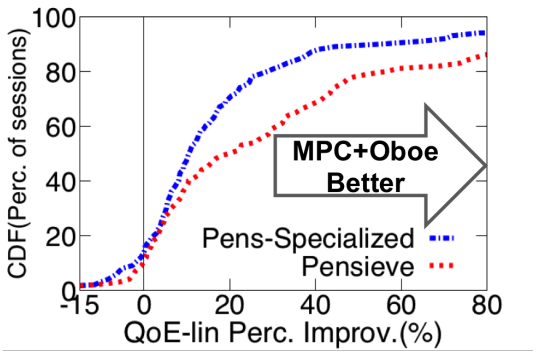


Fig. 3.12.: Benefits of specializing Pensieve models. Each curve shows the QoE improvement of MPC+Oboe relative to each Pensieve model.

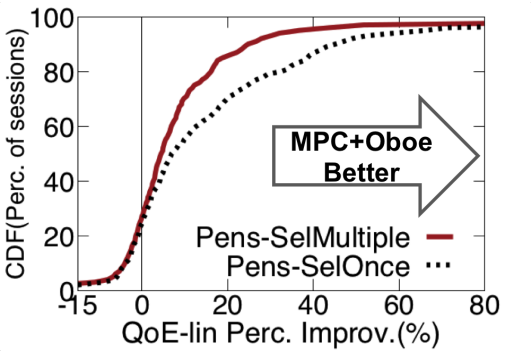


Fig. 3.13.: QoE improvement of MPC+Oboe over two ways of dynamically selecting from specialized Pensieve models.

Figure 3.13 shows CDFs of per-session `QoE-lin` improvement of MPC+Oboe over these selectors. MPC+Oboe is able to outperform both Pens-SelMultile and Pens-SelOnce, with average `QoE-lin` improvements of 14.2% and 24.32% respectively. Even though one of the model selection schemes offers some improvements over the 0-6 Mbps Pensieve model, the benefits are modest. We hypothesize that this behavior is due to the dynamic selection of distinct Pensieve models, which can interfere with reinforcement learning's decision choices, since, during training, the reinforcement learning algorithm assumes there is no such third party intervention.

3.4.5 Oboe with other ABR Algorithms

Oboe can also improve other existing ABR algorithms such as BOLA and HYB, which are designed to maximize average bitrate while minimizing rebuffering.

BOLA BOLA+Oboe tunes γ (§3.2), which determines how much the algorithm strives to avoid rebuffering. BOLA, as implemented in Dash.js, uses a fixed default value of $\gamma = -10.28$. Figure 3.14(a) and 3.14(b) show CDFs of per session performance improvement over BOLA with respect to average bitrate and rebuffering

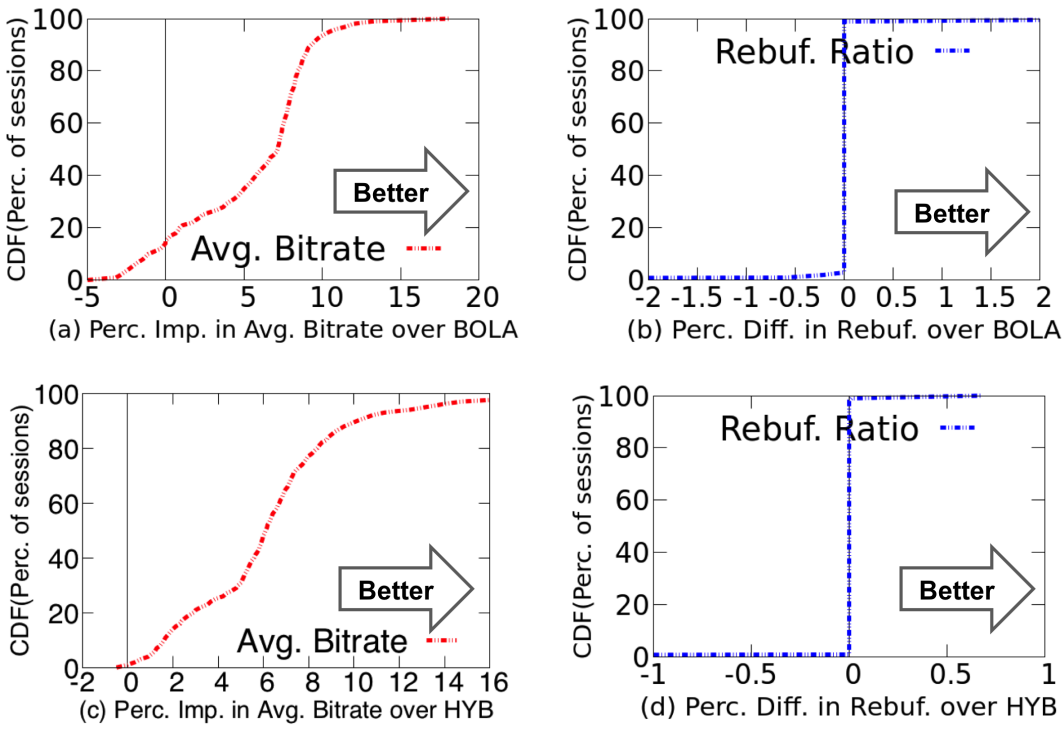


Fig. 3.14.: Percentage improvement in bitrate and rebuffering of BOLA+Oboe over BOLA (a),(b) and HYB+Oboe over HYB (c), (d)

ratio. BOLA+Oboe maintains the rebuffering ratio of BOLA while improving average bitrates for more than 83% of sessions with an overall increase of 7.2% in average across all sessions. For sessions where BOLA+Oboe does not outperform BOLA, its degradation is less than 3.1%.

HYB The performance of HYB is sensitive to the choice of β parameter, which HYB+Oboe tunes. In production, HYB uses $\beta = 0.25$, determined using A/B tests in a large-scale deployment. Figure 3.14(c) and 3.14(d) show CDFs of per session performance improvement of average bitrate and rebuffering ratio over HYB. As with BOLA, HYB+Oboe maintains similar rebuffering ratios as shown in 3.14(d), but improves bitrates for 98% of sessions with an overall average bitrate improvement of 8.32% in average across all sessions.

3.4.6 Sensitivity experiments

Alternative throughput traces To understand how Oboe works on throughput datasets beyond those discussed in §3.4.2, we evaluated Oboe on two other datasets, FCC [124] and HSDPA [125] that have been used in recent work [17, 91]. FCC is a broadband dataset, while HSDPA contains throughput traces collected from video streaming sessions over 3G networks in Norway using mobile devices. Our comparisons use the traces and a Pensieve model pre-trained for those traces available at [114]. We focus our evaluations on MPC+Oboe and Pensieve, given that Pensieve has been shown to out-perform existing ABR schemes including RobustMPC on these traces. Our results show that MPC+Oboe continues to perform better than RobustMPC on these traces. Further, relative to Pensieve, MPC+Oboe improves `QoE-lin` by an average of 6.94% across the FCC dataset and 10.92% across the HSDPA dataset. These improvements are more modest than those in Figure 3.10(a). The vast majority of traces in the FCC and HSDPA set have an average throughput under 3 Mbps (over 95% for FCC and 98% for HSDPA). The results corroborate Figure 3.12 which indicates that MPC+Oboe provides more modest gains over Pensieve when the latter is trained and evaluated on datasets with a narrow throughput range. MPC+Oboe provides larger gains in settings like the traces discussed in §3.4.2, where only 41% traces are under 3 Mbps and 59% are in the 3-6 Mbps range.

Alternative throughput prediction methods Our experiments with RobustMPC rely on throughput prediction based on the harmonic mean of prior throughput samples (following earlier work [17, 91]), with Oboe tuning the configuration to compensate for prediction errors. We next consider if Oboe's benefits hold if RobustMPC were to have more accurate throughput predictions, potentially by using alternate prediction methods [23]. Rather than using a specific prediction technique, we consider an ideal (and unachievable) approach that we denote as Ideal(T), which can exactly predict the average throughput over the next T seconds. Our experiments

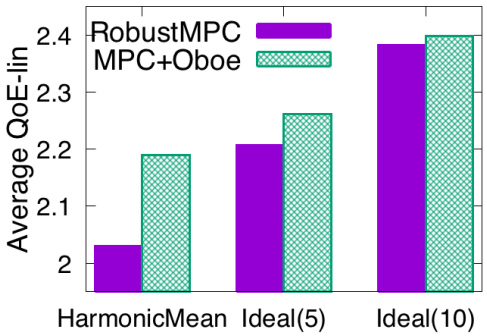


Fig. 3.15.: Average `QoE-lin` of MPC+Oboe with various throughput predictors

were conducted in simulation, using the VirtualPlayer, and the testbed experiment traces (§3.4.2).

Figure 3.15 shows the average `QoE-lin` across the traces for RobustMPC and MPC+Oboe using both the default harmonic mean approach and Ideal(T) for different values of T. Although RobustMPC performs better with an ideal predictor, Oboe still provides benefits, achieving an average improvement in `QoE-lin` of 6.34% for Ideal(5) and of 1.8% for Ideal(10), compared to a 16.1% improvement with the harmonic mean estimator. While the magnitude of benefits is smaller with the ideal prediction approach, in practice Oboe will likely result in larger benefits, since even more sophisticated schemes [23] cannot achieve the ideal predictions, and the errors are likely to grow with larger T.

Oboe can improve performance over RobustMPC even when an Ideal(T) prediction method is used for two reasons. First, T may not match the duration of chunk downloads with RobustMPC, which depends on the exact sequence of bitrates chosen during the look-ahead window. The duration is not known a priori, since RobustMPC itself determines the bitrates based on a provided prediction. Second, the decisions made by RobustMPC are over a small look-ahead window, which may not guarantee optimality over the entire session duration.

3.4.7 Oboe Across Various Settings

In §3.4.5 we have shown that Oboe outperforms other ABR algorithms when compared to their default configurations. We now explore, for HYB, whether Oboe outperforms all parameter settings of HYB and whether it can tune ABRs based on content type and publisher specifications. For these experiments, we use the VirtualPlayer described in §3.3.2.

Comparison against all fixed configurations To explore different fixed configurations, we run HYB with 10 different fixed βs and compare with HYB+Oboe. We summarize the performance for each configuration by considering the (i) median of the average bitrate and the (ii) 90th percentile of the rebuffering ratio across test traces. In this experiment, we also consider an *Oracle* which is the best fixed configuration for each throughput trace with respect to two metrics that HYB tries to optimize.

Figure 3.16(a) and 3.16(b) compare HYB, HYB+Oboe and Oracle over desktop, and mobile traces respectively. While Oracle and HYB+Oboe are depicted as single dots since their performance is uniquely determined, we present a frontier for HYB that shows its performance for different fixed configuration. Figure 3.16(a) shows that HYB+Oboe outperforms HYB in the sense that there is no fixed configuration for HYB that does better than HYB+Oboe performance. HYB+Oboe improves the average bitrates of the median session by 3.2%, while achieving similar rebuffering ratio. Alternately, it reduces the 90th percentile rebuffering ratio from 1.9% to 0%, while maintaining similar bitrates. A similar result holds for mobile traces (Figure 3.16(b)). Thus, even if publishers were to find the best fixed parameter choice for HYB, Oboe would outperform that choice because it dynamically adapts the parameters.

Comparison under different publisher specifications Our results so far are for a VoD (video on demand) setting with a maximum buffer size of 2 minutes. Figure 3.16(c) depicts performance for *live video* (which uses a maximum buffer size

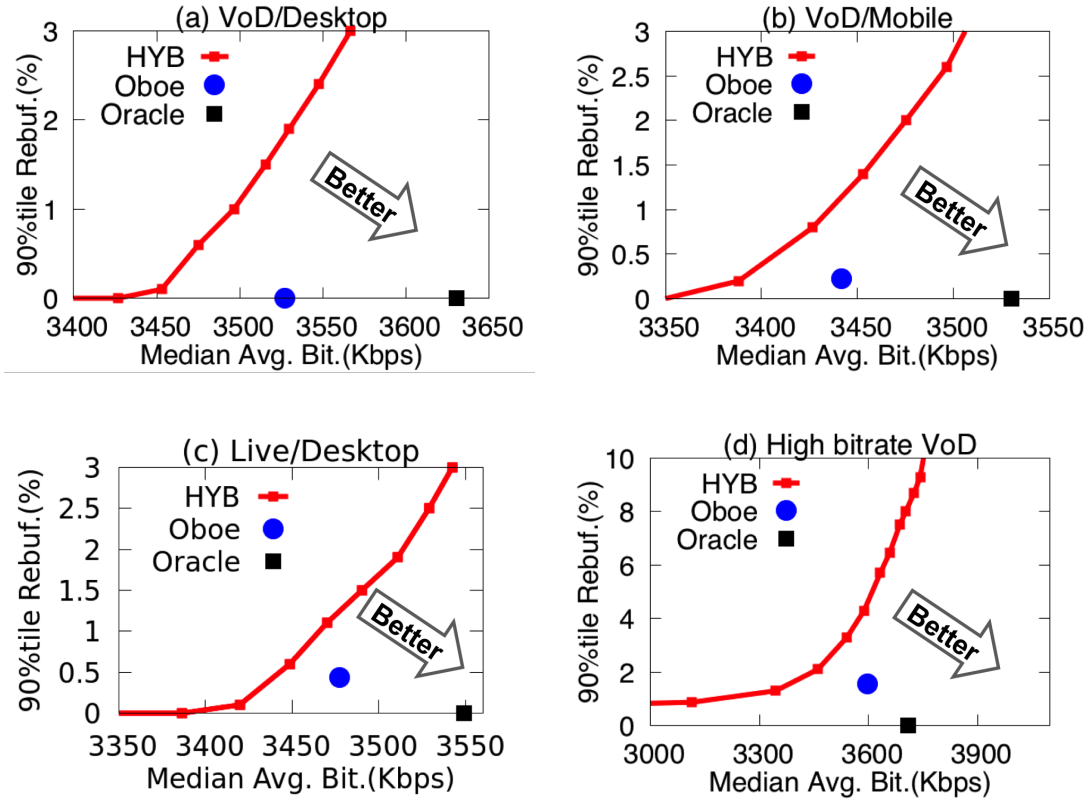


Fig. 3.16.: Comparing HYB with multiple fixed configurations and HYB+Oboe for various settings

of 20 seconds to mimic live settings). HYB+Oboe outperforms HYB for this setting, though we note that the bitrate of both approaches degrades relative to the VoD setting since the baseline HYB switches to higher bitrates more conservatively owing to the smaller buffer sizes.

Finally, Figure 3.16(d) depicts performance for higher bitrate levels ($\{1002, 1434, 2738, 3585, 4661, 5886\} kbps$) and a chunk size of 5 seconds. Even for these choices, HYB+Oboe outperforms HYB, demonstrating its ability to adapt to different publisher specification.

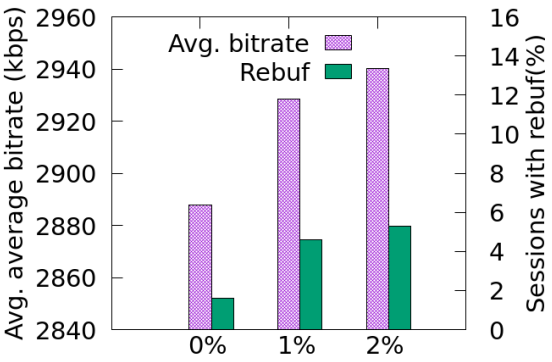


Fig. 3.17.: Avg. of avg. bitrate and fraction of sessions with rebuffering for HYB+Oboe and different publisher preferences

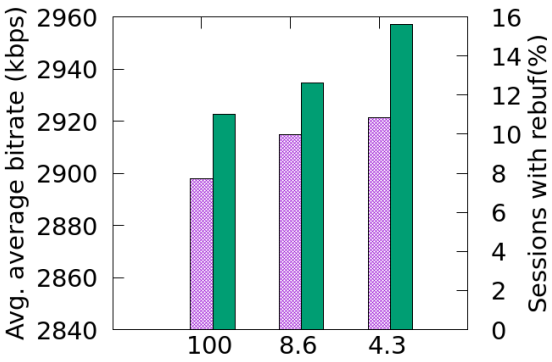


Fig. 3.18.: Avg. of avg. bitrate and fraction of sessions with rebuffering for RobustMPC and different publisher preferences

Accommodating publisher's rebuffering tolerance Oboe allows the publisher to optionally specify explicit rebuffering preferences (§3.3). ABR algorithms such as RobustMPC which use the `QoE-lin` function may permit this indirectly by adjusting `QoE-lin` weights (§3.2.1). Figure 3.17 shows the effectiveness of these approaches, showing the average of average bitrates, and the fraction of sessions with rebuffering for HYB+Oboe. As the publisher makes its rebuffering preference stricter (from 2%-0%), HYB+Oboe achieves lower rebuffering ratios close to the target rebuffering tolerance. In contrast, Figure 3.18 shows that RobustMPC is less effective at controlling rebuffering by adjusting its rebuffering penalty when the weight on the rebuffering term is varied between 100 (strictly avoid rebuffering) to 4.3.[5] We find that even with a very high rebuffering penalty of 100, RobustMPC causes rebuffering in 11% of the sessions. This shows the benefit of Oboe's approach which gives direct control over the underlying metrics.

[5]We used a change penalty of 0 for fair comparison.

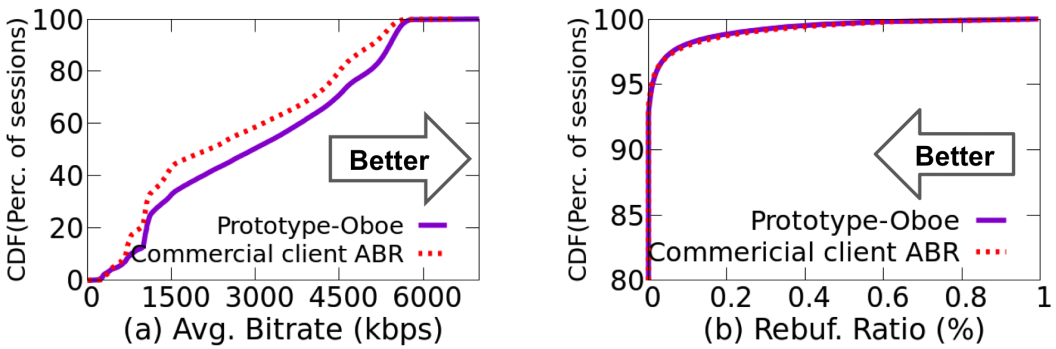


Fig. 3.19.: Comparing prototype Oboe with commercial client side ABR implementation in average bitrate and rebuffering ratio.

3.4.8 Oboe Overhead

Computing the ConfigMap incurs a one-time cost, since the map can be reused across all clients once the it is built. Computing the best parameter configuration for one network state takes about 12 seconds on a single core. This task is perfectly parallelizable, so computing 10K network states (§3.3) will take approximately 3.5 hours to explore with two machines of 48 cores each. We have also analyzed the processing overhead incurred by the ChangeDetector module of Oboe. We measure the time taken by ChangeDetector for every decision cycle across our experiments, and the measurement indicates that the median processing time of ChangeDetector is around 14 ms. Since each decision is made at a chunk boundary and chunks are 4 seconds, ChangeDetector accounts for less than 0.35% overhead.

3.5 Deployment Considerations

The offline stage of Oboe can be run on the cloud, but several choices exist for the online stage, ranging from embedding the online stage entirely in the client player, or moving some or all of the online stage to the cloud. In our implementation, Oboe's components run on the server side. This mimics a cloud implementation, which has

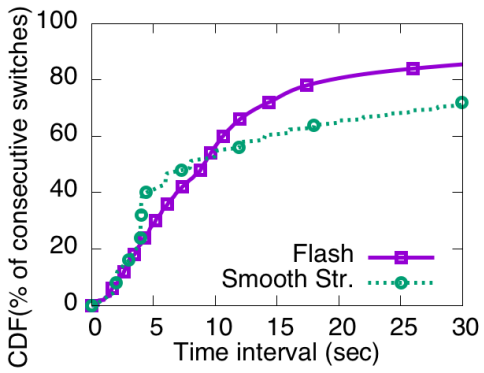


Fig. 3.20.: Time between consecutive bitrate switches for two commercial ABRs

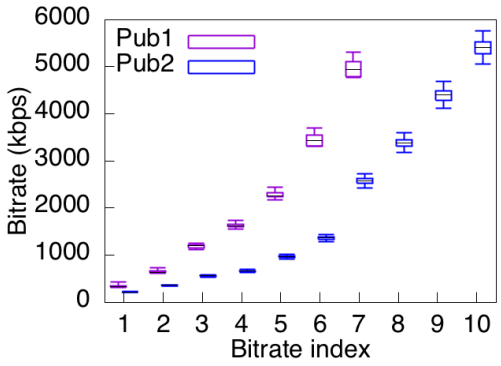


Fig. 3.21.: Variance in bitrate levels across videos from two content publishers.

the benefits of other cloud software: fast update deployment, device independence, etc. [126]. We leave a detailed comparison of these choices to future work, but explore, in this section, the *feasibility* of running the online stage on the cloud.

To this end, we have implemented a restricted version of HYB+Oboe on AWS. This limited version of Oboe implements HYB and incorporates tuning based on publisher specifications but not network state. In our implementation, a client player periodically reports player state (such as buffer length and current bitrate) and throughput samples to a Oboe cloud server and receives bitrate decisions in return. For 10 player features and two chunk downloads per second, the communication overhead is 6.4 Kbps, negligibly small compared to the size of video chunks. Figures 3.19(a) and 3.19(b) compare the performance of this implementation against a client player running HYB over 20K sessions collected during a two-week pilot deployment. Oboe is comparable in performance to the client side player and even improves bitrate slightly (because it was tuned to this publisher's specification).

We expected a cloud implementation would perform worse because of the latency induced by client-server communication. However, we found that most of the bitrate switching decisions occur on timescales much longer than the client-server la-

tencies. Figure 3.20 shows the CDF of the time interval between consecutive bitrate switches for ABR algorithms in two widely used video players(Adobe's Flash [127] and Microsoft Smooth Streaming [128]). The figure shows that over 95% of switching decisions occur at intervals higher than 1 second for both players. This suggests that a cloud-based deployment is viable.

3.6 Discussion and Future Work

Performance improvements for all sessions As our results (*e.g.*, Figure 3.6(a)) show, Oboe improves the performance for most but not all sessions relative to the ABR algorithm it tunes. For instance, after inspecting the results in §3.4.3, we have found that MPC+Oboe typically improves performance relative to RobustMPC by reducing rebuffering and/or the magnitude of bitrate changes, but at the expense of slightly lower bitrates. The resulting `QoE-lin` is improved for most sessions, indicating Oboe does a good job of properly balancing the various factors, but some sessions see lower `QoE-lin`. More generally, designing an ABR approach that can optimize the performance of all sessions is a hard problem that needs more research.

Sharing ConfigMap across videos Oboe need not perform offline precomputation for each individual video, as it can use a single ConfigMap for a *class* of videos that follow a similar bitrate encoding scheme. Figure 3.21 shows that two popular video publishers use similar encoding schemes across two thousand videos each. Publisher 1 uses 7 distinct bitrate levels, and the coefficient of variance across bitrates within each level is only 0.13, while Publisher2, uses 10 distinct bitrate levels, and the coefficient of variance across bitrates within each level is only 0.067. This indicates the potential to share a single ConfigMap across videos.

Generality of Oboe While we have shown that Oboe can tune a variety of configuration parameters across several ABR algorithms, whether Oboe can tune all algorithms and all parameters is an open question. It is unclear if Oboe can directly

augment Pensieve, since a model learned by reinforcement learning may not interact well with intermediaries such as Oboe. However, combining the benefits of Oboe and Pensieve in other ways is an interesting avenue for future work.

3.7 Related Work

Tuning ABR Algorithm Configurations The BBA2 algorithm [90] tunes its lower reservoir based on buffer occupancy dynamics, while MPC [91] adapts its throughput discount factor based on past prediction errors (§3.2). In contrast to such ad-hoc heuristics, Oboe selects configuration parameters based on network state, and publisher specifications. The approach is generically applicable to many ABR algorithms. Newer congestion control protocols like BBR [129] estimate network throughput, which if exposed, could benefit Oboe.

Learning ABR Algorithms Among ABR algorithms that use Reinforcement Learning and other machine learning techniques [17, 130–133], Pensieve [17] has been shown to perform the best. While Pensieve does not specialize to different throughput regimes, Oboe performs better by specializing parameter values for each network state independently.

Other work in self-tuning Beyond ABR algorithms, self-tuning approaches have been explored in other contexts. Winstein et al. [134] used simulations to determine TCP parameters for different settings, while Semke et al. [135] proposed tuning TCP socket buffers to ensure high throughput. More generally, Google Vizier [136] performs such black-box tuning as a service. While Vizier can potentially be used to implement the offline phase of Oboe, our work identifies underlying principles (such as the piecewise stationarity of available throughput) that forms the basis for the tuning.

Video QoE Several researchers have pointed out that sub-optimal ABR performance can significantly impact user-engagement and hence revenue [137,138]. Others have looked at quality issues that occur when multiple players start to compete for bandwidth [6, 33, 139, 140] In contrast, Oboe improves the QoE performance of several ABR algorithms across a range of different network conditions by automatically tuning their parameters.

3.8 Conclusion

Oboe is a system for automatically tuning ABR algorithms by adapting ABR configurations in realtime to match the current network state. Picking configurations in a manner informed by network state and publisher preferences distinguishes Oboe's approach from heuristics used today that do not consider these factors. Oboe significantly improves the performance of BOLA, HYB and RobustMPC; further, for nearly 80% of the sessions in our dataset, Oboe integrated with RobustMPC improves `QoE-lin` relative to Pensieve and the improvements exceed 20% for 25% of the sessions.

4. XATU: EXPLOITING A RICHER THROUGHPUT MODEL FOR VIDEO STREAMING THROUGH NEURAL NETWORKS

4.1 Introduction

Recent years have seen a tremendous growth in Internet video, and by some projections [141], video traffic is expected to account for 82% of all Internet traffic by 2022. While access and core network capacity continues to grow, optimizing Internet video delivery will remain a challenge since we are only beginning to see the adoption of technologies such as 4K video [8] which may involve bitrates of tens of Megabits per second, and since there exists (and will likely remain), a wide disparity in the quality of both fixed and mobile broadband connections across households globally [124].

Video streaming today typically involves splitting video into chunks, each encoded at multiple bitrates. Clients pick bitrates for each chunk based on local estimations of throughput [33]. However, predicting throughput is challenging, often leading to overly conservative bitrate selections, or aggressive selections that result in rebuffering. While much research has sought to tackle these challenges through the design of new Adaptive Bitrate (ABR) algorithms [15–18, 33, 98], the status quo of Internet video streaming is far from satisfactory today. In a recent consumer survey, 34% of respondents reported that they encountered buffering problems once in every three video programs and 24% experienced buffering once in every five videos [142].

In this chapter, we are motivated by two questions: (i) what information can be made available that can aid in predicting throughput perceived by streaming applications (henceforth, referred to as application throughput)? (ii) how do we design frameworks that can leverage such information to improve the accuracy of prediction for streaming applications? We make the following **contributions**:

First, we present observations from an analysis of a dataset of nearly 100K video session traces from real users. The analysis indicates that (i) while features such as the ISP and CDN involved in streaming the video do aid prediction, the state-of-the-art approach [23] of partitioning data by pre-clustering sessions based on a combination of these features and learning only from sessions in the same cluster can hurt prediction accuracy; and (ii) factors such as the Time to First Byte (TTFB) and chunk size impact application throughput. Yet such factors are not explicitly considered today, and are not easily incorporated into the state-of-the-art predictor [23].

Second, motivated by these observations, we present Xatu[1], a prediction framework for video streaming applications. Xatu considers *static* features that do not vary during a session (e.g., ISP, CDN or city), and learns from other sessions with similar static features, but without apriori partitioning data. Further, Xatu models sequences involving multiple *temporal* features that vary across chunks in a session (e.g., TTFB, chunk size), and not just throughput. To achieve its goals, Xatu proposes a gated mask neural network mechanism that uses static features to modify the output of the sequence model, and leverages neural sequence models (specifically, Long Short Term Memory networks (LSTMs) [143]) that can handle multiple temporal features. We show that our output-mask approach gives better accuracy in our task than the traditional approach of combining static and temporal features closer to the input stage of the sequence model used in other domains [144–146]. We also show that these masks help interpret the role played by static features, providing a naturally automated way of clustering sessions in contrast to pre-defining clusters [23].

Third, we evaluate Xatu using a combination of trace-driven experiments using the above dataset, and emulation experiments. Our key results are:

• Xatu reduces the median of the prediction error across sessions relative to CS2P by 23.8%. The benefits come from both the ability of Xatu to exploit static features without pre-clustering data, and from its ability to explicitly model multiple chunk-dependent features including TTFB and size.

[1]Named after a Pokemon character that can see into the future.

• We illustrate the extensibility of Xatu through a case study that explores the impact of the CDN layer an object is served from on application throughput. We show that Xatu can easily exploit new information not traditionally exposed to video streaming applications to improve prediction accuracies.

• To show a potential impact of Xatu when integrated with ABRs, we combine Xatu with MPC [16], a widely studied ABR algorithm. While a more thorough study requires, we show that potential benefit of a composite QoE metric improvement by 38.9% in the median across sessions relative to CS2P based on heuristic approach in order to combine Xatu with ABRs.

4.2 ABR algorithms and prediction

In HTTP-based streaming, videos are split into chunks (corresponding to a few seconds of playtime), with each chunk encoded at multiple bitrates (corresponding to different qualities). A video player selects a bitrate level for each chunk taking into account the amount of data already buffered locally, and a prediction of how long downloading a chunk at a given bitrate would take. In doing so the player seeks to balance multiple video delivery metrics. These metrics include (i) the average bitrate across chunks; (ii) the rebuffering ratio, i.e., the fraction of the video session for which the video player encounters rebuffering; (iii) the extent to which bitrate levels change in the video; and (iv) the join time. A variety of Adaptive Bitrate (ABR) algorithms have been developed to make these decisions [15–18, 24, 33], which primarily differ based on how the algorithms choose which bitrate to use for each chunk by reconciling the above factors.

While some researchers have investigated the design of ABR algorithms that only take client buffer occupancy into account [15,18], the vast majority of ABR algorithms

rely on predictions of chunk download times. We discuss factors that impact chunk download times, and current algorithms.

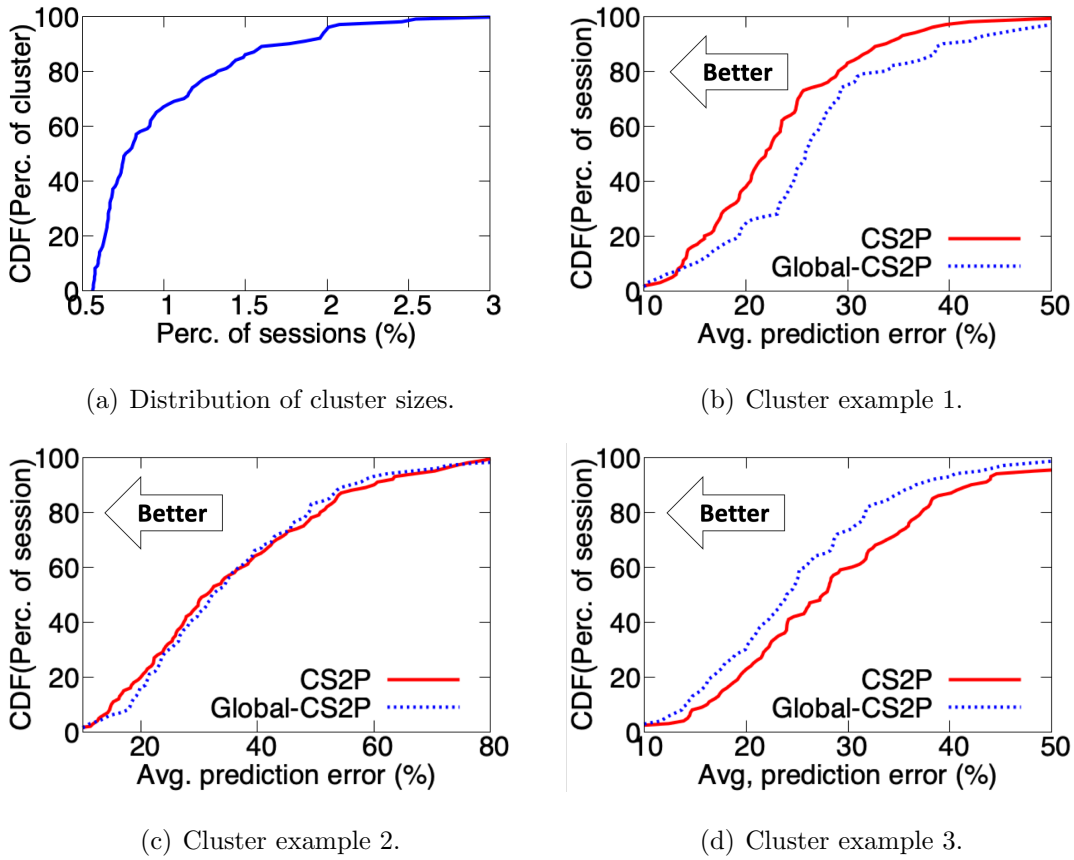
Many factors impact chunk download times. The chunk download time refers to the time from when a HTTP request is sent for a chunk to when the chunk download is finished. The download time D_i of chunk i may be written as:

$$D_i = \text{TTFB}_i + S_i / \text{BW}_i \tag{4.1}$$

Here, TTFB_i represents the Time To First Byte (TTFB) (i.e., the time from when a HTTP request for a chunk is made to when the first byte of a response is received) for chunk i, S_i is the size of the chunk, and BW_i is the network throughput. S_i / BW_i represents the receive time (i.e., the time from when the first byte of the response is received to the last byte).

ABR algorithms mainly focus on throughput. Much of the work in the video streaming literature (e.g., [16, 24, 33]) focus on the application throughput, i.e., the throughput of each downloaded chunk as perceived by the video player ($AppTh_i = S_i/D_i$ using the above notation). These works do not explicitly model the impact of chunk size or TTFB. A commonly used approach, first proposed in Festive [33], and later adopted by others [16, 24], predicts the application throughput of a future chunk based on the harmonic mean of the application throughput of prior chunks. More recently, CS2P [23] (the state-of-the-art throughput prediction approach for ABR streaming), has explored the use of Hidden Markov Model (HMM) for throughput predictions, leveraging the observation that throughput within a session exhibits stateful characteristics and depends on hidden states (e.g., number of flows sharing a bottleneck link). Other works (e.g., the implementation of the Dash.js video player [99]) only consider network throughput and do not consider the impact of TTFB on chunk download times.

The state-of-the-art approach clusters sessions to aid prediction. CS2P [23] also observed that sessions sharing a combination of similar features such as ISP, CDN, user location, and time-of-day tend to have similar throughput patterns. Based on



(a) Distribution of cluster sizes.

(b) Cluster example 1.

(c) Cluster example 2.

(d) Cluster example 3.

Fig. 4.1.: (a) cluster size distribution, and (b)-(d) are throughput prediction error of CS2P and Global-CS2P on example clusters.

these observations, CS2P (i) clusters sessions based on the above features; and (ii) trains a HMM model for each combination of features, rather than using a generic HMM trained across all sessions.

4.3 Motivating data analysis

In this section, we analyze a dataset of real video streaming sessions to understand the implications for prediction approaches used in ABR algorithms today.

Datasets. Our dataset comprises of approximately 100,000 video session traces from real users collected by a streaming TV measurement and intelligence platform. These

measurements were collected over the course of three months in 2017 from a single publisher in the United States that serves short videos. For each video session trace, and for each chunk, the data includes information such as the (i) chunk size; (ii) TTFB and (iii) the start and end times of the download. Each trace also has anonymized IDs representing each of the CDN and ISP that served the video session, an anonymized ID representing the geo-location of user (*e.g.,* city and country), and the time-of-day when the video session started. Video sessions in our dataset are served by 2 CDNs and 89 ISPs across 1406 cities, and an average application throughput of sessions varies from 1.31 Kbps to 94.89 Mbps.

We analyze the data with a view to analyzing two questions (i) does clustering data in a fashion similar to CS2P [23] improve prediction accuracy; and (ii) the extent to which factors such as TTFB and chunk size impact application throughput perceived by video players.

4.3.1 Impact of clustering

We clustered video sessions based on features which include ISP, CDN, city and time-of-day (we use 6 hour windows starting from midnight which is in the range suggested in [23]), and trained a HMM for the specific combination of features as done in CS2P [23].

After clustering data as above, we found many clusters with fewer than 150 sessions, which we filtered as suggested by [23]. After the filtering, we were left with approximately 100 clusters with up to 800 video sessions in each cluster, and a total of 27,000 video traces. Figure 4.1(a) shows a CDF of the percentage of sessions that falls inside each of the remaining clusters. The clusters range in size from about 0.5% of all sessions to about 3% of the sessions. Since CS2P [23] uses each cluster separately in its training, the apriori clustering approach implies that only a small portion of the data is available for the training in any given cluster.

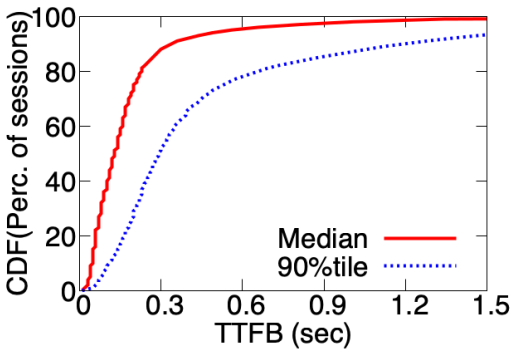
For sessions in each cluster, we compare the error in predicting application throughput using a HMM trained at each cluster (which we refer to as *CS2P*), against that of using a HMM trained over all sessions across all clusters (which we refer to as *Global-CS2P*). We present training methodology details in §4.5.1, and report the normalized absolute application throughput prediction error per session averaged across chunks (more formally defined in §4.6.1). Figure 4.1 compares the prediction error of *CS2P* against *Global-CS2P* for three different clusters. In each graph, a curve to the left corresponds to lower error. While clustering is beneficial in some cases (Fig. 4.1(b)), it does not always provide benefits (Fig. 4.1(c)) and can even hurt (Fig. 4.1(d)).

More generally, we find that clustering helps if the sessions within the cluster have sufficiently similar network characteristics, where limiting the HMM to these similar sessions is beneficial. However, if sessions within a cluster exhibit disparate network characteristics, the benefits of clustering must be weighed against the fact that the overall training data is much smaller. Further, pre-clustering data prevents learning from sessions in other related clusters. For instance, it may be beneficial to learn from sessions in clusters corresponding to slightly different times of day, or nearby cities since these sessions may have similar network patterns. We explore this further in §4.6.1.

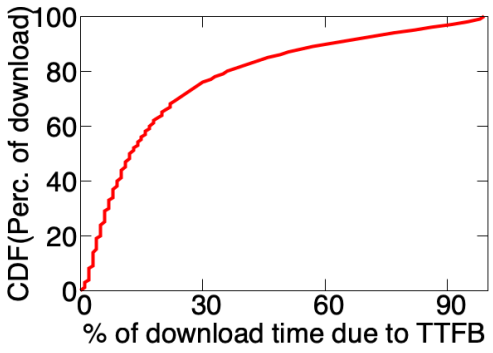
4.3.2 Impact of TTFB and chunk size

To understand the impact of TTFB and chunk size on prediction, we analyze the original dataset but only consider sessions corresponding to the country ID with the most sessions, which was known to be the US. We summarize our findings:

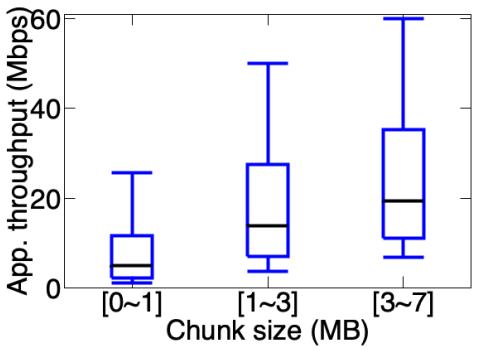
TTFB can significantly impact chunk download times. For each video session, we consider the median TTFB and 90%ile TTFB obtained across all chunks downloaded in that session. Figure 4.2(a) shows the distribution of the median and 90%ile TTFB across the sessions. While the median TTFB for most sessions is small, 23% of the sessions have a median TTFB that exceeds 200 milliseconds, even exceeding 1



(a) Median and 90%ile TTFB per session.



(b) % of download time due to TTFB.



(c) Distribution of application throughput across chunks for different size ranges.

Fig. 4.2.: Impact of TTFB and chunk size on application throughput and chunk download time.

second in a few cases. Further, the 90%ile TTFB of most sessions is high exceeding 300 milliseconds for half the sessions, and exceeding 1 second for nearly 15% of the sessions.

Figure 4.2(b) shows the percentage of the chunk download time that may be attributed to the TTFB across all video chunks of all sessions. For 40% of the chunk downloads, more than 20% of the download time may be attributed to TTFB, while for 15% of chunks downloads, more than half the download time is attributable to

the TTFB. These results indicate that TTFB is a significant contributor to chunk download time. Yet, TTFB is rarely considered in prediction approaches.

Application throughput depends on chunk size. Fig. 4.2(c) shows the application throughput observed by the clients across all chunks in video sessions. The chunks are categorized by size into different ranges, and each boxplot shows the the distribution of application throughput for each chunk size range. The box corresponds to the 25^{th} and 75^{th} percentiles, the horizontal bar to the median, and the end points of the vertical line correspond to the 10^{th} and 90^{th} percentiles. The figure clearly shows that the application throughput tends to be higher for larger chunk sizes, a fact that has not been traditionally modeled in video streaming algorithms.

Modeling the impact of TTFB and chunk size with HMMs is non-trivial. Explicitly modeling the impact of chunk size and TTFB on download times is not straightforward with the state-of-the-art approach for prediction in video streaming [23] which uses HMMs. One approach is to build a separate HMM for each range of chunk sizes. However, this approach is more appropriate if sizes stay within the range throughout the session. Since a HMM models the network as having a sequence of state transitions, and given that the size itself might change during the session, it is not clear how to design a model with multiple HMMs that interact.[2]

Another option is to model network state as a multivariate distribution whose output is the tuple that comprises both the TTFB and the network throughput. This multivariate HMM jointly predicts the TTFB and network throughput for any given network state. For a given chunk size, the download time is appropriately predicted from the predicted TTFB and network throughput, at least in theory. However, when a HMM is trained for this task, it minimizes the prediction error of the tuple TTFB and network throughout, without caring how these prediction errors impact the final download time prediction. Directly predicting the final download time is difficult in HMMs, as we need to explicitly model the inverse distribution of TTFB_i

[2]CS2P [23] was trained on datasets of video streaming sessions where clients picked a bitrate at the start of the session and ABR was not employed - consequently modeling the dependence on size may have been less critical.

and BW_i from Equation (4.1), which does not have a closed form expression for typical assumptions about the distributions of $TTFB_i$ and BW_i. In fact, we did try a multivariate HMM approach and found it led to significantly higher prediction error. Specifically, the average prediction error for the median session with the multivariate HMM was 72.4% in contrast to CS2P using a single HMM based on application throughput which was 43.4%.

4.4 Xatu design

In this section, we present Xatu, which is motivated by the observations in §4.3.

4.4.1 Xatu overview

Xatu's approach is motivated by two key ideas:

Learn from relevant sessions without apriori clustering and data pre-partitioning. Xatu uses features such as the ISP, CDN, time-of-day and city to aid predictions. We henceforth refer to such features as *static features* since they do not change during the session.[3] However, rather than apriori clustering data based on static features (as done by CS2P in §4.3.1), Xatu uses these features as inputs, and learns across all sessions. This allows Xatu to automatically learn from sessions with related but not identical features (e.g., sessions with different cities, or different values of time-of-day that may yet have similar network characteristics).

Model sequences with multiple chunk-dependent features (e.g., TTFB, chunk size). Besides static features, Xatu models multiple features associated with each prior chunk in the session including the TTFB, chunk size, network throughput, and download time. We henceforth refer to these features as *temporal features*, since they change across chunks within a session. The ability to model multiple temporal features distinguishes Xatu from current ABR prediction approaches [23, 33] which

[3]Xatu currently models the CDN as a static feature since switching of CDNs during a session occurs relatively infrequently, but it is easy to move the CDN to a temporal feature if beneficial.

only consider a single temporal feature (application throughput), and do not easily generalize to multiple features (§4.3.2).

Approach. To achieve these goals, Xatu uses a Long Short Term Memory network (LSTM) [143], which is a special type of Recurrent Neural Networks (RNN) capable of learning long-term dependencies. There are two key reasons why we use LSTMs: First, it is a sequence model that can be used to predict the next value —or next few values— in a time series. Second, it is capable of handling multivariate inputs, automatically finding complex (including possibly non-linear) relationships between inputs and outputs. LSTMs have been used successfully in various contexts including language modeling, text classification, time series modeling, and anomaly detection. However, using an off-the-shelf LSTM is not enough. An important design component of Xatu is how we combine both static and temporal features, as seen next.

4.4.2 Xatu Architecture

Xatu takes two classes of inputs: (i) static features such as ISP and location (that do not vary during the session); and (ii) temporal features such as TTFB and download time that vary across chunks. Given n static features of a video session, denoted $s_1^{(j)}, \ldots s_n^{(j)}$, where each feature can take any of an appropriate set of discrete values. Likewise, we use $d_{i,t}^{(j)}$ to denote temporal feature i (TTFB, size, etc.) of chunk t of video session j. We use the notation $d_t^{(j)} = (d_{1,t}^{(j)}, \cdots d_{m,t}^{(j)})$ to refer to an m-dimensional vector that contains all temporal features associated with the t-th chunk of video session j.

In architecting Xatu, an important question is how best to combine the static and temporal features. A conventional approach, used in question and answer systems [147–149], is to use static features in the input, as in the architecture shown in Figure 4.6(a) where the static and temporal features are concatenated at the input stage. However, a key issue with directly adapting this approach to our context is

that the LSTM is not directly aware of which part of the information stays the same and it may take multiple samples to learn the invariant portions of the inputs.

Rather, Xatu introduces an alternate approach to combine static and temporal features, shown in Figure 4.3. The architecture comprises separate static and temporal feature blocks. We use the static features to generate a gate mask: each neuron of the LSTM output (output related to the temporal features) can be turned off depending on the static features. The gate mask directly encodes the effect of static features in the model predictions. In practice, the gate mask is implemented to assume values in the range $(0, 1)$, to ensure that the static input model is differentiable. To our knowledge, the closest work to our gated approach is the tangentially related use of attention for position encoding in Zhang et al. [150]. We empirically evaluate the benefits of our approach (Fig. 4.3) over the architecture in Fig. 4.6(a) in §4.6.2.

Another alternative to our approach is to concatenate the output of the static and temporal feature blocks in Figure 4.3 before going through a fully connected layer. We did not adopt this approach since the static and temporal features interact in ways that are not easy to interpret. Instead, Xatu's approach allows us to interpret the impact of static features as illustrated in §4.6.1.

4.4.3 Design details

We discuss the key building blocks below:

Embedding layer of static features. Figure 4.3 left shows the embedding block that takes the static features of session j and output the mask $z^{(j)}$. It starts with Xatu assigning one-hot encodings to all our n categorical features (e.g., ISP name, CDN name, city, time-of-day, etc.), denoted $s_1^{(j)}, \ldots, s_n^{(j)}$ for session j. We classify the time-of-day into multiple bins (4 bins of 6 hours each in our evaluations), and treat it as a categorical feature because of its cyclic nature. These n inputs are then each passed through n multilayer perceptrons (MLPs), fully connected linear layers with non-polynomial activation functions, $\text{Embed}_1^{(j)}, \ldots, \text{Embed}_n^{(j)}$; there are n

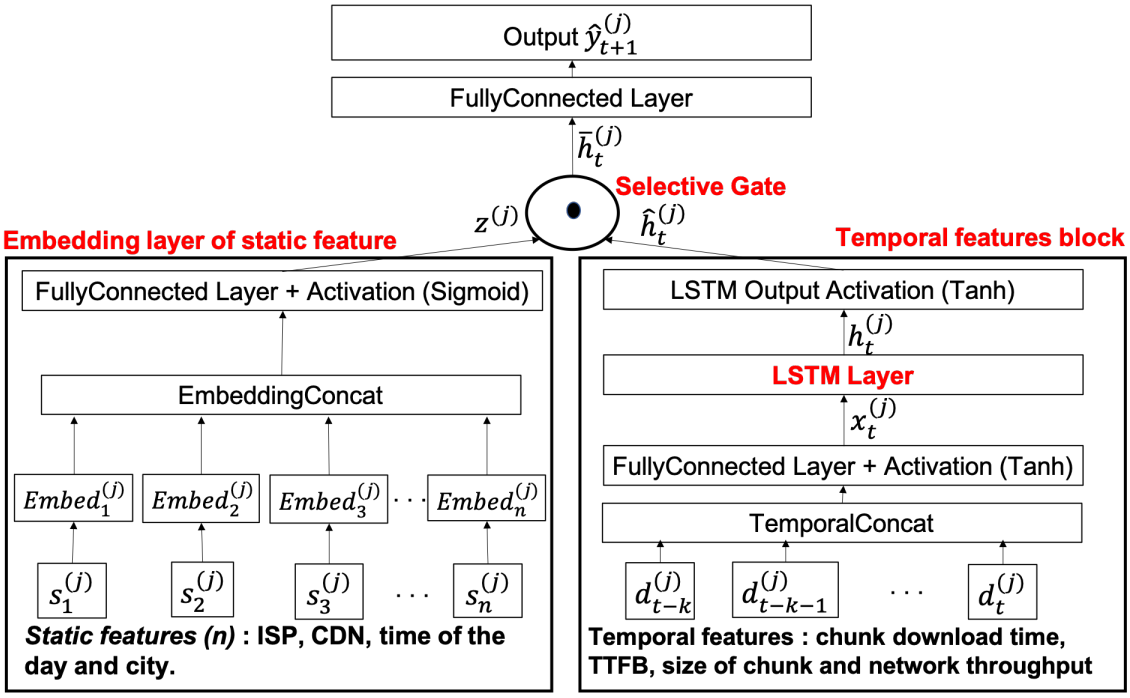


Fig. 4.3.: Xatu architecture.

distinct MLPs, one for each input. The output of these MLPs are then concatenated into a single output and passed through another MLP with sigmoid activations (as many output neurons as there are neurons in the $h_t^{(j)}$ output (Fig. 4.4) of the LSTM), resulting in a final output mask $z^{(j)}$. This mask will be latter elementwise multiplied with $\hat{h}_t^{(j)}$ in the selective gate to ensure the throughput prediction is influenced by the static features of session j.

Temporal features block. Here, we consider predicting the download time of chunk $t+1$ at session j from the sequence $d_1^{(j)}, d_2^{(j)}, \ldots, d_t^{(j)}$ —as defined earlier— and the desired chunk size at time $t+1$. First, our sequence model mixes elements of a k-th order Markov chain with the theoretically infinite memory capacity of an LSTM. That is, rather than only considering the features of chunk t, Xatu considers the features of the past k chunks of the session, where k is a hyperparameter (k is referred to as an *input frame*, and we typically use $k = 5$). So the input of LSTM is the concatenation of temporal features from the last k chunks which results in an

$m \times k$ matrix CONCAT$(d_{t-k}^{(j)}, \ldots, d_t^{(j)})$. Even though in theory an LSTM can learn both long and short term historical dependencies of inputs, explicitly providing the last k chunks ensures it pays close attention to recent history. Doing so rather than just providing the last chunk as input improves prediction accuracy. Another input is the chunk size that has been requested for time $t+1$.

These inputs are fed into an MLP with hyperbolic tangent (tanh) activations outputting vector $x_t^{(j)}$. For completeness, we also show the schematic of the LSTM in Figure 4.4. Let $c_t^{(j)}$ denote the internal memory of the LSTM layer at the end of time step t (in our context, this corresponds to chunk t of a given video session, with the memory being set to null state at the start of the session). Given an input $x_t^{(j)}$ at time step t, the LSTM combines this input with the memory of the previous step ($c_{t-1}^{(j)}$), and the previous output ($h_{t-1}^{(j)}$) to produce an output $h_t^{(j)}$, and updates its memory to $c_t^{(j)}$. In doing so, the LSTM uses multiple gates as described below.

The forget gate layer (with output $f_t^{(j)}$) determines which information from the previous memory is retained and discarded by the LSTM layer. The input gate layer (output $i_t^{(j)}$) decides which values in the memory are updated, while the tanh layer produces $m_t^{(j)}$ which determines new information to be added to the memory. The memory is updated by combining $i_t^{(j)}$ and $m_t^{(j)}$. The output $h_t^{(j)}$ is based on combining the memory (after it goes through a tanh layer), with $o_t^{(j)}$ (the result of the output gate which determines which parts of the memory to output). Finally, the temporal features block outputs $\hat{h}_t^{(j)}$ at download of chunk t after the activation.

Combining output of static and temporal blocks. The final output combines $z^{(j)}$ — which is a function of the static session features $s_1^{(j)}, \ldots, s_n^{(j)}$ — with the output of temporal features block $\hat{h}_t^{(j)}$, resulting in the download time prediction $\hat{y}_{t+1}^{(j)} = \mathbf{w}_{\text{final}}^T \overline{h}_t^{(j)}$, where $\mathbf{w}_{\text{final}}$ is a vector of parameters of the same size as $\overline{h}_t$ and

$\overline{h}_t^{(j)} = z^{(j)} \odot \hat{h}_t^{(j)}$ is the Hadamard product ($\odot$) –element-wise multiplication– between the mask of the static features $z^{(j)}$ and the output of temporal features block $\hat{h}_t^{(j)}$.

Training Xatu. Our whole framework is simply a function

$$\hat{y}_{t+1}^{(j)}, c_t^{(j)}, h_t^{(j)} = F\Big(\mathbf{s}^{(j)}, d_{t-k}^{(j)}, ..., d_t^{(j)}, b_{t+1}^{(j)}, c_{t-1}^{(j)}, h_{t-1}^{(j)}, \mathbf{W}\Big),$$

where $\mathbf{s}^{(j)}$ is the static session features $s_1^{(j)}, \ldots, s_n^{(j)}$, $b_{t+1}^{(j)}$ is the chunk size requested for time $t+1$, $\hat{y}_{t+1}^{(j)}$ is a prediction of $t+1^{\text{st}}$ chunk download time, and $\mathbf{W}$ refers to the set of neural network parameters in all the layers.

To find parameters $\mathbf{W}$ by training Xatu, we first define a loss function to evaluate a prediction accuracy. A loss function L is an sum (or a average) of prediction error l of chunk t

$$L\Big(y_1^{(j)}, \cdots, y_{C^{(j)}}^{(j)}, \hat{y}_1^{(j)}, \cdots, \hat{y}_{C^{(j)}}^{(j)} | \mathbf{W}\Big) = \sum_{t=1}^{C^{(j)}} l\Big(y_t^{(j)}, \hat{y}_t^{(j)}; \mathbf{W}\Big),$$

where $C^{(j)}$ is a total number of chunks downloaded in session j, and $y_t^{(j)}$ and $\hat{y}_t^{(j)}$ are the actual and predicted t-th chunk download times of session j, respectively. The prediction error $l\left(y_t^{(j)}, \hat{y}_t^{(j)}\right)$ can be any form of an error metric such as a square error or an absolute error between a predicted download time and an observed download time. Then we update parameters $\mathbf{W}$ by gradient descent, calculating the loss gradient as

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{t=1}^{C^{(j)}} \frac{\partial}{\partial \mathbf{W}} l\left(y_t^{(j)}, \hat{y}_t^{(j)}; \mathbf{W}\right).$$

The overall optimization samples a session j without replacement from the dataset. Once all sessions are exhausted, an epoch has ended, and we start again by sampling sessions without replacement from the dataset. Because $h_t^{(j)}$ affects all the losses $\left\{l\left(y_{t'}^{(j)}, \hat{y}_{t'}^{(j)}; \mathbf{W}\right)\right\}_{t'>t}$ the computation of the gradient requires a technique called backpropagation through time (BPTT). However, if the length of video session j, $C^{(j)}$, is very long, the gradient can either diverge (explode) or vanish (become zero). To avoid these optimization issues, we split a video session into shorter segments. Note that each BPTT block will use the last hidden state (for both $c_t^{(j)}$ and $h_t^{(j)}$) from

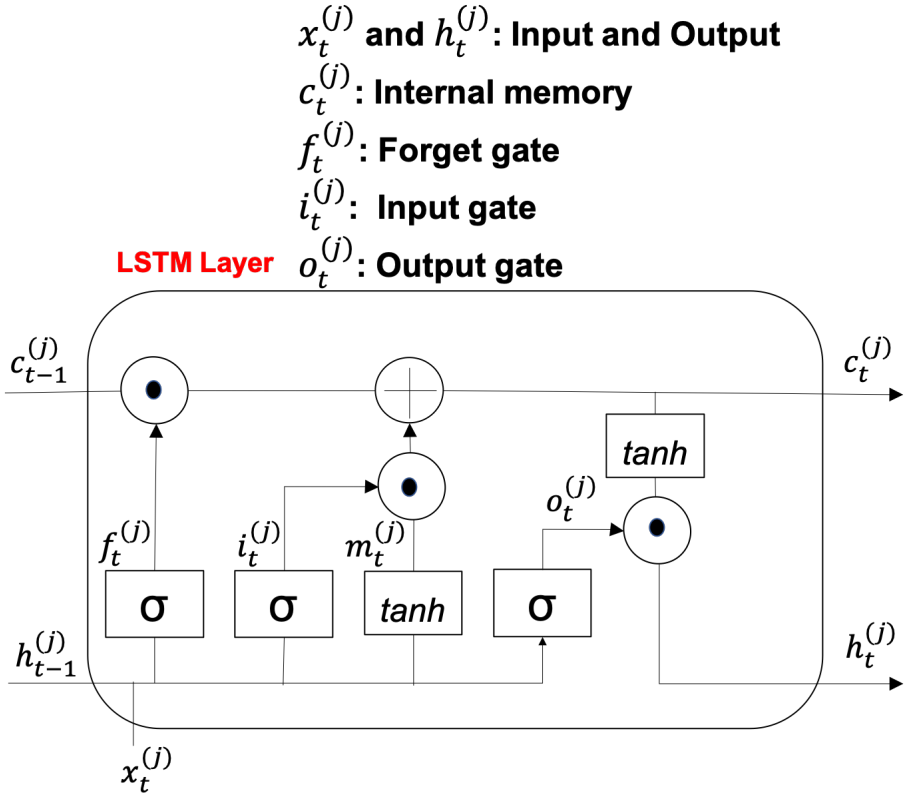


Fig. 4.4.: Detail of LSTM layer.

its previous BPTT block as an initial state, only it will not propagate the gradient back. At the start of a session, variables $c_1^{(j)}$ and $h_1^{(j)}$ are both set to zero.

4.5 Evaluation methodology

Our evaluations compare the throughput prediction accuracy achieved by Xatu relative to CS2P [23] (the state-of-the-art throughput prediction approach for ABR streaming) based on trace-driven experiments. Our trace-driven experiments use the datasets described in §4.3 to evaluate the prediction accuracy achieved with Xatu.

In this section, we explain implementation details of throughput prediction algorithms (Xatu and CS2P) and training methodology we use to evaluate different throughput prediction algorithms.

4.5.1 Prediction schemes compared

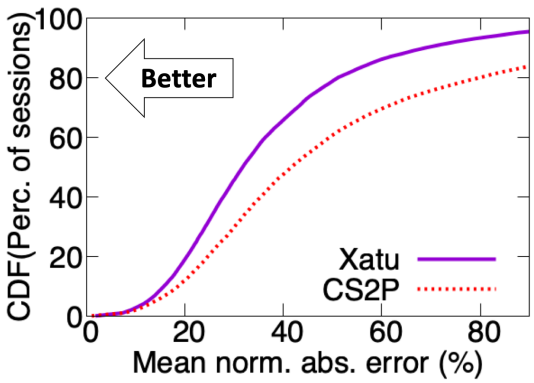
We compared Xatu and CS2P implemented as follows:

Xatu. We implement Xatu using the PyTorch package [151]. We used a BPTT size of ten and an input frame size (k) of five in our implementation. We arrived at these parameters after tuning experiments where we varied each of the parameters between two and twenty, and chose the settings that resulted in the best accuracy in our validation dataset (though we note that the performance was comparable for a wide range of parameter choices). Based also on a hyper-parameter search on the same validations data, we used 516 hidden units for each layer in Xatu. By default, the sum of absolute prediction error in each BPTT, loss $= \Sigma_{p=0}^{B} \mid y_{t-p}^{(j)} - \hat{y}_{t-p}^{(j)} \mid$, where B is the BPTT size. We have also considered other loss functions, as we discuss in §4.6.2.

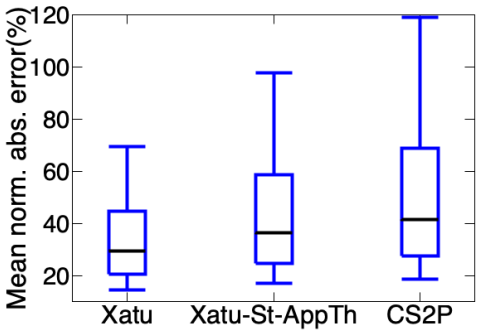
CS2P. We implemented CS2P using the hmmlearn python package [152]. We use the dataset described in §4.3 into clusters using the features described in §4.3 and build a separate HMM for each cluster following [23]. The performance of CS2P depends on the number of hidden states (N), We vary N between 2 and 20 as suggested by [23], and pick the best choice of N for each cluster that results in the lowest average prediction error for that cluster on the validation data.

4.5.2 Datasets and training

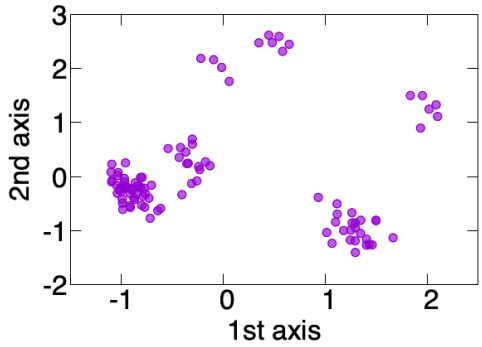
Our evaluations were mainly conducted using data collected from real-world video streaming sessions(§4.3). Specifically, we use the set of 27K traces described in §4.3.1 which we henceforth refer to as the *Primary DataSet*. Recall that this set was obtained after clustering sessions based on a combination of static features and filtering out traces belonging to small clusters(§4.3.1). We use the *Primary DataSet* to ensure that the performance of CS2P was not negatively impacted by the presence of traces from clusters that were too small to allow per-cluster HMM training.



(a) Mean NAE(%) with Xatu and CS2P in the validation set of the *Primary DataSet*.



(b) Understanding how Xatu obtains its benefits.



(c) 2D projection of $z^{(j)}$ for each cluster through PCA.

Fig. 4.5.: Throughput prediction errors with various schemes and understanding Xatu's approach.

For CS2P, we clustered data using the combination of the features mentioned above. For each cluster, we used 60% of the traces for training a HMM for that cluster, 20% for validation, and 20% for testing. Since Xatu does not need a separate model for each cluster, we merged the training sets from all clusters to train Xatu (we used a learning rate of 0.01). Likewise, when we evaluate Global-CS2P which involves a single HMM across all traces, we used the same merged training set across all clusters. Our experiments related to throughput prediction accuracy report results for the validation set.

4.6 Results

In this section, we evaluate the effectiveness of Xatu in improving throughput prediction accuracy compared to CS2P and its interpretability (§4.6.1). We also evaluate the benefits of Xatu's architecture, its sensitivity to various loss functions (§4.6.2), and its capability of specialization .

4.6.1 Xatu vs. CS2P: Prediction accuracy

We compare the prediction error in application throughput by considering each session, taking the normalized absolute error (NAE) per chunk, and computing the mean across chunks. That is, we report

$$\frac{1}{C^{(j)}} \sum_{t=1}^{C^{(j)}} \left| \frac{y_t^{(j)} - \hat{y}_t^{(j)}}{y_t^{(j)}} \right|,$$

where $C^{(j)}$ is the number of chunks downloaded, and $y_t^{(j)}$ and $\hat{y}_t^{(j)}$ are the predicted and actual throughput for chunk t for session j. Fig. 4.5(a) shows the CDF of the mean NAE across sessions for Xatu and CS2P in the validation set of the *Primary DataSet* (§4.5.2). Xatu reduces the median and 90%ile of the mean NAE across sessions by 23.8% and 41.8% respectively. We have also considered a wide range of other metrics to summarize prediction error, and find Xatu consistently out-performs CS2P. For instance, Xatu reduces the median NAE across chunks in a session by 10.3% for the median session, and the 90%ile NAE across chunks by 20% for the median session. Tab. 4.1 summaries improvements in several other prediction error metrics. For instance, we obtained the result in the first row as follows. For each scheme, we computed the mean of the absolute error across chunks for each session, and considered the median across sessions. The table shows that Xatu achieves an improvement of 15.5% for this metric. The other rows may be interpreted similarly.

Table 4.1.: Median improvements of Xatu over CS2P in various prediction error metrics.

Prediction error metrics (per session)	Improvement (in a median session)
Mean of absolute error	15.5%
Median of absolute error	16.2%
90%tile of absolute error	13.8%
Mean of square error	25.2%
90%tile of square error	10.1%

Regardless of error metrics, Xatu constantly improves prediction accuracy ranging from 10.1% to 25.2% over CS2P.

Breaking down factors that help Xatu's performance. There are two factors that help Xatu compared to CS2P. First, rather than pre-partitioning data into clusters, Xatu jointly learns from all data but uses the static features to tailor its predictions. Second, Xatu considers multiple temporal features such as TTFB and chunk size. Recall that these features are not easily incorporated in CS2P's HMM model which only considers application throughput (§4.3.2). To separate these benefits, we consider a variant of Xatu that includes all static features but considers application throughput as the only temporal feature (without considering TTFB and chunk size). We refer this model as *Xatu-St-AppTh*.

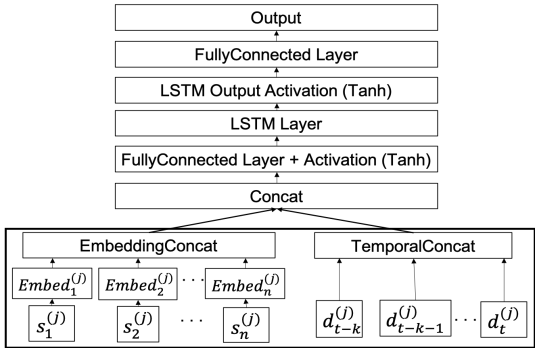
Fig. 4.5(b) shows boxplots which depict the distribution of the mean NAE across sessions for Xatu, Xatu-St-AppTh and CS2P. Across all percentiles, Xatu-St-AppTh achieves lower mean NAE than CS2P, and these benefits may be attributed to Xatu learning across all relevant sessions rather than only learning from sessions in the same cluster. Likewise Xatu achieves lower mean NAE across all percentiles, which

may be attributed to Xatu using all temporal features including TTFB, and chunk size, while Xatu-St-AppTh only considers application throughput.

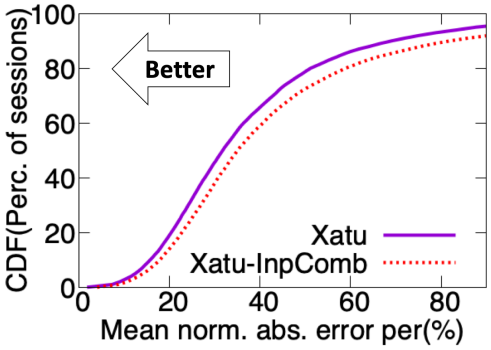
Interpreting results from Xatu. A key aspect of the Xatu architecture is that it facilitates interpretation of how it uses the static features (§4.4.2). We consider the gate mask, $z^{(j)}$ (§4.4.2, Fig. 4.3)) associated with each distinct combination of static features. Fig. 4.5(c) shows results obtained by using Principal Component Analysis (PCA) [153] to project $z^{(j)}$ (a vector of 516 dimensions in our experiments) into a 2 dimensional space to see whether gate masks for different static feature combinations are similar (each dot corresponds to a different combination which is referred to as a cluster by CS2P). Several static feature combinations are bunched together indicating that learning across sessions corresponding to these different combinations may be beneficial, a fact that Xatu exploits. In contrast, CS2P treats sessions corresponding to each combination (cluster) independently. Interestingly, in Fig. 4.5(c), we found that the top three groups of clusters are from one CDN, and the bottom three groups of clusters are from another CDN (we have 2 CDNs in our dataset). For each CDN, the separation into 3 clusters is largely due to time-of-day, (with one cluster corresponding to both 12am-6am, and 6am-12pm). and the other clusters corresponding to 12pm-6pm, and 6pm-12am. These results indicate that Xatu found it important to have specialized predictions for sessions based on their CDN and time-of-day. This further points to the advantage of Xatu's neural network architecture (§4.4.2) which facilitates interpretation of how static features are used.

4.6.2 Sensitivity study

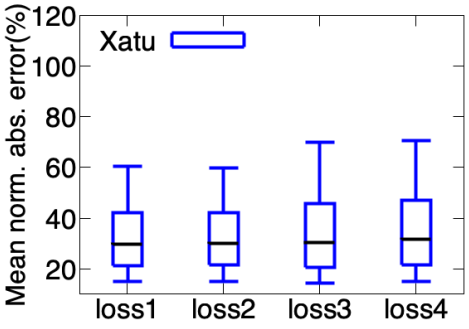
Benefits of Xatu architecture. Xatu uses a neural network architecture that combines static and temporal features at the output with each unique set of static features resulting in a unique gate mask (§4.4.2). In this section, we compare this architecture of Xatu with an alternate architecture (Fig. 4.6(a)) which we refer to as Xatu-InpComb that combines the static and temporal features at the input. We



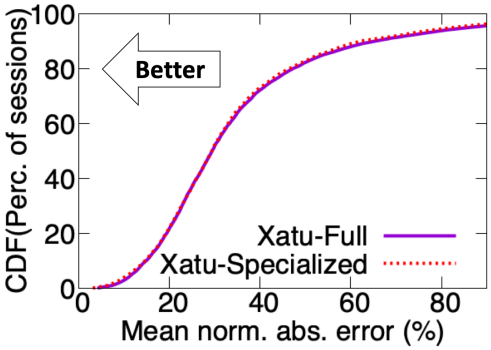
(a) Alternative architecture to Xatu combining static features at the input.



(b) Benefits of Xatu's architecture



(c) Sensitivity of Xatu to loss functions.



(d) Ability of Xatu trained on a larger throughput range to specialize to a smaller throughput range.

Fig. 4.6.: (a)-(b) an ablation study of Xatu architecture, (c) sensitivity of Xatu to various loss functions, and (d) Xatu's Ability of specialization.

perform an ablation study comparing the two architectures. Fig. 4.6(b) shows the CDF of mean NAE for Xatu and Xatu-InpComb. Xatu improves the median and 90%ile prediction error across sessions by 9.4% and 15.7% over Xatu-NoGate, showing the benefits of the gate mask of Xatu in the architecture.

Sensitivity to loss function. Xatu can use any form of loss function during the training (§4.5.1). To check the impact of various loss functions on throughput prediction accuracy of Xatu, we tried four different loss functions; (i) $loss1 = \Sigma_{p=0}^{B} |\, y_{t-p}^{(j)} - \hat{y}_{t-p}^{(j)} \,|$, (ii) $loss2 = loss1/C^{(j)}$, (iii) $loss3 = \Sigma_{p=0}^{B} \left(y_{t-p}^{(j)} - \hat{y}_{t-p}^{(j)} \right)^2$, and

(iv) $loss4 = loss3/C^{(j)}$, where B and $C^{(j)}$ are a size of BPTT and length of session j respectively, and $\hat{y}_t^{(j)}$ and $y_t^{(j)}$ are a predicted and an actual throughput at download of chunk t at session j respectively. While loss1 and loss3 use sum of absolute or square error, loss2 and loss4 divide the sum by a length of a session ($C^{(j)}$ of session j). Fig. 4.6(c) shows the distribution of mean NAE with Xatu across sessions for above loss functions. We conclude that even when the evaluation criteria is NAE, the performance across loss functions shows similar throughput prediction accuracy, pointing to the robustness of Xatu w.r.t. the chosen loss objective.

Ability of specialization. In the §3.4.4, we point out that Pensieve builds a single model which does not *specialize* to different throughput ranges. To evaluate how effectively Xatu can specialize on different throughput range, we build two models of Xatu: (i) Xatu-Full, which is trained with all traces in the full throughput range from 0 to 100Mbps; and (ii) Xatu-Specialized, trained only with a subset of traces in the training set with average throughput in the range 0 to 10Mbps. Both models used the version of Xatu without static features and with limited temporal features as described above. We then evaluated the prediction accuracy achieved by both Xatu models on the validation set, only considering traces in the range 0 to 10 Mbps. Fig. 4.6(d) shows the mean NAE achieved with the two models. The curves are almost indistinguishable and the degradation with Xatu-Full is modest. The median of the mean NAE across sessions increased only 0.12% and while the 90%ile increased by 5.2%. We hypothesize that Xatu specializes better that Pensieve, and this is potentially because it is solving an easier problem (throughput prediction) compared to Pensieve which seeks to determine best bitrates to pick. More advances may be needed to get a deeper understanding of the true ability of a neural network model to specialize in subpopulations of the data.

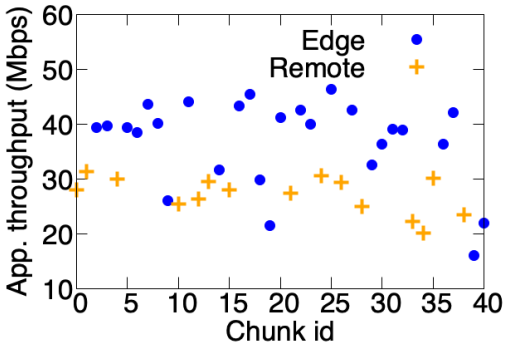
4.7 Extensibility of Xatu to new information

Predicting application throughput only based on information currently available to video players is an inherently hard problem since the underlying factors that impact throughput are not visible to the application. In this section, we consider an example to illustrate (i) how additional auxiliary information not traditionally available to video streaming applications can aid in the prediction; and (ii) show how Xatu easily generalizes to incorporate such information when available.

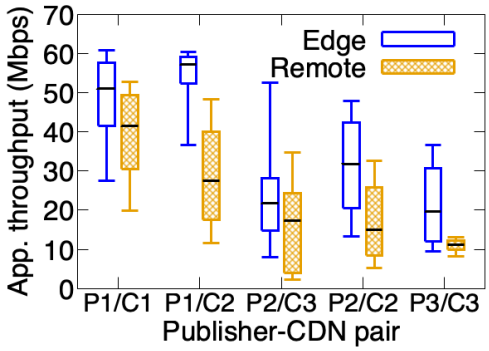
A video content is traditionally served using CDNs, which typically are organized as a hierarchy of caches [154]. A user request that arrives at a server in an edge cluster could "hit" at that server, or experience a cache miss in which case the request may be directed to other upstream servers in the CDN hierarchy, and if necessary, ultimately the CDN origin, or origin server. We explore the interplay between where a video chunk is served from, and the impact on application throughput. We then discuss the implications for prediction.

Controlled measurement. Since the dataset in §4.3 does not include information on which CDN layer a video chunk was retrieved from, we obtain such a dataset using a controlled measurement study. Our study was conducted by streaming 500 videos from 3 different popular video publishers (Twitch, Vimeo and ESPN) from a home network over a 7 day period in October 2018. Two of the publishers provide the number of views for the selected videos, and based on this at least 204 of these videos had more than 10K views.

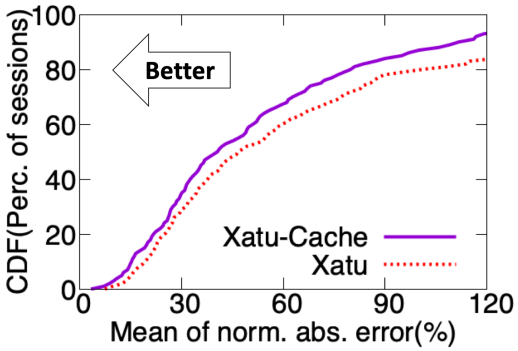
We focused on these publishers because (i) they are popular publishers (within Alexa top 100 US rank [155]); (ii) they provide videos that can be viewed without subscription fees, yet CDN caching is not disabled; and (iii) they use CDNs such as Akamai [156], Fastly [157] and CloudFront [158] that support special HTTP headers which allow us to identify which layer in the hierarchy a video chunk was a hit. For example, for the publishers we considered, CloudFront returns an X-Cache header as part of every HTTP response it serves, which indicates if the object was served



(a) Time series for an example video session.



(b) Throughput distribution for chunks served from *Edge* and *Remote* servers.



(c) Xatu's improvement in prediction accuracy with additional information.

Fig. 4.7.: Benefits of exposing where objects are served from (*Edge* or *Remote*), and ability of Xatu to leverage such information.

by Cloudfront or not. For each video session, and each chunk, we collected the time at which the request was made, the chunk download time, chunk size, and TTFB, and other information that enabled us to identify where the object was served from. More details about the information collected and how we classified which CDN layer a video chunk is served from is discussed in the Appendix. While a more fine-grained

classification is possible, we classify objects as being served from an edge server, or a remote server (henceforth referred to as *Edge*, and *Remote* respectively).[4]

Findings. We next discuss key findings from analyzing the data collected above. Fig. 4.7(a) shows the time series for an example session from Vimeo corresponding to a popular video with more than 228K views. The X-axis indicates the chunk id, and the Y-axis shows the application throughput of each chunk download. For each chunk, we indicate whether it was served from an *Edge* or a *Remote* server using different symbols for each. The graph shows that even within a session, chunks may be served from either an *Edge* or a *Remote* server with no obvious pattern. When all videos with more than 10K views were considered, we found that for 50% of the sessions, 43.1% of chunks or more were served from a *Remote* server. Further, 73.1% of these video sessions involved video chunks retrieved from both an *Edge* and a *Remote* server.

Fig. 4.7(b) shows how the application throughput depends on where video chunks are served from. For each combination of publisher and CDN (e.g., P1/C1 denotes publisher P1 and CDN C1), we show two boxplots that correspond to the distribution of application throughput for chunks being served from an *Edge* or a *Remote* server. Note that each publisher could use multiple CDNs. Across all combinations, the throughput is significantly better when objects are served from an *Edge* server.

Potential for better predictions. We next show the potential to provide better predictions with Xatu by leveraging information when available. We add an additional feature to Xatu which consists of a single bit that indicates whether the next video chunk will be served from an *Edge* or a *Remote* server (we refer to this version as Xatu-Cache). We use 400 of the 500 traces collected from our controlled measurements above as our training set, and the remaining 100 traces as a validation set. We disabled all static features on Xatu and Xatu-Cache, and trained both schemes on

[4]For Cloudfront and Fastly, *Edge* indicates an object is served by the CDN, and *Remote* indicates otherwise (*e.g.,* S3 [159]). For Akamai, we classified an object as being served by a *Remote* server in some cases where multiple servers in the hierarchy experienced a miss, and we were unable to distinguish whether the object was served from a remote CDN server or the origin

the training set. Fig. 4.7(c) shows the mean NAE across sessions in the validation set with both schemes. Xatu improves the median and 90%ile prediction error across sessions by 13.1% and 31.5% respectively.

Architectural changes to facilitate sharing of the information above is an important question in its own right. In the above example, a CDN could provide a video client with information on whether the next chunk can be fetched from an *Edge* server or not. Alternately, if the CDN server were itself to run the ABR algorithm (e.g., [17, 24, 160, 161]) it may be easier to share the information. Nevertheless, the results show the potential benefits of exposing more information to the video application than available today, and the ability of Xatu to easily leverage such information if available to improve prediction accuracies.

4.8 Potential QoE improvement when integrated with ABRs

To show the potential benefits of Xatu in terms of video delivery performance, we have integrated both Xatu and CS2P with MPC, a widely used ABR algorithm [16], replacing its default harmonic mean throughput predictor. To integrate Xatu with MPC, we adopted a heuristic approach since there are potential biases due to causality between chunk sizes and throughputs on our dataset obtained by real-world video sessions while a more thorough approach (*e.g.,* using a casual model) is left for an interesting future direction. In this section, we present the causality issue while integrating Xatu with MPC, an evaluation setup, and potential QoE improvement by Xatu.

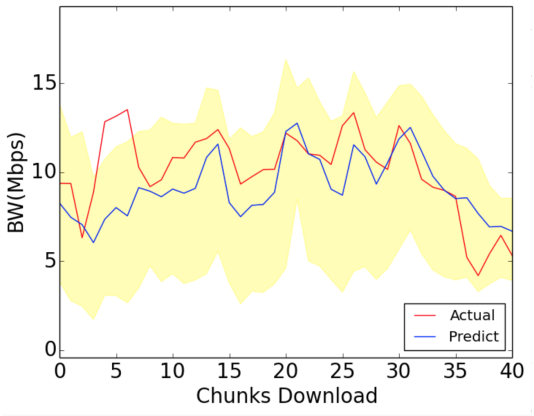


Fig. 4.8.: Actual throughputs (a red line), a range of predicted throughputs (a yellow region) by Xatu depending on chunk sizes, and a predicted throughput based on a heuristic approach (a blue line) obtained from the emulation set-up.

4.8.1 Integrating Xatu with MPC and causality issue between size of chunks and throughputs on the training data

Unlike existing approaches [23, 33], Xatu provides application throughput predictions that model the dependence on the chunk size [5]. However, the model learnt by Xatu is not immune to biases towards predicting lower throughput for smaller chunk sizes, and larger throughput for larger chunk sizes [162, 163]. This is because Xatu has been trained on data that has the actual size selected by a real-world ABR algorithm deployed in the wild, and the size selected by the deployed algorithm correlates with the network conditions. Fig. 4.8 shows actual throughputs (a red line) and a range of predicted throughputs by Xatu depending on chunk sizes obtained from the emulation set-up (descried in §4.8.2). Even though actual throughputs are in a range of predicted throughputs, predicted throughputs vary depending on chunk sizes due to causality issue as we described above.

This problem can occur in any data-driven approaches due to a potential data skew, and can be solved by casual inference [164, 165]. While a more thorough ap-

[5]Given a chunk size, Xatu predicts download time by default, which can be converted to a chunk-size dependent application throughput

proach is possible by creating a casual model, we left it as an interesting future direction and used an intuitive heuristic approach as follow to show potential QoE improvement given such potential biases. Given the fact that actual throughput (a red line in Fig. 4.8) fall well within the range of predicted throughputs depending on chunk sizes (a yellow region in Fig. 4.8), MPC provides the median size of each of the next r chunks for each bitrate candidate. Xatu predicts application throughput for each of these sizes, and MPC uses the median of the set of predicted throughput values (a blue line in Fig. 4.8). While our empirical results with this heuristic approach show QoE improvement, alternate ways to avoid and compensate for the bias may require more thorough studies and provide stronger performance benefits in the future.

4.8.2 Evaluation Testbed

To evaluate the performance of ABR algorithms with different predictors, we create an emulation setup involving video being streamed to a Dash.js video player on Chrome [99] from an Apache server hosting the video. We emulate traces from the testing set described above (§4.5.2), and vary both the network throughput and the TTFB as per the trace. Like previous work [17, 23, 24], we modify the Dash.js video player so requests for which bitrate to select are sent to a server (which we refer to as an ABR server). The server queries the appropriate prediction model (CS2P, or Xatu) and picks the bitrate chosen by MPC for that prediction. Likewise, in the Pensieve experiments, the server queries the Pensieve model to decide the bitrate. The client then requests the appropriate chunk from the video server. Based on the static features (ISP, CDN etc.) of the particular video trace being emulated, the HMM model for that set of features is consulted in CS2P experiments, while the static features are used in the model lookup process for the Xatu experiments.

We used one of the reference videos from DASH in all our experiments which has a total length of 192 seconds [166], with chunk durations of 4 seconds and supporting

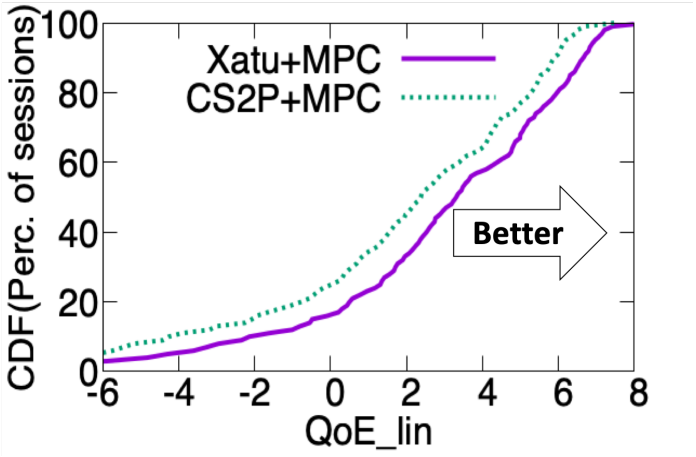


Fig. 4.9.: `QoE-lin` of Xatu+MPC and CS2P+MPC from the testbed emulation on the testing set.

the bitrates {895, 2600, 4729, 9104} Kbps which are corresponding to 480p, 720p, 1080p and 1440p resolution respectively.

The ABR server, video hosting server and a video client run on the same 8-core, 4 Ghz, Intel i7 desktop with 12 GB RAM running Ubuntu 16.04. To emulate different network conditions between servers and client, we used the Chrome DevTools API [167]. This allows us to emulate the network throughput, and RTT between the client and servers using the Chrome-Remote-Interface based on our traces [168]. All our testbed experiments use a client buffer of 1 minute like previous work [17].

Like prior work [16, 23, 24], we used `QoE-lin` to summarize video performance of an ABR scheme which is a linear combination of the average bitrate, rebuffering ratio and the average of bitrate change magnitude. Specifically, $\texttt{QoE-lin}\,(P1, P2) = \frac{1}{C} * \left[\Sigma_{t=1}^{C}\,(R_t - P2 * V_t) - P1 * T\right]$ where C is a number of chunk downloaded in a video session, R_t and V_t are a bitrate and a bitrate change magnitude (Mbps) for a chunk t, T is a total rebuffering time (sec) for the video session, and $P1$ and $P2$ are scaling penalties applied to rebuffering and bitrate change magnitude. We use $P1 = 9.1$ and $P2 = 1$ as our default penalties since the maximum bitrate of our video is 9.1 Mbps.

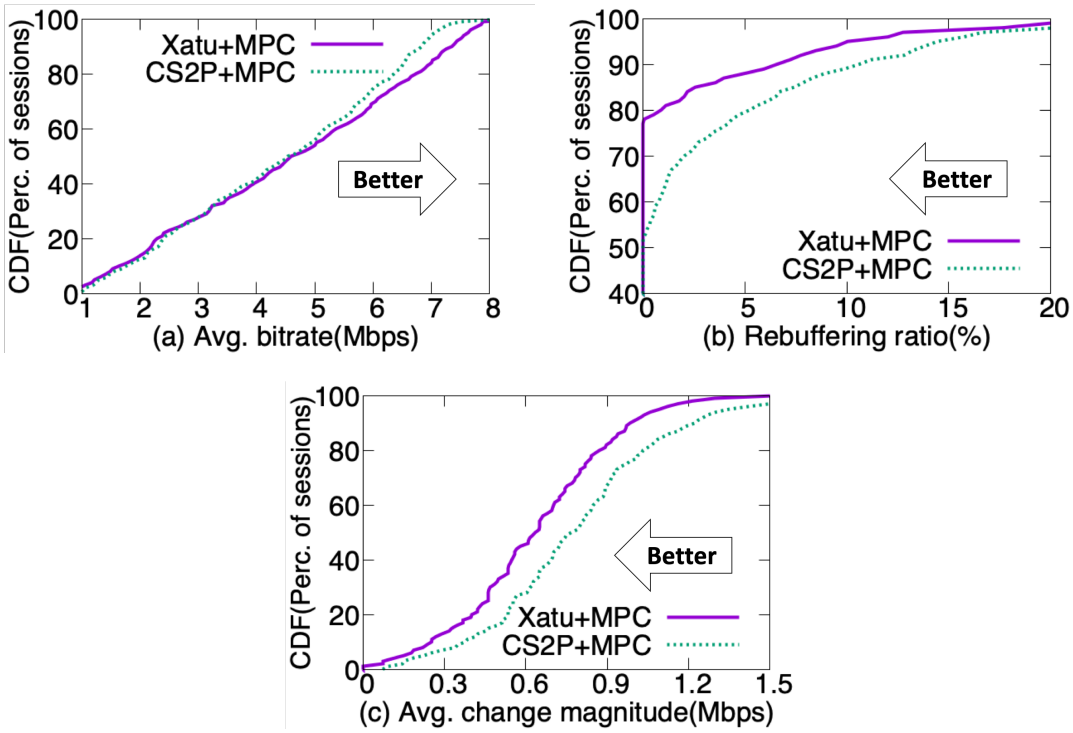


Fig. 4.10.: Individual metrics of `QoE-lin` for Xatu+MPC and CS2P+MPC.

4.8.3 Xatu vs. CS2P: ABR algorithm impact

Following the analysis and heuristic in §4.8.1, we next evaluate whether the improved throughput prediction accuracy of Xatu potentially translates to better performance in terms of video delivery metrics by comparing the performance of the MPC algorithm when integrated with Xatu (Xatu+MPC), and when integrated with CS2P (CS2P+MPC). We conduct these evaluations using the setup described in §4.8.2. We use a random subset of 500 traces chosen from the testing set of the *Primary DataSet* (§4.5.2), but only considering traces that have an average throughput less than 10 Mbps since the highest bitrate of the video is 9.1 Mbps similar to previous works [16, 17, 24].

Fig. 4.9 shows the CDF of `QoE-lin` from the emulation testbed (§4.8.2) on the testing data (§4.5.2). Xatu+MPC improves the median and 90%ile `QoE-lin` by 38.9% and 13.2% respectively over CS2P+MPC. Fig. 4.10(a), Fig. 4.10(b) and Fig. 4.10(c)

show CDFs of the individual components of `QoE-lin`, – the average bitrate (Mbps), rebuffering ratio (%) and the average bitrate change magnitude (Mbps) – for the two schemes. While Xatu+MPC maintains similar average bitrate (higher is better) compared to CS2P+MPC, it reduced the number of sessions that have a rebuffering event (lower is better) by 26% and improves the median of the average bitrate change magnitude (lower is better) by 17.4%.

4.8.4 Xatu+MPC vs. Pensieve

We next compare Xatu+MPC to Pensieve, an ABR algorithm that combines reinforcement learning with deep learning to select bitrates for each chunk. Pensieve has been shown to out-perform MPC, when a harmonic mean predictor is used [17]. We compare its performance with MPC when the more sophisticated Xatu predictor is used.

Comparison methodology. We perform the comparison with a subset of 9K traces chosen from the 27K traces in the *Primary DataSet*. Our comparisons use a smaller set because the Pensieve implementation involves CPU-based training and the training time grows with the number of traces. We used the Pensieve implementation provided by the authors [169] and retrained a Pensieve model using the methodology described in [17, 24] for our dataset.

The current implementation of Pensieve does not consider static features (ISP, CDN etc.) and does not consider TTFB. To ensure a fair comparison, and ensure both approaches consider comparable input features, we consider a limited version of Xatu, where we disabled all static features, and only considered chunk size and download time as the temporal features (disabling TTFB and network throughput).

We trained both Xatu and Pensieve picking 60% of the 9K traces randomly as the training set, and picking 20% for the validation and another 20% for the testing set (with the training, validation and testing sets being the same for both schemes). We only considered traces in the testing set with average throughput under 10 Mbps

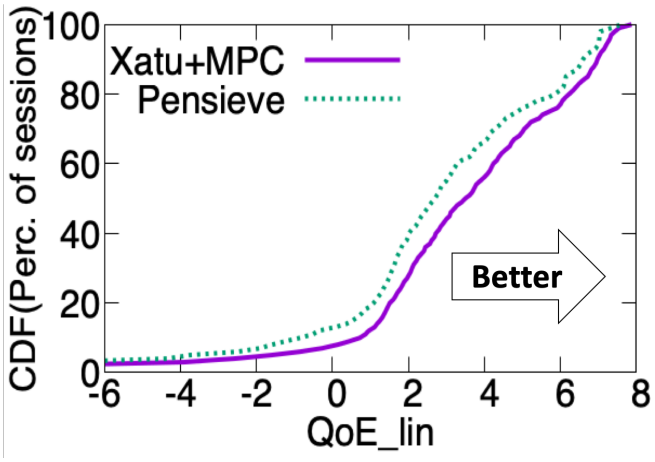


Fig. 4.11.: `QoE-lin` of Xatu+MPC and Pensieve from the testbed emulation. Note these experiments use a subset of the *Primary DataSet* and many features of Xatu are disabled for fair comparison.

like §4.8.3, and select a random subset of 500 of these traces to evaluate the two approaches in the testbed experiment.

Results. Fig. 4.11 shows the CDF of `QoE-lin` of Xatu+MPC and Pensieve from the testbed video streaming emulation on the testing set. Xatu+MPC improves the median and 90%ile `QoE-lin` by 29.2% and 5.8% respectively. We hypothesize that a key contributing factor to this result is the finding that Pensieve is unable to specialize to different throughput ranges in §3.4.4. Specifically, we show that the performance with Pensieve on traces with throughput in the range $(0, T_1)$, $T_1 > 0$, degrades when a model trained on throughput in the range $(0, T_2)$ is used relative to a model trained on the range $(0, T_1)$, where $T1 < T2$. On the other hand, Xatu effectively specializes the model for different throughput ranges as we shown in §4.6.2.

4.9 Related work

ML in adaptive video streaming. Several works [170–173], most notably Pensieve [17], propose ABR algorithms based on Reinforcement Learning (RL) or bayesian

bandits [174] to decide which bitrates to select. In contrast, Xatu uses a neural network for the more limited task of predicting application throughput alone. We have shown that MPC when combined with Xatu performs better than Pensieve, potentially because the Xatu model is able to specialize (§4.8.4), while Pensieve has been shown to not specialize as well [24]. Understanding the underlying causes, and developing ML models that can both specialize and solve the more complex task of bitrate selection is an interesting avenue for future work, and recent developments may hold promise [175].

Complementary video related research. Oboe [24] develops ways to tune ABR algorithm parameters to achieve better performance, and is complementary to Xatu which focuses on prediction. Oboe when combined with Xatu can potentially achieve even better performance than either approach alone. Deep neural networks have been used to learn a mapping from a low quality video to a higher quality version to reduce dependency on bandwidth [176]. Prior work [177] has shown the theoretical possibility of oscillations when video is streamed with a caching server. In contrast, we present measurement results to show video chunks in a session may be served from edge and remote CDN servers in practice, and explore the potential of improving prediction with additional information.

Throughput prediction. Prior work has explored TCP network throughput prediction using Support Vector Regression (SVR) [178] and auto-regression techniques [179], or developed approximate analytical models of TCP throughput [180]. Newer congestion control protocols like BBR [129] estimate network throughput. However, these works focus exclusively on TCP network throughput, which we argue is just one factor that impacts the download time of video chunks. For instance, we have shown that TTFB, and the interplay with cache hierarchy can significantly impact download times. Further, we have compared our approach with CS2P [23] which has been shown to out-perform many of these techniques. While SVR techniques can include both static and dynamic features as inputs, Xatu's approach of combining them at the output offers better interpretability. In the video streaming context [181] argues

the benefits of throughput prediction but does not consider other factors, or provide a solution.

Applications of LSTMs. LSTMs [143] have been widely applied to many domains including language modeling and speech recognition (e.g., [144, 144, 182–184]). Recently, LSTMs have been applied to prediction tasks in networking [185–189]. In contrast, beyond applying LSTMs, Xatu introduces a neural network architecture that combines static and temporal features in an interpretable manner, and empirically explores its benefits(§4.4.2,§4.6.2). Among recent methods for natural language tasks, self-attention mechanism [150] and attention mechanisms of question-and-answer systems (e.g., Transformer [146]) modify the input according to a question we want an answer, or give the question also as input (e.g. BERT [144]). In contrast, Xatu uses a different type of procedure, where the extra static information is placed in a gate mask so that Xatu can use static session information and simultaneously learn from similar sessions in other clusters.

4.10 Conclusion

In this chapter, we make three contributions. First, we have shown that prediction frameworks for video streaming (i) must not only consider network throughput, but also a richer set of temporal features including TTFB, and size; and (ii) should avoid apriori clustering sessions based on static features. Second, we present Xatu, a general framework to achieve these goals which combines LSTMs with a new gated mask neural network mechanism, to jointly learn a neural network sequence model with an interpretable automatic session clustering method. Third, our evaluations show the benefits of Xatu. Relative to CS2P, Xatu improves the median of the mean NAE across sessions by 23.8%. Also we show the benefits of Xatu's neural network architecture, its ability of specialization and its extensibility. Finally we show the potential of QoE improvement when Xatu is integrated with ABRs.

5. CONCLUSIONS

Video dominates today's Internet and will play a larger role going forward. While core and access network capacities continue to grow, optimizing Internet video delivery will remain a challenge despite recent efforts, as new forms of video and technology keep emerging, and content publishers continue to seek higher QoE of users due to its correlations with user engagement and revenue. In this chapter, we present a summary of our key contributions and future directions.

5.1 Contributions

Creating a deeper understanding of video management plane. We shed light on video management plane by characterizing it, at scale, along three key dimensions: streaming protocols, playback devices and platforms and CDNs based on *more than 100 content publishers data spanning 27 months*. We provide a deeper understanding on how each dimension/how many instances of each dimension have evolved over time and across video publishers. We found significant diversity with respect to those dimensions and the increasing trend in diversity. We also take an initial step towards proposing new metrics to measure impacts of diversity of three dimensions on *complexity* of video management plane operations, and found that the complexity of many management tasks is sub-linearly correlated with the number of hours a publisher's content is viewed. Additionally, we demonstrate that today's management plane practices may not be well suited for content *syndication* (*e.g.,* redundancies in CDN storage) through the case study.

Oboe: Enhancing video control plane by increasing the dynamic range of ABRs. We develop Oboe, a system that improves the dynamic range of ABR algorithms by automatically tuning ABR behavior to the current *network state* of a

client connection, specifically to throughput and throughput variability to improve QoE of a wide range of users. We demonstrate several aspects of Oboe performance through real testbed experiments and trace driven simulations, and show the practical viability of this architecture with results from a pilot deployment. we integrated Oboe with a couple of existing ABR algorithms [16,18] such as MPC, BOLA and HYB and showed significantly improvement in several QoE metrics by 7.2% - 38%. Oboe also betters a recently proposed reinforcement learning based ABR [17], Pensieve in part because it is able to better specialize ABR behavior across different network states.

Xatu: Improving video control plane by a better throughput framework. We propose a new throughput prediction framework, Xatu, to address the challenges in existing prediction methods in ABRs. Xatu jointly learns a neural network sequence model with an interpretable automatic session clustering method. Xatu learns clustering rules across all sessions it deems relevant, and models sequences with multiple chunk-dependent features (e.g., TTFB, size) rather than just throughput. We evaluate Xatu over datasets of real video sessions along with trace-driven experiment. Our results show that Xatu significantly improves throughput prediction accuracy by 23.8% relative to CS2P [23] (the state-of-the-art predictor), and Xatu is extensible and can achieve better accuracies if additional information was exposed to video streaming algorithms through an example of hierarchical CDN. We also show potential improvement in QoE by Xatu while integrated with ABRs through emulation experiment based on the heuristic approach.

5.2 Future directions

Network assisted video streaming. QoE of video streaming is largely dependent on a client side logic (*e.g.,* ABRs), and this setting makes it difficult to ensure high quality of video streaming since local inference of network state based only on client side information is challenging, and interaction across multiple video players in the same network without knowing the state of others further complicates the problem.

Given recent trends to facilitate computing at edge networks, deploying some video streaming functionalities into edge networks (*e.g.,* ISP Points of Presence) offers several potential benefits. It allows the incorporation of network-side information as well as other video players in the same network to make an bitrate adaptation decisions in video streaming more efficiently. It is also beneficial for network providers to coordinate their resources across users while maintaining fairness among users.

Pitfalls in data-driven approaches. While recent efforts of data-driven approaches in video streaming, including Xatu, show promising improvements, such approaches could lead to inaccurate or sub-optimal results. As a recent study mentioned [164], one of major sources of this sub-optimality stems from potential data skews. Data collected from the real-world may contain a biased causality or be insufficient to make a reliable model or data-driven decisions. Creating a casual model to capture such potential biases is an interesting future direction.

Emergence of new forms of interactive video. Beyond traditional video streaming, we are starting to see the emergence of interactive video [190, 191] including 360-degree video (e.g., [7]). Such video has much higher bandwidth requirements than regular video streaming [192, 193], and the traditional ABR approach, which can fetch chunks ahead of time for VoD applications, does not suffice since a shift in user orientation may make the content in the client buffer irrelevant. Also, proactively sending video corresponding to multiple orientations may lead to wasted data. In practice, it becomes necessary to make trade-offs with regard to the objectives of responsive interactions and acceptable bandwidth overheads, and modeling such trade-offs efficiently is an interesting future direction.

REFERENCES

REFERENCES

[1] "US Adults Spend 5.5 Hours with Video Content Each Day," http://www.emarketer.com/Article/US-Adults-Spend-55-Hours-with-Video-Content-Each-Day/1012362, Apr. 2016.

[2] "Sandvine Global Internet Phenomena 2016," June 2016.

[3] "The Zettabyte Era—Trends and Analysis," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.pdf, June 2016.

[4] "Cisco: It Came to Me in a Stream... ," https://www.cisco.com/web/about/ac79/docs/sp/Online-Video-Consumption_Consumers.pdf, 2012.

[5] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the Impact of Video Quality on User Engagement," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, 2011.

[6] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12, 2012.

[7] "YouTube 360 Video," https://youtube-creators.googleblog.com/2015/03/a-new-way-to-see-and-share-your-world.html, 2015.

[8] "Can I stream Netflix in ultra hd?" available at https://help.netflix.com/en/node/13444.

[9] G. D. I. Technology, "Emerging Tech Trends: Virtual and Augmented Reality Devices," https://gdit.com/resources/blog/emerging-tech-trends-virtual-and-augmented-reality-devices.

[10] "Bitmovin: Video Developer Survey," https://bitmovin.com/whitepapers/Bitmovin-Developer-Survey.pdf, Sep. 2017.

[11] "DASH-IF: Survey of European Broadcaster on MPEG-DASH," http://dashif.org/wp-content/uploads/2015/04/Survey-of-the-European-Broadcasters-on-MPEG-DASH-Whitepaper-V2.1.pdf, 2013.

[12] "encoding.com," http://1yy04i3k9fyt3vqjsf2mv610yvm-wpengine.netdna-ssl.com/files/2017-Global-Media-Formats-Report.pdf.

[13] "Ooyala: Global Video Index," http://go.ooyala.com/rs/447-EQK-225/images/Ooyala-Global-Video-Index-Q4-2017.pdf.

[14] "Level2: Over the top video delivery," http://www.level3.com/~/media/files/white-paper/en_cdn_wp_ovrtopvddlvry.ashx.

[15] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14, 2014.

[16] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, London, United Kingdom, 2015.

[17] H. Mao, R. Netravali, and M. Alizadeh, "Neural Adaptive Video Streaming with Pensieve," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, 2017.

[18] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal Bitrate Adaptation for Online Videos," in *Proceedings of the IEEE International Conference on Computer Communications*, ser. INFOCOM, 2016.

[19] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang, "C3: Internet-Scale Control Plane for Video Quality Optimization," in *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15*, 2015.

[20] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang, "CFA: A Practical Prediction System for Video QoE Optimization," in *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 16*, 2016.

[21] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, "Practical, Real-time Centralized Control for CDN-based Live Video Delivery," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. ACM, 2015.

[22] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11, 2011.

[23] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, 2016.

[24] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. M. Ribeiro, J. Zhan, and H. Zhang, "Oboe:Auto-tuning video ABR algorithms to network conditions," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18, 2018.

[25] "Conviva," https://www.conviva.com/.

[26] "Sandvine: Global Internet phenomena report ," https://www.sandvine.com/trends/global-internet-phenomena/, 2015.

[27] "Cisco: Visual Networking Index: Global Mobile Data Traffic Forecast Update 2016-2021 ," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html, 2017.

[28] "The Wall Street Journal: YouTube Tops 1 Billion Hours of Video a Day, on Pace to Eclipse TV," https://www.wsj.com/articles/youtube-tops-1-billion-hours-of-video-a-day-on-pace-to-eclipse-tv-1488220851.

[29] "YouTube: You know what's cool? A billion hours," https://youtube.googleblog.com/2017/02/you-know-whats-cool-billion-hours.html.

[30] "Netflix: 2017 on Netflix - A Year in Bingeing," https://media.netflix.com/en/press-releases/2017-on-netflix-a-year-in-bingeingl.

[31] "Recode: Facebook Says Video Is Huge – 100-Million-Hours-Per-Day Huge," https://www.recode.net/2016/1/27/11589140/facebook-says-video-is-huge-100-million-hours-per-day-huge.

[32] "comScore: OTT breaks out of its Netflix shell ," https://www.comscore.com/Insights/Blog/OTT-Breaks-Out-of-Its-Netflix-Shell, 2017.

[33] J. Jiang, V. Sekar, and H. Zhang, "Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, 2012.

[34] "Conviva: The 2017 OTT Streaming Market Year in Review," https://www.conviva.com/blog/2017-ott-streaming-market-year-review/.

[35] M. K. Mukerjee, I. N. Bozkurt, D. Ray, B. M. Maggs, S. Seshan, and H. Zhang, "Redesigning CDN-Broker Interactions for Improved Content Delivery," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17, 2017.

[36] "ITU: H.264," https://www.itu.int/rec/T-REC-H.264.

[37] "ITU: H.265," https://www.itu.int/rec/T-REC-H.265.

[38] "WebM: VP9," https://www.webmproject.org/vp9/.

[39] "Apple HTTP Live Streaming," https://developer.apple.com/streaming/.

[40] "Microsoft: Microsoft Smooth Streaming," http://www.iis.net/downloads/microsoft/smooth-streaming.

[41] "Adobe HTTP Dynamic Streaming," www.adobe.com/products/hds-dynamic-streaming.html.

[42] "DASH-IF: MPEG-DASH," https://mpeg.chiariglione.org/standards/mpeg-dash.

[43] "Apple: fMP4 support on Apple devices," https://developer.apple.com/streaming/examples/, 2016.

[44] "Apple: Technical Note 2224 for HLS Streaming," https://developer.apple.com/library/content/technotes/tn2224/_index.html.

[45] J. Jiang, S. Sun, V. Sekar, and H. Zhang, "Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation," in *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, 2017, pp. 393–406.

[46] M. K. Mukerjee, I. N. Bozkurt, B. Maggs, S. Seshan, and H. Zhang, "The Impact of Brokers on the Future of Content Delivery," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16, 2016.

[47] "Alexa: Top 500 sites on the web," https://www.alexa.com/topsites/category.

[48] V. Almeida, A. Bestavros, M. Crovella, and A. De Oliveira, "Characterizing reference locality in the WWW," in *Parallel and Distributed Information Systems, 1996., Fourth International Conference on.* IEEE, 1996, pp. 92–103.

[49] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 1999, pp. 126–134.

[50] M. Siekkinen, E. Masala, and T. Kämäräinen, "A First Look at Quality of Mobile Live Streaming Experience: The Case of Periscope," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16, 2016.

[51] B. Wang, X. Zhang, G. Wang, H. Zheng, and B. Y. Zhao, "Anatomy of a Personalized Livestreaming System," in *Proceedings of the 2016 Internet Measurement Conference*, ser. IMC '16, 2016.

[52] "Facebook Live," https://live.fb.com/.

[53] "Streaming Learning Center: DASH or HLS? Which is the best format today?" https://streaminglearningcenter.com/blogs/dash-or-hls-which-is-the-best-format-today.html.

[54] "Theoplayer," https://www.theoplayer.com/.

[55] "JW Player," https://www.jwplayer.com/.

[56] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15, 2015.

[57] L. Wei and J. Heidemann, "Does Anycast Hang up on You?" in *IEEE International Workshop on Traffic Monitoring and Analysis*, Dublin, Ireland, 2017.

[58] "The Chromium Projects: Flash Usage Trends," https://www.chromium.org/flash-roadmap/flash-usage-trends.

[59] "Apple: AVFoundation framework ," https://developer.apple.com/av-foundation/.

[60] "Nexplayer: Nexplayer Software Development Kit," https://nexplayersdk.com/.

[61] "Xbox: XDK Software Development Kit," https://www.xbox.com/en-US/developers.

[62] "Unified Streaming," http://www.unified-streaming.com.

[63] "Telestream," http://www.telestream.net/.

[64] "Amazon AWS Media Package," https://aws.amazon.com/mediapackage/.

[65] M. Ghasemi, P. Kanuparthy, A. Mansy, T. Benson, and J. Rexford, "Performance Characterization of a Commercial Video Streaming Service," in *Proceedings of the ACM Conference on Internet Measurement Conference*, ser. IMC, 2016.

[66] J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "Shedding Light on the Structure of Internet Video Quality Problems in the Wild," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13, 2013.

[67] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Understanding Website Complexity: Measurements, Metrics, and Implications," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11, 2011.

[68] T. Benson, A. Akella, and D. Maltz, "Unraveling the Complexity of Network Management," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'09, 2009.

[69] "Conviva: Precision Delivery Intelligence," https://www.conviva.com/whitepapers/.

[70] "Netflix: Per-title Encode Optimization," Dec 2015, https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2.

[71] Balachandran, Athula and Sekar, Vyas and Akella, Aditya and Seshan, Srinivasan and Stoica, Ion and Zhang, Hui, "Developing a Predictive Model of Quality of Experience for Internet Video," in *Proceedings of the 2013 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '13, 2013.

[72] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An Analysis of Facebook Photo Caching," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13, 2013.

[73] V. K. Adhikari, S. Jain, and Z.-L. Zhang, "YouTube Traffic Dynamics and Its Interplay with a Tier-1 ISP: An ISP Perspective," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10, 2010.

[74] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the Expansion of Google's Serving Infrastructure," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13, 2013.

[75] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube Traffic Characterization: A View from the Edge," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07, 2007.

[76] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of YouTube Network Traffic at a Campus Network - Measurements, Models, and Implications," *Comput. Netw.*, Mar. 2009.

[77] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '07, 2007.

[78] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "YouTube Everywhere: Impact of Device and Infrastructure Synergies on User Experience," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11, 2011.

[79] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting Video Server Selection Strategies in the YouTube CDN," in *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ser. ICDCS '11, 2011.

[80] Y. Ding, Y. Du, Y. Hu, Z. Liu, L. Wang, K. Ross, and A. Ghose, "Broadcast Yourself: Understanding YouTube Uploaders," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11, 2011.

[81] J. Zhou, Y. Li, V. K. Adhikari, and Z.-L. Zhang, "Counting YouTube Videos via Random Prefix Sampling," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11, 2011.

[82] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, Z.-L. Zhang, M. Varvello, and M. Steiner, "Measurement Study of Netflix, Hulu, and a Tale of Three CDNs," *IEEE/ACM Trans. Netw.*, vol. 23, no. 6, Dec. 2015.

[83] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "A Hypergiant's View of Internet," in *SIGCOMM Computer Communication Review*, ser. CCR '18, 2018.

[84] H. Abrahamsson and M. Nordmark, "Program Popularity and Viewer Behaviour in a Large TV-on-demand System," in *Proceedings of the 2012 Internet Measurement Conference*, ser. IMC '12, 2012.

[85] J. Erman, A. Gerber, K. K. Ramadrishnan, S. Sen, and O. Spatscheck, "Over the Top Video: The Gorilla in Cellular Networks," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11, 2011.

[86] Z. Li, J. Lin, M.-I. Akodjenou, G. Xie, M. A. Kaafar, Y. Jin, and G. Peng, "Watching Videos from Everywhere: A Study of the PPTV Mobile VoD System," in *Proceedings of the 2012 Internet Measurement Conference*, ser. IMC '12, 2012.

[87] "Streamingmedia: Video: The Pros and Cons of a Multi-CDN Strategy," http://www.streamingmedia.com/Articles/Editorial/Short-Cuts/Video-The-Pros-and-Cons-of-a-Multi-CDN-Strategy-112351.aspx.

[88] K. Fukuda, H. Asai, and K. Nagami, "Tracking the Evolution and Diversity in Network Usage of Smartphones," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15, 2015.

[89] T. Petsas, A. Papadogiannakis, M. Polychronakis, E. P. Markatos, and T. Karagiannis, "Rise of the Planet of the Apps: A Systematic Study of the Mobile App Ecosystem," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13, 2013.

[90] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, 2014.

[91] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, 2015.

[92] "Toward A Practical Perceptual Video Quality Metric," https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652.

[93] H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz, "Analyzing Stability in Wide-area Network Performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 25, pp. 2–12, 1997.

[94] Y. Zhang and N. Duffield, "On the Constancy of Internet Path Properties," in *Proceedings of the ACM SIGCOMM Workshop on Internet Measurement*, 2001.

[95] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante, "Characterizing and Predicting TCP Throughput on the Wide Area Network," in *IEEE International Conference on Distributed Computing Systems*, ser. ICDCS, 2005.

[96] J. Jobin, M. Faloutsos, S. K. Tripathi, and S. V. Krishnamurthy, "Understanding the Effects of Hotspots in Wireless Cellular Networks," in *Proceedings of the Conference of the IEEE Computer and Communications Societies*, ser. INFOCOM, 2004.

[97] G. Urvoy-Keller, "On the Stationarity of TCP Bulk Data Transfers." in *Proceedings of the Passive and Active Measurement Conference*, ser. PAM, 2005.

[98] G. Tian and Y. Liu, "Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '12*, 2012.

[99] "DASH Industry Forum," https://github.com/Dash-Industry-Forum/dash.js.

[100] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, T. Karim, E. Katz-Bassett, and R. Govindan, "An Internet-Wide Analysis of Traffic Policing," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, 2016.

[101] H. Pishro-Nik, *Introduction to Probability, Statistics and Random Processes*. Kappa Research, 2014.

[102] W. A. Fuller, *Introduction to Statistical Time Series*. John Wiley and Sons, 1976.

[103] "Bayesian Changepoint Detection," https://github.com/hildensia/bayesian_changepoint_detection.

[104] R. P. Adams and D. J. MacKay, "Bayesian Online Changepoint Detection," in *arXiv:0710.3742v1*, 2007.

[105] L. Wei and E. Keogh, "Semi-supervised Time Series Classification," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, ser. SIGKDD, 2006.

[106] F. Desobry, M. Davy, and C. Doncarli, "An Online Kernel Change Detection Algorithm," *IEEE Transactions on Signal Processing*, vol. 53, no. 8, pp. 2961–2974, 2005.

[107] D. R. Jeske, V. Montes De Oca, W. Bischoff, and M. Marvasti, "Cusum Techniques for Timeslot Sequences with Applications to Network Surveillance," *Computational Statistics and Data Analysis*, vol. 53, pp. 4332–4344, 2009.

[108] K. Yamanishi and J.-i. Takeuchi, "A Unifying Framework for Detecting Outliers and Change Points from Non-stationary Time Series Data," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, ser. SIGKDD, 2002.

[109] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani, "An Online Algorithm for Segmenting Time Series," in *Proceedings of the IEEE International Conference on Data Mining*, ser. ICDM, 2001.

[110] T. Rakthanmanon, E. J. Keogh, S. Lonardi, and S. Evans, "Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data," in *Proceedings of the International Conference on Data Mining*, ser. ICML, 2011.

[111] D. Barry and J. A. Hartigan, "A Bayesian Analysis for Change Point Problems," *Journal of the American Statistical Society*, vol. 88, no. 421, pp. 309–319, 1993.

[112] P. Fernhead, "Exact and Efficient Bayesian Inference for Multiple Changepoint Problems," *Statistics and Computing*, vol. 16, no. 2, pp. 203–213, 2006.

[113] X. Xiang and K. Murphy, "Modelling Changing Dependency Structure in Multivariate Time Series," in *Proceedings of the International Conference on Data Mining*, ser. ICML, 2007.

[114] "Pensieve," https://github.com/hongzimao/pensieve.

[115] "DASH Industry Forum," https://dash.akamaized.net/envivio/EnvivioDash3.

[116] "Google-Chrome: Chrome DevTools Protocol," https://chromedevtools.github.io/devtools-protocol/tot/Network/.

[117] "Chrome Remote Interface," https://github.com/cyrus-and/chrome-remote-interface.

[118] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[119] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of the International Conference on Machine Learning*, ser. ICML, 2016.

[120] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[121] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," in *Proceedings of the Association for Advancement of Artificial Intelligence*, ser. AAAI, 2018.

[122] R. J. Williams and J. Peng, "Function Optimization using Connectionist Reinforcement Learning Algorithms," *Connection Science*, vol. 3, no. 3, pp. 241–268, 1991.

[123] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, ser. SIGKDD, 2016.

[124] "Federal Communications Commission. Raw Data - Measuring Broadband America." www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016.

[125] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications," in *Proceedings of the ACM Multimedia Systems Conference*, ser. MMSys, 2013.

[126] "Oracle: 5 Reasons to Consider SaaS for Your Business Applications," http://www.oracle.com/us/solutions/cloud/saas-business-applications-1945540.pdf.

[127] "Adobe OSMF player," http://www.osmf.org.

[128] "Microsoft Smooth Streaming," http://www.iis.net/downloads/microsoft/smooth-streaming.

[129] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, pp. 20–53, 2016.

[130] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A Learning-based Algorithm for Improved Bandwidth-awareness of Adaptive Streaming Clients," in *Symposium on Integrated Network Management*, ser. IM, 2015.

[131] V. Martín, J. Cabrera, and N. García, "Design, Optimization and Evaluation of a Q-learning HTTP Adaptive Streaming Client," *IEEE Transactions on Consumer Electronics*, vol. 62, no. 4, pp. 380–388, 2016.

[132] M. Claeys, S. Latré, J. Famaey, T. Wu, W. V. Leekwijck, and F. D. Turck, "Design and Optimisation of a (FA)Q-learning-based HTTP Adaptive Streaming Client," *Connection Science*, vol. 26, no. 1, pp. 25–43, 2014.

[133] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard, "Online Learning Adaptation Strategy for DASH Clients," in *Proceedings of the International Conference on Multimedia Systems*, ser. MMSys, 2016.

[134] K. Winstein and H. Balakrishnan, "TCP Ex Machina: Computer-generated Congestion Control," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, 2013.

[135] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP Buffer Tuning," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, 1998.

[136] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google Vizier: A Service for Black-Box Optimization," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*, ser. SIGKDD, 2017.

[137] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "Developing a Predictive Model of Quality of Experience for Internet Video," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM, 2013.

[138] S. S. Krishnan and R. K. Sitaraman, "Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs," in *Proceedings of the ACM Conference on Internet Measurement Conference*, ser. IMC, 2012.

[139] R. Houdaille and S. Gouache, "Shaping HTTP Adaptive Streams for a Better User Experience," in *Proceedings of the Multimedia Systems Conference*, ser. MMSys, 2012.

[140] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis, "What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth?" in *the International Workshop on Network and Operating System Support for Digital Audio and Video*, ser. NOSSDAV, 2012.

[141] "Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[142] "New research reveals buffer rage as tech's newest epidemic," https://www.prnewswire.com/news-releases/new-research-reveals-buffer-rage-as-techs-newest-epidemic-300237001.html.

[143] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[144] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[145] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," *ICLR*, 2017.

[146] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," in *NIPS*, 2017.

[147] A. Jabri, A. Joulin, and L. Van Der Maaten, "Revisiting visual question answering baselines," in *European conference on computer vision*. Springer, 2016, pp. 727–739.

[148] M. Malinowski, M. Rohrbach, and M. Fritz, "Ask your neurons: A neural-based approach to answering questions about images," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1–9.

[149] D. Wang and E. Nyberg, "A long short-term memory model for answer sentence selection in question answering," in *ACL*, vol. 2, 2015, pp. 707–712.

[150] Y. Zhang, V. Zhong, D. Chen, G. Angeli, and C. D. Manning, "Position-aware attention and supervised data improve slot filling," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 35–45.

[151] "PyTorch," https://pytorch.org/.

[152] "hmmlearn," https://hmmlearn.readthedocs.io/en/latest/#.

[153] "Principal component analysis." https://en.wikipedia.org/wiki/Principal_component_analysis.

[154] S. Puzhavakath Narayanan, Y. S. Nam, A. Sivakumar, B. Chandrasekaran, B. Maggs, and S. Rao, "Reducing latency through page-aware management of web objects by content delivery networks," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 1. ACM, 2016, pp. 89–100.

[155] "US Alexa Rank." https://www.alexa.com/topsites/countries/US.

[156] "Akamai," https://www.akamai.com/.

[157] "Fastly," https://www.fastly.com/.

[158] "Amazon CloudFront," https://aws.amazon.com/cloudfront/?nc=sn&loc=0.

[159] "Amazon S3." https://aws.amazon.com/s3/.

[160] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "A case for a coordinated internet video control plane," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 359–370, 2012.

[161] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang, "C3: Internet-Scale Control Plane for Video Quality Optimization," in *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15*, 2015.

[162] J. Pearl, *Causality*. Cambridge university press, 2009.

[163] S. S. Krishnan and R. K. Sitaraman, "Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs," in *Proceedings of the ACM Conference on Internet Measurement Conference*, ser. IMC, 2012.

[164] M. Bartulovic, J. Jiang, S. Balakrishnan, V. Sekar, and B. Sinopoli, "Biases in data-driven networking, and what to do about them," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017, pp. 192–198.

[165] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering what-if deployment and configuration questions with wise," in *ACM SIGCOMM'08*, 2008. [Online]. Available: http://dx.doi.org/10.1145/1402946.1402971

[166] "DASH IF Test Assets Database," http://testassets.dashif.org/#testvector/list.

[167] "Google-Chrome: Chrome DevTools Protocol." https://chromedevtools.github.io/devtools-protocol/tot/Network/.

[168] "Chrome Remote Interface." https://github.com/cyrus-and/chrome-remoteinterface.

[169] "Pensieve Github." https://github.com/hongzimao/pensieve.

[170] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard, "Online learning adaptation strategy for dash clients," in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys '16. New York, NY, USA: ACM, 2016, pp. 8:1–8:12. [Online]. Available: http://doi.acm.org/10.1145/2910017.2910603

[171] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck, "Design of a q-learning-based client quality selection algorithm for http adaptive video streaming," in *Proceedings of the 2013 Workshop on Adaptive and Learning Agents (ALA), Saint Paul (Minn.), USA*, 2013, pp. 30–37.

[172] ——, "Design and optimisation of a (fa) q-learning-based http adaptive streaming client," *Connection Science*, vol. 26, no. 1, pp. 25–43, 2014.

[173] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 131–138.

[174] B. Alt, T. Ballard, R. Steinmetz, H. Koeppl, and A. Rizk, "Cba: Contextual quality adaptation for adaptive bitrate video streaming (extended version)," *arXiv preprint arXiv:1901.05712*, 2019.

[175] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," *arXiv preprint arXiv:1807.02264*, 2018.

[176] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han, "Neural adaptive content-aware internet video delivery," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI) 18)*, 2018, pp. 645–661.

[177] D. H. Lee, C. Dovrolis, and A. C. Begen, "Caching in http adaptive streaming: Friend or foe?" in *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. ACM, 2014, p. 31.

[178] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A machine learning approach to tcp throughput prediction," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1. ACM, 2007, pp. 97–108.

[179] Q. He, C. Dovrolis, and M. Ammar, "On the predictability of large transfer tcp throughput," in *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4. ACM, 2005, pp. 145–156.

[180] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling tcp throughput: A simple model and its empirical validation," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 303–314, 1998.

[181] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha, "Can accurate predictions improve video streaming in cellular networks?" in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM, 2015, pp. 57–62.

[182] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[183] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[184] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The microsoft 2017 conversational speech recognition system," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5934–5938.

[185] L. Mei, R. Hu, H. Cao, Y. Liu, Z. Han, F. Li, and J. Li, "Realtime mobile bandwidth prediction using lstm neural network," in *International Conference on Passive and Active Network Measurement*. Springer, 2019, pp. 34–47.

[186] A. Narayanan, S. Verma, E. Ramadan, P. Babaie, and Z.-L. Zhang, "Making content caching policies' smart'using the deepcache framework," *ACM SIGCOMM Computer Communication Review*, vol. 48, no. 5, pp. 64–69, 2019.

[187] H. D. Trinh, L. Giupponi, and P. Dini, "Mobile traffic prediction from raw data using lstm networks," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. IEEE, 2018, pp. 1827–1832.

[188] C. Zhang and P. Patras, "Long-term mobile traffic forecasting using deep spatio-temporal neural networks," in *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2018, pp. 231–240.

[189] A. Azzouni and G. Pujolle, "Neutm: A neural network-based framework for traffic matrix prediction in sdn," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–5.

[190] G. Cheung, A. Ortega, N.-M. Cheung, and B. Girod, "On media data structures for interactive streaming in immersive applications," in *Visual Communications and Image Processing 2010*. International Society for Optics and Photonics, 2010.

[191] N.-M. Cheung and G. Cheung, "Coding for interactive navigation in high-dimensional media data," *Emerging Technologies for 3D Video: Creation, Coding, Transmission and Rendering*, pp. 162–186, 2013.

[192] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. ACM, 2016, pp. 1–6.

[193] X. Liu, Q. Xiao, V. Gopalakrishnan, B. Han, F. Qian, and M. Varvello, "360 innovations for panoramic video streaming," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017, pp. 50–56.

VITA

VITA

Yun Seong Nam is a Ph.D. Candidate in the School of Electrical and Computer Engineering at Purdue University, advised by Professor Sanjay Rao. His research interests are in Computer Networking Systems. His research currently focuses on techniques and novel architectures to deliver high quality video across diverse and variable network conditions. Prior to his doctoral studies, he received his M.S and B.S. degree from Columbia University, NYC, U.S. and Yonsei University, Seoul, S. Korean respectively.